

**OCEANBASE**



# OceanBase 数据库

## 数据迁移

| 产品版本：V4.3.5


| 文档版本：20250106

# 声明

**北京奥星贝斯科技有限公司版权所有©2024，并保留一切权利。**

未经北京奥星贝斯科技有限公司事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

## 商标声明

 **OCEANBASE** 及其他 OceanBase 相关的商标均为北京奥星贝斯科技有限公司所有。本文档涉及的第三方的注册商标，依法由权利人所有。

## 免责声明

由于产品版本升级、调整或其他原因，本文档内容有可能变更。北京奥星贝斯科技有限公司保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在北京奥星贝斯科技有限公司授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过北京奥星贝斯科技有限公司授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

# 通用约定

格式	说明	样例
 危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 危险 重置操作将丢失用户配置数据。
 警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 警告 重启操作将导致业务中断，恢复业务时间约十分钟。
 注意	用于警示信息、补充说明等，是用户必须了解的内容。	 注意 权重设置为0，该服务器不会再接受新请求。
 说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击 <b>设置&gt; 网络&gt; 设置网络类型</b> 。
<b>粗体</b>	表示按键、菜单、页面名称等UI元素。	在 <b>结果确认</b> 页面，单击 <b>确定</b> 。
Courier字体	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <code>Instance_ID</code>
[ ] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

# 目录

1	数据迁移概述 .....	16
1.1	应用场景 .....	16
1.2	迁移方案 .....	16
1.2.0.1	说明 .....	16
2	使用 OMS 从 MySQL 数据库迁移数据到 OceanBase 数据库 MySQL 租户 .....	18
2.1	背景信息 .....	18
2.2	相关文档 .....	18
3	使用 mydumper 和 myloader 从 MySQL 数据库迁移数据到 OceanBase 数据库 .....	
3.1	mydumper 和 myloader 简介 .....	19
3.1.0.1	注意 .....	19
3.2	操作步骤 .....	19
3.2.1	步骤一：环境准备 .....	21
3.2.2	步骤二：备份操作 .....	21
3.2.2.1	备份全库 .....	21
3.2.2.2	备份指定数据库 test .....	21
3.2.2.3	备份指定数据库指定表(t1,t2) .....	22
3.2.2.4	仅备份表数据 .....	22
3.2.2.5	备份 t1 表的数据，开并行和数据压缩 .....	22
3.2.3	步骤三：恢复操作 .....	22
3.2.3.1	禁用外键检查约束 .....	22
3.2.3.2	使用 source 命令恢复数据 .....	23
3.2.3.3	使用 myloader 命令恢复数据 .....	23
4	使用 DBCAT 迁移 MySQL 表结构到 OceanBase 数据库 .....	24
4.0.0.1	注意 .....	24
4.1	环境准备 .....	24
4.2	导出 MySQL 数据库的表结构 .....	25
4.3	导入 OceanBase 数据库 .....	30
4.3.0.1	注意 .....	30
4.4	导数结果验证 .....	30

5	使用 DataX 迁移 MySQL 表数据到 OceanBase 数据库	33
5.1	框架设计	33
5.1.1	mysqlreader 插件	34
5.1.2	oceanbasev10writer 插件	34
5.1.3	DataX 配置文件	34
5.1.3.1	注意	36
5.2	环境准备	37
5.3	使用 DataX 迁移 MySQL 数据到 OceanBase 示例	38
5.3.0.1	注意	41
5.3.0.2	注意	41
5.3.0.3	说明	42
5.3.0.4	注意	43
5.4	更多信息	43
6	使用 CloudCanal 从 MySQL 数据库迁移数据到 OceanBase 数据库	44
6.0.0.1	功能适用性	44
6.1	前置条件	44
6.2	数据迁移操作步骤	44
6.2.1	添加数据源	44
6.2.2	创建任务	47
6.2.3	查看任务状态	50
6.3	相关文档	51
7	使用 Canal 从 MySQL 数据库同步数据到 OceanBase 数据库	52
7.1	架构原理	52
7.2	操作步骤	52
7.2.1	步骤一：MySQL 相关设置	52
7.2.2	步骤二：Canal 的下载和安装	54
7.2.3	步骤三：部署 RDB 适配器	55
7.2.3.1	说明	58
7.3	功能限制	59
8	使用 Flink CDC 从 MySQL 数据库同步数据到 OceanBase 数据库	60
8.1	Flink CDC 环境准备	60
8.2	准备数据	60
8.2.1	准备 MySQL 数据库数据	60
8.2.2	准备 OceanBase 数据库数据	61
8.3	启动 Flink 集群和 Flink SQL CLI	62
8.3.0.1	说明	63
8.3.1	设置 checkpoint	64
8.3.2	创建 MySQL CDC 表	64
8.3.3	创建 OceanBase CDC 表	65
8.3.4	在 Flink SQL CLI 中将数据写入 OceanBase 数据库中	66
8.3.4.1	说明	66
8.4	查看关联数据写入 OceanBase 数据库情况	67
8.5	查看数据更新情况	67

9	使用 ChunJun 从 MySQL 数据库迁移数据到 OceanBase 数据库	69
9.1	场景描述	69
9.2	前提条件	69
9.3	操作步骤	70
9.3.1	步骤一：ChunJun 环境准备	70
9.3.2	步骤二：配置 json 文件	70
9.3.3	步骤三：运行 json 配置文件	72
9.3.4	步骤四：查看数据迁移情况	74
9.4	相关文档	75
10	使用 OMS 从 OceanBase 数据库 MySQL 租户迁移数据到 MySQL 数据库	76
10.1	背景信息	76
10.1.0.1	说明	76
10.2	相关文档	76
11	使用 OMS 从 OceanBase 数据库 Oracle 租户迁移增量数据到 MySQL 数据库	77
11.1	背景信息	77
11.2	相关文档	77
12	使用 DBCAT 迁移 OceanBase 表结构到 MySQL 数据库	78
12.0.0.1	注意	78
12.1	环境准备	78
12.2	导出 OceanBase 数据库的 MySQL 租户下的表结构	80
12.3	导入 MySQL 数据库	85
12.3.0.1	注意	85
12.4	导出数据验证	85
13	使用 Datax 迁移 OceanBase 表数据到 MySQL 数据库	88
13.1	框架设计	88
13.1.1	oceanbasev10reader 插件	88
13.1.2	mysqlwriter 插件	89
13.1.3	DataX 配置文件	89
13.1.3.1	注意	91
13.2	环境准备	92
13.3	使用 DataX 迁移 OceanBase 数据到 MySQL 数据库示例	93
13.3.0.1	注意	96
13.3.0.2	注意	96
13.3.0.3	说明	97
13.3.0.4	注意	98
13.4	更多信息	98
14	使用 Canal 从 OceanBase 数据库同步数据到 MySQL 数据库	99
14.1	OceanBase CDC 实现逻辑	99
14.2	操作步骤	99
14.2.1	步骤一：安装 oblogproxy	99
14.2.2	步骤二：安装 canal server	99
14.2.3	步骤三：配置 RDB 适配器	101
14.3	功能限制	103

15 使用 CloudCanal 从 OceanBase 数据库迁移数据到 MySQL 数据库 .....	104
15.0.0.1 功能适用性 .....	104
15.1 前置条件 .....	104
15.2 操作步骤 .....	104
15.2.1 添加数据源 .....	104
15.2.2 创建任务 .....	107
15.2.3 查看任务状态 .....	110
15.3 相关文档 .....	110
16 使用 Flink CDC 从 OceanBase 数据库迁移数据到 MySQL 数据库 .....	111
16.1 环境准备 .....	111
16.1.1 配置 OceanBase 数据库 oblogproxy 服务 .....	111
16.1.2 Flink 环境设置 .....	112
16.2 准备数据 .....	112
16.2.1 准备 OceanBase 数据库数据 .....	112
16.2.2 准备 MySQL 数据库数据 .....	113
16.3 启动 Flink 集群和 Flink SQL CLI .....	114
16.3.0.1 说明 .....	115
16.3.1 设置 checkpoint .....	116
16.3.2 创建 OceanBase CDC 表 .....	116
16.3.3 创建 MySQL CDC 表 .....	118
16.3.4 在 Flink SQL CLI 中将数据写入 MySQL 数据库中 .....	118
16.3.4.1 说明 .....	119
16.4 查看关联数据写入 MySQL 数据库情况 .....	119
17 使用 ChunJun 从 OceanBase 数据库迁移数据到 MySQL 数据库 .....	120
17.1 场景描述 .....	120
17.2 前提条件 .....	121
17.3 操作步骤 .....	121
17.3.1 步骤一：ChunJun 环境准备 .....	121
17.3.2 步骤二：配置 json 文件 .....	122
17.3.3 步骤三：运行 json 配置文件 .....	124
17.3.4 步骤四：对 OceanBase 数据库数据增量修改 .....	126
17.3.5 步骤五：查看数据迁移情况 .....	127
17.4 相关文档 .....	128
18 使用 OMS 从 Oracle 数据库迁移数据到 OceanBase 数据库 MySQL 租户 .....	129
18.1 背景信息 .....	129
18.2 相关文档 .....	129
19 使用 OMS 从 Oracle 数据库迁移数据到 OceanBase 数据库 Oracle 租户 .....	130
19.1 背景信息 .....	130
19.2 相关文档 .....	130

20 使用 DBCAT 迁移 Oracle 表结构到 OceanBase 数据库 .....	131
20.0.0.1 注意 .....	131
20.1 环境准备 .....	131
20.2 导出 Oracle 数据库的表结构 .....	132
20.3 导入 OceanBase 数据库 .....	137
20.3.0.1 注意 .....	137
20.4 导数结果验证 .....	137
21 使用 DataX 迁移 Oracle 表数据到 OceanBase 数据库 .....	140
21.1 框架设计 .....	140
21.1.1 oraclereader 插件 .....	140
21.1.2 oceanbasev10writer 插件 .....	141
21.1.3 DataX 配置文件 .....	141
21.1.3.1 注意 .....	142
21.2 环境准备 .....	144
21.3 使用 DataX 迁移 Oracle 数据到 OceanBase 示例 .....	145
21.3.0.1 注意 .....	147
21.3.0.2 注意 .....	148
21.3.0.3 说明 .....	148
21.3.0.4 注意 .....	149
21.4 更多信息 .....	149
22 使用 OMS 从 OceanBase 数据库 Oracle 租户迁移数据到 Oracle 数据库 .....	150
22.1 背景信息 .....	150
22.2 相关文档 .....	150
23 使用 DBCAT 迁移 OceanBase 表结构到 Oracle 数据库 .....	151
23.0.0.1 注意 .....	151
23.1 环境准备 .....	151
23.2 导出 OceanBase 数据库 Oracle 租户下的表结构 .....	153
23.3 导入 Oracle 数据库 .....	158
23.3.0.1 注意 .....	158
23.4 导数结果验证 .....	158
24 使用 Datax 迁移 OceanBase 表数据到 Oracle 数据库 .....	161
24.1 框架设计 .....	161
24.1.1 oceanbasev10reader 插件 .....	161
24.1.2 oraclewriter 插件 .....	162
24.1.3 DataX 配置文件 .....	162
24.1.3.1 注意 .....	163
24.2 环境准备 .....	165
24.3 使用 DataX 迁移 OceanBase 数据到 Oracle 数据库示例 .....	166
24.3.0.1 注意 .....	169
24.3.0.2 注意 .....	169
24.3.0.3 说明 .....	170
24.3.0.4 注意 .....	171
24.4 更多信息 .....	171



25 使用 OMS 从 DB2 LUW 数据库迁移数据到 OceanBase 数据库 MySQL 租户	172
25.1 背景信息	172
25.2 相关文档	172
26 使用 OMS 从 DB2 LUW 数据库迁移数据到 OceanBase 数据库 Oracle 租户	173
26.1 背景信息	173
26.2 相关文档	173
27 使用 DBCAT 迁移 DB2 LUW 表结构到 OceanBase 数据库	174
27.0.0.1 注意	174
27.1 环境准备	174
27.2 导出 DB2 LUW 数据库的表结构	175
27.3 导入 OceanBase 数据库	180
27.3.0.1 注意	180
27.4 导出数据验证	180
28 使用 OMS 从 OceanBase 数据库 MySQL 租户迁移数据到 DB2 LUW 数据库	183
28.1 背景信息	183
28.2 相关文档	183
29 使用 OMS 从 OceanBase 数据库 Oracle 租户迁移数据到 DB2 LUW 数据库	184
29.1 背景信息	184
29.2 相关文档	184
30 使用 OMS 从 TiDB 数据库迁移数据到 OceanBase 数据库 MySQL 租户	185
30.1 背景信息	185
30.2 相关文档	185
31 使用 OMS 从 PostgreSQL 数据库迁移数据到 OceanBase 数据库 MySQL 租户	186
31.1 背景信息	186
31.2 相关文档	186
32 使用 DataX 迁移 CSV 文件到 OceanBase 数据库	187
32.1 框架设计	187
32.1.1 txtfilereader 插件	187
32.1.2 oceanbasev10writer 插件	188
32.1.3 DataX 配置文件	188
32.1.3.1 注意	189
32.2 环境准备	191
32.3 使用 DataX 迁移 CSV 文件到 OceanBase 示例	192
32.3.0.1 注意	195
32.3.0.2 注意	195
32.3.0.3 说明	196
32.3.0.4 注意	197
32.4 更多信息	197

33 使用 LOAD DATA 语句导入数据	198
33.1 使用限制	198
33.2 注意事项	198
33.3 使用场景	198
33.3.0.1 说明	198
33.4 LOAD DATA 语法	198
33.4.1 获取 LOAD DATA 执行权限	199
33.5 示例	199
33.5.0.1 说明	199
33.5.1 从服务器端文件导入数据	200
33.5.1.1 注意	200
33.5.2 从客户端（本地）文件导入数据	202
33.5.2.1 注意	203
33.6 异常处理	204
33.6.1 日志文件	204
33.7 相关文档	205
34 从 SQL 文件导入数据到 OceanBase 数据库	206
34.1 配置 SQL 文件信息	206
34.2 使用命令行界面导入数据	206
34.3 使用 ODC 导入数据	208
34.3.0.1 注意	211
34.4 使用 obloader 导入数据	213
34.4.1 操作步骤	213
34.4.1.1 说明	213
34.4.2 导入 SQL 文件数据示例	213
34.4.2.1 导入表结构	214
34.4.2.2 注意	214
34.4.2.3 导入表数据	217
34.4.2.4 注意	217
35 使用 OMS 从 OceanBase 数据库迁移数据到 OceanBase 数据库同类型租户	222
35.1 背景信息	222
35.2 相关文档	222
36 使用 OMS 创建 OceanBase 数据库同类型租户容灾双活项目	223
36.1 背景信息	223
36.2 相关文档	223

37 使用 OceanBase 导数工具在 OceanBase 集群 MySQL 租户间迁移数据 .....	224
37.1 OceanBase 导数工具简介 .....	224
37.2 操作步骤 .....	224
37.2.0.1 说明 .....	224
37.2.0.2 说明 .....	225
37.3 示例 .....	225
37.3.1 表结构迁移 .....	225
37.3.1.1 说明 .....	225
37.3.1.2 导出 DDL 定义文件 .....	226
37.3.1.3 导入 DDL 定义文件 .....	230
37.3.2 表数据迁移 .....	233
37.3.2.1 导出 CSV 数据文件 .....	234
37.3.2.2 导入 CSV 数据文件 .....	238
38 使用 OceanBase 导数工具从 OceanBase 集群 MySQL 租户迁移数据到 Oracle 租户 .....	242
38.1 OceanBase 导数工具简介 .....	242
38.2 操作步骤 .....	242
38.2.0.1 说明 .....	242
38.2.0.2 说明 .....	243
38.3 导出和导入数据示例 .....	243
38.3.1 表结构迁移 .....	243
38.3.1.1 说明 .....	243
38.3.1.2 导出 DDL 定义文件 .....	244
38.3.1.3 导入 DDL 定义文件 .....	248
38.3.1.4 注意 .....	249
38.3.2 表数据迁移 .....	253
38.3.2.1 导出 CSV 数据文件 .....	253
38.3.2.2 导入 CSV 数据文件 .....	257
38.3.2.3 注意 .....	257
39 使用 OceanBase 导数工具在 OceanBase 集群 Oracle 租户间迁移数据 .....	262
39.1 OceanBase 导数工具简介 .....	262
39.2 操作步骤 .....	262
39.2.0.1 说明 .....	262
39.2.0.2 说明 .....	263
39.3 导出和导入数据示例 .....	263
39.3.1 表结构迁移 .....	263
39.3.1.1 说明 .....	263
39.3.1.2 导出 DDL 定义文件 .....	264
39.3.1.3 导入 DDL 定义文件 .....	268
39.3.2 表数据迁移 .....	272
39.3.2.1 导出 CSV 数据文件 .....	272
39.3.2.2 导入 CSV 数据文件 .....	276

40 使用 OceanBase 导数工具从 OceanBase 集群 Oracle 租户迁移数据到 MySQL 租户	
40.1 OceanBase 导数工具简介 .....	281 281
40.2 操作步骤 .....	281
40.2.0.1 说明 .....	281
40.2.0.2 说明 .....	282
40.3 导出和导入数据示例 .....	282
40.3.1 表结构迁移 .....	282
40.3.1.1 说明 .....	282
40.3.1.2 导出 DDL 定义文件 .....	283
40.3.1.3 导入 DDL 定义文件 .....	287
40.3.1.4 注意 .....	288
40.3.2 表数据迁移 .....	291
40.3.2.1 导出 CSV 数据文件 .....	291
40.3.2.2 导入 CSV 数据文件 .....	296
40.3.2.3 注意 .....	296
41 表与表之间的数据迁移 .....	301
41.1 表与表之间数据迁移 .....	301
41.1.1 语法 1 .....	301
41.1.1.1 注意 .....	301
41.1.2 语法 2 .....	301
41.1.2.1 功能适用性 .....	301
41.1.2.2 注意 .....	302
41.1.3 示例 1 .....	302
41.1.4 示例 2 .....	304
41.2 相关文档 .....	305
42 资源单元迁移 .....	306
42.0.0.1 功能适用性 .....	306
42.1 语法 .....	306
42.2 示例 .....	306
42.3 相关文档 .....	307

43 使用 OUTFILE 语句导出数据 .....	308
43.1 背景信息 .....	308
43.1.0.1 注意 .....	308
43.2 权限要求 .....	308
43.3 语法 .....	308
43.4 参数解释 .....	310
43.4.0.1 说明 .....	311
43.4.0.2 说明 .....	312
43.4.0.3 注意 .....	312
43.4.0.4 说明 .....	312
43.4.1 field_term .....	313
43.4.2 line_term .....	313
43.4.3 file_option .....	313
43.4.3.1 注意 .....	313
43.4.3.2 说明 .....	314
43.5 示例 .....	314
43.5.1 导出数据文件到本地 .....	314
43.5.1.1 注意 .....	314
43.5.1.2 说明 .....	316
43.5.2 导出数据文件到 OSS .....	317
43.6 更多信息 .....	317
44 旁路导入概述 .....	318
44.0.0.1 注意 .....	318
44.1 功能概述 .....	318
44.2 使用场景 .....	318
44.3 相关文档 .....	319

45 全量旁路导入 .....	320
45.1 注意事项 .....	320
45.2 使用 LOAD DATA 语句旁路导入数据 .....	320
45.2.0.1 注意 .....	320
45.2.1 使用限制 .....	320
45.2.2 使用语法 .....	321
45.2.2.1 注意 .....	322
45.2.2.2 说明 .....	323
45.2.3 使用示例 .....	324
45.2.3.1 说明 .....	324
45.2.3.2 说明 .....	324
45.2.3.3 注意 .....	325
45.2.3.4 说明 .....	327
45.2.3.5 注意 .....	327
45.3 使用 INSERT INTO SELECT 语句旁路导入数据 .....	329
45.3.1 使用限制 .....	330
45.3.2 使用语法 .....	330
45.3.2.1 说明 .....	331
45.3.3 使用示例 .....	331
45.4 使用 CREATE TABLE AS SELECT 语句旁路导入数据 .....	342
45.4.1 使用语法 .....	342
45.4.2 使用示例 .....	343
45.5 相关文档 .....	347

46 增量旁路导入 .....	348
46.1 注意事项 .....	348
46.2 使用 LOAD DATA 语句旁路导入数据 .....	348
46.2.1 使用限制 .....	348
46.2.2 使用语法 .....	349
46.2.2.1 注意 .....	349
46.2.2.2 注意 .....	350
46.2.2.3 说明 .....	351
46.2.2.4 说明 .....	352
46.2.3 使用示例 .....	352
46.2.3.1 说明 .....	352
46.2.3.2 说明 .....	352
46.2.3.3 注意 .....	353
46.2.3.4 说明 .....	355
46.2.3.5 注意 .....	355
46.3 使用 INSERT INTO SELECT 语句旁路导入数据 .....	357
46.3.1 使用限制 .....	358
46.3.2 使用语法 .....	358
46.3.2.1 说明 .....	359
46.3.3 使用示例 .....	359
46.4 使用 CREATE TABLE AS SELECT 语句旁路导入数据 .....	371
46.4.1 使用语法 .....	371
46.4.1.1 注意 .....	372
46.4.2 使用示例 .....	372
46.5 相关文档 .....	376
47 使用 OBLADER 旁路导入数据 .....	377
47.0.0.1 注意 .....	377
47.1 导入示例 .....	379
47.2 旁路导入模式相关参数 .....	379
47.3 注意事项 .....	380

# 1 数据迁移概述

数据迁移是日常运维操作的一种常见操作，是调整集群负载和机房搬迁的必备操作。虽然集群内部、表与表之间数据归档、磁盘水位均衡、资源单元搬迁等操作在 OceanBase 数据库中可以通过简单命令快速发起，但是涉及异构数据源和集群间的数据同步等功能时就需要借助外部工具。

本文档主要介绍几种常用的数据迁移方法及工具。

## 1.1 应用场景

数据迁移是数据库运维常见的操作，主要有如下的应用场景：

- 调整集群负载和机房搬迁。
- 数据导出到文件、从文件导入到数据库。
- 数据库替换。
- 数据库逻辑复制，包括读写分离、数据库容灾、业务多活等。
- 业务上的数据复制需求。

## 1.2 迁移方案

OceanBase 数据库提供丰富的数据迁移复制方法，包括：

- 使用 OMS 迁移（建议）
- 使用 obloader & obdumper 迁移
  - 使用 obloader 导入数据
  - 使用 obdumper 导出数据
- 使用 SQL 脚本迁移
- 使用 MyDumper 迁移
- 使用 DataX 迁移
- 使用 OUTFILE 语句迁移
- 使用 DBCAT 迁移

### 1.2.0.1 说明

您也可以通过备份恢复功能来实现 OceanBase 数据库租户的数据迁移。有关 OceanBase 数据库备份恢复功能的详细信息，参见 [物理备份与恢复概述](#)。



迁移方案具体支持情况如下：

迁移方案	结构迁移	全量数据迁移	增量数据迁移	数据校验	支持的数据源
OMS	支持	支持	支持	支持	OceanBase MySQL Oracle PostgreSQL DB2_LUW TiDB Kafka RocketMQ Datahub
obloader & obdumper	支持	支持	不支持	不支持	OceanBase
SQL 脚本迁移	支持	支持	不支持	不支持	主流数据库，但无法跨实例做数据迁移
MyDumper	支持	支持	不支持	不支持	MySQL 兼容的数据源
DataX	不支持	支持	不支持	不支持	较多，具体范围查看官方文档
OUTFILE 语句	不支持	支持	不支持	不支持	MySQL OceanBase
DBCAT	支持	不支持	不支持	不支持	较多，具体范围查看官方文档

## 2 使用 OMS 从 MySQL 数据库迁移数据到 OceanBase 数据库 MySQL 租户

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 MySQL 数据库迁移数据到 OceanBase 数据库 MySQL 租户中的背景信息。

### 2.1 背景信息

在 OMS 控制台创建从自建 MySQL 数据库迁移数据至 OceanBase 数据库 MySQL 租户的数据迁移项目，您可以通过结构迁移、全量迁移和增量同步，无缝迁移源端数据库中的存量业务数据和增量数据至 OceanBase 数据库 MySQL 租户。

MySQL 数据库支持单主库、单备库和主备库等模式。迁移 MySQL 数据库的数据至 OceanBase 数据库 MySQL 租户时，不同类型的数据源支持的操作也不同。

类型	支持的操作
单主库	结构迁移 + 全量迁移 + 增量同步 + 全量校验 + 反向增量
单备库	结构迁移 + 全量迁移 + 全量校验
主备库	主库：支持增量同步 + 反向增量 备库：支持结构迁移 + 全量迁移 + 全量校验

### 2.2 相关文档

更多使用 OMS 从 MySQL 数据库迁移数据到 OceanBase 数据库 MySQL 租户的操作信息，请参见 [迁移 MySQL 数据库的数据至 OceanBase 数据库 MySQL 租户](#)。

## 3 使用 mydumper 和 myloader 从 MySQL 数据库迁移数据到 OceanBase 数据库

本文将介绍如何使用 mydumper 和 myloader 从 MySQL 数据库迁移数据至 OceanBase 数据库（MySQL 模式）。

### 3.1 mydumper 和 myloader 简介

- **mydumper**: 是一款开源的多线程备份工具，用于备份 MySQL 数据库。mydumper 支持备份整个数据库、单个表或多个表，支持多线程备份，可以加速数据备份过程。mydumper 还支持备份数据的压缩和加密，可以从备份文件中恢复数据。mydumper 是一款命令行工具，需要安装并使用命令行进行操作。
- **myloader**: 是一款用于恢复 mydumper 备份文件的工具。myloader 可以快速恢复备份的 MySQL 数据库，并支持从备份文件中恢复数据，不需要先创建数据库。myloader 还支持恢复到指定时间点的备份文件，可以指定要恢复的数据表和数据行。myloader 是一款命令行工具，需要安装并使用命令行进行操作。

mydumper 和 myloader 都是开源的工具，具有高效、安全、可靠的特点，并且支持多种备份方式和备份数据的压缩和加密。

#### 3.1.0.1 注意

使用 MyDumper 仅支持导出 OceanBase 数据库 MySQL 模式租户中的数据。

### 3.2 操作步骤

1. 环境准备。
2. 备份操作。

备份 MySQL 数据库数据示例语句如下：

```
mydumper -h xx.xx.xx.xx -P3306 -u username -p ***** -B test_db -o /data  
/backup/mysql/
```

mysdumper 命令常用参数说明：

参数	参数全称	说明
-h	--host	连接的主机名或 IP 地址
-P	--port	连接的端口
-u	--user	备份使用的用户名
-p	--password	备份使用的用户密码
-B	--database	备份的数据库名，默认备份所有库
-T	--tables-list	备份的表名，多张表名字用逗号隔开
-i	--ignore-engines	备份忽略的存储引擎，用逗号分割
-m	--no-schemas	不备份表结构
-t	--threads	开启的并行备份线程数，默认是 4
-c	--compress	对输出文件进行压缩
-o	--outputdir	备份文件输出目录

### 3. 恢复操作。

备份数据后，您可通过 `source` 命令或 `myloader` 命令恢复数据。

- 使用 `source` 命令恢复数据。

若无法直接还原单表，可以找到对应的单表 sql 文件，进入命令行，使用 `source` 命令恢复数据。

```
source <表名>-schema.sql #还原表结构
```

```
source <表名>.sql #还原表数据
```

- 下述示例语句展示了如何使用 `myloader` 恢复 OceanBase 数据库中的数据：

```
myloader -h xx.xx.xx.xx -P2883 -u 'user@tenantname#clustenamer' -p ***** -o  
-d /data/backup/mysql/
```

myloader 命令常用参数说明：

参数	参数全称	说明
-h	--host	连接的主机名
-P	--port	连接的端口
-u	--user	恢复使用的用户名
-p	--password	恢复使用的用户密码
-B	--database	恢复的数据库名
-d	--directory	恢复文件的目录
-o	--overwrite-tables	如果要恢复的表存在，则先 drop 掉该表；使用该参数，需要备份时候要备份表结构
-B	--database	需要还原的数据库

## 3.2.1 步骤一：环境准备

### 1. 下载安装包。

请根据需要在 [下载安装包地址](#) 中，下载对应的安装包并安装。

### 2. 在数据库主机上解压缩安装包。

以 mydumper-0.12.7-2-zstd.el7.x86\_64.rpm 为示例。

```
rpm2cpio mydumper-0.12.7-2-zstd.el7.x86_64.rpm | cpio -div
```

### 3. 验证是否正确安装。

```
./usr/bin/mydumper --help
```

在备份目的文件夹中，会生成 metadata 文件和对应的表结构或表数据文件。

## 3.2.2 步骤二：备份操作

### 3.2.2.1 备份全库

```
mydumper -h xx.xx.xx.xx -P3306 -u username -p ***** -o /data/backup/mysql/
```

### 3.2.2.2 备份指定数据库 test

```
mydumper -h xx.xx.xx.xx -P3306 -u username -p ***** -B test -o /data/backup  
/mysql/
```

### 3.2.2.3 备份指定数据库指定表(t1,t2)

```
mydumper -h xx.xx.xx.xx -P3306 -u username -p ***** -B test -T t1,t2 -o /data  
/backup/mysql/
```

### 3.2.2.4 仅备份表数据

```
mydumper -h xx.xx.xx.xx -P3306 -u username -p ***** -B test -T t1 -m -o /data  
/backup/mysql/
```

### 3.2.2.5 备份 t1 表的数据，开并行和数据压缩

```
mydumper -h xx.xx.xx.xx -P3306 -u username -p ***** -B test -T t1 -t 6 -c -o /data  
/backup/mysql/
```

## 3.2.3 步骤三：恢复操作

### 3.2.3.6 禁用外键检查约束

在备份的表结构语句里，可能包含外键。在导入 OceanBase MySQL 里时，如果外键依赖的表没有创建，导入脚本会报错，因此在导入之前需要禁用外键检查约束。

```
MySQL [oceanbase]> SET GLOBAL foreign_key_checks=off;
```

```
Query OK, 0 rows affected
```

```
MySQL [oceanbase]> SHOW GLOBAL VARIABLES LIKE '%foreign%';
```

```
+-----+-----+
```

```
| Variable_name | Value |
```

```
+-----+-----+
```

```
| foreign_key_checks | OFF |
```

```
+-----+-----+
1 row in set
```

备份数据后，您可通过 `source` 命令或 `myloader` 命令恢复数据。

### 3.2.3.7 使用 `source` 命令恢复数据

若无法直接还原单表，可以找到对应的单表 sql 文件，进入命令行，使用 `source` 命令恢复数据。

```
source test.t1-schema.sql 还原表结构
```

```
source test.t1.00000.sql 还原表数据
```

### 3.2.3.8 使用 `myloader` 命令恢复数据

- 导入表结构和数据。

```
myloader -h xx.xx.xx.xx -P2883 -u 'user@tenantname#clustenamer' -p ***** -B
test -o -d /data/backup/mysql/
```

- 导入数据库（若目标库不存在则会新建）。

```
myloader -h xx.xx.xx.xx -P2883 -u 'user@tenantname#clustenamer' -p ***** -B
test -s test1 -o -d /data/backup/mysql/
```

更多关于 `mydumper` 和 `myloader` 的使用方法，请参见 <https://github.com/mydumper/mydumper>。

## 4 使用 DBCAT 迁移 MySQL 表结构到 OceanBase 数据库

DBCAT 是一款轻量级的命令行工具，可用于提供数据库之间 DDL 转换和 Schema 比对等功能。这里以 DBCAT 迁移表结构为例进行介绍。

DBCAT 安装包文件名为 `dbcat-[版本号]-SNAPSHOT.tar.gz`，下载后解压缩即可使用，可执行文件名为 `dbcat`。

### 4.0.0.1 注意

DBCAT 是 OMS 的一个组件，在社区版环境中推荐使用 OMS 导出。

## 4.1 环境准备

DBCAT 能运行在 CentOS、macOS 和 Windows 下。需要安装 JDK 1.8 以上（含）版本。可以使用 OpenJDK，安装好后配置环境变量 `JAVA_HOME`。

CentOS 安装 OpenJDK 示例：

```
$sudo yum -y install java-1.8.0-openjdk.x86_64

$which java
/usr/local/java/jdk1.8.0_261/bin/java

echo 'export JAVA_HOME=/usr/local/java/jdk1.8.0_261/' >> ~/.bash_profile
. ~/.bash_profile
```

解压安装文件：

```
tar zxvf dbcat-1.8.0-SNAPSHOT.tar.gz
cd dbcat-1.8.0-SNAPSHOT/
chmod +x bin/dbcat

$tree -L 3 --filelimit 30
```



```
.
├── bin
│   ├── dbcat
│   ├── dbcat.bat
│   └── dbcat-debug
├── conf
│   ├── dbcat.properties
│   └── logback.xml
├── docs
│   ├── README.docx
│   ├── README.md
│   └── README.txt
├── LEGAL.md
├── lib [45 entries exceeds filelimit, not opening dir]
├── LICENSE
├── meta
│   └── README
└── NOTICE

5 directories, 12 files
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。
conf	日志文件配置目录。
lib	运行时期依赖的包。
meta	离线转换场景下，导出字典表数据。
~/output	SQL 文件与报告文件，运行时生成。

## 4.2 导出 MySQL 数据库的表结构

DBCAT 具有在线转换功能，该功能是指 DBCAT 能直连源端数据库，将数据库中的对象导出。当对象非常多时（如超过 1 万），导出过程可能会有点慢。

dbcat 导出命令如下：

```
bin/dbcat convert -H<host> -P<port> -u<user> -p<password> -D <database> --from  
<from> --to <to> --all
```

您可运行命令 `bin/dbcat help convert` 查看更多参数信息。

必选参数：

选项	有无参数	中文描述
-H/--host	Y	数据库服务器的 IP 地址
-P/--port	Y	数据库服务器的端口
-u/--user	Y	登录数据库的用户名
-t/--tenant	Y	连接 OceanBase 集群需要提供租户名
-c/--cluster	Y	连接 OceanBase 集群需要提供集群名
-p/--password	Y	登录数据库的密码
-D/--database	Y	数据库名（源库），DB2 须区分数据库名和模式名
--service-id	Y	连接 Oracle 数据库需要提供服务 ID
--service-name	Y	连接 Oracle 数据库需要提供服务名
--as-sysdba	N	连接 Oracle 数据库 sysdba 角色
--sys-user	Y	连接 OceanBase 集群系统租户的用户名
--sys-password	Y	连接 OceanBase 集群系统租户的密码
--schema	Y	模式名（源库），非 DB2，模式名与数据名相同
--from	Y	源库的类型
--to	Y	目标库的类型
--all	N	所有的数据库对象

可选参数：

选项	有无参数	中文描述
-f/--file	Y	sql 文件的输出路径
--offline	N	使用离线模式
--target-schema	Y	模式名（目标库）
--table	Y	导出的表
--view	Y	导出的视图
--trigger	Y	导出的触发器
--synonym	Y	导出的同义词
--sequence	Y	导出的序列
--function	Y	导出的函数
--procedure	Y	导出的存储过程
--dblink	Y	导出所有的 DBLink
--type	Y	导出的 type
--type-body	Y	导出的 type body
--package	Y	导出的 package
--package-body	Y	导出的 package body
--no-quote	N	产生的 DDL 不带引号
--no-schema	N	产生的 DDL 不带模式名
--target-schema	Y	产生的 DDL 中使用指定的模式名
--exclude-type	Y	搭配 --all 使用，如：--all --exclude-type 'TABLE' 表示排除 TABLE 类型

这里以导出 MySQL 5.7 版本下 database 为 test 的所有对象的结构，并将其迁移到 4.0.0 版本的 OceanBase 集群中的 MySQL 租户为示例。

```
bin/dbcat convert -H 11.161.xxx.xxx -P 3306 -uroot -pxxxxxxx -D test --from mysql57
--to obmysql40 --all
```

特别说明：

- dbcat 不需要直接安装在数据库主机上，安装在可直连数据库主机的主机上即可。

- 参数中的 --from 和 --to 为源端和目的端的数据库类型，需要详细到版本号。当前 dbcat 支持的源端和目标端数据库详细如下：

源端数据库类型	目标端数据库类型
TiDB	OBMYSQL
PG	OBMYSQL
SYBASE	OBORACLE
MYSQL	OBMYSQL
ORACLE	OBORACLE
ORACLE	OBMYSQL
DB2 IBM i	OBORACLE
DB2 LUW	OBORACLE
DB2 LUW	OBMYSQL
OBMYSQL	MYSQL
OBORACLE	ORACLE

其中 OBMYSQL 为 OceanBase 数据库的 MySQL 租户，OBORACLE 为 OceanBase 数据库的 Oracle 租户。

- 当前支持的源端和目标端数据库详细的版本，详情如下。

数据库类型	数据库版本
TiDB	tidb4 tidb5
PG	pgsql10
SYBASE	sybase15
DB2 IBM i	db2ibmi71
DB2 LUW	db2luw970 db2luw1010 db2luw1050 db2luw111 db2luw115
MYSQL	mysql56 mysql57 mysql</80>

ORACLE	oracle9i oracle10g oracle11g oracle12c oracle18c oracle19c
OBMYSQL	obmysql14x obmysql21x obmysql22x obmysql200 obmysql211 obmysql2210 obmysql2230 obmysql2250 obmysql2271 ~ obmysql2277 obmysql30x obmysql31x obmysql32x obmysql322 obmysql40
OBORACLE	oboracle2220 oboracle2230 oboracle2250 oboracle2270 ~ oboracle2277 oboracle21x oboracle22x oboracle30x oboracle31x oboracle32x oboracle322 oboracle40

运行后的输出文件在用户 `home` 目录的 `output` 下。

```
$tree ~/output/dbcat-20xx-xx-xx-164533/
/home/qing.meiq/output/dbcat-20xx-xx-xx-164533/
|____ tpccdb
|  |____ TABLE-schema.sql
|____ tpccdb-conversion.html

1 directory, 2 files
```

## 4.3 导入 OceanBase 数据库

使用 DBCAT 导出的文件格式为 SQL 文件，这里可以通过 ODC 的导入功能批量将表结构导入 OceanBase 数据库，更多信息请参考 [批量导出与导入](#)。

也可以使用 `source` 命令将 SQL 文件数据导入 OceanBase 数据库，示例如下：

```
obclient [test]> source TABLE-schema.sql
Query OK, 0 rows affected (0.044 sec)
```

### 4.3.0.1 注意

如果 SQL 文件不在当前目录下，则需要使用绝对地址。

## 4.4 导数结果验证

示例：查看一个表结构在 MySQL 里的书写方式和 OceanBase 数据库里的表结构。

查看源数据库 MySQL 的表 `bmsql_customer` 的建表 SQL：

```
MySQL [test]> show create table bmsql_customer \G
***** 1. row *****
Table: bmsql_customer
Create Table: CREATE TABLE `bmsql_customer` (
  `c_w_id` bigint(20) NOT NULL,
  `c_d_id` bigint(20) NOT NULL,
  `c_id` bigint(20) NOT NULL,
  `c_discount` decimal(4,4) DEFAULT NULL,
  `c_credit` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_last` varchar(16) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_first` varchar(16) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_credit_lim` decimal(12,2) DEFAULT NULL,
  `c_balance` decimal(12,2) DEFAULT NULL,
  `c_ytd_payment` decimal(12,2) DEFAULT NULL,
  `c_payment_cnt` bigint(20) DEFAULT NULL,
  `c_delivery_cnt` bigint(20) DEFAULT NULL,
```

```
`c_street_1` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
`c_street_2` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
`c_city` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
`c_state` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,
`c_zip` char(9) COLLATE utf8_unicode_ci DEFAULT NULL,
`c_phone` char(16) COLLATE utf8_unicode_ci DEFAULT NULL,
`c_since` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
`c_middle` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,
`c_data` varchar(500) COLLATE utf8_unicode_ci DEFAULT NULL,
PRIMARY KEY (`c_w_id`,`c_d_id`,`c_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
1 row in set (0.01 sec)
```

查看目标库数据库 OceanBase 的表 bmsql\_customer 的表结构：

```
obclient [test]> desc bmsql_customer;
+-----+-----+-----+-----+-----+
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
+-----+
| c_w_id | bigint(20) | NO | PRI | NULL | |
| c_d_id | bigint(20) | NO | PRI | NULL | |
| c_id | bigint(20) | NO | PRI | NULL | |
| c_discount | decimal(4,4) | YES | | NULL | |
| c_credit | char(2) | YES | | NULL | |
| c_last | varchar(16) | YES | | NULL | |
| c_first | varchar(16) | YES | | NULL | |
| c_credit_lim | decimal(12,2) | YES | | NULL | |
| c_balance | decimal(12,2) | YES | | NULL | |
```

```
| c_ytd_payment | decimal(12,2) | YES || NULL ||  
| c_payment_cnt | bigint(20) | YES || NULL ||  
| c_delivery_cnt | bigint(20) | YES || NULL ||  
| c_street_1 | varchar(20) | YES || NULL ||  
| c_street_2 | varchar(20) | YES || NULL ||  
| c_city | varchar(20) | YES || NULL ||  
| c_state | char(2) | YES || NULL ||  
| c_zip | char(9) | YES || NULL ||  
| c_phone | char(16) | YES || NULL ||  
| c_since | timestamp | NO || CURRENT_TIMESTAMP | ON UPDATE CURRENT_TIMESTAMP |  
| c_middle | char(2) | YES || NULL ||  
| c_data | varchar(500) | YES || NULL ||  
+-----+-----+-----+-----+-----+  
+-----+  
21 rows in set (0.004 sec)
```

经对比是一致的。

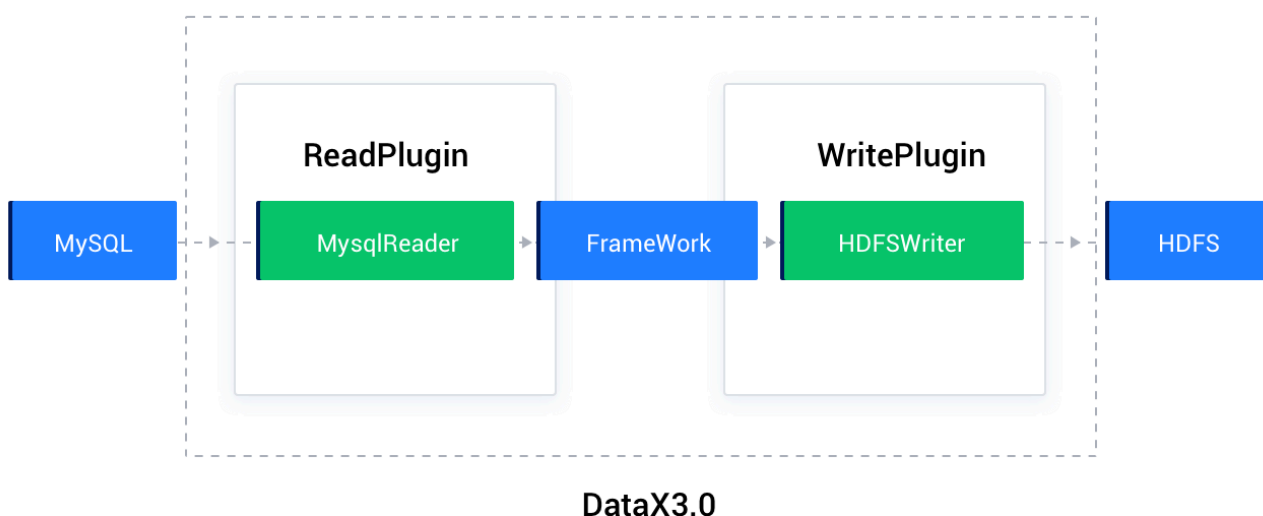


## 5 使用 DataX 迁移 MySQL 表数据到 OceanBase 数据库

DataX 是阿里云 DataWorks 数据集成的开源版本，是阿里巴巴集团内被广泛使用的离线数据同步工具/平台。DataX 实现了包括 MySQL、Oracle、SQLserver、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、Hologres、DRDS、OceanBase 等各种异构数据源之间高效的数据同步功能。

OceanBase 数据库社区版客户，可以在 [DataX 开源网站](#) 内下载源码，自行编译。编译时，可根据需要剔除在 `pom.xml` 中不用的数据库插件，否则编译出来的包会非常大。

### 5.1 框架设计



DataX 作为离线数据同步框架，采用 "Framework + Plugin" 模式构建。将数据源读取和写入抽象为 Reader/Writer 插件，纳入到整个同步框架中。

- Reader 作为数据采集模块，负责采集数据源的数据，将数据发送给 Framework。
- Writer 作为数据写入模块，负责不断向 Framework 获取数据，并将数据写入到目的端。
- Framework 用于连接 Reader 和 Writer，作为两者的数据传输通道，并处理缓冲、流控、并发、数据转换等核心技术问题。

DataX 以任务的形式迁移数据。每个任务只处理一个表，每个任务有一个 `json` 格式的配置文件，配置文件里包含 `reader` 和 `writer` 两部分。`reader` 和 `writer` 分别对 DataX 支持的数据库读写插件，例如，将 MySQL 表数据迁移到 OceanBase 数据库操作时，需要从

MySQL 读取数据写入 OceanBase 数据库，因此使用的插件为 MySQL 数据库的 `mysqlreader` 插件和 OceanBase 数据库的 `oceanbasev10writer` 插件来搭配完成。这里介绍下 `mysqlreader` 和 `oceanbasev10writer` 插件。

## 5.1.1 mysqlreader 插件

`mysqlreader` 插件实现了从 MySQL 数据库上读取数据的功能。在底层实现上，`MySQLReader` 通过 JDBC 连接远程 MySQL 数据库，并执行相应的 SQL 语句将数据从 MySQL 库中 SELECT 出来。

实现原理方面，简而言之，`MySQLReader` 通过 JDBC 连接器连接到远程的 MySQL 数据库，并根据用户配置的信息生成查询语句，然后发送到远程 MySQL 数据库，远程的 MySQL 数据库将该 SQL 执行的返回结果使用 DataX 自定义的数据类型拼装为抽象的数据集，再传递给下游 Writer 处理。

详细功能和参数说明请参考官方说明：[mysqlreader 插件](#)。

## 5.1.2 oceanbasev10writer 插件

`oceanbasev10writer` 插件实现了将数据写入到 OceanBase 数据库的目标表中的功能。在底层实现上，`oceanbasev10writer` 通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并执行相应的 insert sql 语句来将数据写入到 OceanBase 数据库，OceanBase 数据库内部会分批次提交入库。

实现原理方面，`oceanbasev10writer` 通过 DataX 框架获取 Reader 生成的协议数据，生成 insert 语句。在写入数据时，如果出现主键或唯一键冲突，OceanBase 数据库的 MySQL 租户可以通过 replace 模式来更新表中的所有字段；OceanBase 数据库的 Oracle 租户当前只能使用 Insert 方式。出于性能考虑，写入采用 batch 方式批量写，当行数累计到预定阈值时，才发起写入请求。

## 5.1.3 DataX 配置文件

配置文件示例：

```
{
  "job": {
    "content": [
      {
```

```
"reader": {
  "name": "streamreader",
  "parameter": {
    "sliceRecordCount": 10,
    "column": [
      {
        "type": "long",
        "value": "10"
      },
      {
        "type": "string",
        "value": "hello, 你好, 世界-DataX"
      }
    ]
  },
  "writer": {
    "name": "streamwriter",
    "parameter": {
      "encoding": "UTF-8",
      "print": true
    }
  }
},
"setting": {
  "speed": {
    "channel": 2
  }
}
```

```
}  
}  
}
```

### 5.1.3.1 注意

datax 仅迁移表数据，需要提前在目标端创建好对应的表对象结构。

将 json 配置文件放到 DataX 的目录 job 下，或者自定义路径。执行方法如下：

```
$bin/datax.py job/stream2stream.json
```

输出信息：

```
<.....>
```

```
2021-08-26 11:06:09.217 [job-0] INFO JobContainer - PerfTrace not enable!
```

```
2021-08-26 11:06:09.218 [job-0] INFO StandAloneJobContainerCommunicator - Total  
20 records, 380 bytes | Speed 38B/s, 2 records/s | Error 0 records, 0 bytes | All Task  
WaitWriterTime 0.000s | All Task WaitReaderTime 0.000s | Percentage 100.00%
```

```
2021-08-26 11:06:09.223 [job-0] INFO JobContainer -
```

```
任务启动时刻：2021-08-26 11:05:59
```

```
任务结束时刻：2021-08-26 11:06:09
```

```
任务总计耗时：10s
```

```
任务平均流量：38B/s
```

```
记录写入速度：2rec/s
```

```
读出记录总数：20
```

```
读写失败总数：0
```

DataX 任务执行结束会有个简单的任务报告，包含上述输出的平均流量、写入速度和读写失败总数等。

DataX 的 job 的参数 settings 可以指定速度参数和错误记录容忍度等。

```
"setting": {
```

```
"speed": {
```

```
"channel": 10
},
"errorLimit": {
"record": 10,
"percentage": 0.1
}
}
```

参数说明：

- `errorLimit` 表示报错记录数的容忍度，超出这个限制后任务就中断退出。
- `channel` 是并发数，理论上并发越大，迁移性能越好。但实际操作中也要考虑源端的读压力、网络传输性能以及目标端写入性能。

## 5.2 环境准备

下载 tar 包地址：<http://datax-opensource.oss-cn-hangzhou.aliyuncs.com/datax.tar.gz>

解压安装文件：

```
tar zxvf datax.tar.gz
cd datax
```

目录如下：

```
$tree -L 1 --filelimit 30
.
|—— bin
|—— conf
|—— job
|—— lib
|—— log
|—— log_perf
|—— plugin
```

```
|—— script
|—— tmp
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。该目录下的 datax.py 为 DataX 任务的启动脚本
conf	日志文件配置目录。该目录下存放 datax 与任务无关的配置文件
lib	运行时期依赖的包。该目录存放 DataX 运行所需要的全局 jar 文件
job	该目录下有一个用于测试验证 datax 安装的任务配置文件
log	日志文件目录。该目录下存放 datax 任务运行的日志；datax 运行时，默认会将日志输出到标准输出，同时写入到 log 目录下
plugin	插件文件目录。该目录下保存 DataX 支持的各种数据源插件

## 5.3 使用 DataX 迁移 MySQL 数据到 OceanBase 示例

将 MySQL 数据迁移到 OceanBase，如果源端和目标端不能同时跟 DataX 服务器网络联通，那么可以通过 CSV 文件中转。如果源端数据库和目标端数据库能同时跟 DataX 所在服务器联通，则可以使用 DataX 直接将数据从源端迁移到目标端。

示例：从 MySQL 迁移 tpccdb.bmsql\_order 表数据到 OceanBase MySQL 模式下的 tpcc.bmsql\_order。

myjob.json 配置文件如下：

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 4
      }
    }
  },
}
```

```
"errorLimit": {
  "record": 0,
  "percentage": 0.1
},
"content": [
  {
    "reader": {
      "name": "mysqlreader",
      "parameter": {
        "username": "tpcc",
        "password": "*****",
        "column": ["*"],
        "connection": [
          {
            "table": ["bmsql_order"],
            "jdbcUrl": ["jdbc:mysql://127.0.0.1:3306/tpccdb?
useUnicode=true&characterEncoding=utf8"]
          }
        ]
      }
    },
    "writer": {
      "name": "oceanbasev10writer",
      "parameter": {
        "obWriteMode": "insert",
        "column": ["*"],
        "preSql": ["truncate table bmsql_order"],
        "connection": [
```

```
{
  "jdbcUrl": "jdbc:oceanbase://127.0.0.1:2883/tpcc?",
  "table": ["bmsql_order"]
},
{
  "username": "tpcc",
  "password": "*****",
  "writerThreadCount": 10,
  "batchSize": 1000,
  "memstoreThreshold": "0.9"
}
}
```

#### 参数说明

参数	描述
name	描述的是连接数据库的 reader 或 writer 对应的数据库插件的名称。其中 MySQL 的 reader 插件为：mysqlreader，OceanBase 的 writer 插件为 oceanbasev10writer。具体 reader 和 writer 的插件可以参考 datax 的文档： <a href="#">DataX 数据源指南</a> 。



jdbcUrl	<p>描述的是到连接的数据库的 JDBC 信息，使用 JSON 的数组描述，并支持一个库填写多个连接地址。您在 JSON 数组中填写一个 JDBC 连接即可。jdbcUrl 按照 MySQL 官方规范，并可以填写连接附件控制信息。具体请参见 <a href="#">MySQL 官方文档</a>。</p> <h3>5.3.3.1 注意</h3> <ul style="list-style-type: none"><li>• jdbcUrl 必须包含在 connection 配置单元中。</li><li>• OceanBase 数据库需要通过 obproxy 进行连接，端口默认 2883。</li><li>• writer 端的 jdbcUrl，无需在连接串两端加 []，reader 端的 jdbcUrl，必需在连接串两端加 []</li><li>• 必选：是</li><li>• 默认值：无</li></ul>
username	<p>数据源的用户名</p> <p>必选：是</p> <p>默认值：无</p>
password	<p>数据源指定用户名的密码</p> <p>必选：是</p> <p>默认值：无</p>
table	<p>所选取的需要同步的表。使用 JSON 的数组描述，因此支持多张表同时抽取。当配置为多张表时，用户自己需确保多张表是同一 schema 结构，MySQLReader 不予检查表是否同一逻辑表。</p> <h3>5.3.3.2 注意</h3> <p>table 必须包含在 connection 配置单元中。</p> <ul style="list-style-type: none"><li>• 必选：是</li><li>• 默认值：无</li></ul>

<p>column</p>	<p>所配置的表中需要同步的列名集合，使用 JSON 的数组描述字段信息。columns 不建议配置为 <code>['*']</code>，因为表结构发生变化时该配置也会发生变化。建议的配置方式是指定具体的列名。支持列裁剪，即列可以挑选部分列进行导出。支持列换序，即列可以不按照表 schema 信息进行导出。支持常量配置，用户需要按照 MySQL SQL 语法格式：<code>["id", "`table`", "1", "bazhen.csy", "null", "to_char(a + 1)", "2.3", "true"]</code>。</p> <h3>5.3.3.3 说明</h3> <ul style="list-style-type: none"><li>• <code>id</code> 为普通列名。</li><li>• <code>table</code> 为包含保留字的列名；</li><li>• <code>1</code> 为整形数字常量。</li><li>• <code>bazhen.csy</code> 为字符串常量。</li><li>• <code>null</code> 为空指针。</li><li>• <code>to_char(a + 1)</code> 为表达式。</li><li>• <code>2.3</code> 为浮点数。</li><li>• <code>true</code> 为布尔值。</li><li>• 必选：是</li><li>• 默认值：无</li></ul>
---------------	---

where

筛选条件，MySQLReader 根据指定的column、table、where 条件拼接 SQL，并根据这个 SQL 进行数据抽取。在实际业务场景中，往往会选择当天的数据进行同步，可以将where 条件指定为 `gmt_create > $bizdate`。

### 5.3.3.4 注意

不可以将 `where` 条件指定为 `limit 10`，`limit` 不是 SQL 的合法 `where` 子句。`where` 条件可以有序地进行业务增量同步。如果不填写 `where` 语句，包括不提供 `where` 的 `key` 或者 `value`，DataX 均视作同步全量数据。

- 必选：否
- 默认值：无

配置 job 文件后，执行该 job。命令如下：

```
python datax.py ../job/myjob.json
```

## 5.4 更多信息

关于 DataX 的开源代码和更多信息，请参见 [DataX](#)。

## 6 使用 CloudCanal 从 MySQL 数据库迁移数据到 OceanBase 数据库

CloudCanal 是一款数据迁移同步工具，帮助企业快速构建高质量数据流通通道，产品包含 SaaS 模式和私有输出专享模式。开发团队核心成员来自大厂，具备数据库内核、大规模分布式系统、云产品构建背景，懂数据库，懂分布式，懂云产品商业和服务模式。

本文将介绍如何使用 CloudCanal 社区版 v2.2.5.36 将 MySQL 数据库中的数据同步到目标端 OceanBase 数据库 MySQL 模式中。

### 6.0.0.1 功能适用性

- CloudCanal 社区版从 v2.2.0.7 版本开始支持从 MySQL 数据库迁移数据至 OceanBase 数据库 MySQL 模式。详情请参见 [v2.2.0.7](#)。
- CloudCanal 暂时只支持 MySQL 数据库 V5.6 之后的版本作为源库。

## 6.1 前置条件

参考 [全新安装\(Linux/MacOS\)](#) 完成 CloudCanal 社区版的安装部署以及申请免费 License 并激活。

## 6.2 数据迁移操作步骤

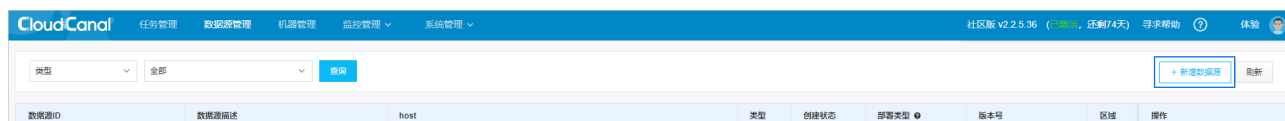
1. 添加数据源。
2. 创建任务。
3. 查看任务。

### 6.2.1 添加数据源

1. 登录 CloudCanal 平台。



## 2. 进入数据源管理界面，点击新增数据源。



## 3. 在新增数据源页面，填写数据源信息。

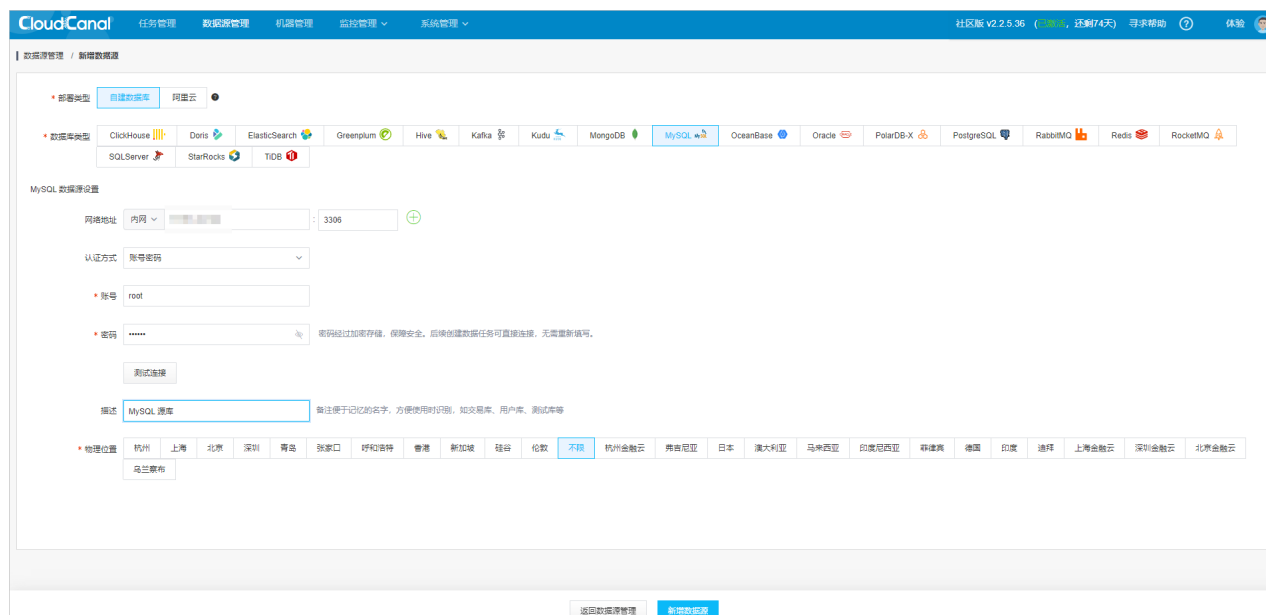
- 部署类型：有 **自建数据库** 和 **阿里云**。
  - 阿里云：用户在阿里云上购买的数据库实例。
  - 自建数据库：用户自己部署的数据库实例。
- 数据库类型：选择数据源类型。

新增两个数据源 MySQL 和 OceanBase，分别作为同步的源库和目标库：

- 选择自建数据库中 MySQL，添加自己部署的 MySQL 数据库实例。

MySQL 数据源设置：

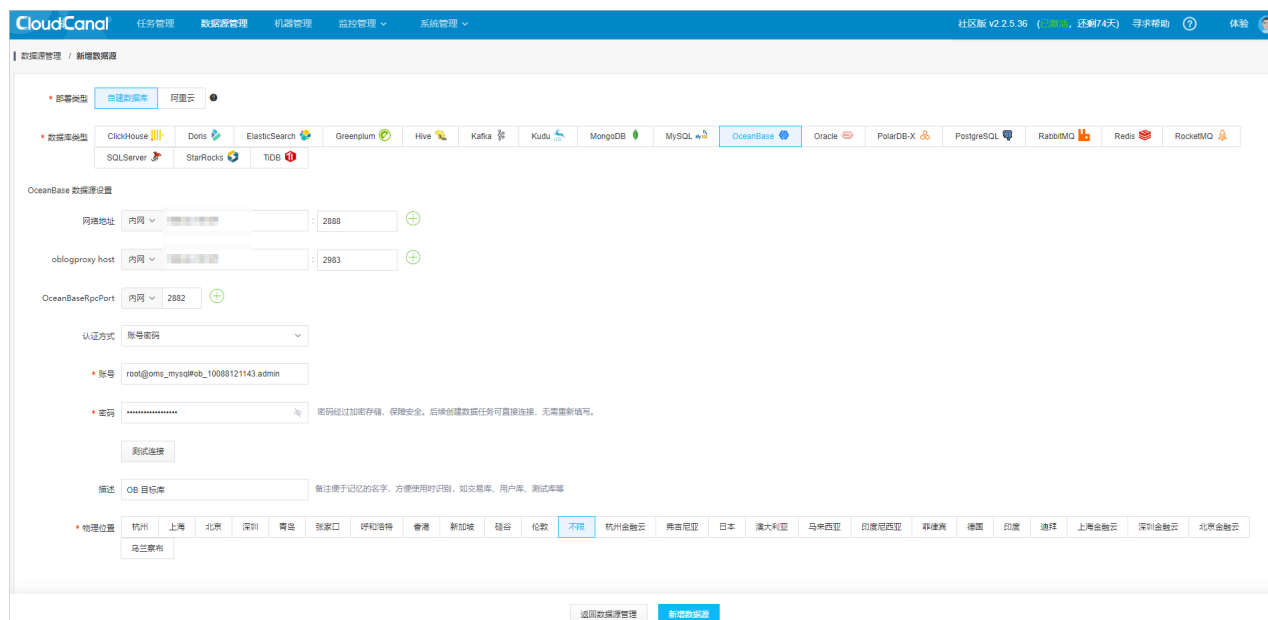
- 网络地址：填写 MySQL 数据库的 IP。
- 认证方式：有 **账号密码**、**有账号密码** 和 **无账号密码** 三种方式。默认 **账号密码** 方式。
- 账号：连接 MySQL 数据库的用户名。
- 密码：连接 MySQL 数据库的用户名对应的密码。
- 描述：选填项，备注便于记忆的名字，方便使用时识别，如交易库、用户库、测试库等。



- 选择自建数据库中 OceanBase，添加自己部署的 OceanBase 数据库实例。

OceanBase 数据源设置：

- 网络地址：填写连接 OceanBase 数据库的 IP，直连或通过 ODP 连接。
- oblogproxy host: oblogproxy 的 IP 地址。OceanBase 数据库作为源库增量同步时，不可以为空；OceanBase 数据库作为目标库时，可为空。有关 oblogproxy 的详细信息，请参考 [oblogproxy 介绍](#)。
- OceanBaseRpcPort: OceanBase Rpc 端口，默认 2882。
- 认证方式：分别为 **账号密码**、**有账号密码** 和 **无账号密码**。默认账号密码。
- 账号：连接 OceanBase 数据库的用户名。直连格式：**用户名@租户名称**；ODP 连接格式：**用户名@租户名称#集群名称**。
- 密码：连接 OceanBase 数据库的用户名对应的密码。
- 描述：选填项，备注便于记忆的名字，方便使用时识别，如交易库、用户库、测试库等。



#### 4. 查看新增的两个数据源。

数据源ID	数据源描述	host	类型	创建状态	部署类型	版本号	区域	操作
my-82ge913pm8zy628	MySQL 5.7	内网	MySQL	创建成功	自建数据库	5.7.26-log	不限	更多
ob-97es9uhq95dc5uu	OB 3231 MySQL 模式	内网	OceanBase	创建成功	自建数据库	5.7.25-OceanBase-v3.2.3.1	不限	更多

## 6.2.2 创建任务

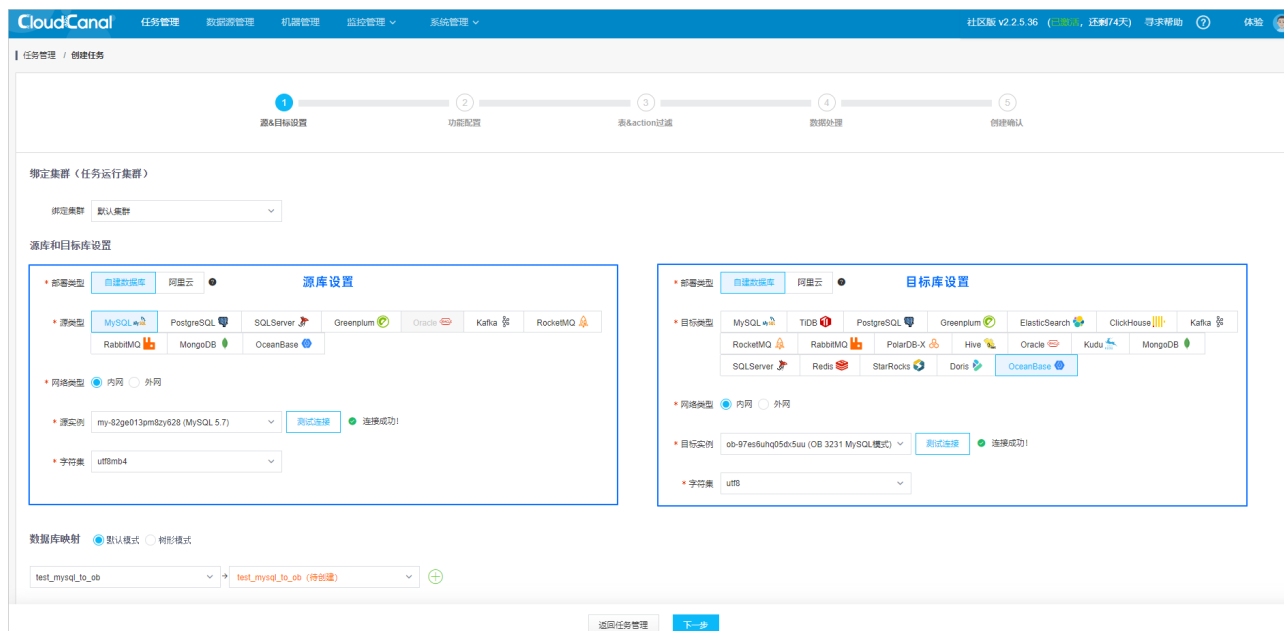
添加好数据源之后可以按照如下步骤进行数据全量迁移、增量同步和结构迁移。

#### 1. 任务管理 -> 创建任务。



#### 2. 源库和目标库设置。

- 选择 **任务运行集群**，任务会被调度到绑定集群的一台机器上执行。社区版部署完成后，会有一个默认的运行集群。
- 选择源库 MySQL 和目标数据库 OceanBase，并点击 **测试连接**。
- 选择需要 **迁移同步或校验的数据库**，指定数据库映射关系。
- 完成设置后点击 **下一步**。

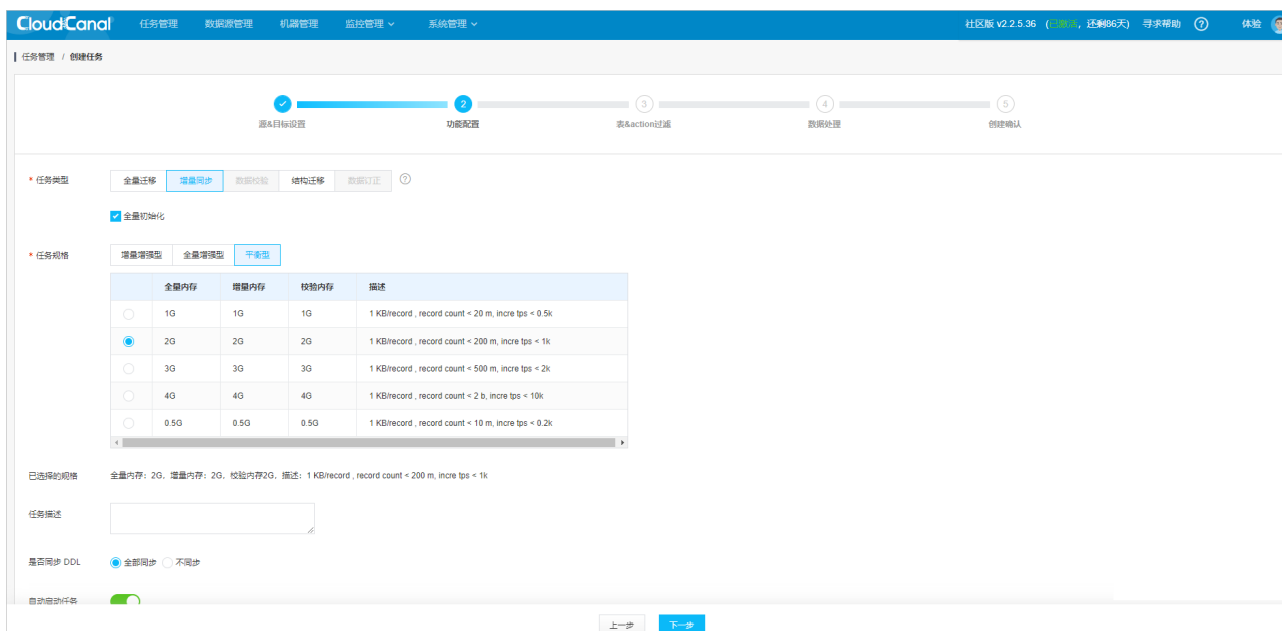


### 3. 功能配置。

选择 **增量同步** 功能，第一次会先查表进行全量同步，之后消费 binlog 增量同步数据。

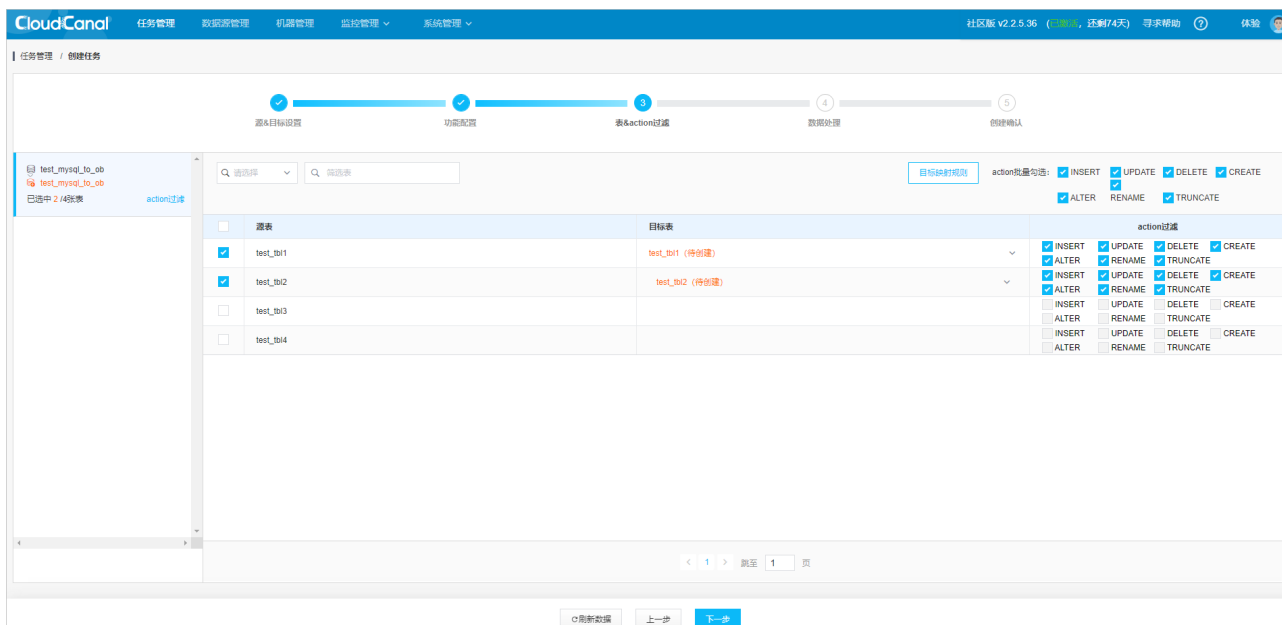
- 任务类型具有以下功能：
  - 全量迁移：以数据迁移为主，适合数据的全量搬迁及短期的增量同步任务。
  - 增量同步：默认选项，默认附带 **全量初始化**。以数据同步为主，适合长期的增量同步任务。
  - 数据校验：对比源端与目标端的数据，一次性或定时多次校验数据迁移的准确性。社区版不支持此功能。
  - 结构迁移：根据所选数据库、表自动创建对应的数据库、表。
  - 数据订正：对比源端与目标端的数据，将不一致的数据自动覆盖成和源端一致。社区版不支持此功能。
- 任务规格：默认 **平衡型**、**2G** 规格即可。
- 完成配置后，点击 **下一步**。





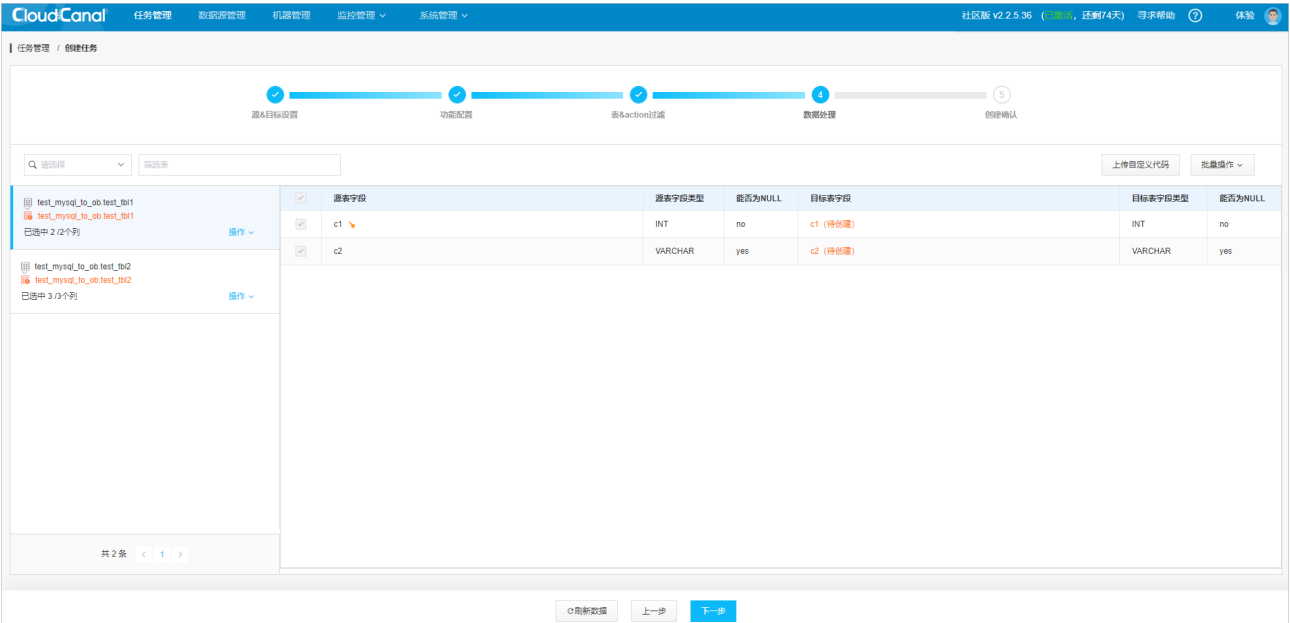
#### 4. 表&action 过滤。

- 选择要同步的表，要保证目标库的 UPDATE 和 DELETE 操作和源库的一致，需要保证源库表中有主键或者唯一约束。
- 完成配置后，点击 下一步。

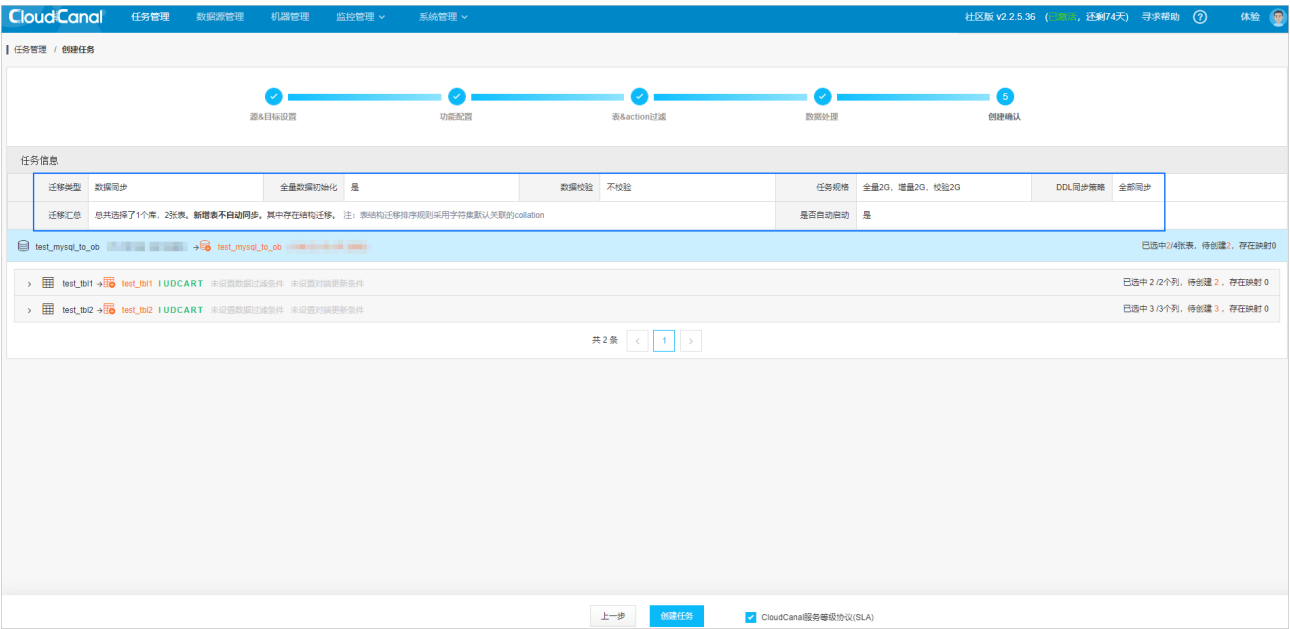


#### 5. 数据处理。

- 选择表中要同步的列。
- 完成设置后，点击 下一步。



6. 确认创建任务。  
最后一步，确认创建内容无误后点击确认创建。



### 6.2.3 查看任务状态

任务创建成功后，会默认进行结构迁移、全量迁移、增量同步，进度条会逐步发生变化。回到 CloudCanal 任务管理控制台，刷新并查看任务实时状态，从结构迁移、数据初始化，到数据同步。



## 6.3 相关文档

有关 CloudCanal 的详细信息，请参见 [CloudCanal 官方文档](#)。

## 7 使用 Canal 从 MySQL 数据库同步数据到 OceanBase 数据库

Canal 是 Alibaba 开源的一个产品，主要用途是基于 MySQL 数据库增量日志解析，提供增量数据订阅和消费。

本文档主要介绍使用 Canal 的 canal.deployer 和 canal.adapter 组件从 MySQL 数据库同步数据至 OceanBase 数据库。

### 7.1 架构原理

- canal deployer：canal 的 server 端，进行 binlog 到 CanalEntry 的转换。
- canal adapter：canal 的客户端适配器，解析 CanalEntry 并将增量变动同步到目的端。

Canal 相关信息访问地址：<https://github.com/alibaba/canal/releases>。

### 7.2 操作步骤

#### 7.2.1 步骤一：MySQL 相关设置

1. 修改 MySQL 配置文件 my.cnf。

配置文件 my.cnf 位置：/etc/my.cnf。先开启 binlog 写入功能，配置 binlog-format 为 ROW 模式，配置后需重启 MySQL 生效。

示例如下：

```
log-bin=mysql-bin # 开启 binlog
binlog-format=ROW # 选择 ROW 模式
server_id=1 # 配置 MySQL replaction 需要定义，不要和 canal 的 slaveId 重复
```

2. 创建一个 MySQL 用户。

创建一个连接 MySQL 的用户，用户名：canal，密码：\*\*\*\*\*。并且为 canal 授予所有库的读写权限。

示例如下：

```
MySQL [(none)]> CREATE USER 'canal'@'%' IDENTIFIED BY '*****';
Query OK, 0 rows affected
```

```
MySQL [(none)]> GRANT SELECT,REPLICATION SLAVE,REPLICATION CLIENT ON *.*  
TO 'canal'@'%';
```

Query OK, 0 rows affected

```
MySQL [(none)]> FLUSH PRIVILEGES;
```

Query OK, 0 rows affected

### 3. 创建一个测试数据库。

创建数据库 `test_mysql_to_ob`，表 `tbl1` 和 `tbl2`，并插入数据。

```
MySQL [(none)]> CREATE DATABASE test_mysql_to_ob;
```

Query OK, 1 row affected

```
MySQL [(none)]> USE test_mysql_to_ob;
```

Database changed

```
MySQL [test_mysql_to_ob]> CREATE TABLE tbl1(col1 INT PRIMARY KEY, col2  
VARCHAR(20),col3 INT);
```

Query OK, 0 rows affected

```
MySQL [test_mysql_to_ob]> INSERT INTO tbl1 VALUES(1,'China',86),(2,'Taiwan',  
886),(3,'Hong Kong',852),(4,'Macao',853),(5,'North Korea',850);
```

Query OK, 5 rows affected

Records: 5 Duplicates: 0 Warnings: 0

```
MySQL [test_mysql_to_ob]> CREATE TABLE tbl2(col1 INT PRIMARY KEY,col2  
VARCHAR(20));
```

Query OK, 0 rows affected

```
MySQL [test_mysql_to_ob]> INSERT INTO tbl2 VALUES(86,'+86'),(886,'+886'),
```

```
(852,'+852'),(853,'+853'),(850,'+850');
```

```
Query OK, 5 rows affected
```

```
Records: 5 Duplicates: 0 Warnings: 0
```

## 7.2.2 步骤二：Canal 的下载和安装

1. 下载软件包。

下载 [canal.deployer-1.1.5.tar.gz](https://github.com/alibaba/canal/releases/download/canal-1.1.5/canal.deployer-1.1.5.tar.gz)。

```
wget https://github.com/alibaba/canal/releases/download/canal-1.1.5/canal.deployer-1.1.5.tar.gz
```

2. 将压缩包解压至目录 `/Canal_Home/canal`。

```
mkdir /Canal_Home/canal && tar zxvf canal.deployer-1.1.5.tar.gz -C /Canal_Home/canal
```

3. 修改配置文件。

canal.deployer 默认的配置文件 `conf/canal.properties` 和 `conf/example/instance.properties`。这个是默认创建了一个 instance 叫 `example`。需要修改 `example` 的实例配置文件，修改数据库连接地址、用户名和密码。`canal.instance.connectionCharset` 代表数据库的编码方式对应到 Java 中的编码类型，比如 `UTF-8`，`GBK`，`ISO-8859-1`。

示例如下：

```
vi conf/example/instance.properties

# mysql serverId
canal.instance.mysql.slaveId = 1234
#position info, 需要改成自己的数据库信息
canal.instance.master.address = xxx.xxx.xxx.xxx:3306
canal.instance.master.journal.name =
canal.instance.master.position =
```

```
canal.instance.master.timestamp =  
#canal.instance.standby.address =  
#canal.instance.standby.journal.name =  
#canal.instance.standby.position =  
#canal.instance.standby.timestamp =  
#username/password, 需要改成自己的数据库信息  
canal.instance.dbUsername = canal  
canal.instance.dbPassword = *****  
canal.instance.defaultDatabaseName =  
canal.instance.connectionCharset = UTF-8  
#table regex  
canal.instance.filter.regex = .*\\\.*
```

4. 启动 Canal Server。

```
cd /Canal_Home/canal && sh bin/startup.sh
```

5. 查看 server 日志。

```
cat logs/canal/canal.log
```

6. 查看 instance 的日志。

```
tail -f logs/canal/canal.log  
tail -f logs/example/example.log
```

7. 如需停止服务可执行下述命令。

```
cd /Canal_Home/canal && sh bin/stop.sh
```

## 7.2.3 步骤三：部署 RDB 适配器

Canal Adapter 提供了对多种目标容器的支持，对于 OceanBase 来说，主要使用它的 rdb 模块，目的端容器为 OceanBase。

1. 下载软件包。

下载 [canal.adapter-1.1.5.tar.gz](https://github.com/alibaba/canal-adapter-rdb)。

```
wget https://github.com/alibaba/canal/releases/download/canal-1.1.5/canal.adapter-1.1.5.tar.gz
```

2. 将压缩包解压至目录 `/Canal_Home/adapter`。

```
mkdir /Canal_Home/adapter && tar zxvf canal.adapter-1.1.5.tar.gz -C /Canal_Home/adapter
```

3. 修改启动器配置。

修改启动器配置：`conf/application.yml`。首先指定 `adapter` 源端类型，通过 `mode` 指定，这里选择 `tcp`。后面就要指定 `canal.tcp` 相关属性，包括 `canal server` 的 IP 和端口，数据库的连接用户和密码。之后指定 `adapter` 目标端连接信息。`instance` 是源端实例名称，在 `canal` 部署的时候定义的。`key` 是自定义，名字后面有用。`jdbc` 相关属性是目标端 OceanBase 的连接方式，可以使用 MySQL 自带的驱动。

示例如下：

```
mode: tcp #tcp kafka rocketMQ rabbitMQ
flatMessage: true
zookeeperHosts:
syncBatchSize: 1000
retries: 0
timeout:
accessKey:
secretKey:
consumerProperties:
# canal tcp consumer
canal.tcp.server.host: 127.0.0.1:11111
canal.tcp.zookeeper.hosts:
canal.tcp.batch.size: 500
canal.tcp.username:
canal.tcp.password:
canalAdapters:
```



```
- instance: example # canal instance Name or mq topic name
groups:
- groupId: g1
outerAdapters:
- name: logger
- name: rdb
key: test_mysql_to_ob
properties:
jdbc.driverClassName: com.mysql.jdbc.Driver
jdbc.url: jdbc:mysql://xxx.xxx.xxx.xxx:2883/test_data?useUnicode=true
jdbc.username: root@mysql001#test4000
jdbc.password: *****
```

#### 4. RDB 映射文件。

修改 `conf/rdb/mytest_user.yml` 文件。其中, `destination` 指定的是 `canal` `instance` 名称; `outerAdapterKey` 是前面定义的 `key`; `mirrorDb` 指定数据库级别 DDL 和 DML 镜像同步。

映射有两种：一是按表映射；二是整库映射。下面以整库映射为例进行配置，注释部分为按表映射的配置：

```
[root@obce00 adapter]# cat conf/rdb/mytest_user.yml
#dataSourceKey: defaultDS
#destination: example
#groupId: g1
#outerAdapterKey: mysql1
#concurrent: true
#dbMapping:
# database: mytest
# table: user
# targetTable: mytest
# targetPk:
```

```
# id: id
# mapAll: true
# targetColumns:
# id:
# name:
# role_id:
# c_time:
# test1:
# etlCondition: "where c_time>={}"
# commitBatch: 3000 # 批量提交的大小
# Mirror schema synchronize config
dataSourceKey: defaultDS
destination: example
groupId: g1
outerAdapterKey: test_mysql_to_ob
concurrent: true
dbMapping:
mirrorDb: true
database: test_data
commitBatch: 1000
```

5. 启动 RDB。

## 7.2.3.1 说明

如果使用了 OceanBase 的驱动，则将目标库 OceanBase 驱动包放入 `lib` 文件夹。

启动 canal-adapter 启动器。

```
``bash
cd /Canal_Home/adapter && sh bin/startup.sh
``
```

6. 查看 RDB 日志。

```
tail -f logs/adapter/adapter.log
```

7. 停止服务可执行下述命令。

```
cd /Canal_Home/adapter && bin/stop.sh
```

8. 查看数据同步情况。

在 MySQL 源端写入数据，在 OceanBase 目标端查看数据同步。

## 7.3 功能限制

- 同步的表必须有主键。否则，源端删除无主键表的任意一笔记录，同步到目标端会导致整个表被删除。
- DDL 支持新建表、新增列。

## 8 使用 Flink CDC 从 MySQL 数据库同步数据到 OceanBase 数据库

Flink CDC (CDC Connectors for Apache Flink) 是 Apache Flink 的一组 Source 连接器，它支持从大多数数据库中实时地读取存量历史数据和增量变更数据。Flink CDC 能够将数据库的全量和增量数据同步到消息队列和数据仓库中。Flink CDC 也可以用于实时数据集成，您可以使用它将数据库数据实时导入数据湖或者数据仓库。同时，Flink CDC 还支持数据加工，您可以通过它的 SQL Client 对数据库数据做实时关联、打宽、聚合，并将结果写入到各种存储中。CDC (Change Data Capture, 即变更数据捕获) 能够帮助您监测并捕获数据库的变动。CDC 提供的数据可以做很多事情，比如：做历史库、做近实时缓存、提供给消息队列 (MQ)，用户消费 MQ 做分析和审计等。

以下将介绍使用 Flink CDC 从 MySQL 数据库同步数据到 OceanBase 数据库。

### 8.1 Flink CDC 环境准备

下载 Flink 和所需要的依赖包：

1. 通过 [下载地址](#) 下载 Flink。本文档使用的是 Flink 1.15.3，并将其解压至目录 `/FLINK_HOME/flink-1.15.3`。
2. 下载下面列出的依赖包，并将它们放到目录 `/FLINK_HOME/flink-1.15.3/lib/` 下。
  - [flink-sql-connector-mysql-cdc-2.1.1.jar](#)
  - [flink-connector-jdbc-1.15.3.jar](#)
  - [mysql-connector-java-5.1.47.jar](#)

### 8.2 准备数据

#### 8.2.1 准备 MySQL 数据库数据

在 MySQL 数据库中准备测试数据，作为导入 OceanBase 数据库的源数据。

1. 进入 MySQL 数据库。

```
[xxx@xxx /...]  
$mysql -hxxx.xxx.xxx -P3306 -uroot -p*****  
<Omit echo information>  
  
MySQL [(none)]>
```

2. 创建数据库 `test_mysql_to_ob`，表 `tbl1` 和 `tbl2`，并插入数据。

```
MySQL [(none)]> CREATE DATABASE test_mysql_to_ob;
```

```
Query OK, 1 row affected
```

```
MySQL [(none)]> USE test_mysql_to_ob;
```

```
Database changed
```

```
MySQL [test_mysql_to_ob]> CREATE TABLE tbl1(col1 INT PRIMARY KEY, col2  
VARCHAR(20),col3 INT);
```

```
Query OK, 0 rows affected
```

```
MySQL [test_mysql_to_ob]> INSERT INTO tbl1 VALUES(1,'China',86),(2,'Taiwan',  
886),(3,'Hong Kong',852),(4,'Macao',853),(5,'North Korea',850);
```

```
Query OK, 5 rows affected
```

```
Records: 5 Duplicates: 0 Warnings: 0
```

```
MySQL [test_mysql_to_ob]> CREATE TABLE tbl2(col1 INT PRIMARY KEY,col2  
VARCHAR(20));
```

```
Query OK, 0 rows affected
```

```
MySQL [test_mysql_to_ob]> INSERT INTO tbl2 VALUES(86,'+86'),(886,'+886'),  
(852,'+852'),(853,'+853'),(850,'+850');
```

```
Query OK, 5 rows affected
```

```
Records: 5 Duplicates: 0 Warnings: 0
```

## 8.2.2 准备 OceanBase 数据库数据

在 OceanBase 数据库中创建存放源数据的表。

1. 登录 OceanBase 数据库。

使用 `user001` 用户登录集群的 `mysql001` 租户。

```
[xxx@xxx /...]  
$obclient -h10.10.10.2 -P2881 -uuser001@mysql001 -p -A  
Enter password:  
Welcome to the OceanBase. Commands end with ; or \g.  
Your OceanBase connection id is 3221536981  
Server version: OceanBase 4.0.0.0 (r100000302022111120-  
7cef93737c5cd03331b5f29130c6e80ac950d33b) (Built Nov 11 2022 20:38:33)  
  
Copyright (c) 2000, 2018, OceanBase and/or its affiliates. All rights reserved.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
obclient [(none)]>
```

2. 创建数据库 `test_mysql_to_ob` 和表 `mysql_tbl1_and_tbl2`。

```
obclient [(none)]> CREATE DATABASE test_mysql_to_ob;  
Query OK, 1 row affected  
  
obclient [(none)]> USE test_mysql_to_ob;  
Database changed  
obclient [test_mysql_to_ob]> CREATE TABLE mysql_tbl1_and_tbl2(col1 INT  
PRIMARY KEY,col2 INT,col3 VARCHAR(20),col4 VARCHAR(20));  
Query OK, 0 rows affected
```

## 8.3 启动 Flink 集群和 Flink SQL CLI

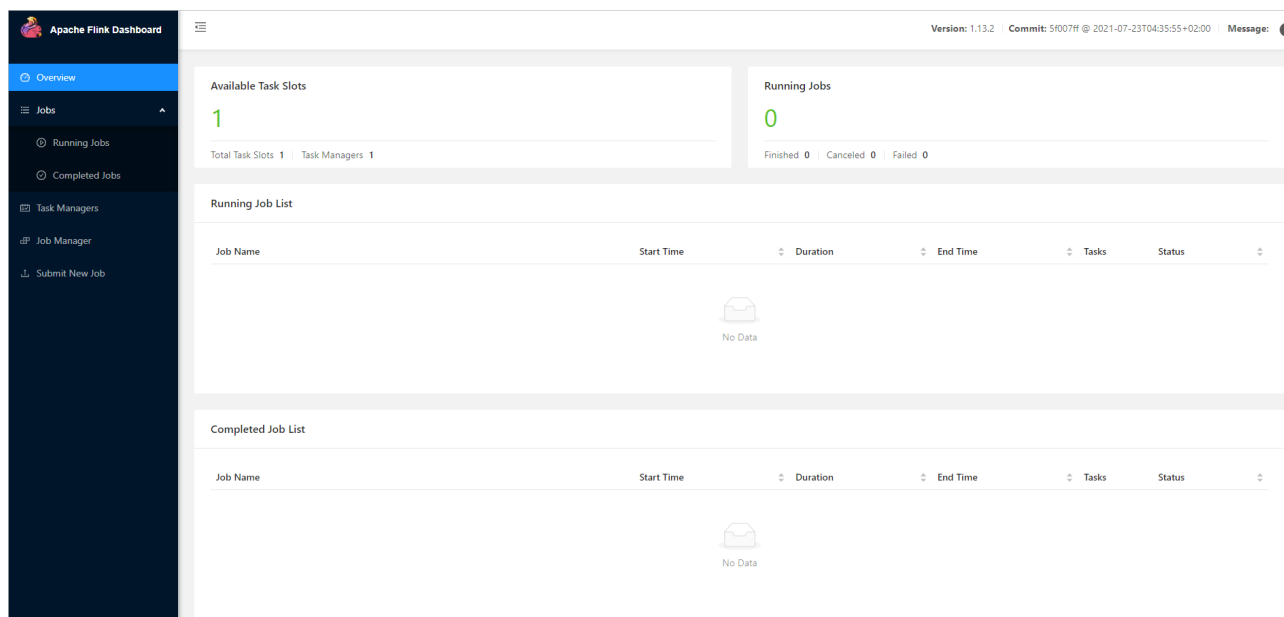
1. 使用下面的命令跳转至 Flink 目录下。

```
[xxx@xxx /FLINK_HOME]  
#cd flink-1.15.3
```

2. 使用下面的命令启动 Flink 集群。

```
[xxx@xxx /FLINK_HOME/flink-1.15.3]
#./bin/start-cluster.sh
```

启动成功的话，可以在 `http://localhost:8081/` 访问到 Flink Web UI，如下所示：



### 8.3.2.1 说明

执行 `./bin/start-cluster.sh` 后，如果提示：`bash: ./bin/start-cluster.sh: Permission denied`。需要把 `flink-1.15.3` 目录下的所有 `-rw-rw-r--` 权限的文件权限都设置为 `-rwxrwxrwx` 权限。

示例如下：

```
[xxx@xxx /FLINK_HOME/flink-1.15.3]
# chmod -R 777 /FLINK_HOME/flink-1.15.3/*
```

3. 使用下面的命令启动 Flink SQL CLI。

```
[xxx@xxx /FLINK_HOME/flink-1.15.3]
#./bin/sql-client.sh
```

启动成功后，可以看到如下的页面：



### 8.3.3 设置 checkpoint

在 Flink SQL CLI 中开启 `checkpoint`，每隔 3 秒做一次 `checkpoint`。

```
Flink SQL> SET execution.checkpointing.interval = 3s;
```

```
[INFO] Session property has been set.
```

### 8.3.4 创建 MySQL CDC 表

在 Flink SQL CLI 中创建 MySQL 数据库对应的表。

对于 MySQL 数据库中 `test_mysql_to_ob` 的表 `tbl1` 和 `tbl2` 使用 Flink SQL CLI 创建对应的表，用于同步这些底层数据库表的数据。



```
Flink SQL> CREATE TABLE mysql_tbl1 (  
col1 INT PRIMARY KEY,  
col2 VARCHAR(20),  
col3 INT)  
WITH (  
'connector' = 'mysql-cdc',  
'hostname' = 'xxx.xxx.xxx.xxx',  
'port' = '3306',  
'username' = 'root',  
'password' = '*****',  
'database-name' = 'test_mysql_to_ob',  
'table-name' = 'tbl1');  
[INFO] Execute statement succeed.  
  
Flink SQL> CREATE TABLE mysql_tbl2 (col1 INT PRIMARY KEY,  
col2 VARCHAR(20))  
WITH ('connector' = 'mysql-cdc',  
'hostname' = 'xxx.xxx.xxx.xxx',  
'port' = '3306',  
'username' = 'root',  
'password' = '*****',  
'database-name' = 'test_mysql_to_ob',  
'table-name' = 'tbl2');  
[INFO] Execute statement succeed.
```

有关 MySQL CDC Connector WITH 选项的详细信息，请参见 [Connector Options](#)。

## 8.3.5 创建 OceanBase CDC 表

在 Flink SQL CLI 中创建 OceanBase 数据库对应的表。创建 `mysql_tbl1_and_tbl2` 表，用来将关联后的数据写入 OceanBase 数据库中。

```
Flink SQL> CREATE TABLE mysql_tbl1_and_tbl2(
col1 INT PRIMARY KEY,
col2 INT,col3 VARCHAR(20),
col4 VARCHAR(20))
WITH ('connector' = 'jdbc',
'url' = 'jdbc:mysql://10.10.10.2:2881/test_mysql_to_ob',
'username' = 'root@mysql001',
'password' = '*****',
'table-name' = 'mysql_tbl1_and_tbl2');
[INFO] Execute statement succeed.
```

有关 JDBC SQL Connector WITH 选项的详细信息，请参见 [Connector Options](#)。

## 8.3.6 在 Flink SQL CLI 中将数据写入 OceanBase 数据库中

使用 Flink SQL 将表 tbl1 与表 tbl2 关联，并将关联后的信息写入 OceanBase 数据库中。

```
Flink SQL> INSERT INTO mysql_tbl1_and_tbl2
SELECT t1.col1,t1.col3,t1.col2,t2.col2
FROM mysql_tbl1 t1,mysql_tbl2 t2
WHERE t1.col3=t2.col1;
[INFO] Submitting SQL update statement to the cluster...
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is
`com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and
manual loading of the driver class is generally unnecessary.
[INFO] SQL update statement has been successfully submitted to the cluster:
Job ID: c5ee92498addf813858e448ec25e85af
```

### 8.3.6.2 说明

本文档测试示例使用的 MySQL 驱动（`com.mysql.jdbc.Driver`）是 MySQL Connector/J 5.1.47 版本。新版本 MySQL 驱动（`com.mysql.cj.jdbc.Driver`）请使用 MySQL Connector/J 8.x 版本。

## 8.4 查看关联数据写入 OceanBase 数据库情况

登录 OceanBase 数据库，在 `test_mysql_to_ob` 库中查看表 `mysql_tbl1_and_tbl2` 的数据。

```
obclient [test_mysql_to_ob]> SELECT * FROM mysql_tbl1_and_tbl2;
```

```
+-----+-----+-----+-----+
```

```
| col1 | col2 | col3 | col4 |
```

```
+-----+-----+-----+-----+
```

```
| 1 | 86 | China | +86 |
```

```
| 2 | 886 | Taiwan | +886 |
```

```
| 3 | 852 | Hong Kong | +852 |
```

```
| 4 | 853 | Macao | +853 |
```

```
| 5 | 850 | North Korea | +850 |
```

```
+-----+-----+-----+-----+
```

```
5 rows in set
```

## 8.5 查看数据更新情况

1. 在 MySQL 数据库的表 `tbl1` 和 `tbl2` 中插入分别插入一条数据。

```
MySQL [test_mysql_to_ob]> INSERT INTO tbl1 VALUES(6,'code',673);
```

```
Query OK, 1 row affected
```

```
MySQL [test_mysql_to_ob]> INSERT INTO tbl2 VALUES(673,'+673');
```

```
Query OK, 1 row affected
```

2. 在 OceanBase 数据库中查看数据是否同步。

```
obclient [test_mysql_to_ob]> SELECT * FROM mysql_tbl1_and_tbl2;
```

```
+-----+-----+-----+-----+
```

```
| col1 | col2 | col3 | col4 |
```

```
+-----+-----+-----+-----+
```

```
| 1 | 86 | China | +86 |
```

```
| 2 | 886 | Taiwan | +886 |
```

```
| 3 | 852 | Hong Kong | +852 |
```

```
| 4 | 853 | Macao | +853 |
```

```
| 5 | 850 | North Korea | +850 |
```

```
| 6 | 673 | code | +673 |
```

```
+-----+-----+-----+-----+
```

```
6 rows in set
```

## 9 使用 ChunJun 从 MySQL 数据库迁移数据到 OceanBase 数据库

纯钧（ChunJun，原名 FlinkX）是一款稳定、易用、高效、批流一体的数据集成框架，基于实时计算引擎 Flink，支持 JSON 模版配置任务，兼容 Flink SQL 语法、支持分布式运行，支持 flink-standalone、yarn-session、yarn-per job 等多种提交方式和支持全量同步、增量同步等特点。

以下将介绍使用 ChunJun 的 Local 模式从 MySQL 数据库迁移数据到 OceanBase 数据库（MySQL 模式）。

### 9.1 场景描述

将 MySQL 数据库表 `tbl1` 的数据迁移至 OceanBase 数据库表 `test_tbl1` 中，数据库信息如下：

MySQL 数据库信息（源库）	示例值
主机地址	xxx.xxx.xxx.xxx
端口号	3306
用户名称	root
用户的密码	*****
Schema 库名称	test_mysql_to_ob
表名	tbl1

OceanBase 数据库信息（目标库）	示例值
集群名	test4000
主机地址	10.10.10.2
端口号	2881
业务租户名称（MySQL 模式）	mysql001
用户名称	root
用户的密码	*****
Schema 库名称	test_data
表名	test_tbl1

### 9.2 前提条件

安装 JDK 1.8，并配置好 `JAVA_HOME` 环境变量。

## 9.3 操作步骤

### 9.3.1 步骤一：ChunJun 环境准备

1. 下载 ChunJun 压缩包并解压。

下载 [chunjun-dist.tar.gz](https://github.com/DTStack/chunjun/releases/download/v1.12.6/chunjun-dist.tar.gz)。

```
wget https://github.com/DTStack/chunjun/releases/download/v1.12.6/chunjun-dist.tar.gz
```

将压缩包解压至目录 `/chunjun_Home/chunjun`。

```
mkdir /ChunJun_Home/chunjun && tar zxvf chunjun-dist.tar.gz -C /ChunJun_Home/chunjun
```

2. 配置 ChunJun 环境变量。

```
export ChunJun_HOME=/ChunJun_Home/chunjun-dist
```

### 9.3.2 步骤二：配置 json 文件

根据环境信息配置将 MySQL 数据库表 `tbl1` 的数据迁移至 OceanBase 数据库表 `test_tbl1` 的 json 文件。

示例如下：

```
[root@xxx /]  
$cd /ChunJun_Home/chunjun-dist/chunjun-examples/json  
  
[root@xxx /ChunJun_Home/chunjun-dist/chunjun-examples/json]  
$mkdir test_data  
  
[root@xxx /ChunJun_Home/chunjun-dist/chunjun-examples/json]  
$cd mkdir test_data
```

```
[root@xxx /ChunJun_Home/chunjun-dist/chunjun-examples/json/test_data]
$vi test_mysql_to_ob.json

{
  "job": {
    "setting": {
      "errorLimit": {
        "record": 0,
        "percentage": 0.02},
      "speed": {"bytes": 0,
        "channel": 1}
    },
    "content": [
      {
        "reader": {
          "name": "mysqlreader",
          "parameter": {
            "username": "root",
            "password": "*****",
            "column": [
              "*"
            ],
            "connection": [
              {
                "jdbcUrl": [
                  "jdbc:mysql://xxx.xxx.xxx.xxx:3306/test_mysql_to_ob"
                ],
                "table": [
                  "tbl1"
                ]
              }
            ]
          }
        }
      }
    ]
  }
}
```

```
]
}
]
}
},
"writer": {
  "name": "mysqlwriter",
  "parameter": {
    "column": [
      "*"
    ],
    "connection": [
      {
        "jdbcUrl": "jdbc:mysql://10.10.10.2:2881/test_data",
        "table": [
          "test_tbl1"
        ]
      }
    ],
    "username": "root@mysql001",
    "password": "*****"
  }
}
}
]
}
```

### 9.3.3 步骤三：运行 json 配置文件

启动一个 JVM 进程执行 ChunJun 任务。



进入到 `/ChunJun_Home/chunjun-dist` 目录，执行命令如下命令：

```
[root@xxx /]
$cd /ChunJun_Home/chunjun-dist

[admin@xxx /ChunJun_Home/chunjun-dist]
$sh bin/chunjun-local.sh -job chunjun-examples/json/test_data/test_mysql_to_ob.
json

# #
# #
#
#####
# # # # #
# # # # #
# # # # #
# # # # #
#####
#
####

Reference site: https://dtstack.github.io/chunjun

chunjun is starting ...
CHUNJUN_HOME is auto set /ChunJun_Home/chunjun-dist
FLINK_HOME is empty!
HADOOP_HOME is empty!
...
<ellipsis>
...
*****
```

```
numWrite | 3
last_write_num_0 | 0
conversionErrors | 0
writeDuration | 20130
duplicateErrors | 0
numRead | 3
snapshotWrite | 0
otherErrors | 0
readDuration | 78
byteRead | 389
last_write_location_0 | 0
byteWrite | 389
nullErrors | 0
nErrors | 0
*****

2023-01-06 19:18:57,306 - 50207 INFO [main] com.dtstack.chunjun.Main:program
Flink_Job execution success
```

### 9.3.4 步骤四：查看数据迁移情况

登录到 OceanBase 数据库查看数据是否迁移成功。

```
[admin@xxx /home/admin]
$obclient -h10.10.10.2 -P2881 -uroot@mysql001 -p -A
Enter password:
Welcome to the OceanBase. Commands end with ; or \g.
Your OceanBase connection id is 3221630190
Server version: OceanBase 4.0.0.0 (r101000022022120716-
0d7927892ad6d830e28437af099f018b0ad9a322) (Built Dec 7 2022 16:22:15)
```

Copyright (c) 2000, 2018, OceanBase and/or its affiliates. All rights reserved.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
obclient [(none)]> use test_data;
```

Database changed

```
obclient [test_data]> SELECT * FROM test_tbl1;
```

```
+-----+-----+-----+
```

```
| COL1 | COL2 | COL3 |
```

```
+-----+-----+-----+
```

```
| 1 | China | 86 |
```

```
| 2 | Taiwan | 886 |
```

```
| 3 | Hong Kong | 852 |
```

```
+-----+-----+-----+
```

3 rows in set

## 9.4 相关文档

- 更多 ChunJun 特点的信息介绍，请参见 [Features of ChunJun](#)。
- ChunJun 快速入门的详细信息，请参见 [快速开始](#)。
- MySQL 数据库作为源库配置 json 文件的详细信息，请参见 [MySQL Source](#)。
- 更多脚本示例，请参见项目内 `/chunjun-dist/chunjun-examples` 文件夹下的文件。

## 10 使用 OMS 从 OceanBase 数据库 MySQL 租户迁移数据到 MySQL 数据库

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 OceanBase 数据库 MySQL 租户迁移数据到 MySQL 数据库中的背景信息。

### 10.1 背景信息

在 OMS 控制台创建从 OceanBase 数据库 MySQL 租户迁移数据至 MySQL 数据库的数据迁移项目。您可以通过结构迁移、全量迁移和增量同步，无缝迁移源端数据库中的存量业务数据和增量数据至 MySQL 数据库。

MySQL 数据库支持单主库、单备库和主备库等模式。迁移 OceanBase 数据库 MySQL 租户的数据至 MySQL 数据库时，不同类型的数据源支持的操作也不同。

类型	支持的操作
单主库	结构迁移 + 全量迁移 + 增量同步 + 全量校验 + 反向增量
单备库	不支持将单备库的 MySQL 数据源作为数据迁移的目标端
主备库	<div>主库：支持结构迁移 + 全量迁移 + 增量同步 + 反向增量</div> <div>备库：支持全量校验</div> <div><h4>10.1.0.1 说明</h4><p>当选择主库+备库的数据源时，OMS 将使用备库作为全量校验的主节点。</p></div>

### 10.2 相关文档

更多使用 OMS 从 OceanBase 数据库 MySQL 租户迁移数据到 MySQL 数据库的操作信息，请参见 [迁移 OceanBase 数据库 MySQL 租户的数据至 MySQL 数据库](#)。

# 11 使用 OMS 从 OceanBase 数据库 Oracle 租户迁移增量数据到 MySQL 数据库

本文将简单介绍使用 OceanBase 迁移服务 (OceanBase Migration Service, OMS) 从 OceanBase 数据库 Oracle 租户迁移增量数据到 MySQL 数据库中的背景信息。

## 11.1 背景信息

在 OMS 控制台创建从 OceanBase 数据库 Oracle 租户迁移数据至 MySQL 数据库的数据迁移项目。您可以通过增量同步，迁移源 OceanBase 数据库 Oracle 租户中的增量数据至 MySQL 数据库。

## 11.2 相关文档

更多使用 OMS 从 OceanBase 数据库 Oracle 租户迁移增量数据到 MySQL 数据库的操作信息，请参见 [迁移 OceanBase 数据库 Oracle 租户的增量数据至 MySQL 数据库](#)。

## 12 使用 DBCAT 迁移 OceanBase 表结构到 MySQL 数据库

DBCAT 是一款轻量级的命令行工具，可用于提供数据库之间 DDL 转换和 Schema 比对等功能。这里以 DBCAT 迁移表结构为例进行介绍。

DBCAT 安装包文件名为 `dbcat-[版本号]-SNAPSHOT.tar.gz`，下载后解压缩即可使用，可执行文件名为 `dbcat`。

### 12.0.0.1 注意

DBCAT 是 OMS 的一个组件，在社区版环境中推荐使用 OMS 导出。

## 12.1 环境准备

DBCAT 能运行在 CentOS、macOS 和 Windows 下。需要安装 JDK 1.8 以上（含）版本。可以使用 OpenJDK，安装好后配置环境变量 `JAVA_HOME`。

CentOS 安装 OpenJDK 示例：

```
$sudo yum -y install java-1.8.0-openjdk.x86_64

$which java
/usr/local/java/jdk1.8.0_261/bin/java

echo 'export JAVA_HOME=/usr/local/java/jdk1.8.0_261/' >> ~/.bash_profile
. ~/.bash_profile
```

解压安装文件：

```
tar zxvf dbcat-1.8.0-SNAPSHOT.tar.gz
cd dbcat-1.8.0-SNAPSHOT/
chmod +x bin/dbcat

$tree -L 3 --filelimit 30
```

```
.
├── bin
│   ├── dbcat
│   ├── dbcat.bat
│   └── dbcat-debug
├── conf
│   ├── dbcat.properties
│   └── logback.xml
├── docs
│   ├── README.docx
│   ├── README.md
│   └── README.txt
├── LEGAL.md
├── lib [45 entries exceeds filelimit, not opening dir]
├── LICENSE
├── meta
│   └── README
└── NOTICE

5 directories, 12 files
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。
conf	日志文件配置目录。
lib	运行时期依赖的包。
meta	离线转换场景下，导出字典表数据。
~/output	SQL 文件与报告文件，运行时生成。

## 12.2 导出 OceanBase 数据库的 MySQL 租户下的表结构

DBCAT 具有在线转换功能，该功能是指 DBCAT 能直连源端数据库，将数据库中的对象导出。当对象非常多时（如超过 1 万），导出过程可能会有点慢。

dbcat 导出命令如下：

```
bin/dbcat convert -H<host> -P<port> -u<user> -p<*****> -D <database> --from  
<from> --to <to> --all
```

您可运行命令 `bin/dbcat help convert` 查看更多参数信息。



必选参数：

选项	有无参数	中文描述
-H/--host	Y	数据库服务器的 IP 地址
-P/--port	Y	数据库服务器的端口
-u/--user	Y	登录数据库的用户名
-t/--tenant	Y	连接 OceanBase 集群需要提供租户名
-c/--cluster	Y	连接 OceanBase 集群需要提供集群名
-p/--password	Y	登录数据库的密码
-D/--database	Y	数据库名（源库），DB2 须区分数据库名和模式名
--service-id	Y	连接 Oracle 数据库需要提供服务 ID
--service-name	Y	连接 Oracle 数据库需要提供服务名
--as-sysdba	N	连接 Oracle 数据库 sysdba 角色
--sys-user	Y	连接 OceanBase 集群系统租户的用户名
--sys-password	Y	连接 OceanBase 集群系统租户的密码
--schema	Y	模式名（源库），非DB2，模式名与数据名相同
--from	Y	源库的类型
--to	Y	目标库的类型
--all	N	所有的数据库对象

可选参数：

选项	有无参数	中文描述
-f/--file	Y	sql 文件的输出路径
--offline	N	使用离线模式
--target-schema	Y	式名（目标库）
--table	Y	导出的表
--view	Y	导出的视图
--trigger	Y	导出的触发器
--synonym	Y	导出的同义词
--sequence	Y	导出的序列
--function	Y	导出的函数
--procedure	Y	导出的存储过程
--dblink	Y	导出所有的 DBLink
--type	Y	导出的 type
--type-body	Y	导出的 type body
--package	Y	导出的 package
--package-body	Y	导出的 package body
--no-quote	N	产生的 DDL 不带引号
--no-schema	N	产生的 DDL 不带模式名
--target-schema	Y	产生的 DDL 中使用指定的模式名
--exclude-type	Y	搭配 --all 使用，如：--all --exclude-type 'TABLE' 表示排除 TABLE 类型

这里以导出 4.0.0 版本的 OceanBase 集群中的 MySQL 租户下 database 为 test 的所有对象的结构，并将其迁移到 MySQL 5.7 版本下为示例。

```
bin/dbcat convert -H 172.30.xxx.xxx -P 2883 -uroot -pxxxxxx -D test --from
obmysql40 --to mysql57 --all
```

特别说明：

- dbcat 不需要直接安装在数据库主机上，安装在可直连数据库主机的主机上即可。

- 参数中的 --from 和 --to 为源端和目的端的数据库类型，需要详细到版本号。当前 dbcat 支持的源端和目标端数据库详细如下：

源端数据库类型	目标端数据库类型
TiDB	OBMYSQL
PG	OBMYSQL
SYBASE	OBORACLE
MYSQL	OBMYSQL
ORACLE	OBORACLE
ORACLE	OBMYSQL
DB2 IBM i	OBORACLE
DB2 LUW	OBORACLE
DB2 LUW	OBMYSQL
OBMYSQL	MYSQL
OBORACLE	ORACLE

其中 OBMYSQL 为 OceanBase 数据库的 MySQL 租户，OBORACLE 为 OceanBase 数据库的 Oracle 租户。

- 当前支持的源端和目标端数据库详细的版本，详情如下。

数据库类型	数据库版本
TiDB	tidb4 tidb5
PG	pgsql10
SYBASE	sybase15
DB2 IBM i	db2ibmi71
DB2 LUW	db2luw970 db2luw1010 db2luw1050 db2luw111 db2luw115
MYSQL	mysql56 mysql57 mysql</80>

ORACLE	oracle9i oracle10g oracle11g oracle12c oracle18c oracle19c
OBMYSQL	obmysql14x obmysql21x obmysql22x obmysql200 obmysql211 obmysql2210 obmysql2230 obmysql2250 obmysql2271 ~ obmysql2277 obmysql30x obmysql31x obmysql32x obmysql322 obmysql40
OBORACLE	oboracle2220 oboracle2230 oboracle2250 oboracle2270 ~ oboracle2277 oboracle21x oboracle22x oboracle30x oboracle31x oboracle32x oboracle322 oboracle40

运行后的输出文件在用户 `home` 目录的 `output` 下。

```
$tree ~/output/dbcat-20xx-xx-xx-164533/
/home/qing.meiq/output/dbcat-20xx-xx-xx-164533/
|—— tpccdb
|  |—— TABLE-schema.sql
|—— tpccdb-conversion.html

1 directory, 2 files
```

## 12.3 导入 MySQL 数据库

使用 DBCAT 导出的文件格式为 SQL 文件，可以使用 source 命令，示例如下：

```
MySQL [test]> source TABLE-schema.sql
Query OK, 0 rows affected (0.044 sec)
```

### 12.3.0.1 注意

如果 sql 文件不在当前目录下，则需要使用绝对地址。

## 12.4 导数结果验证

示例：查看一个表结构在 MySQL 里的书写方式 和 OceanBase 数据库里的表结构。

查看源数据库 OceanBase 的表 bmsql\_customer 的表结构：

```
obclient [test]> desc bmsql_customer;
+-----+-----+-----+-----+-----+
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
+-----+
| c_w_id | bigint(20) | NO | PRI | NULL | |
| c_d_id | bigint(20) | NO | PRI | NULL | |
| c_id   | bigint(20) | NO | PRI | NULL | |
| c_discount | decimal(4,4) | YES | | NULL | |
| c_credit | char(2) | YES | | NULL | |
| c_last | varchar(16) | YES | | NULL | |
| c_first | varchar(16) | YES | | NULL | |
| c_credit_lim | decimal(12,2) | YES | | NULL | |
| c_balance | decimal(12,2) | YES | | NULL | |
| c_ytd_payment | decimal(12,2) | YES | | NULL | |
| c_payment_cnt | bigint(20) | YES | | NULL | |
| c_delivery_cnt | bigint(20) | YES | | NULL | |
```

```
| c_street_1 | varchar(20) | YES || NULL ||
| c_street_2 | varchar(20) | YES || NULL ||
| c_city | varchar(20) | YES || NULL ||
| c_state | char(2) | YES || NULL ||
| c_zip | char(9) | YES || NULL ||
| c_phone | char(16) | YES || NULL ||
| c_since | timestamp | NO || CURRENT_TIMESTAMP | ON UPDATE CURRENT_TIMESTAMP |
| c_middle | char(2) | YES || NULL ||
| c_data | varchar(500) | YES || NULL ||
+-----+-----+-----+-----+-----+
+-----+
21 rows in set (0.004 sec)
```

查看目标数据库 MySQL 的表 bmsql\_customer 的建表 SQL:

```
MySQL [test]> show create table bmsql_customer \G
***** 1. row *****
Table: bmsql_customer
Create Table: CREATE TABLE `bmsql_customer` (
  `c_w_id` bigint(20) NOT NULL,
  `c_d_id` bigint(20) NOT NULL,
  `c_id` bigint(20) NOT NULL,
  `c_discount` decimal(4,4) DEFAULT NULL,
  `c_credit` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_last` varchar(16) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_first` varchar(16) COLLATE utf8_unicode_ci DEFAULT NULL,
  `c_credit_lim` decimal(12,2) DEFAULT NULL,
  `c_balance` decimal(12,2) DEFAULT NULL,
  `c_ytd_payment` decimal(12,2) DEFAULT NULL,
  `c_payment_cnt` bigint(20) DEFAULT NULL,
  `c_delivery_cnt` bigint(20) DEFAULT NULL,
```

```
`c_street_1` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,  
`c_street_2` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,  
`c_city` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,  
`c_state` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,  
`c_zip` char(9) COLLATE utf8_unicode_ci DEFAULT NULL,  
`c_phone` char(16) COLLATE utf8_unicode_ci DEFAULT NULL,  
`c_since` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE  
CURRENT_TIMESTAMP,  
`c_middle` char(2) COLLATE utf8_unicode_ci DEFAULT NULL,  
`c_data` varchar(500) COLLATE utf8_unicode_ci DEFAULT NULL,  
PRIMARY KEY (`c_w_id`,`c_d_id`,`c_id`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci  
1 row in set (0.01 sec)
```

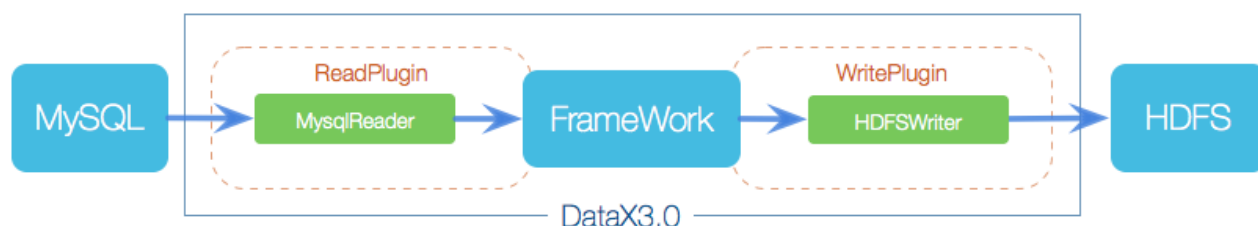
经对比是一致的。

## 13 使用 Datax 迁移 OceanBase 表数据到 MySQL 数据库

DataX 是阿里云 DataWorks 数据集成的开源版本，是阿里巴巴集团内被广泛使用的离线数据同步工具/平台。DataX 实现了包括 MySQL、Oracle、SQLserver、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、Hologres、DRDS、OceanBase 等各种异构数据源之间高效的数据同步功能。

OceanBase 数据库社区版客户，可以在 [DataX 开源网站](#) 内下载源码，自行编译。编译时，可根据需要剔除在 `pom.xml` 中不用的数据库插件，否则编译出来的包会非常大。

### 13.1 框架设计



DataX 作为离线数据同步框架，采用 "Framework + Plugin" 模式构建。将数据源读取和写入抽象为 Reader/Writer 插件，纳入到整个同步框架中。

- Reader 作为数据采集模块，负责采集数据源的数据，将数据发送给 Framework。
- Writer 作为数据写入模块，负责不断向 Framework 获取数据，并将数据写入到目的端。
- Framework 用于连接 Reader 和 Writer，作为两者的数据传输通道，并处理缓冲、流控、并发、数据转换等核心技术问题。

DataX 以任务的形式迁移数据。每个任务只处理一个表，每个任务有一个 `json` 格式的配置文件，配置文件里包含 `reader` 和 `writer` 两部分。`reader` 和 `writer` 分别对 DataX 支持的数据库读写插件，例如，将 OceanBase 表数据迁移到 MySQL 数据库操作时，需要从 OceanBase 数据库读取数据写入 MySQL 数据库，因此使用的插件为 OceanBase 数据库的 `oceanbasev10reader` 插件和 MySQL 数据库的 `mysqlwriter` 插件来搭配完成。这里介绍下 `oceanbasev10reader` 和 `mysqlwriter` 插件。

#### 13.1.1 oceanbasev10reader 插件



实现原理方面，`oceanbasev10reader` 通过 DataX 框架获取 Reader 生成的协议数据，生成 insert 语句。在写入数据时，如果出现主键或唯一键冲突，OceanBase 数据库的 MySQL 租户可以通过 `replace` 模式来更新表中的所有字段；OceanBase 数据库的 Oracle 租户当前只能使用 Insert 方式。

`oceanbasev10reader` 插件实现了从 OceanBase 数据库读取数据的功能。在底层实现上，`oceanbasev10reader` 插件通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并执行相应的 SQL 语句将数据从 OceanBase 数据库中 SELECT 出来。

实现原理方面，简而言之，OceanBase 数据库通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并根据用户配置的信息生成查询语句，然后发送到远程 OceanBase 数据库，远程的 OceanBase 数据库将该 SQL 执行的返回结果使用 DataX 自定义的数据类型拼装为抽象的数据集，再传递给下游 Writer 处理。

## 13.1.2 mysqlwriter 插件

`mysqlwriter` 插件实现了将数据写入到 MySQL 主库的目标表中的功能。在底层实现上，`mysqlwriter` 插件通过 JDBC 连接远程 MySQL 数据库，并执行相应的 `insert into ...` 或者 `replace into ...` 的 sql 语句将数据写入 MySQL 数据库，MySQL 数据库内部会分批次提交入库。在使用 `mysqlwriter` 插件迁移数据时，需要 MySQL 数据库使用 innodb 引擎。

实现原理方面，`mysqlwriter` 插件通过 DataX 框架获取 Reader 生成的协议数据，根据用户配置的 writeMode 生成 `insert into ...` 或者 `replace into ...` 的 sql 语句写数据到 MySQL 数据库主库。

详细功能和参数说明请参考官方说明：[mysqlwriter 插件](#)。

## 13.1.3 DataX 配置文件

配置文件示例：

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "streamreader",
```

```
"parameter": {
  "sliceRecordCount": 10,
  "column": [
    {
      "type": "long",
      "value": "10"
    },
    {
      "type": "string",
      "value": "hello, 你好, 世界-DataX"
    }
  ]
},
"writer": {
  "name": "streamwriter",
  "parameter": {
    "encoding": "UTF-8",
    "print": true
  }
},
"setting": {
  "speed": {
    "channel": 2
  }
}
```

```
}  
}
```

### 13.1.3.1 注意

datax 仅迁移表数据，需要提前在目标端创建好对应的表对象结构。

将 `json` 配置文件放到 DataX 的目录 `job` 下，或者自定义路径。执行方法如下：

```
$bin/datax.py job/stream2stream.json
```

输出信息：

```
<.....>
```

```
2021-08-26 11:06:09.217 [job-0] INFO JobContainer - PerfTrace not enable!
```

```
2021-08-26 11:06:09.218 [job-0] INFO StandAloneJobContainerCommunicator - Total  
20 records, 380 bytes | Speed 38B/s, 2 records/s | Error 0 records, 0 bytes | All Task  
WaitWriterTime 0.000s | All Task WaitReaderTime 0.000s | Percentage 100.00%
```

```
2021-08-26 11:06:09.223 [job-0] INFO JobContainer -
```

任务启动时刻：2021-08-26 11:05:59

任务结束时刻：2021-08-26 11:06:09

任务总计耗时：10s

任务平均流量：38B/s

记录写入速度：2rec/s

读出记录总数：20

读写失败总数：0

DataX 任务执行结束会有个简单的任务报告，包含上述输出的平均流量、写入速度和读写失败总数等。

DataX 的 `job` 的参数 `settings` 可以指定速度参数和错误记录容忍度等。

```
"setting": {  
  "speed": {  
    "channel": 10
```

```
},  
"errorLimit": {  
  "record": 10,  
  "percentage": 0.1  
}  
}
```

参数说明：

- `errorLimit` 表示报错记录数的容忍度，超出这个限制后任务就中断退出。
- `channel` 是并发数，理论上并发越大，迁移性能越好。但实际操作中也要考虑源端的读压力、网络传输性能以及目标端写入性能。

## 13.2 环境准备

下载 tar 包地址：<http://datax-opensource.oss-cn-hangzhou.aliyuncs.com/datax.tar.gz>

解压安装文件：

```
tar zxvf datax.tar.gz  
cd datax
```

目录如下：

```
$tree -L 1 --filelimit 30  
.  
├── bin  
├── conf  
├── job  
├── lib  
├── log  
├── log_perf  
└── plugin
```

```
|—— script
|—— tmp
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。该目录下的 <code>datax.py</code> 为 DataX 任务的启动脚本
conf	日志文件配置目录。该目录下存放 <code>datax</code> 与任务无关的配置文件
lib	运行时期依赖的包。该目录存放 DataX 运行所需要的全局 jar 文件
job	该目录下有一个用于测试验证 <code>datax</code> 安装的任务配置文件
log	日志文件目录。该目录下存放 <code>datax</code> 任务运行的日志； <code>datax</code> 运行时，默认会将日志输出到标准输出，同时写入到 <code>log</code> 目录下
plugin	插件文件目录。该目录下保存 DataX 支持的各种数据源插件

## 13.3 使用 DataX 迁移 OceanBase 数据到 MySQL 数据库示例

将 OceanBase 数据迁移到 MySQL，如果源端和目标端不能同时跟 DataX 服务器网络联通，那么可以通过 CSV 文件中转。如果源端数据库和目标端数据库能同时跟 DataX 所在服务器联通，则可以使用 DataX 直接将数据从源端迁移到目标端。

示例：从 OceanBase 迁移 `test.t1` 表数据到 MySQL 模式下的 `test.t1`。

`myjob.json` 配置文件如下：

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 4
      },
      "errorLimit": {
```

```
"record": 0,
"percentage": 0.1
},
"content": [
{
"reader": {
"name": "oceanbasev10reader",
"parameter": {
"username": "*****",
"password": "*****",
"column": ["*"],
"connection": [
{
"table": ["t1"],
"jdbcUrl": ["jdbc:oceanbase://172.30.xxx.xxx:2883/test"]
}
]
}
},
"writer": {
"name": "mysqlwriter",
"parameter": {
"obWriteMode": "insert",
"column": ["*"],
"preSql": ["truncate table t1"],
"connection": [
{
"jdbcUrl": "jdbc:mysql://100.88.xxx.xxx:3308/test",
```

```
"table": ["t1"]
}
],
"username": "*****",
"password": "*****",
"writerThreadCount": 10,
"batchSize": 1000,
"memstoreThreshold": "0.9"
}
}
}
]
```

#### 参数说明

参数	描述
name	描述的是连接数据库的 reader 或 writer 对应的数据库插件的名称。其中 MySQL 的 reader 插件为：mysqlreader，OceanBase 的 writer 插件为 oceanbasev10writer。具体 reader 和 writer 的插件可以参考 datax 的文档： <a href="#">DataX 数据源指南</a> 。

jdbcUrl	<p>描述的是到连接的数据库的 JDBC 信息，使用 JSON 的数组描述，并支持一个库填写多个连接地址。您在 JSON 数组中填写一个 JDBC 连接即可。jdbcUrl 按照 MySQL 官方规范，并可以填写连接附件控制信息。具体请参见 <a href="#">MySQL 官方文档</a>。</p> <h3>13.3.3.1 注意</h3> <ul style="list-style-type: none"><li>• jdbcUrl 必须包含在 connection 配置单元中。</li><li>• OceanBase 数据库需要通过 obproxy 进行连接，端口默认 2883。</li><li>• writer 端的 jdbcUrl，无需在连接串两端加 []，reader 端的 jdbcUrl，必需在连接串两端加 []</li><li>• 必选：是</li><li>• 默认值：无</li></ul>
username	<p>数据源的用户名</p> <p>必选：是</p> <p>默认值：无</p>
password	<p>数据源指定用户名的密码</p> <p>必选：是</p> <p>默认值：无</p>
table	<p>所选取的需要同步的表。使用 JSON 的数组描述，因此支持多张表同时抽取。当配置为多张表时，用户自己需确保多张表是同一 schema 结构，MySQLReader 不予检查表是否同一逻辑表。</p> <h3>13.3.3.2 注意</h3> <p>table 必须包含在 connection 配置单元中。</p> <ul style="list-style-type: none"><li>• 必选：是</li><li>• 默认值：无</li></ul>



<p>column</p>	<p>所配置的表中需要同步的列名集合，使用 JSON 的数组描述字段信息。columns 不建议配置为 <code>['*']</code>，因为表结构发生变化时该配置也会发生变化。建议的配置方式是指定具体的列名。支持列裁剪，即列可以挑选部分列进行导出。支持列换序，即列可以不按照表 schema 信息进行导出。支持常量配置，用户需要按照 MySQL SQL 语法格式：<code>["id", "`table`", "1", "bazhen.csy", "null", "to_char(a + 1)", "2.3", "true"]</code>。</p> <h3>13.3.3.3 说明</h3> <ul style="list-style-type: none"><li>• <code>id</code> 为普通列名。</li><li>• <code>table</code> 为包含保留字的列名；</li><li>• <code>1</code> 为整形数字常量。</li><li>• <code>bazhen.csy</code> 为字符串常量。</li><li>• <code>null</code> 为空指针。</li><li>• <code>to_char(a + 1)</code> 为表达式。</li><li>• <code>2.3</code> 为浮点数。</li><li>• <code>true</code> 为布尔值。</li><li>• 必选：是</li><li>• 默认值：无</li></ul>
---------------	--

where

筛选条件，MySQLReader 根据指定的column、table、where 条件拼接 SQL，并根据这个 SQL 进行数据抽取。在实际业务场景中，往往会选择当天的数据进行同步，可以将where 条件指定为 `gmt_create > $bizdate`。

### 13.3.3.4 注意

不可以将 where 条件指定为 `limit 10`，`limit` 不是 SQL 的合法 where 子句。where 条件可以有序地进行业务增量同步。如果不填写 where 语句，包括不提供 where 的 key 或者 value，DataX 均视作同步全量数据。

- 必选：否
- 默认值：无

配置 job 文件后，执行该 job。命令如下：

```
python datax.py ../job/myjob.json
```

## 13.4 更多信息

关于 DataX 的开源代码和更多信息，请参见 [DataX](#)。

# 14 使用 Canal 从 OceanBase 数据库同步数据到 MySQL 数据库

本文档主要介绍使用 Canal 和 oblogproxy 组件从 OceanBase 数据库同步数据至 MySQL 数据库。

## 14.1 OceanBase CDC 实现逻辑

oblogproxy 是 OceanBase 数据库的增量日志代理服务。基于 libobcdc (原名: liboblog), 以服务的形式提供实时增量链路接入和管理能力, 方便应用接入 OceanBase 增量日志。能够解决在网络隔离的情况下, 订阅增量日志的需求, 并提供多种链路接入方式。

Canal 通过解析 MySQL 的 Binlog 日志, 将数据库的增量数据抽取出来, 并将其发送到订阅者进行消费和处理。这使得用户可以方便地实现数据的实时同步、实时分析等需求。

数据链路:

```
ob_cluster -> oblogreader -> oblogmsg -> canal_server -> canal_client -> mysql
```

组件介绍:

- libobcdc 是 OceanBase CDC 的基本组件, libobcdc 衍生出 oblogproxy, libobcdc 依赖 oblogmsg。libobcdc 是 C++ 动态库, 将从 OceanBase 集群中拉到的增量日志按事务提交顺序向外透出, 透出的方式是按 LogMessage 协议创建相关对象给使用方使用。
- ObLogReader 对 libobcdc 的 C++ 封装, 在 libobcdc 的基础上, 对创建出的 LogMessage 对象适配多种序列化输出。
- logproxy 增量日志代理, 并不对增量数据进行转发, 而只是对新建连接请求创建对应的 oblogreader 子进程, 并由该 oblogreader 子进程通过该连接直接向客户端发送按 LogMessage 序列化后的增量日志记录。LogProxy 用于对同机器上所有 ObLogReader 子进程的生命周期进行管理。
- logclient 对字节流按 LogMessage 协议反序列化后生成 LogMessage 对象给使用方调用, 比如和 canal 对接。

## 14.2 操作步骤

### 14.2.1 步骤一: 安装 oblogproxy

请参考 [oblogproxy 安装](#) 完成 oblogproxy 的安装部署。

### 14.2.2 步骤二: 安装 canal server

1. 下载软件包。

下载 [canal-for-ob.deployer.tar.gz](https://github.com/oceanbase/canal/releases/download/canal-for-ob-1.1.6-alpha/canal-for-ob.deployer.tar.gz)。

```
wget https://github.com/oceanbase/canal/releases/download/canal-for-ob-1.1.6-alpha/canal-for-ob.deployer.tar.gz
```

2. 将压缩包解压至目录 `/Canal_Home/canal-for-ob`。

```
mkdir /Canal_Home/canal-for-ob && tar zxvf canal-for-ob.deployer.tar.gz -C /Canal_Home/canal-for-ob
```

3. 修改配置文件 `canal.properties`。

修改 `/Canal_Home/canal-for-ob/conf/canal.properties`。

```
canal.serverMode = tcp
canal.destinations = example
canal.instance.global.spring.xml = classpath:spring/ob-default-instance.xml
```

4. 配置 canal instance。

将 `/Canal_Home/canal-for-ob/conf/example` 下的 `instance.properties` 删除，然后将 `ob-instance.properties` 重命名为 `instance.properties`。

修改 `instance.properties`，示例如下：

```
cd /Canal_Home/canal-for-ob/conf/example
vi instance.properties

# OceanBase集群参数
canal.instance.oceanbase.rsList=10.10.10.1:2882:2881;10.10.10.2:2882:2881;
10.10.10.3:2882:2881
canal.instance.oceanbase.username=root@mysql001#test4000
canal.instance.oceanbase.password=*****
canal.instance.oceanbase.startTimestamp=0

# oceanbase logproxy参数
canal.instance.oceanbase.logproxy.address=10.10.10.1:2983
```

```
canal.instance.oceanbase.logproxy.sslEnabled=false
canal.instance.oceanbase.logproxy.serverCert=../conf/${canal.instance.
destination:}/ca.crt
canal.instance.oceanbase.logproxy.clientCert=../conf/${canal.instance.
destination:}/client.crt
canal.instance.oceanbase.logproxy.clientKey=../conf/${canal.instance.
destination:}/client.key

# 是否要在库名中去掉租户前缀。logproxy 输出的日志中库名默认为 [tenant].[db]
canal.instance.oceanbase.tenant=mysql001
canal.instance.parser.excludeTenantInDbName=true

# 日志过滤。格式为 [tenant].[database].[table]，支持正则
canal.instance.filter.regex=mysql001.*.*
```

#### 5. 启动 Canal Server。

```
cd /Canal_Home/canal-for-ob && sh bin/startup.sh
```

## 14.2.3 步骤三：配置 RDB 适配器

#### 1. 下载软件包。

下载 [canal-for-ob.adapter.tar.gz](https://github.com/oceanbase/canal/releases/download/canal-for-ob-1.1.6-alpha/canal-for-ob.adapter.tar.gz)。

```
wget https://github.com/oceanbase/canal/releases/download/canal-for-ob-
1.1.6-alpha/canal-for-ob.adapter.tar.gz
```

#### 2. 将压缩包解压至目录 /Canal\_Home/canal-adapter-for-ob 。

```
mkdir /Canal_Home/canal-adapter-for-ob && tar zxvf canal-for-ob.adapter.tar.
gz -C /Canal_Home/canal-adapter-for-ob
```

#### 3. 修改 application.yml 。

```
cd /Canal_Home/canal-adapter-for-ob/conf
vi application.yml

canalAdapters:
- instance: example # canal instance Name or mq topic name
groups:
- groupId: g1
outerAdapters:
- name: logger
- name: rdb
key: mysql1
properties:
jdbc.driverClassName: com.mysql.jdbc.Driver
jdbc.url: jdbc:mysql://10.10.10.1:3306/test_data?useUnicode=false
jdbc.username: root
jdbc.password: *****
```

#### 4. 修改适配器库/表映射（以库映射为例）。

```
cd /Canal_Home/canal-adapter-for-ob/conf/rdb
vi application.yml

## Mirror schema synchronize config
dataSourceKey: defaultDS
destination: example
groupId: g1
outerAdapterKey: mysql1
concurrent: true
dbMapping:
```

```
mirrorDb: true  
database: test_data
```

5. 启动 canal client。

```
cd /Canal_Home/canal-adapter-for-ob && sh bin/startup.sh
```

6. 查看数据同步情况。

在 OceanBase 源端写入数据，在 MySQL 目标端查看数据同步。

## 14.3 功能限制

- 同步的表必须有主键。否则，源端删除无主键表的任意一笔记录，同步到目标端会导致整个表被删除。
- DDL 支持新建表、新增列。

# 15 使用 CloudCanal 从 OceanBase 数据库迁移数据到 MySQL 数据库

CloudCanal 是一款数据迁移同步工具，帮助企业快速构建高质量数据流通通道，产品包含 SaaS 模式和私有输出专享模式。开发团队核心成员来自大厂，具备数据库内核、大规模分布式系统、云产品构建背景，懂数据库，懂分布式，懂云产品商业和服务模式。

本文将介绍如何使用 CloudCanal 社区版 v2.2.6.9 将 OceanBase 数据库 MySQL 模式中的数据迁移同步到对端 MySQL 数据库中。

## 15.0.0.1 功能适用性

- CloudCanal 社区版从 2.2.3.0 版本开始支持从 OceanBase 数据库 MySQL 模式迁移数据至 MySQL 数据库。详情请参见 [2.2.3.0](#)。
- CloudCanal 暂时只支持 OceanBase 数据库 V3.2.3.0 之前的版本作为源库。

## 15.1 前置条件

1. 参考 [全新安装\(Linux/MacOS\)](#) 完成 CloudCanal 社区版的安装部署。
2. 参考 [oblogproxy 安装](#) 完成 oblogproxy 的安装部署。

## 15.2 操作步骤

1. 添加数据源。
2. 创建任务。
3. 查看任务。

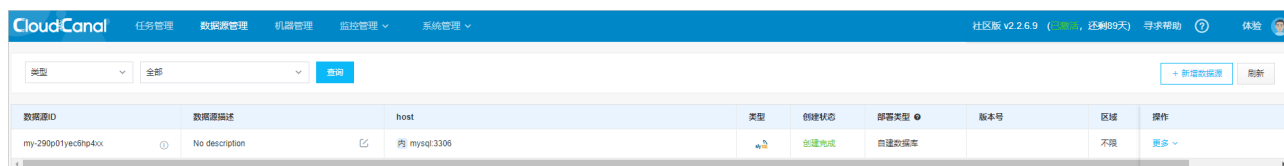
### 15.2.1 添加数据源

1. 登录 CloudCanal 平台。





## 2. 进入数据源管理界面，点击新增数据源。



## 3. 在新增数据源页面，填写数据源信息。

- 部署类型：有 **自建数据库** 和 **阿里云** 两种选项。
  - 阿里云：用户在阿里云上购买的数据库实例。
  - 自建数据库：用户自己部署的数据库实例。
- 数据库类型：选择数据源类型。

新增两个数据源 OceanBase 和 MySQL，分别作为同步的源库和目标库：

- 选择自建数据库中 OceanBase，添加自己部署的 OceanBase 数据库实例。

OceanBase 数据源设置：

- 网络地址：填写连接 OceanBase 数据库的 IP，直连或通过 ODP 连接。
- oblogproxy host: oblogproxy 的 IP 地址。OceanBase 数据库作为源库增量同步时，不可以为空；OceanBase 数据库作为目标库时，可为空。有关 oblogproxy 的详细信息，请参考 [oblogproxy 介绍](#)。
- OceanBaseRpcPort: OceanBase Rpc 端口，默认 2882。
- 认证方式：分别为 **账号密码**、**有账号密码** 和 **无账号密码**。默认账号密码。
- 账号：连接 OceanBase 数据库的用户名。直连格式：**用户名@租户名称**；ODP 连接格式：**用户名@租户名称#集群名称**。
- 密码：连接 OceanBase 数据库的用户名对应的密码。

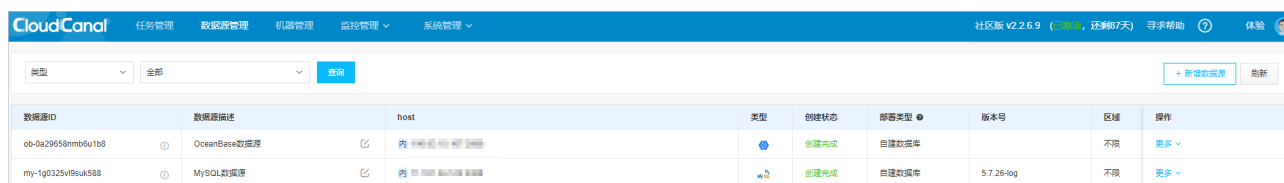
- **描述**：选填项，备注便于记忆的名字，方便使用时识别，如交易库、用户库、测试库等。

- 选择自建数据库中 MySQL，添加自己部署的 MySQL 数据库实例。

MySQL 数据源设置：

- **网络地址**：填写 MySQL 数据库的 IP。
- **认证方式**：有 **账号密码**、**有账号密码** 和 **无账号密码** 三种方式。默认 **账号密码** 方式。
- **账号**：连接 MySQL 数据库的用户名。
- **密码**：连接 MySQL 数据库的用户名对应的密码。
- **描述**：选填项，备注便于记忆的名字，方便使用时识别，如交易库、用户库、测试库等。

#### 4. 查看新增的两个数据源。



数据源ID	数据源描述	host	类型	创建状态	部署类型	版本号	区域	操作
ob-9a29658nmb6u1b8	OceanBase数据源			创建完成	自建数据库		不限	更多
my-1g0325v8uak588	MySQL数据源			创建完成	自建数据库	5.7.26-log	不限	更多

## 15.2.2 创建任务

添加好数据源之后可以按照如下步骤进行数据全量迁移、增量同步和结构迁移。

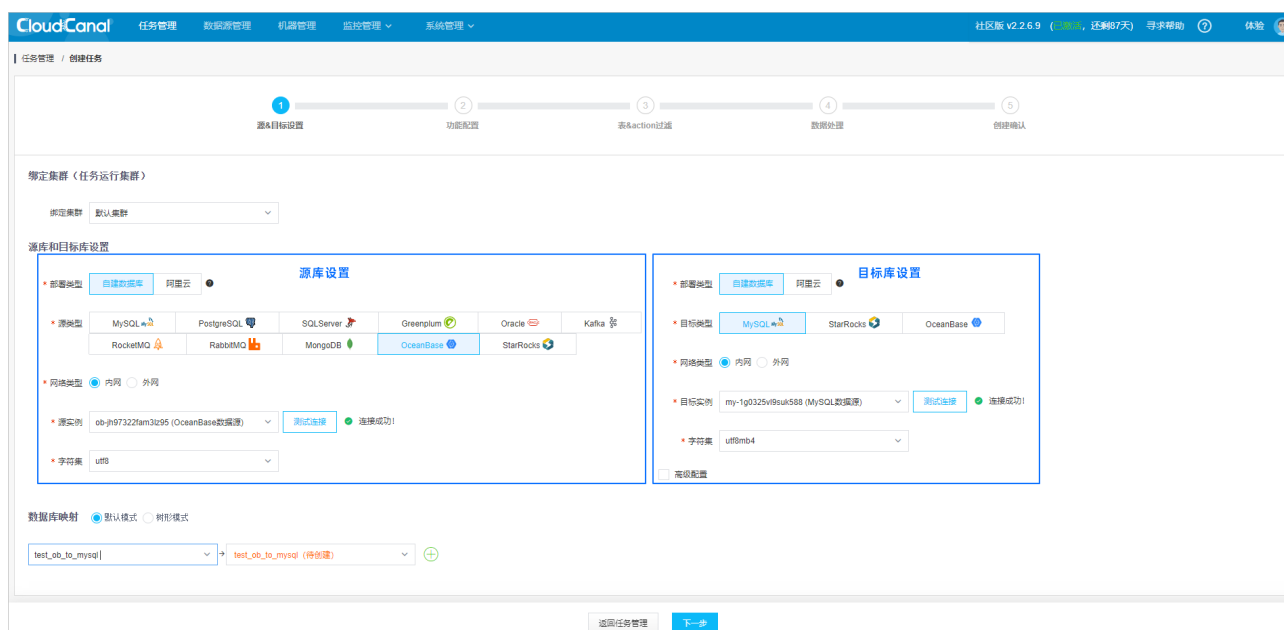
#### 1. 任务管理 -> 创建任务。



任务ID/描述	类型	创建时间	创建人	状态	进度	操作
您还没有相关任务						

#### 2. 源库和目标库设置。

- 选择 **任务运行集群**，任务会被调度到绑定集群的一台机器上执行。社区版部署完成后，会有一个默认的运行集群。
- 选择源库 OceanBase 和目标数据库 MySQL，并点击 **测试连接**。
- 选择需要 **迁移同步或校验的数据库**，指定数据库映射关系。
- 完成设置后点击 **下一步**。



绑定集群 (任务运行集群)

绑定集群: 默认集群

源库和目标库设置

部署类型: 自建数据库 | 阿里云

源库设置

- 源类型: MySQL, PostgreSQL, SQL Server, Greenplum, Oracle, Kafka, RocketMQ, RabbitMQ, MongoDB, OceanBase, StarRocks
- 网络类型: 内网 (selected), 外网
- 源实例: ob-j97322f4m3z95 (OceanBase数据源) | 测试连接 | 连接成功!
- 字符集: utf8

目标库设置

- 部署类型: 自建数据库 | 阿里云
- 目标类型: MySQL, StarRocks, OceanBase
- 网络类型: 内网 (selected), 外网
- 目标实例: my-1g0325v8uak588 (MySQL数据源) | 测试连接 | 连接成功!
- 字符集: utf8mb4
- 高级配置

数据库映射: 默认模式 | 树形模式

test\_ob\_to\_mysql | test\_ob\_to\_mysql (待创建) | +

返回任务管理 | 下一步

#### 3. 功能配置。

选择 **增量同步** 功能，第一次会先查表进行全量同步，之后消费 binlog 增量同步数据。

- 任务类型具有以下功能：
  - 全量迁移：以数据迁移为主，适合数据的全量搬迁及短期的增量同步任务。

- 增量同步：默认选项，默认附带 **全量初始化**。以数据同步为主，适合长期的增量同步任务。
  - 数据校验：对比源端与目标端的数据，一次性或定时多次校验数据迁移的准确性。社区版不支持此功能。
  - 结构迁移：根据所选数据库、表自动创建对应的数据库、表。
  - 数据订正：对比源端与目标端的数据，将不一致的数据自动覆盖成和源端一致。社区版不支持此功能。
- 任务规格：默认 **平衡型**、**2G** 规格即可。
  - 完成配置后，点击 **下一步**。

CloudCanal 任务管理 数据源管理 机器管理 监控管理 系统管理 社区版 v2.2.6.9 (100%, 还剩67天) 寻求帮助 体验

任务管理 / 创建任务

源&目标设置 功能配置 表&action过滤 数据处理 创建确认

任务类型 全量迁移 增量同步 数据校验 结构迁移 数据订正

全量初始化 ☒

任务规格 增量增强型 全量增强型 平衡型

全量内存	增量内存	校验内存	描述
<input type="radio"/> 1G	1G	1G	1 KB/record, record count < 20 m, incre tps < 0.5k
<input checked="" type="radio"/> 2G	2G	2G	1 KB/record, record count < 200 m, incre tps < 1k
<input type="radio"/> 3G	3G	3G	1 KB/record, record count < 500 m, incre tps < 2k
<input type="radio"/> 4G	4G	4G	1 KB/record, record count < 2 b, incre tps < 10k
<input type="radio"/> 0.5G	0.5G	0.5G	1 KB/record, record count < 10 m, incre tps < 0.2k

已选择的规格 全量内存: 2G, 增量内存: 2G, 校验内存: 2G, 描述: 1 KB/record, record count < 200 m, incre tps < 1k

任务描述

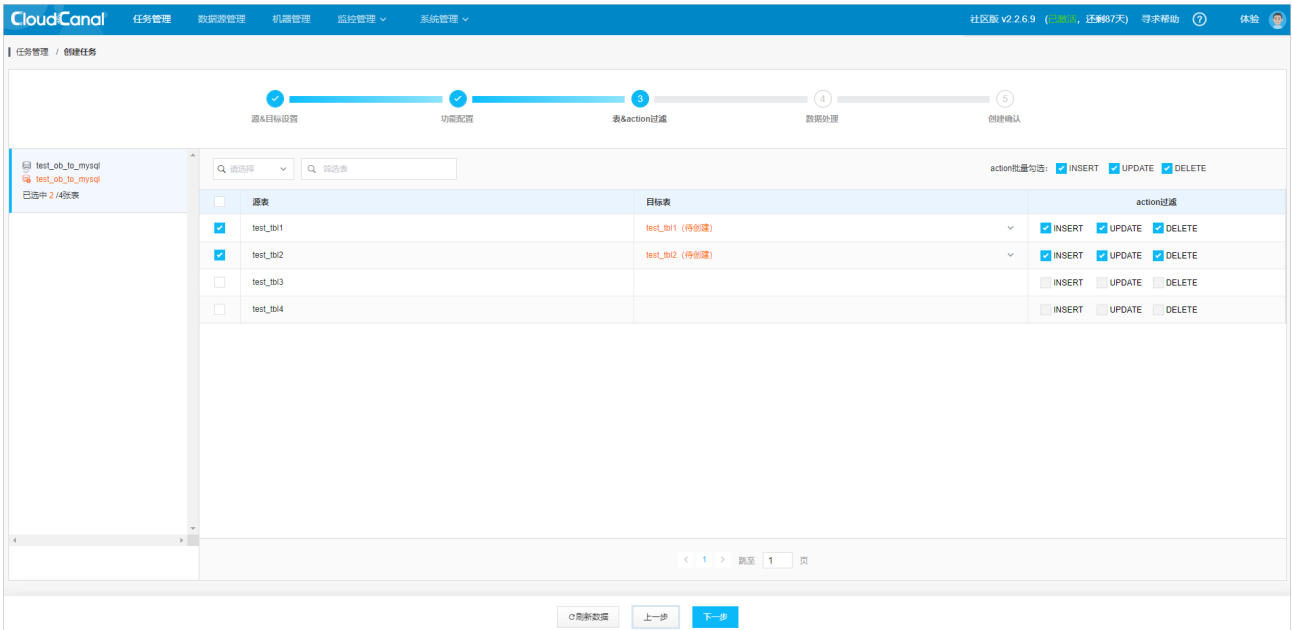
是否同步 DDL ☒ 全部同步 ☐ 不同步

自动启动任务 ☒

上一步 下一步

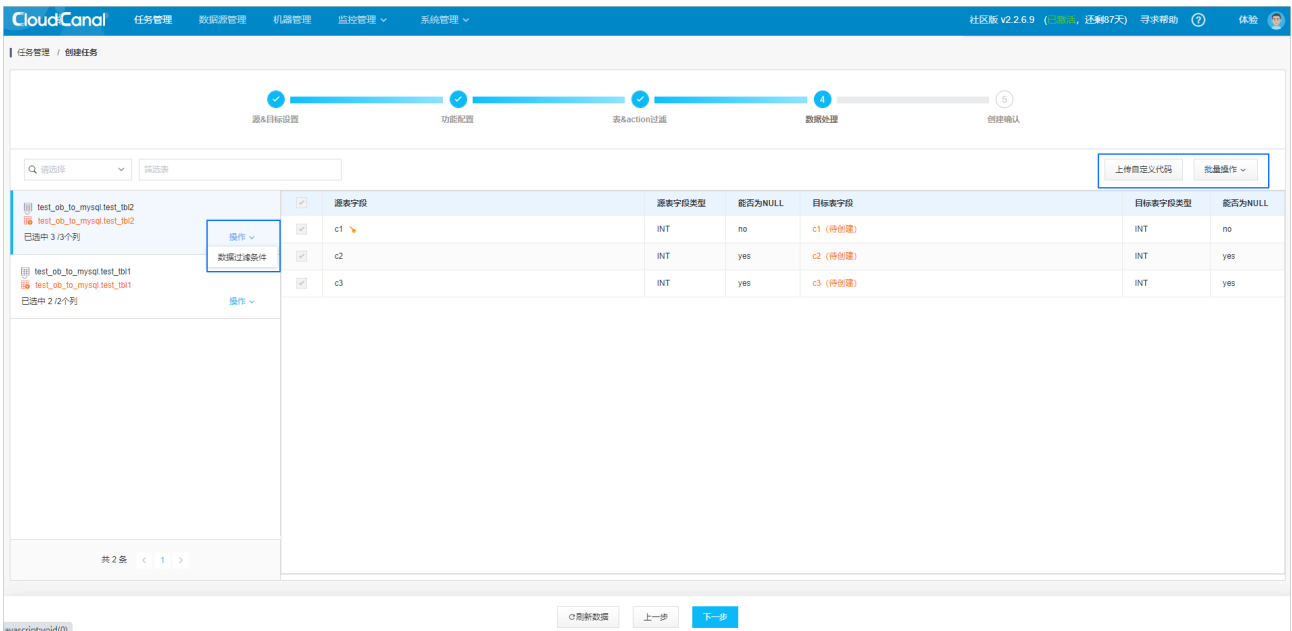
#### 4. 表&action 过滤。

- 选择要同步的表，要保证目标库的 **UPDATE** 和 **DELETE** 操作和源库的一致，需要保证源库表中有主键或者唯一约束。
- 完成配置后，点击 **下一步**。



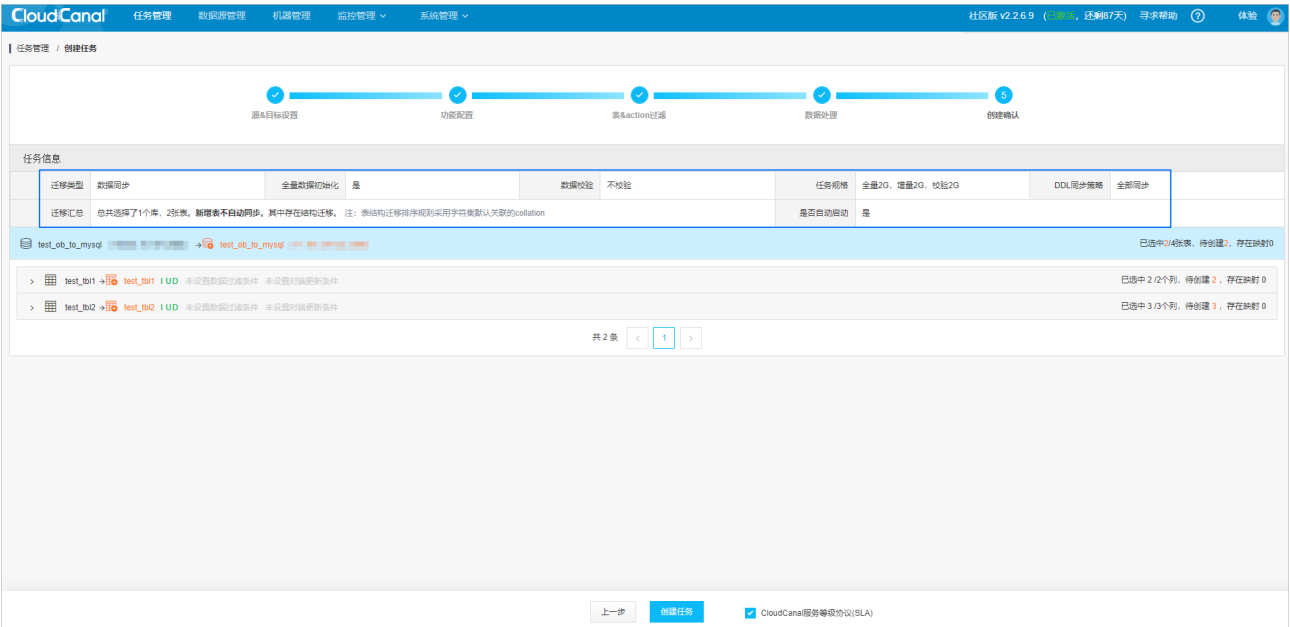
5. 数据处理。

- 可以进行添加 **数据过滤条件**、**上传自定义代码** 和 **批量操作**。
  - 数据过滤条件：在数据处理页面左侧对应表的 **操作** 中添加 **数据过滤条件**。
  - 上传自定义代码：自定义代码实时加工允许用户使用 Java 语言编写自定义的数据行处理逻辑，上传 CloudCanal 平台后，数据同步任务执行全量、增量时会自动应用用户的自定义处理逻辑，然后再下入对端数据源。
  - 批量操作：有 **批量添加数据过滤条件** 和 **批量裁剪列** 选项。
- 完成设置后，点击 **下一步**。



6. 确认创建任务。

最后一步，确认创建内容无误后点击 **创建任务**。



### 15.2.3 查看任务状态

增量同步任务创建成功后，会默认进行 **结构迁移**、**全量迁移**、**增量同步**。  
回到 CloudCanal 任务管理控制台，单击右上角 **刷新** 并查看任务实时状态。



### 15.3 相关文档

有关 CloudCanal 的详细信息，请参见 [CloudCanal 官方文档](#)。

## 16 使用 Flink CDC 从 OceanBase 数据库迁移数据到 MySQL 数据库

Flink CDC (CDC Connectors for Apache Flink) 是 Apache Flink 的一组 Source 连接器，它支持从大多数数据库中实时地读取存量历史数据和增量变更数据。Flink CDC 能够将数据库的全量和增量数据同步到消息队列和数据仓库中。Flink CDC 也可以用于实时数据集成，您可以使用它将数据库数据实时导入数据湖或者数据仓库。同时，Flink CDC 还支持数据加工，您可以通过它的 SQL Client 对数据库数据做实时关联、打宽、聚合，并将结果写入到各种存储中。CDC（Change Data Capture，即变更数据捕获）能够帮助您监测并捕获数据库的变动。CDC 提供的数据可以做很多事情，比如：做历史库、做近实时缓存、提供给消息队列（MQ），用户消费 MQ 做分析和审计等。

本文将介绍使用 Flink CDC 从 OceanBase 数据库迁移数据到 MySQL 数据库。

### 16.1 环境准备

#### 16.1.1 配置 OceanBase 数据库 oblogproxy 服务

oblogproxy 是 OceanBase 数据库的增量日志代理服务。oblogproxy 支持实时增量链路接入和管理，方便应用接入 OceanBase 数据库的增量日志。同时支持在网络隔离时订阅增量日志。

1. 在 OceanBase 集群 sys 租户中，为 oblogproxy 创建一个带密码的用户。

示例如下：

```
obclient [(none)]> SHOW TENANT;

+-----+
| Current_tenant_name |
+-----+
| sys |
+-----+

1 row in set

obclient [(none)]> CREATE USER sys_user001 IDENTIFIED BY '*****';
Query OK, 0 rows affected
```

```
obclient [(none)]> GRANT ALL PRIVILEGES ON *.* TO sys_user001 WITH GRANT
OPTION;
Query OK, 0 rows affected
```

2. 参考 [oblogproxy 安装](#) 完成 oblogproxy 的安装部署。

## 16.1.2 Flink 环境设置

下载 Flink 和所需要的依赖包：

1. 通过 [下载地址](#) 下载 Flink。本文档使用的是 Flink 1.15.3，并将其解压至目录 `flink-1.15.3`。
2. 下载下面列出的依赖包，并将它们放到目录 `flink-1.15.3/lib/` 下。
  - [flink-sql-connector-oceanbase-cdc-2.2.0.jar](#)
  - [flink-connector-jdbc-1.15.3.jar](#)
  - [mysql-connector-java-5.1.47.jar](#)

## 16.2 准备数据

### 16.2.3 准备 OceanBase 数据库数据

在 OceanBase 数据库中准备测试数据，作为导入 MySQL 数据库的源数据。

1. 登录 OceanBase 数据库。

使用 `root` 用户登录集群的 `mysql001` 租户。

```
[xxx@xxx /home/admin]
$obclient -h10.10.10.2 -P2881 -uroot@mysql001 -p***** -A
Welcome to the OceanBase. Commands end with ; or \g.
Your OceanBase connection id is 3221536981
Server version: OceanBase 4.0.0.0 (r100000302022111120-
7cef93737c5cd03331b5f29130c6e80ac950d33b) (Built Nov 11 2022 20:38:33)

Copyright (c) 2000, 2018, OceanBase and/or its affiliates. All rights reserved.
```



Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
obclient [(none)]>
```

2. 创建数据库 `test_ob_to_mysql`，表 `tbl1` 和 `tbl2`，并插入数据。

```
obclient [(none)]> CREATE DATABASE test_ob_to_mysql;
```

Query OK, 1 row affected

```
obclient [(none)]> USE test_ob_to_mysql;
```

Database changed

```
obclient [test_ob_to_mysql]> CREATE TABLE tbl1(col1 INT PRIMARY KEY, col2  
VARCHAR(20),col3 INT);
```

Query OK, 0 rows affected

```
obclient [test_ob_to_mysql]> INSERT INTO tbl1 VALUES(1,'China',86),(2,'Taiwan',  
886),(3,'Hong Kong',852),(4,'Macao',853),(5,'North Korea',850);
```

Query OK, 5 rows affected

Records: 5 Duplicates: 0 Warnings: 0

```
obclient [test_ob_to_mysql]> CREATE TABLE tbl2(col1 INT PRIMARY KEY,col2  
VARCHAR(20));
```

Query OK, 0 rows affected

```
obclient [test_ob_to_mysql]> INSERT INTO tbl2 VALUES(86,'+86'),(886,'+886'),  
(852,'+852'),(853,'+853'),(850,'+850');
```

Query OK, 5 rows affected

Records: 5 Duplicates: 0 Warnings: 0

## 16.2.4 准备 MySQL 数据库数据

在 MySQL 数据库中创建存放源数据的表。

1. 进入 MySQL 数据库。

```
[xxx@xxx /home/admin]
$mysql -hxxx.xxx.xxx.xxx -P3306 -uroot -p*****
<Omit echo information>

MySQL [(none)]>
```

2. 创建数据库 `test_ob_to_mysql` 和表 `ob_tbl1_and_tbl2`。

```
MySQL [(none)]> CREATE DATABASE test_ob_to_mysql;
Query OK, 1 row affected

MySQL [(none)]> USE test_ob_to_mysql;
Database changed

MySQL [test_ob_to_mysql]> CREATE TABLE ob_tbl1_and_tbl2(col1 INT PRIMARY
KEY,col2 INT,col3 VARCHAR(20),col4 VARCHAR(20));
Query OK, 0 rows affected
```

## 16.3 启动 Flink 集群和 Flink SQL CLI

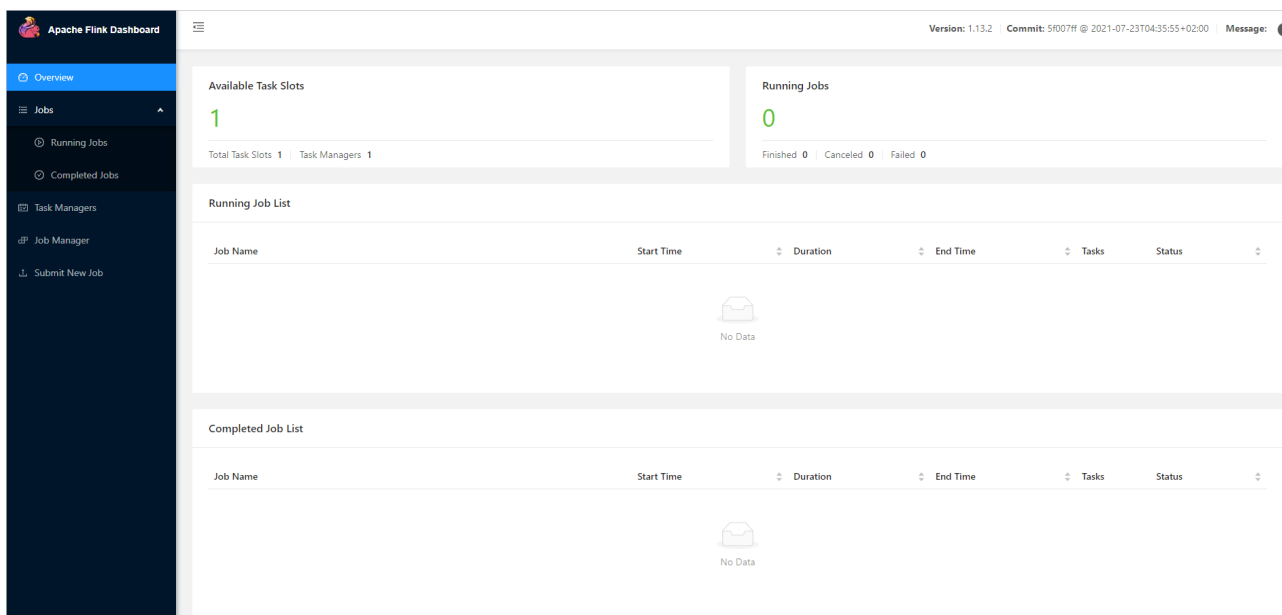
1. 使用下面的命令跳转至 Flink 目录下。

```
[xxx@xxx /FLINK_HOME]
#cd flink-1.15.3
```

2. 使用下面的命令启动 Flink 集群。

```
[xxx@xxx /FLINK_HOME/flink-1.15.3]
#./bin/start-cluster.sh
```

启动成功的话，可以在 `http://localhost:8081/` 访问到 Flink Web UI，如下所示：



## 16.3.4.1 说明

执行 `./bin/start-cluster.sh` 后，如果提示：`bash: ./bin/start-cluster.sh: Permission denied`。需要把 `flink-1.15.3` 录下的所有 `-rw-rw-r--` 权限的文件都设置为 `-rwxrwxrwx` 权限。

示例如下：

```
[xxx@xxx /.../flink-1.15.3]
# chmod -R 777 /FLINK_HOME/flink-1.15.3/*
```

3. 使用下面的命令启动 Flink SQL CLI。

```
[xxx@xxx /FLINK_HOME/flink-1.15.3]
# ./bin/sql-client.sh
```

启动成功后，可以看到如下的页面：



## 16.3.5 设置 checkpoint

在 Flink SQL CLI 中开启 `checkpoint`，每隔 3 秒做一次 `checkpoint`。

```
Flink SQL> SET execution.checkpointing.interval = 3s;
```

```
[INFO] Session property has been set.
```

## 16.3.6 创建 OceanBase CDC 表

在 Flink SQL CLI 中创建 OceanBase 数据库对应的表。对于 OceanBase 数据库中 `test_ob_to_mysql` 的表 `tbl1` 和 `tbl2` 使用 Flink SQL CLI 创建对应的表，用于同步这些底层数据库表的数据。

```
Flink SQL> CREATE TABLE ob_tbl1 (  
col1 INT PRIMARY KEY,  
col2 VARCHAR(20),  
col3 INT)  
WITH ('connector' = 'oceanbase-cdc',  
'scan.startup.mode' = 'initial',  
'tenant-name' = 'mysql001',  
'username' = 'root@mysql001',  
'password' = '*****',  
'database-name' = 'test_ob_to_mysql',  
'table-name' = 'tbl1',  
'hostname' = '10.10.10.2',  
'port' = '2881',  
'rootserver-list' = '10.10.10.2:2882:2881',  
'logproxy.host' = '10.10.10.2',  
'logproxy.port' = '2983');  
[INFO] Execute statement succeed.  
  
Flink SQL> CREATE TABLE ob_tbl2 (col1 INT PRIMARY KEY,  
col2 VARCHAR(20))  
WITH ('connector' = 'oceanbase-cdc',  
'scan.startup.mode' = 'initial',  
'tenant-name' = 'mysql001',  
'username' = 'root@mysql001',  
'password' = '*****',  
'database-name' = 'test_ob_to_mysql',  
'table-name' = 'tbl2',  
'hostname' = '10.10.10.2',  
'port' = '2881',
```

```
'rootserver-list' = '10.10.10.2:2882:2881',  
'logproxy.host' = '10.10.10.2',  
'logproxy.port' = '2983');  
[INFO] Execute statement succeed.
```

有关 OceanBase CDC WITH 选项的详细信息，请参见 [OceanBase CDC Connector](#)。

## 16.3.7 创建 MySQL CDC 表

在 Flink SQL CLI 中创建 MySQL 数据库对应的表。创建 `ob_tbl1_and_tbl2` 表，用来将同步的数据写入 MySQL 数据库中。

```
Flink SQL> CREATE TABLE ob_tbl1_and_tbl2(  
col1 INT PRIMARY KEY,  
col2 INT,col3 VARCHAR(20),  
col4 VARCHAR(20))  
WITH ('connector' = 'jdbc',  
'url' = 'jdbc:mysql://xxx.xxx.xxx.xxx:3306/test_ob_to_mysql',  
'username' = 'root',  
'password' = '*****',  
'table-name' = 'ob_tbl1_and_tbl2');  
[INFO] Execute statement succeed.
```

有关 JDBC SQL Connector WITH 选项的详细信息，请参见 [JDBC SQL Connector](#)。

## 16.3.8 在 Flink SQL CLI 中将数据写入 MySQL 数据库中

使用 Flink SQL 将表 `tbl1` 与表 `tbl2` 关联，并将关联后的信息写入 MySQL 数据库中。

```
Flink SQL> INSERT INTO ob_tbl1_and_tbl2  
SELECT t1.col1,t1.col3,t1.col2,t2.col2  
FROM ob_tbl1 t1,ob_tbl2 t2  
WHERE t1.col3=t2.col1;  
Flink SQL> INSERT INTO ob_tbl1_and_tbl2
```

```
> SELECT t1.col1,t1.col3,t1.col2,t2.col2
> FROM ob_tbl1 t1,ob_tbl2 t2
> WHERE t1.col3=t2.col1;
[INFO] Submitting SQL update statement to the cluster...
Loading class `com.mysql.jdbc.Driver`. This is deprecated. The new driver class is
`com.mysql.cj.jdbc.Driver`. The driver is automatically registered via the SPI and
manual loading of the driver class is generally unnecessary.
[INFO] SQL update statement has been successfully submitted to the cluster:
Job ID: 9cd180a65cb4e2c4d1a5a91465aa38a3
```

## 16.3.8.2 说明

本文档测试示例使用的 MySQL 驱动（`com.mysql.jdbc.Driver`）是 MySQL Connector/J 5.1.47 版本。新版本 MySQL 驱动（`com.mysql.cj.jdbc.Driver`）请使用 MySQL Connector/J 8.x 版本。

## 16.4 查看关联数据写入 MySQL 数据库情况

登录 MySQL 数据库，在 `test_ob_to_mysql` 库中查看表 `ob_tbl1_and_tbl2` 的数据。

```
MySQL [test_ob_to_mysql]> SELECT * FROM ob_tbl1_and_tbl2;
```

```
+-----+-----+-----+-----+
```

```
| col1 | col2 | col3 | col4 |
```

```
+-----+-----+-----+-----+
```

```
| 1 | 86 | China | +86 |
```

```
| 2 | 886 | Taiwan | +886 |
```

```
| 3 | 852 | Hong Kong | +852 |
```

```
| 4 | 853 | Macao | +853 |
```

```
| 5 | 850 | North Korea | +850 |
```

```
+-----+-----+-----+-----+
```

```
5 rows in set
```

## 17 使用 ChunJun 从 OceanBase 数据库迁移数据到 MySQL 数据库

纯钧（ChunJun，原名 FlinkX）是一款稳定、易用、高效、批流一体的数据集成框架，目前基于实时计算引擎 Flink 实现多种异构数据源之间的数据同步与计算。OceanBase CDC 组件 libobcdc 和 oblogproxy 提供了拉取 OceanBase 增量 commit log 的能力。chunjun-connector-oceanbasecdc 在内部集成了 oblogclient 来连接 oblogproxy，获取相关的日志数据，可以进行 OceanBase 数据库的增量数据迁移。

以下将介绍使用 ChunJun 的 Local 模式从 OceanBase 数据库（MySQL 模式）迁移数据到 MySQL 数据库。

### 17.1 场景描述



将 OceanBase 数据库表 `test_tbl1` 的数据迁移至 MySQL 数据库表 `tbl1` 中，数据库信息如下：

OceanBase 数据库信息（源库）	示例值
集群名	test4000
主机地址	10.10.10.2
端口号	2881
oblogproxy IP	10.10.10.3
oblogproxy 端口	2983
业务租户名称（MySQL 模式）	mysql001
用户名称	root
用户的密码	*****
Schema 库名称	test_data
表名	test_tbl1

MySQL 数据库信息（目标库）	示例值
主机地址	10.10.10.1
端口号	3306
用户名称	root
用户的密码	*****
Schema 库名称	test_ob_to_mysql
表名	tbl1

## 17.2 前提条件

- 安装 JDK 1.8，并配置好 `JAVA_HOME` 环境变量。
- 完成 oblogproxy 的安装部署。详细信息，请参见 [oblogproxy 安装](#)。

## 17.3 操作步骤

### 17.3.1 步骤一：ChunJun 环境准备

1. 下载 ChunJun 压缩包并解压。

下载 [chunjun-dist.tar.gz](#)。

```
wget https://github.com/DTStack/chunjun/releases/download/v1.12.6
/chunjun-dist.tar.gz
```

将压缩包解压至目录 `/chunjun_Home/chunjun`。

```
mkdir /ChunJun_Home/chunjun && tar zxvf chunjun-dist.tar.gz -C
/ChunJun_Home/chunjun
```

## 2. 配置 ChunJun 环境变量。

```
export ChunJun_HOME=/ChunJun_Home/chunjun-dist
```

## 17.3.2 步骤二：配置 json 文件

根据环境信息配置将 OceanBase 数据库表 `test_tbl1` 的数据迁移至 MySQL 数据库表 `tbl1` 的 json 文件。

示例如下：

```
[root@xxx /]
$cd /ChunJun_Home/chunjun-dist/chunjun-examples/json

[root@xxx /ChunJun_Home/chunjun-dist/chunjun-examples/json]
$mkdir test_data

[root@xxx /ChunJun_Home/chunjun-dist/chunjun-examples/json]
$cd mkdir test_data

[root@xxx /ChunJun_Home/chunjun-dist/chunjun-examples/json/test_data]
$vi test_ob_to_mysql.json

{
"job": {
"setting": {
```

```
"errorLimit": {
  "record": 0,
  "percentage": 0.02},
  "speed": {"bytes": 0,
  "channel": 1}
},
"content": [
{
  "reader": {
    "name": "oceanbasecdcreader",
    "table": {
      "tableName": "test_tbl1"},
    "parameter": {
      "logProxyHost": "10.10.10.3",
      "logProxyPort": 2983,
      "obReaderConfig": {
        "rsList": "10.10.10.2:2882:2881",
        "username": "root",
        "password": "*****"
      },
      "cat": "insert,delete,update",
      "column": [
        "*"
      ]
    }
  },
  "writer": {
    "name": "mysqlwriter",
    "parameter": {
```

```
"column": [  
  "*"   
],  
"connection": [  
  {  
    "jdbcUrl": "jdbc:mysql://10.10.10.1:3306/test_ob_to_mysql",  
    "table": [  
      "tbl1"  
    ]  
  }  
],  
"username": "root",  
"password": "*****"  
}  
}  
}  
]  
}  
}
```

### 17.3.3 步骤三：运行 json 配置文件

启动一个 JVM 进程执行 ChunJun 任务。

进入到 `/ChunJun_Home/chunjun-dist` 目录，执行命令以下命令：

```
[root@xxx /]  
$cd /ChunJun_Home/chunjun-dist  
  
[admin@xxx /ChunJun_Home/chunjun-dist]  
$sh bin/chunjun-local.sh -job chunjun-examples/json/test_data/test_ob_to_mysql.  
json
```

```
# #  
# #  
#  
#####  
# # # # #  
# # # # #  
# # # # #  
# # # # #  
#####  
#  
#####
```

Reference site: <https://dtstack.github.io/chunjun>

chunjun is starting ...

CHUNJUN\_HOME is auto set /ChunJun\_Home/chunjun-dist/

FLINK\_HOME is empty!

HADOOP\_HOME is empty!

...

<ellipsis>

...

\*\*\*\*\*

numWrite | 3

last\_write\_num\_0 | 0

conversionErrors | 0

writeDuration | 20149

duplicateErrors | 0

numRead | 3

snapshotWrite | 0

```
otherErrors | 0
readDuration | 95
byteRead | 389
last_write_location_0 | 0
byteWrite | 389
nullErrors | 0
nErrors | 0
*****

2023-01-06 20:34:47,448 - 50317 INFO [main] com.dtstack.chunjun.Main:program
Flink_Job execution success
```

## 17.3.4 步骤四：对 OceanBase 数据库数据增量修改

登录到 OceanBase 数据库（MySQL 模式），并在表 `test_tbl1` 中插入一条数据。

```
[admin@xxx /home/admin]
$obclient -h10.10.10.2 -P2881 -uroot@mysql001 -p***** -Dtest_data
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the OceanBase. Commands end with ; or \g.
Your OceanBase connection id is 3221709678
Server version: OceanBase 4.0.0.0 (r101000022022120716-
0d7927892ad6d830e28437af099f018b0ad9a322) (Built Dec 7 2022 16:22:15)

Copyright (c) 2000, 2018, OceanBase and/or its affiliates. All rights reserved.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
obclient [test_data]> INSERT INTO test_tbl1 VALUES(4,'Macao',853);
Query OK, 1 row affected
```

## 17.3.5 步骤五：查看数据迁移情况

登录到 MySQL 数据库查看数据迁移情况。

```
[admin@xxx /home/admin]
$mysql -h10.10.10.1 -P3306 -uroot -p*****
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MySQL connection id is 189769
Server version: 5.7.26-log MySQL Community Server (GPL)

<ellipsis>

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> use test_ob_to_mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [test_ob_to_mysql]> SELECT * FROM tbl1;
+-----+-----+-----+
| col1 | col2 | col3 |
+-----+-----+-----+
| 1 | China | 86 |
| 2 | Taiwan | 886 |
| 3 | Hong Kong | 852 |
| 4 | Macao | 853 |
```

```
+-----+-----+-----+
```

```
4 rows in set
```

## 17.4 相关文档

- ChunJun 快速入门的详细信息，请参见 [快速开始](#)。
- MySQL 数据库作为目标库配置 json 文件的详细信息，请参见 [MySQL Sink](#)。
- 更多脚本示例，请参见项目内 /chunjun-dist/chunjun-examples 文件夹下的文件。



# 18 使用 OMS 从 Oracle 数据库迁移数据到 OceanBase 数据库 MySQL 租户

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 Oracle 数据库迁移数据到 OceanBase 数据库 MySQL 租户中的背景信息。

## 18.1 背景信息

在 OMS 控制台创建从 Oracle 数据库迁移数据至 OceanBase 数据库 MySQL 租户的数据迁移项目，您可以通过结构迁移、全量迁移和增量数据同步，无缝迁移源端数据库中的存量业务数据和增量数据至 OceanBase 数据库 MySQL 租户。

Oracle 数据库支持单主库、单备库和主备库等模式。迁移 Oracle 数据库的数据至 OceanBase 数据库 MySQL 租户的数据库时，不同类型的数据源支持的操作也不同。

类型	支持的操作
单主库	结构迁移 + 全量迁移 + 增量同步 + 全量校验 + 反向增量
单备库	结构迁移 + 全量迁移 + 增量同步 + 全量校验
主备库	主库：支持反向增量 备库：支持结构迁移 + 全量迁移 + 增量同步 + 全量校验

## 18.2 相关文档

更多使用 OMS 从 Oracle 数据库迁移数据到 OceanBase 数据库 MySQL 租户的操作信息，请参见 [迁移 Oracle 数据库的数据至 OceanBase 数据库 MySQL 租户](#)。

# 19 使用 OMS 从 Oracle 数据库迁移数据到 OceanBase 数据库 Oracle 租户

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 Oracle 数据库迁移数据到 OceanBase 数据库 Oracle 租户中的背景信息。

## 19.1 背景信息

在 OMS 控制台创建从 Oracle 数据库迁移数据至 OceanBase 数据库 Oracle 租户的数据迁移项目，您可以通过结构迁移、全量迁移和增量数据同步，无缝迁移源端数据库中的存量业务数据和增量数据至 OceanBase 数据库 Oracle 租户。

Oracle 数据库支持单主库、单备库和主备库等模式。迁移 Oracle 数据库的数据至 OceanBase 数据库 Oracle 租户的数据库时，不同类型的数据源支持的操作也不同。

类型	支持的操作
单主库	结构迁移 + 全量迁移 + 增量同步 + 全量校验 + 反向增量
单备库	结构迁移 + 全量迁移 + 增量同步 + 全量校验
主备库	主库：支持反向增量 备库：支持结构迁移 + 全量迁移 + 增量同步 + 全量校验

## 19.2 相关文档

更多使用 OMS 从 Oracle 数据库迁移数据到 OceanBase 数据库 Oracle 租户的操作信息，请参见 [迁移 Oracle 数据库的数据至 OceanBase 数据库 Oracle 租户](#)。

## 20 使用 DBCAT 迁移 Oracle 表结构到 OceanBase 数据库

DBCAT 是一款轻量级的命令行工具，可用于提供数据库之间 DDL 转换和 Schema 比对等功能。这里以 DBCAT 迁移表结构为例进行介绍。

DBCAT 安装包文件名为 `dbcat-[版本号]-SNAPSHOT.tar.gz`，下载后解压缩即可使用，可执行文件名为 `dbcat`。

### 20.0.0.1 注意

DBCAT 是 OMS 的一个组件，在社区版环境中推荐使用 OMS 导出。

## 20.1 环境准备

DBCAT 能运行在 CentOS、macOS 和 Windows 下。需要安装 JDK 1.8 以上（含）版本。可以使用 OpenJDK，安装好后配置环境变量 `JAVA_HOME`。

CentOS 安装 OpenJDK 示例：

```
$sudo yum -y install java-1.8.0-openjdk.x86_64

$which java
/usr/local/java/jdk1.8.0_261/bin/java

echo 'export JAVA_HOME=/usr/local/java/jdk1.8.0_261/' >> ~/.bash_profile
. ~/.bash_profile
```

解压安装文件：

```
tar zxvf dbcat-1.8.0-SNAPSHOT.tar.gz
cd dbcat-1.8.0-SNAPSHOT/
chmod +x bin/dbcat

$tree -L 3 --filelimit 30
```

```
.
├── bin
│   ├── dbcat
│   ├── dbcat.bat
│   └── dbcat-debug
├── conf
│   ├── dbcat.properties
│   └── logback.xml
├── docs
│   ├── README.docx
│   ├── README.md
│   └── README.txt
├── LEGAL.md
├── lib [45 entries exceeds filelimit, not opening dir]
├── LICENSE
├── meta
│   └── README
└── NOTICE

5 directories, 12 files
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。
conf	日志文件配置目录。
lib	运行时期依赖的包。
meta	离线转换场景下，导出字典表数据。
~/output	SQL 文件与报告文件，运行时生成。

## 20.2 导出 Oracle 数据库的表结构

DBCAT 具有在线转换功能，该功能是指 DBCAT 能直连源端数据库，将数据库中的对象导出。当对象非常多时（如超过 1 万），导出过程可能会有点慢。

dbcat 导出命令如下：

```
bin/dbcat convert -H<host> -P<port> -u<user> -p<*****> -D<database> --service-name<service-name> --service-id <service-id> --from <from> --to <to> --all
```

您可运行命令 `bin/dbcat help convert` 查看更多参数信息。

必选参数：

选项	有无参数	中文描述
-H/--host	Y	数据库服务器的 IP 地址
-P/--port	Y	数据库服务器的端口
-u/--user	Y	登录数据库的用户名
-t/--tenant	Y	连接 OceanBase 集群需要提供租户名
-c/--cluster	Y	连接 OceanBase 集群需要提供集群名
-p/--password	Y	登录数据库的密码
-D/--database	Y	数据库名（源库），DB2 须区分数据库名和模式名
--service-id	Y	连接 Oracle 数据库需要提供服务 ID
--service-name	Y	连接 Oracle 数据库需要提供服务名
--as-sysdba	N	连接 Oracle 数据库 sysdba 角色
--sys-user	Y	连接 OceanBase 集群系统租户的用户名
--sys-password	Y	连接 OceanBase 集群系统租户的密码
--schema	Y	模式名（源库），非 DB2，模式名与数据名相同
--from	Y	源库的类型
--to	Y	目标库的类型
--all	N	所有的数据库对象

可选参数：

选项	有无参数	中文描述
-f/--file	Y	sql 文件的输出路径
--offline	N	使用离线模式
--target-schema	Y	模式名（目标库）
--table	Y	导出的表
--view	Y	导出的视图
--trigger	Y	导出的触发器
--synonym	Y	导出的同义词
--sequence	Y	导出的序列
--function	Y	导出的函数
--procedure	Y	导出的存储过程
--dblink	Y	导出所有的 DBLink
--type	Y	导出的 type
--type-body	Y	导出的 type body
--package	Y	导出的 package
--package-body	Y	导出的 package body
--no-quote	N	产生的 DDL 不带引号
--no-schema	N	产生的 DDL 不带模式名
--target-schema	Y	产生的 DDL 中使用指定的模式名
--exclude-type	Y	搭配 --all 使用，如：--all --exclude-type 'TABLE' 表示排除 TABLE 类型

这里以导出 Oracle 12c 版本下 database 为 test 的所有对象的结构，并将其迁移到 4.0.0 版本的 OceanBase 集群中的 Oracle 租户为示例。

```
bin/dbcat convert -H 100.88.xxx.xxx -P 1521 -uxxx -pxxxxxxx -D test --service-name
xxx --service-id xxx --from oracle12c --to oboracle40 --all
```

特别说明：

- dbcat 不需要直接安装在数据库主机上，安装在可直连数据库主机的主机上即可。

- 参数中的 --from 和 --to 为源端和目的端的数据库类型，需要详细到版本号。当前 dbcat 支持的源端和目标端数据库详细如下：

源端数据库类型	目标端数据库类型
TiDB	OBMYSQL
PG	OBMYSQL
SYBASE	OBORACLE
MYSQL	OBMYSQL
ORACLE	OBORACLE
ORACLE	OBMYSQL
DB2 IBM i	OBORACLE
DB2 LUW	OBORACLE
DB2 LUW	OBMYSQL
OBMYSQL	MYSQL
OBORACLE	ORACLE

其中 OBMYSQL 为 OceanBase 数据库的 MySQL 租户，OBORACLE 为 OceanBase 数据库的 Oracle 租户。

- 当前支持的源端和目标端数据库详细的版本，详情如下。

数据库类型	数据库版本
TiDB	tidb4 tidb5
PG	pgsql10
SYBASE	sybase15
DB2 IBM i	db2ibmi71
DB2 LUW	db2luw970 db2luw1010 db2luw1050 db2luw111 db2luw115
MYSQL	mysql56 mysql57 mysql</80>

ORACLE	oracle9i oracle10g oracle11g oracle12c oracle18c oracle19c
OBTMySQL	obmysql14x obmysql21x obmysql22x obmysql200 obmysql211 obmysql2210 obmysql2230 obmysql2250 obmysql2271 ~ obmysql2277 obmysql30x obmysql31x obmysql32x obmysql322 obmysql40
OBTOracle	oboracle2220 oboracle2230 oboracle2250 oboracle2270 ~ oboracle2277 oboracle21x oboracle22x oboracle30x oboracle31x oboracle32x oboracle322 oboracle40

运行后的输出文件在用户 `home` 目录的 `output` 下。

```
$tree ~/output/dbcat-20xx-xx-xx-164533/
/home/qing.meiq/output/dbcat-20xx-xx-xx-164533/
|—— tpccdb
|  |—— TABLE-schema.sql
|—— tpccdb-conversion.html

1 directory, 2 files
```



## 20.3 导入 OceanBase 数据库

使用 DBCAT 导出的文件格式为 SQL 文件，这里可以通过 ODC 的导入功能批量将表结构导入 OceanBase 数据库，更多信息请参考 [批量导出与导入](#)。

也可以使用 `source` 命令将 SQL 文件数据导入 OceanBase 数据库，示例如下：

```
obclient [test]> source TABLE-schema.sql
Query OK, 0 rows affected (0.044 sec)
```

### 20.3.0.1 注意

如果 SQL 文件不在当前目录下，则需要使用绝对地址。

## 20.4 导数结果验证

示例：查看一个表结构在 Oracle 数据库的表结构和 OceanBase 数据库里的表结构。

查看源数据库 Oracle 的表 `bmsql_customer` 的表结构：

```
SQL> desc bmsql_customer;
Name Null? Type
-----
C_W_ID NOT NULL NUMBER(38)
C_D_ID NOT NULL NUMBER(38)
C_ID NOT NULL NUMBER(38)
C_DISCOUNT NUMBER(4,4)
C_CREDIT CHAR(2)
C_LAST VARCHAR2(16)
C_FIRST VARCHAR2(16)
C_CREDIT_LIM NUMBER(12,2)
C_BALANCE NUMBER(12,2)
C_YTD_PAYMENT NUMBER(12,2)
C_PAYMENT_CNT NUMBER(38)
C_DELIVERY_CNT NUMBER(38)
C_STREET_1 VARCHAR2(20)
```

```

C_STREET_2 VARCHAR2(20)
C_CITY VARCHAR2(20)
C_STATE CHAR(2)
C_ZIP CHAR(9)
C_PHONE CHAR(16)
C_SINCE TIMESTAMP(6)
C_MIDDLE CHAR(2)
C_DATA VARCHAR2(500)

```

查看目标库数据库 OceanBase 的表 bmsql\_customer 的表结构：

```

obclient [SYS]> desc bmsql_customer;
+-----+-----+-----+-----+-----+-----+
| FIELD | TYPE | NULL | KEY | DEFAULT | EXTRA |
+-----+-----+-----+-----+-----+-----+
| C_W_ID | NUMBER(38) | NO | PRI | NULL | NULL |
| C_D_ID | NUMBER(38) | NO | PRI | NULL | NULL |
| C_ID | NUMBER(38) | NO | PRI | NULL | NULL |
| C_DISCOUNT | NUMBER(4,4) | YES | NULL | NULL | NULL |
| C_CREDIT | CHAR(2) | YES | NULL | NULL | NULL |
| C_LAST | VARCHAR2(16) | YES | NULL | NULL | NULL |
| C_FIRST | VARCHAR2(16) | YES | NULL | NULL | NULL |
| C_CREDIT_LIM | NUMBER(12,2) | YES | NULL | NULL | NULL |
| C_BALANCE | NUMBER(12,2) | YES | NULL | NULL | NULL |
| C_YTD_PAYMENT | NUMBER(12,2) | YES | NULL | NULL | NULL |
| C_PAYMENT_CNT | NUMBER(38) | YES | NULL | NULL | NULL |
| C_DELIVERY_CNT | NUMBER(38) | YES | NULL | NULL | NULL |
| C_STREET_1 | VARCHAR2(20) | YES | NULL | NULL | NULL |
| C_STREET_2 | VARCHAR2(20) | YES | NULL | NULL | NULL |
| C_CITY | VARCHAR2(20) | YES | NULL | NULL | NULL |
| C_STATE | CHAR(2) | YES | NULL | NULL | NULL |

```

```
| C_ZIP | CHAR(9) | YES | NULL | NULL | NULL |  
| C_PHONE | CHAR(16) | YES | NULL | NULL | NULL |  
| C_SINCE | TIMESTAMP(6) | YES | NULL | sysdate | NULL |  
| C_MIDDLE | CHAR(2) | YES | NULL | NULL | NULL |  
| C_DATA | VARCHAR2(500) | YES | NULL | NULL | NULL |  
+-----+-----+-----+-----+-----+-----+  
21 rows in set (0.002 sec)
```

经对比是一致的。

## 21 使用 DataX 迁移 Oracle 表数据到 OceanBase 数据库

DataX 是阿里云 DataWorks 数据集成的开源版本，是阿里巴巴集团内被广泛使用的离线数据同步工具/平台。DataX 实现了包括 MySQL、Oracle、SQLserver、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、Hologres、DRDS、OceanBase 等各种异构数据源之间高效的数据同步功能。

OceanBase 数据库社区版客户，可以在 [DataX 开源网站](#) 内下载源码，自行编译。编译时，可根据需要剔除在 pom.xml 中不用的数据库插件，否则编译出来的包会非常大。

### 21.1 框架设计

DataX 作为离线数据同步框架，采用 "Framework + Plugin" 模式构建。将数据源读取和写入抽象为 Reader/Writer 插件，纳入到整个同步框架中。

- Reader 作为数据采集模块，负责采集数据源的数据，将数据发送给 Framework。
- Writer 作为数据写入模块，负责不断向 Framework 获取数据，并将数据写入到目的端。
- Framework 用于连接 Reader 和 Writer，作为两者的数据传输通道，并处理缓冲、流控、并发、数据转换等核心技术问题。

DataX 以任务的形式迁移数据。每个任务只处理一个表，每个任务有一个 json 格式的配置文件，配置文件里包含 reader 和 writer 两部分。reader 和 writer 分别对 DataX 支持的数据库读写插件，例如，将 Oracle 表数据迁移到 OceanBase 数据库操作时，需要从 Oracle 读取数据写入 OceanBase 数据库，因此使用的插件为 Oracle 数据库的 oraclereader 插件和 OceanBase 数据库的 oceanbasev10writer 插件来搭配完成。这里介绍下 oraclereader 和 oceanbasev10writer 插件。

#### 21.1.1 oraclereader 插件

oraclereader 插件实现了从 Oracle 数据库上读取数据的功能。在底层实现上，oraclereader 插件通过 JDBC 连接远程 Oracle 数据库，并执行相应的 SQL 语句将数据从 Oracle 库中 SELECT 出来。

实现原理方面，简而言之，oraclereader 插件通过 JDBC 连接器连接到远程的 Oracle 数据库，并根据用户配置的信息生成查询语句，然后发送到远程 Oracle 数据库，远程的 Oracle 数据库将该 SQL 执行的返回结果使用 DataX 自定义的数据类型拼装为抽象的数据集，再传递给下游 Writer 处理。

详细功能和参数说明请参考官方说明：[oraclereader 插件](#)。

## 21.1.2 oceanbasev10writer 插件

`oceanbasev10writer` 插件实现了将数据写入到 OceanBase 数据库的目标表中的功能。在底层实现上，`oceanbasev10writer` 通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并执行相应的 insert sql 语句来将数据写入到 OceanBase 数据库，OceanBase 数据库内部会分批次提交入库。

实现原理方面，`oceanbasev10writer` 通过 DataX 框架获取 Reader 生成的协议数据，生成 insert 语句。在写入数据时，如果出现主键或唯一键冲突，OceanBase 数据库的 MySQL 租户可以通过 `replace` 模式来更新表中的所有字段；OceanBase 数据库的 Oracle 租户当前只能使用 Insert 方式。出于性能考虑，写入采用 batch 方式批量写，当行数累计到预定阈值时，才发起写入请求。

## 21.1.3 DataX 配置文件

配置文件示例：

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "sliceRecordCount": 10,
            "column": [
              {
                "type": "long",
                "value": "10"
              },
              {
                "type": "string",
```

```
"value": "hello, 你好, 世界-DataX"
}
]
}
},
"writer": {
  "name": "streamwriter",
  "parameter": {
    "encoding": "UTF-8",
    "print": true
  }
}
}
},
"setting": {
  "speed": {
    "channel": 2
  }
}
}
}
```

## 21.1.3.1 注意

datax 仅迁移表数据，需要提前在目标端创建好对应的表对象结构。

将 `json` 配置文件放到 DataX 的目录 `job` 下，或者自定义路径。执行方法如下：

```
$bin/datax.py job/stream2stream.json
```

输出信息：

```
<.....>
```

```
2021-08-26 11:06:09.217 [job-0] INFO JobContainer - PerfTrace not enable!
2021-08-26 11:06:09.218 [job-0] INFO StandAloneJobContainerCommunicator - Total
20 records, 380 bytes | Speed 38B/s, 2 records/s | Error 0 records, 0 bytes | All Task
WaitWriterTime 0.000s | All Task WaitReaderTime 0.000s | Percentage 100.00%
2021-08-26 11:06:09.223 [job-0] INFO JobContainer -
任务启动时刻：2021-08-26 11:05:59
任务结束时刻：2021-08-26 11:06:09
任务总计耗时：10s
任务平均流量：38B/s
记录写入速度：2rec/s
读出记录总数：20
读写失败总数：0
```

DataX 任务执行结束会有个简单的任务报告，包含上述输出的平均流量、写入速度和读写失败总数等。

DataX 的 `job` 的参数 `settings` 可以指定速度参数和错误记录容忍度等。

```
"setting": {
  "speed": {
    "channel": 10
  },
  "errorLimit": {
    "record": 10,
    "percentage": 0.1
  }
}
```

参数说明：

- `errorLimit` 表示报错记录数的容忍度，超出这个限制后任务就中断退出。

- `channel` 是并发数，理论上并发越大，迁移性能越好。但实际操作中也要考虑源端的读压力、网络传输性能以及目标端写入性能。

## 21.2 环境准备

下载 tar 包地址：<http://datax-opensource.oss-cn-hangzhou.aliyuncs.com/datax.tar.gz>

解压安装文件：

```
tar zxvf datax.tar.gz
cd datax
```

目录如下：

```
$tree -L 1 --filelimit 30
.
├── bin
├── conf
├── job
├── lib
├── log
├── log_perf
├── plugin
├── script
└── tmp
```



安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。该目录下的 datax.py 为 DataX 任务的启动脚本
conf	日志文件配置目录。该目录下存放 datax 与任务无关的配置文件
lib	运行时期依赖的包。该目录存放 DataX 运行所需要的全局 jar 文件
job	该目录下有一个用于测试验证 datax 安装的任务配置文件
log	日志文件目录。该目录下存放 datax 任务运行的日志；datax 运行时，默认会将日志输出到标准输出，同时写入到 log 目录下
plugin	插件文件目录。该目录下保存 DataX 支持的各种数据源插件

## 21.3 使用 DataX 迁移 Oracle 数据到 OceanBase 示例

将 Oracle 数据迁移到 OceanBase，如果源端和目标端不能同时跟 DataX 服务器网络联通，那么可以通过 CSV 文件中转。如果源端数据库和目标端数据库能同时跟 DataX 所在服务器联通，则可以使用 DataX 直接将数据从源端迁移到目标端。

示例：从 Oracle 迁移 ZJSZY.TECTM03 表数据到 OceanBase Oracle 租户下的 ZJSZY.TECTM03。

myjob.json 配置文件如下：

```
{
  "job":{
    "setting":{
      "speed":{
        "channel":32
      },
      "errorLimit":{
        "percentage": 0.1
      }
    }
  }
```

```
,
"content": [
{
"reader": {
"name": "oraclereader",
"parameter": {
"username": "user_name",
"password": "*****",
"column": ["*"],
"connection": [
{
"table": ["ZJSZY.TECTM03"],
"jdbcUrl": ["jdbc:oracle:thin:@Oracle_ip:1521:orcl"]
}
]
}
},
"writer": {
"name": "oceanbasev10writer",
"parameter": {
"writeMode": "insert",
"batchSize": 5000,
"memstoreThreshold": "90",
"username": "user@tenet#cluster",
"password": "*****",
"column": ["*"],
"connection": [
{
"table": ["ZJSZY.TECTM03"],
```

```
"jdbcUrl":"jdbc:oceanbase://odp_ip:3306/zjszy"
}
]
}
}
}
}
]
}
}
```

### 参数说明

参数	描述
name	描述的是连接数据库的 reader 或 writer 对应的数据库插件的名称。其中 Oracle 的 reader 插件为：oraclereader，OceanBase 的 writer 插件为 oceanbasev10writer。具体 reader 和 writer 的插件可以参考 datax 的文档： <a href="#">DataX 数据源指南</a> 。
jdbcUrl	<p>描述的是到连接的数据库的 JDBC 信息，使用 JSON 的数组描述，并支持一个库填写多个连接地址。您在 JSON 数组中填写一个 JDBC 连接即可。jdbcUrl 按照 MySQL 官方规范，并可以填写连接附件控制信息。具体请参见 <a href="#">MySQL 官方文档</a>。</p> <h2>21.3.3.1 注意</h2> <ul style="list-style-type: none"><li>• jdbcUrl 必须包含在 connection 配置单元中。</li><li>• OceanBase 数据库需要通过 obproxy 进行连接，端口默认 2883。</li><li>• writer 端的 jdbcUrl，无需在连接串两端加 []，reader 端的 jdbcUrl，必需在连接串两端加 []</li><li>• 必选：是</li><li>• 默认值：无</li></ul>
username	<p>数据源的用户名</p> <p>必选：是</p> <p>默认值：无</p>

password	<p>数据源指定用户名的密码</p> <p>必选：是</p> <p>默认值：无</p>
table	<p>所选取的需要同步的表。使用 JSON 的数组描述，因此支持多张表同时抽取。当配置为多张表时，用户自己需确保多张表是同一 schema 结构，OracleReader 不予检查表是否同一逻辑表。</p> <h2>21.3.3.2 注意</h2> <p>table 必须包含在 connection 配置单元中。</p> <ul style="list-style-type: none"><li>• 必选：是</li><li>• 默认值：无</li></ul>
column	<p>所配置的表中需要同步的列名集合，使用 JSON 的数组描述字段信息。columns 不建议配置为 ['*']，因为表结构发生变化时该配置也会发生变化。建议的配置方式是指定具体的列名。支持列裁剪，即列可以挑选部分列进行导出。支持列换序，即列可以不按照表 schema 信息进行导出。支持常量配置，用户需要按照 MySQL SQL 语法格式：["id", "`table`", "1", "bazhen.csy", "null", "to_char(a + 1)", "2.3", "true"]。</p> <h2>21.3.3.3 说明</h2> <ul style="list-style-type: none"><li>• id 为普通列名。</li><li>• table 为包含保留字的列名；</li><li>• 1 为整形数字常量。</li><li>• bazhen.csy 为字符串常量。</li><li>• null 为空指针。</li><li>• to_char(a + 1) 为表达式。</li><li>• 2.3 为浮点数。</li><li>• true 为布尔值。</li><li>• 必选：是</li><li>• 默认值：无</li></ul>

where

筛选条件，OracleReader 根据指定的column、table、where 条件拼接 SQL，并根据这个 SQL 进行数据抽取。在实际业务场景中，往往会选择当天的数据进行同步，可以将where 条件指定为gmt\_create > \$bizdate。

### 21.3.3.4 注意

不可以将 where 条件指定为 limit 10，limit 不是 SQL 的合法 where 子句。where 条件可以有序地进行业务增量同步。如果不填写 where 语句，包括不提供 where 的 key 或者 value，DataX 均视作同步全量数据。

- 必选：否
- 默认值：无

配置 job 文件后，执行该 job。命令如下：

```
python datax.py ../job/myjob.json
```

## 21.4 更多信息

关于 DataX 的开源代码和更多信息，请参见 [DataX](#)。

## 22 使用 OMS 从 OceanBase 数据库 Oracle 租户迁移数据到 Oracle 数据库

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 OceanBase 数据库 Oracle 租户迁移数据到 Oracle 数据库中的背景信息。

### 22.1 背景信息

在 OMS 控制台创建从 OceanBase 数据库 Oracle 租户迁移数据至 Oracle 数据库的数据迁移项目，您可以通过结构迁移、全量迁移和增量同步，无缝迁移源端数据库中的存量业务数据和增量数据至 Oracle 数据库。

Oracle 数据库支持单主库、单备库和主备库等模式。迁移 OceanBase 数据库 Oracle 租户的数据至 Oracle 数据库时，不同类型的数据源支持的操作也不同。

类型	支持的操作
单主库	结构迁移 + 全量迁移 + 增量同步 + 全量校验 + 反向增量
单备库	不支持将单备库的 Oracle 数据源作为数据迁移的目标端
主备库	主库：支持结构迁移 + 全量迁移 + 增量同步 备库：支持全量校验 + 反向增量

同时，OMS 支持将多个 OceanBase 数据库 Oracle 租户的表数据汇聚至 Oracle 数据库的同一张表中，其中无需结构迁移，只需要进行全量迁移和增量同步。该汇聚同步功能的使用限制如下：

- 对于全量迁移和增量同步，源端有的列，目标端必须有。如果不满足该要求，OMS 会报错。
- 主键列在源端表中必须存在。
- 目标表中的列，可以存在源端不存在的列。

### 22.2 相关文档

更多使用 OMS 从 OceanBase 数据库 Oracle 租户迁移数据到 Oracle 数据库的操作信息，请参见 [迁移 OceanBase 数据库 Oracle 租户的数据至 Oracle 数据库](#)。

## 23 使用 DBCAT 迁移 OceanBase 表结构到 Oracle 数据库

DBCAT 是一款轻量级的命令行工具，可用于提供数据库之间 DDL 转换和 Schema 比对等功能。这里以 DBCAT 迁移表结构为例进行介绍。

DBCAT 安装包文件名为 `dbcat-[版本号]-SNAPSHOT.tar.gz`，下载后解压缩即可使用，可执行文件名为 `dbcat`。

### 23.0.0.1 注意

DBCAT 是 OMS 的一个组件，在社区版环境中推荐使用 OMS 导出。

## 23.1 环境准备

DBCAT 能运行在 CentOS、macOS 和 Windows 下。需要安装 JDK 1.8 以上（含）版本。可以使用 OpenJDK，安装好后配置环境变量 `JAVA_HOME`。

CentOS 安装 OpenJDK 示例：

```
$sudo yum -y install java-1.8.0-openjdk.x86_64

$which java
/usr/local/java/jdk1.8.0_261/bin/java

echo 'export JAVA_HOME=/usr/local/java/jdk1.8.0_261/' >> ~/.bash_profile
. ~/.bash_profile
```

解压安装文件：

```
tar zxvf dbcat-1.8.0-SNAPSHOT.tar.gz
cd dbcat-1.8.0-SNAPSHOT/
chmod +x bin/dbcat

$tree -L 3 --filelimit 30
```

```
.
├── bin
│   ├── dbcat
│   ├── dbcat.bat
│   └── dbcat-debug
├── conf
│   ├── dbcat.properties
│   └── logback.xml
├── docs
│   ├── README.docx
│   ├── README.md
│   └── README.txt
├── LEGAL.md
├── lib [45 entries exceeds filelimit, not opening dir]
├── LICENSE
├── meta
│   └── README
└── NOTICE

5 directories, 12 files
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。
conf	日志文件配置目录。
lib	运行时期依赖的包。
meta	离线转换场景下，导出字典表数据。
~/output	SQL 文件与报告文件，运行时生成。



## 23.2 导出 OceanBase 数据库 Oracle 租户下的表结构

DBCAT 具有在线转换功能，该功能是指 DBCAT 能直连源端数据库，将数据库中的对象导出。当对象非常多时（如超过 1 万），导出过程可能会有点慢。

dbcat 导出命令如下：

```
bin/dbcat convert -H<host> -P<port> -u<user> -p<*****> -D <database> --from  
<from> --to <to> --table '*'
```

您可运行命令 `bin/dbcat help convert` 查看更多参数信息。

必选参数：

选项	有无参数	中文描述
-H/--host	Y	数据库服务器的 IP 地址
-P/--port	Y	数据库服务器的端口
-u/--user	Y	登录数据库的用户名
-t/--tenant	Y	连接 OceanBase 集群需要提供租户名
-c/--cluster	Y	连接 OceanBase 集群需要提供集群名
-p/--password	Y	登录数据库的密码
-D/--database	Y	数据库名（源库），DB2 须区分数据库名和模式名
--service-id	Y	连接 Oracle 数据库需要提供服务 ID
--service-name	Y	连接 Oracle 数据库需要提供服务名
--as-sysdba	N	连接 Oracle 数据库 sysdba 角色
--sys-user	Y	连接 OceanBase 集群系统租户的用户名
--sys-password	Y	连接 OceanBase 集群系统租户的密码
--schema	Y	模式名（源库），非 DB2，模式名与数据名相同
--from	Y	源库的类型
--to	Y	目标库的类型
--all	N	所有的数据库对象

可选参数：

选项	有无参数	中文描述
-f/--file	Y	sql 文件的输出路径
--offline	N	使用离线模式
--target-schema	Y	模式名（目标库）
--table	Y	导出的表
--view	Y	导出的视图
--trigger	Y	导出的触发器
--synonym	Y	导出的同义词
--sequence	Y	导出的序列
--function	Y	导出的函数
--procedure	Y	导出的存储过程
--dblink	Y	导出所有的 DBLink
--type	Y	导出的 type
--type-body	Y	导出的 type body
--package	Y	导出的 package
--package-body	Y	导出的 package body
--no-quote	N	产生的 DDL 不带引号
--no-schema	N	产生的 DDL 不带模式名
--target-schema	Y	产生的 DDL 中使用指定的模式名
--exclude-type	Y	搭配 --all 使用，如：--all --exclude-type 'TABLE' 表示排除 TABLE 类型

这里以导出 4.0.0 版本的 OceanBase 集群中的 Oracle 租户下 test 的所有表结构，并将其迁移到 Oracle 12c 版本下为示例。

```
bin/dbcat convert -H 172.30.xxx.xxx -P 2883 -uroot -pxxxxxx -D test --from
oboracle40 --to oracle12c --table '*'
```

特别说明：

- dbcat 不需要直接安装在数据库主机上，安装在可直连数据库主机的主机上即可。

- 参数中的 --from 和 --to 为源端和目的端的数据库类型，需要详细到版本号。当前 dbcat 支持的源端和目标端数据库详细如下：

源端数据库类型	目标端数据库类型
TiDB	OBMYSQL
PG	OBMYSQL
SYBASE	OBORACLE
MYSQL	OBMYSQL
ORACLE	OBORACLE
ORACLE	OBMYSQL
DB2 IBM i	OBORACLE
DB2 LUW	OBORACLE
DB2 LUW	OBMYSQL
OBMYSQL	MYSQL
OBORACLE	ORACLE

其中 OBMYSQL 为 OceanBase 数据库的 MySQL 租户，OBORACLE 为 OceanBase 数据库的 Oracle 租户。

- 当前支持的源端和目标端数据库详细的版本，详情如下。

数据库类型	数据库版本
TiDB	tidb4 tidb5
PG	pgsql10
SYBASE	sybase15
DB2 IBM i	db2ibmi71
DB2 LUW	db2luw970 db2luw1010 db2luw1050 db2luw111 db2luw115
MYSQL	mysql56 mysql57 mysql</80>

ORACLE	oracle9i oracle10g oracle11g oracle12c oracle18c oracle19c
OBMYSQL	obmysql14x obmysql21x obmysql22x obmysql200 obmysql211 obmysql2210 obmysql2230 obmysql2250 obmysql2271 ~ obmysql2277 obmysql30x obmysql31x obmysql32x obmysql322 obmysql40
OBORACLE	oboracle2220 oboracle2230 oboracle2250 oboracle2270 ~ oboracle2277 oboracle21x oboracle22x oboracle30x oboracle31x oboracle32x oboracle322 oboracle40

运行后的输出文件在用户 `home` 目录的 `output` 下。

```
$tree ~/output/dbcat-20xx-xx-xx-164533/
/home/qing.meiq/output/dbcat-20xx-xx-xx-164533/
|—— tpccdb
|  |—— TABLE-schema.sql
|—— tpccdb-conversion.html

1 directory, 2 files
```

## 23.3 导入 Oracle 数据库

使用 DBCAT 导出的文件格式为 SQL 文件，导入 Oracle 数据库，可以进库后使用 @文件名 命令导入 SQL 文件。示例如下：

```
SQL> @TABLE-schema.sql
Query OK, 0 rows affected (0.044 sec)
```

### 23.3.0.1 注意

如果 sql 文件不在当前目录下，则需要使用绝对地址。

## 23.4 导数结果验证

示例：查看一个表结构在 OceanBase 数据库里和 ORACLE 数据库中的表结构的。

查看源数据库 OceanBase 的表 bmsql\_customer 的表结构：

```
obclient [SYS]> desc bmsql_customer;
+-----+-----+-----+-----+-----+-----+
| FIELD | TYPE | NULL | KEY | DEFAULT | EXTRA |
+-----+-----+-----+-----+-----+-----+
| C_W_ID | NUMBER(38) | NO | PRI | NULL | NULL |
| C_D_ID | NUMBER(38) | NO | PRI | NULL | NULL |
| C_ID | NUMBER(38) | NO | PRI | NULL | NULL |
| C_DISCOUNT | NUMBER(4,4) | YES | NULL | NULL | NULL |
| C_CREDIT | CHAR(2) | YES | NULL | NULL | NULL |
| C_LAST | VARCHAR2(16) | YES | NULL | NULL | NULL |
| C_FIRST | VARCHAR2(16) | YES | NULL | NULL | NULL |
| C_CREDIT_LIM | NUMBER(12,2) | YES | NULL | NULL | NULL |
| C_BALANCE | NUMBER(12,2) | YES | NULL | NULL | NULL |
| C_YTD_PAYMENT | NUMBER(12,2) | YES | NULL | NULL | NULL |
| C_PAYMENT_CNT | NUMBER(38) | YES | NULL | NULL | NULL |
| C_DELIVERY_CNT | NUMBER(38) | YES | NULL | NULL | NULL |
| C_STREET_1 | VARCHAR2(20) | YES | NULL | NULL | NULL |
```

```
| C_STREET_2 | VARCHAR2(20) | YES | NULL | NULL | NULL |
| C_CITY | VARCHAR2(20) | YES | NULL | NULL | NULL |
| C_STATE | CHAR(2) | YES | NULL | NULL | NULL |
| C_ZIP | CHAR(9) | YES | NULL | NULL | NULL |
| C_PHONE | CHAR(16) | YES | NULL | NULL | NULL |
| C_SINCE | TIMESTAMP(6) | YES | NULL | sysdate | NULL |
| C_MIDDLE | CHAR(2) | YES | NULL | NULL | NULL |
| C_DATA | VARCHAR2(500) | YES | NULL | NULL | NULL |
+-----+-----+-----+-----+-----+
21 rows in set (0.002 sec)
```

查看目标数据库 Oracle 的表 bmsql\_customer 的表结构：

```
SQL> desc bmsql_customer;
```

```
Name Null? Type
```

```
-----
C_W_ID NOT NULL NUMBER(38)
C_D_ID NOT NULL NUMBER(38)
C_ID NOT NULL NUMBER(38)
C_DISCOUNT NUMBER(4,4)
C_CREDIT CHAR(2)
C_LAST VARCHAR2(16)
C_FIRST VARCHAR2(16)
C_CREDIT_LIM NUMBER(12,2)
C_BALANCE NUMBER(12,2)
C_YTD_PAYMENT NUMBER(12,2)
C_PAYMENT_CNT NUMBER(38)
C_DELIVERY_CNT NUMBER(38)
C_STREET_1 VARCHAR2(20)
C_STREET_2 VARCHAR2(20)
C_CITY VARCHAR2(20)
```

```
C_STATE CHAR(2)
C_ZIP CHAR(9)
C_PHONE CHAR(16)
C_SINCE TIMESTAMP(6)
C_MIDDLE CHAR(2)
C_DATA VARCHAR2(500)
```

经对比是一致的。



## 24 使用 Datax 迁移 OceanBase 表数据到 Oracle 数据库

DataX 是阿里云 DataWorks 数据集成的开源版本，是阿里巴巴集团内被广泛使用的离线数据同步工具/平台。DataX 实现了包括 MySQL、Oracle、SQLserver、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、Hologres、DRDS、OceanBase 等各种异构数据源之间高效的数据同步功能。

OceanBase 数据库社区版客户，可以在 [DataX 开源网站](#) 内下载源码，自行编译。编译时，可根据需要剔除在 `pom.xml` 中不用的数据库插件，否则编译出来的包会非常大。

### 24.1 框架设计

DataX 作为离线数据同步框架，采用 "Framework + Plugin" 模式构建。将数据源读取和写入抽象为 Reader/Writer 插件，纳入到整个同步框架中。

- Reader 作为数据采集模块，负责采集数据源的数据，将数据发送给 Framework。
- Writer 作为数据写入模块，负责不断向 Framework 获取数据，并将数据写入到目的端。
- Framework 用于连接 Reader 和 Writer，作为两者的数据传输通道，并处理缓冲、流控、并发、数据转换等核心技术问题。

DataX 以任务的形式迁移数据。每个任务只处理一个表，每个任务有一个 `json` 格式的配置文件，配置文件里包含 `reader` 和 `writer` 两部分。`reader` 和 `writer` 分别对 DataX 支持的数据库读写插件，例如，将 OceanBase 表数据迁移到 Oracle 数据库操作时，需要从 OceanBase 数据库读取数据写入 Oracle 数据库，因此使用的插件为 OceanBase 数据库的 `oceanbasev10reader` 插件和 Oracle 数据库的 `oraclewriter` 插件来搭配完成。这里介绍下 `oceanbasev10reader` 和 `oraclewriter` 插件。

#### 24.1.1 oceanbasev10reader 插件

实现原理方面，`oceanbasev10reader` 通过 DataX 框架获取 Reader 生成的协议数据，生成 insert 语句。在写入数据时，如果出现主键或唯一键冲突，OceanBase 数据库的 MySQL 租户可以通过 `replace` 模式来更新表中的所有字段；OceanBase 数据库的 Oracle 租户当前只能使用 Insert 方式。

oceanbasev10reader 插件实现了从 OceanBase 数据库读取数据的功能。在底层实现上，oceanbasev10reader 插件通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并执行相应的 SQL 语句将数据从 OceanBase 数据库中 SELECT 出来。

实现原理方面，简而言之，OceanBase 数据库通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并根据用户配置的信息生成查询语句，然后发送到远程 OceanBase 数据库，远程的 OceanBase 数据库将该 SQL 执行的返回结果使用 DataX 自定义的数据类型拼装为抽象的数据集，再传递给下游 Writer 处理。

## 24.1.2 oraclewriter 插件

oraclewriter 插件实现了将数据写入到 Oracle 主库的目标表中的功能。在底层实现上，oraclewriter 插件通过 JDBC 连接远程 Oracle 数据库，并执行相应的 insert into ... 语句将数据写入 Oracle 数据库，Oracle 数据库内部会分批次提交入库。

实现原理方面，OracleWriter 通过 DataX 框架获取 Reader 生成的协议数据，根据用户的配置生成相应的 SQL 语句 insert into... 将数据插入 Oracle 主库的目标表中。

详细功能和参数说明请参考官方说明：[oraclewriter 插件](#)。

## 24.1.3 DataX 配置文件

配置文件示例：

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "sliceRecordCount": 10,
            "column": [
              {
                "type": "long",
```

```
"value": "10"
},
{
  "type": "string",
  "value": "hello, 你好, 世界-DataX"
}
]
},
"writer": {
  "name": "streamwriter",
  "parameter": {
    "encoding": "UTF-8",
    "print": true
  }
}
}
},
"setting": {
  "speed": {
    "channel": 2
  }
}
}
```

### 24.1.3.1 注意

datax 仅迁移表数据，需要提前在目标端创建好对应的表对象结构。

将 json 配置文件放到 DataX 的目录 job 下，或者自定义路径。执行方法如下：

```
$bin/datax.py job/stream2stream.json
```

输出信息：

```
<.....>
```

```
2021-08-26 11:06:09.217 [job-0] INFO JobContainer - PerfTrace not enable!
```

```
2021-08-26 11:06:09.218 [job-0] INFO StandAloneJobContainerCommunicator - Total  
20 records, 380 bytes | Speed 38B/s, 2 records/s | Error 0 records, 0 bytes | All Task  
WaitWriterTime 0.000s | All Task WaitReaderTime 0.000s | Percentage 100.00%
```

```
2021-08-26 11:06:09.223 [job-0] INFO JobContainer -
```

任务启动时刻：2021-08-26 11:05:59

任务结束时刻：2021-08-26 11:06:09

任务总计耗时：10s

任务平均流量：38B/s

记录写入速度：2rec/s

读出记录总数：20

读写失败总数：0

DataX 任务执行结束会有个简单的任务报告，包含上述输出的平均流量、写入速度和读写失败总数等。

DataX 的 `job` 的参数 `settings` 可以指定速度参数和错误记录容忍度等。

```
"setting": {  
  "speed": {  
    "channel": 10  
  },  
  "errorLimit": {  
    "record": 10,  
    "percentage": 0.1  
  }  
}
```

参数说明：

- `errorLimit` 表示报错记录数的容忍度，超出这个限制后任务就中断退出。
- `channel` 是并发数，理论上并发越大，迁移性能越好。但实际操作中也要考虑源端的读压力、网络传输性能以及目标端写入性能。

## 24.2 环境准备

下载 tar 包地址：<http://datax-opensource.oss-cn-hangzhou.aliyuncs.com/datax.tar.gz>

解压安装文件：

```
tar zxvf datax.tar.gz
cd datax
```

目录如下：

```
$tree -L 1 --filelimit 30
.
├── bin
├── conf
├── job
├── lib
├── log
├── log_perf
├── plugin
├── script
└── tmp
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。该目录下的 datax.py 为 DataX 任务的启动脚本
conf	日志文件配置目录。该目录下存放 datax 与任务无关的配置文件
lib	运行时期依赖的包。该目录存放 DataX 运行所需要的全局 jar 文件
job	该目录下有一个用于测试验证 datax 安装的任务配置文件
log	日志文件目录。该目录下存放 datax 任务运行的日志；datax 运行时，默认会将日志输出到标准输出，同时写入到 log 目录下
plugin	插件文件目录。该目录下保存 DataX 支持的各种数据源插件

## 24.3 使用 DataX 迁移 OceanBase 数据到 Oracle 数据库示例

将 OceanBase 数据迁移到 Oracle 数据库，如果源端和目标端不能同时跟 DataX 服务器网络联通，那么可以通过 CSV 文件中转。如果源端数据库和目标端数据库能同时跟 DataX 所在服务器联通，则可以使用 DataX 直接将数据从源端迁移到目标端。

示例：从 OceanBase 迁移 test.t1 表数据到 Oracle 数据库下的 test.t1。

myjob.json 配置文件如下：

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 4
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.1
      }
    }
  }
}
```

```
}  
,  
"content": [  
  {  
    "reader": {  
      "name": "oceanbasev10reader",  
      "parameter": {  
        "username": "*****",  
        "password": "*****",  
        "column": ["*"],  
        "connection": [  
          {  
            "table": ["t1"],  
            "jdbcUrl": ["jdbc:oceanbase://172.30.xxx.xxx:2883/test"]  
          }  
        ]  
      }  
    },  
    "writer": {  
      "name": "oraclewriter",  
      "parameter": {  
        "obWriteMode": "insert",  
        "column": ["*"],  
        "preSql": ["truncate table t1"],  
        "connection": [  
          {  
            "jdbcUrl": "jdbc:oracle:thin:@172.30.xxx.xxx:1521:test",  
            "table": ["TEST.T1"]  
          }  
        ]  
      }  
    }  
  ]  
}
```

```
],  
"username": "*****",  
"password": "*****",  
"writerThreadCount": 10,  
"batchSize": 1000,  
"memstoreThreshold": "0.9"  
}  
}  
}  
]  
}  
}
```

#### 参数说明

参数	描述
name	描述的是连接数据库的 reader 或 writer 对应的数据库插件的名称。其中 MySQL 的 reader 插件为：mysqlreader，OceanBase 的 writer 插件为 oceanbasev10writer。具体 reader 和 writer 的插件可以参考 datax 的文档： <a href="#">DataX 数据源指南</a> 。



jdbcUrl	<p>描述的是到连接的数据库的 JDBC 信息，使用 JSON 的数组描述，并支持一个库填写多个连接地址。您在 JSON 数组中填写一个 JDBC 连接即可。jdbcUrl 按照 MySQL 官方规范，并可以填写连接附件控制信息。具体请参见 <a href="#">MySQL 官方文档</a>。</p> <h3>24.3.3.1 注意</h3> <ul style="list-style-type: none"><li>• jdbcUrl 必须包含在 connection 配置单元中。</li><li>• OceanBase 数据库需要通过 obproxy 进行连接，端口默认 2883。</li><li>• writer 端的 jdbcUrl，无需在连接串两端加 []，reader 端的 jdbcUrl，必需在连接串两端加 []</li><li>• 必选：是</li><li>• 默认值：无</li></ul>
username	<p>数据源的用户名</p> <p>必选：是</p> <p>默认值：无</p>
password	<p>数据源指定用户名的密码</p> <p>必选：是</p> <p>默认值：无</p>
table	<p>所选取的需要同步的表。使用 JSON 的数组描述，因此支持多张表同时抽取。当配置为多张表时，用户自己需确保多张表是同一 schema 结构，MySQLReader 不予检查表是否同一逻辑表。</p> <h3>24.3.3.2 注意</h3> <p>table 必须包含在 connection 配置单元中。</p> <ul style="list-style-type: none"><li>• 必选：是</li><li>• 默认值：无</li></ul>

<p>column</p>	<p>所配置的表中需要同步的列名集合，使用 JSON 的数组描述字段信息。columns 不建议配置为 <code>['*']</code>，因为表结构发生变化时该配置也会发生变化。建议的配置方式是指定具体的列名。支持列裁剪，即列可以挑选部分列进行导出。支持列换序，即列可以不按照表 schema 信息进行导出。支持常量配置，用户需要按照 MySQL SQL 语法格式：<code>["id", "`table`", "1", "bazhen.csy", "null", "to_char(a + 1)", "2.3", "true"]</code>。</p> <h3>24.3.3.3 说明</h3> <ul style="list-style-type: none"><li>• <code>id</code> 为普通列名。</li><li>• <code>table</code> 为包含保留字的列名；</li><li>• <code>1</code> 为整形数字常量。</li><li>• <code>bazhen.csy</code> 为字符串常量。</li><li>• <code>null</code> 为空指针。</li><li>• <code>to_char(a + 1)</code> 为表达式。</li><li>• <code>2.3</code> 为浮点数。</li><li>• <code>true</code> 为布尔值。</li><li>• 必选：是</li><li>• 默认值：无</li></ul>
---------------	--

where

筛选条件，MySQLReader 根据指定的column、table、where 条件拼接 SQL，并根据这个 SQL 进行数据抽取。在实际业务场景中，往往会选择当天的数据进行同步，可以将where 条件指定为gmt\_create > \$bizdate。

### 24.3.3.4 注意

不可以将 where 条件指定为 limit 10，limit 不是 SQL 的合法 where 子句。where 条件可以有序地进行业务增量同步。如果不填写 where 语句，包括不提供 where 的 key 或者 value，DataX 均视作同步全量数据。

- 必选：否
- 默认值：无

配置 job 文件后，执行该 job。命令如下：

```
python datax.py ../job/myjob.json
```

## 24.4 更多信息

关于 DataX 的开源代码和更多信息，请参见 [DataX](#)。

## 25 使用 OMS 从 DB2 LUW 数据库迁移数据到 OceanBase 数据库 MySQL 租户

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 DB2 LUW 数据库迁移数据到 OceanBase 数据库 MySQL 租户中的背景信息。

### 25.1 背景信息

在 OMS 控制台创建从 DB2 LUW 数据库迁移数据至 OceanBase 数据库 MySQL 租户的数据迁移项目，您可以通过结构迁移、全量迁移和增量同步，迁移源端 DB2 LUW 数据库中的数据至 OceanBase 数据库 MySQL 租户。

### 25.2 相关文档

更多使用 OMS 从 DB2 LUW 数据库迁移数据到 OceanBase 数据库 MySQL 租户的操作信息，请参见 [迁移 DB2 LUW 数据库的数据至 OceanBase 数据库 MySQL 租户](#)。

## 26 使用 OMS 从 DB2 LUW 数据库迁移数据到 OceanBase 数据库 Oracle 租户

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 DB2 LUW 数据库迁移数据到 OceanBase 数据库 Oracle 租户中的背景信息。

### 26.1 背景信息

在 OMS 控制台创建从 DB2 LUW 数据库迁移数据至 OceanBase 数据库 Oracle 租户的数据迁移项目，您可以通过结构迁移、全量迁移和增量同步，迁移源端 DB2 LUW 数据库中的数据至 OceanBase 数据库 Oracle 租户。

### 26.2 相关文档

更多使用 OMS 从 DB2 LUW 数据库迁移数据到 OceanBase 数据库 Oracle 租户的操作信息，请参见 [迁移 DB2 LUW 数据库的数据至 OceanBase 数据库 Oracle 租户](#)。

## 27 使用 DBCAT 迁移 DB2 LUW 表结构到 OceanBase 数据库

DBCAT 是一款轻量级的命令行工具，可用于提供数据库之间 DDL 转换和 Schema 比对等功能。这里以 DBCAT 迁移表结构为例进行介绍。

DBCAT 安装包文件名为 `dbcat-[版本号]-SNAPSHOT.tar.gz`，下载后解压缩即可使用，可执行文件名为 `dbcat`。

### 27.0.0.1 注意

DBCAT 是 OMS 的一个组件，在社区版环境中推荐使用 OMS 导出。

## 27.1 环境准备

DBCAT 能运行在 CentOS、macOS 和 Windows 下。需要安装 JDK 1.8 以上（含）版本。可以使用 OpenJDK，安装好后配置环境变量 `JAVA_HOME`。

CentOS 安装 OpenJDK 示例：

```
$sudo yum -y install java-1.8.0-openjdk.x86_64

$which java
/usr/local/java/jdk1.8.0_261/bin/java

echo 'export JAVA_HOME=/usr/local/java/jdk1.8.0_261/' >> ~/.bash_profile
. ~/.bash_profile
```

解压安装文件：

```
tar zxvf dbcat-1.8.0-SNAPSHOT.tar.gz
cd dbcat-1.8.0-SNAPSHOT/
chmod +x bin/dbcat

$tree -L 3 --filelimit 30
```

```
.
├── bin
│   ├── dbcat
│   ├── dbcat.bat
│   └── dbcat-debug
├── conf
│   ├── dbcat.properties
│   └── logback.xml
├── docs
│   ├── README.docx
│   ├── README.md
│   └── README.txt
├── LEGAL.md
├── lib [45 entries exceeds filelimit, not opening dir]
├── LICENSE
├── meta
│   └── README
└── NOTICE

5 directories, 12 files
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。
conf	日志文件配置目录。
lib	运行时期依赖的包。
meta	离线转换场景下，导出字典表数据。
~/output	SQL 文件与报告文件，运行时生成。

## 27.2 导出 DB2 LUW 数据库的表结构

DBCAT 具有在线转换功能，该功能是指 DBCAT 能直连源端数据库，将数据库中的对象导出。当对象非常多时（如超过 1 万），导出过程可能会有点慢。

dbcat 导出命令如下：

```
bin/dbcat convert -H<host> -P<port> -u<user> -p<*****> -D<database> --schema
<schema> --from <from> --to <to> --all
```

您可运行命令 `bin/dbcat help convert` 查看更多参数信息。

必选参数：

选项	有无参数	中文描述
-H/--host	Y	数据库服务器的 IP 地址
-P/--port	Y	数据库服务器的端口
-u/--user	Y	登录数据库的用户名
-t/--tenant	Y	连接 OceanBase 集群需要提供租户名
-c/--cluster	Y	连接 OceanBase 集群需要提供集群名
-p/--password	Y	登录数据库的密码
-D/--database	Y	数据库名（源库），DB2 须区分数据库名和模式名
--service-id	Y	连接 Oracle 数据库需要提供服务 ID
--service-name	Y	连接 Oracle 数据库需要提供服务名
--as-sysdba	N	连接 Oracle 数据库 sysdba 角色
--sys-user	Y	连接 OceanBase 集群系统租户的用户名
--sys-password	Y	连接 OceanBase 集群系统租户的密码
--schema	Y	模式名（源库），非 DB2，模式名与数据名相同
--from	Y	源库的类型
--to	Y	目标库的类型
--all	N	所有的数据库对象



可选参数：

选项	有无参数	中文描述
-f/--file	Y	sql 文件的输出路径
--offline	N	使用离线模式
--target-schema	Y	模式名（目标库）
--table	Y	导出的表
--view	Y	导出的视图
--trigger	Y	导出的触发器
--synonym	Y	导出的同义词
--sequence	Y	导出的序列
--function	Y	导出的函数
--procedure	Y	导出的存储过程
--dblink	Y	导出所有的 DBLink
--type	Y	导出的 type
--type-body	Y	导出的 type body
--package	Y	导出的 package
--package-body	Y	导出的 package body
--no-quote	N	产生的 DDL 不带引号
--no-schema	N	产生的 DDL 不带模式名
--target-schema	Y	产生的 DDL 中使用指定的模式名
--exclude-type	Y	搭配 --all 使用，如：--all --exclude-type 'TABLE' 表示排除 TABLE 类型

这里以导出 DB2 LUW 11.5.x 版本下 database 为 test 的所有对象的结构，并将其迁移到 4.0.0 版本的 OceanBase 集群中的 MySQL 租户为示例。

```
bin/dbcat convert -H xxx.xxx.xxx.xxx -P 50001 -udb2inst2 -p***** --schema TESTDB -
DTESTDB --table bmsql_customer --from db2luw115 --to obmysql32x --all
```

特别说明：

- dbcat 不需要直接安装在数据库主机上，安装在可直连数据库主机的主机上即可。

- 参数中的 --from 和 --to 为源端和目的端的数据库类型，需要详细到版本号。当前 dbcat 支持的源端和目标端数据库详细如下：

源端数据库类型	目标端数据库类型
TiDB	OBMYSQL
PG	OBMYSQL
SYBASE	OBORACLE
MYSQL	OBMYSQL
ORACLE	OBORACLE
ORACLE	OBMYSQL
DB2 IBM i	OBORACLE
DB2 LUW	OBORACLE
DB2 LUW	OBMYSQL
OBMYSQL	MYSQL
OBORACLE	ORACLE

其中 OBMYSQL 为 OceanBase 数据库的 MySQL 租户，OBORACLE 为 OceanBase 数据库的 Oracle 租户。

- 当前支持的源端和目标端数据库详细的版本，详情如下。

数据库类型	数据库版本
TiDB	tidb4 tidb5
PG	pgsql10
SYBASE	sybase15
DB2 IBM i	db2ibmi71
DB2 LUW	db2luw970 db2luw1010 db2luw1050 db2luw111 db2luw115
MYSQL	mysql56 mysql57 mysql</80>

ORACLE	oracle9i oracle10g oracle11g oracle12c oracle18c oracle19c
OBTMySQL	obmysql14x obmysql21x obmysql22x obmysql200 obmysql211 obmysql2210 obmysql2230 obmysql2250 obmysql2271 ~ obmysql2277 obmysql30x obmysql31x obmysql32x obmysql322 obmysql40
OBTOracle	oboracle2220 oboracle2230 oboracle2250 oboracle2270 ~ oboracle2277 oboracle21x oboracle22x oboracle30x oboracle31x oboracle32x oboracle322 oboracle40

运行后的输出文件在用户 `home` 目录的 `output` 下。

```
$tree ~/output/dbcat-20xx-xx-xx-164533/
/home/qing.meiq/output/dbcat-20xx-xx-xx-164533/
|____ tpccdb
|  |____ TABLE-schema.sql
|____ tpccdb-conversion.html

1 directory, 2 files
```

## 27.3 导入 OceanBase 数据库

使用 DBCAT 导出的文件格式为 SQL 文件，这里可以通过 ODC 的导入功能批量将表结构导入 OceanBase 数据库，更多信息请参考 [批量导出与导入](#)。

也可以使用 `source` 命令将 SQL 文件数据导入 OceanBase 数据库，示例如下：

```
obclient [test]> source TABLE-schema.sql
Query OK, 0 rows affected (0.044 sec)
```

### 27.3.0.1 注意

如果 sql 文件不在当前目录下，则需要使用绝对地址。

## 27.4 导数结果验证

示例：查看一个表结构在 DB2 LUW 里的书写方式 和 OceanBase 数据库里的表结构。

查看源数据库 DB2 LUW 的表 `bmsql_customer` 的建表 SQL：

```
db2 => describe table bmsql_customer
```

Data type Column

Column name schema Data type name Length Scale Nulls

```
-----
C_W_ID SYSIBM BIGINT 8 0 No
C_D_ID SYSIBM BIGINT 8 0 No
C_ID SYSIBM BIGINT 8 0 No
C_DISCOUNT SYSIBM DECIMAL 4 4 Yes
C_CREDIT SYSIBM CHARACTER 2 0 Yes
C_LAST SYSIBM VARCHAR 16 0 Yes
C_FIRST SYSIBM VARCHAR 16 0 Yes
C_CREDIT_LIM SYSIBM DECIMAL 12 2 Yes
C_BALANCE SYSIBM DECIMAL 12 2 Yes
C_YTD_PAYMENT SYSIBM DECIMAL 12 2 Yes
C_PAYMENT_CNT SYSIBM INTEGER 4 0 Yes
```

```

C_DELIVERY_CNT SYSIBM INTEGER 4 0 Yes
C_STREET_1 SYSIBM VARCHAR 20 0 Yes
C_STREET_2 SYSIBM VARCHAR 20 0 Yes
C_CITY SYSIBM VARCHAR 20 0 Yes
C_STATE SYSIBM CHARACTER 2 0 Yes
C_ZIP SYSIBM CHARACTER 9 0 Yes
C_PHONE SYSIBM CHARACTER 16 0 Yes
C_SINCE SYSIBM TIMESTAMP 10 6 Yes
C_MIDDLE SYSIBM CHARACTER 2 0 Yes
C_DATA SYSIBM VARCHAR 500 0 Yes

```

21 record(s) selected.

查看目标库数据库 OceanBase 的表 bmsql\_customer 的表结构：

```

obclient [test]> desc bmsql_customer;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| c_w_id | bigint(20) | NO | PRI | NULL | |
| c_d_id | bigint(20) | NO | PRI | NULL | |
| c_id | bigint(20) | NO | PRI | NULL | |
| c_discount | decimal(4,4) | YES | | NULL | |
| c_credit | char(2) | YES | | NULL | |
| c_last | varchar(16) | YES | | NULL | |
| c_first | varchar(16) | YES | | NULL | |
| c_credit_lim | decimal(12,2) | YES | | NULL | |
| c_balance | decimal(12,2) | YES | | NULL | |
| c_ytd_payment | decimal(12,2) | YES | | NULL | |
| c_payment_cnt | int(11) | YES | | NULL | |
| c_delivery_cnt | int(11) | YES | | NULL | |

```

```
| c_street_1 | varchar(20) | YES || NULL ||  
| c_street_2 | varchar(20) | YES || NULL ||  
| c_city | varchar(20) | YES || NULL ||  
| c_state | char(2) | YES || NULL ||  
| c_zip | char(9) | YES || NULL ||  
| c_phone | char(16) | YES || NULL ||  
| c_since | timestamp | YES || NULL ||  
| c_middle | char(2) | YES || NULL ||  
| c_data | varchar(500) | YES || NULL ||  
+-----+-----+-----+-----+-----+-----+  
21 rows in set (0.007 sec)
```

经对比发现表字段的字符类型以及长度会有一定的转变，更多关于表字段转换的信息请参见[创建 DB2 LUW 至 OceanBase 数据库 Oracle 租户的数据迁移项目](#) 中的数据类型映射。

## 28 使用 OMS 从 OceanBase 数据库 MySQL 租户迁移数据到 DB2 LUW 数据库

本文将简单介绍使用 OceanBase 迁移服务 (OceanBase Migration Service, OMS) 从 OceanBase 数据库 MySQL 租户迁移数据到 DB2 LUW 数据库中的背景信息。

### 28.1 背景信息

在 OMS 控制台创建从 OceanBase 数据库 MySQL 租户迁移数据至 DB2 LUW 数据库的数据迁移项目，您可以通过结构迁移、全量迁移和增量同步，迁移源端 OceanBase 数据库 MySQL 租户中的数据至 DB2 LUW 数据库。

### 28.2 相关文档

更多使用 OMS 从 OceanBase 数据库 MySQL 租户迁移数据到 DB2 LUW 数据库的操作信息，请参见 [迁移 OceanBase 数据库 MySQL 租户的数据至 DB2 LUW 数据库](#)。

## 29 使用 OMS 从 OceanBase 数据库 Oracle 租户迁移数据到 DB2 LUW 数据库

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 OceanBase 数据库 Oracle 租户迁移数据到 DB2 LUW 数据库中的背景信息。

### 29.1 背景信息

在 OMS 控制台创建从 OceanBase 数据库 Oracle 租户迁移数据至 DB2 LUW 数据库的数据迁移项目，您可以通过结构迁移、全量迁移和增量同步，迁移源端 OceanBase 数据库 Oracle 租户中的数据至 DB2 LUW 数据库。

### 29.2 相关文档

更多使用 OMS 从 OceanBase 数据库 Oracle 租户迁移数据到 DB2 LUW 数据库的操作信息，请参见 [迁移 OceanBase 数据库 Oracle 租户的数据至 DB2 LUW 数据库](#)。



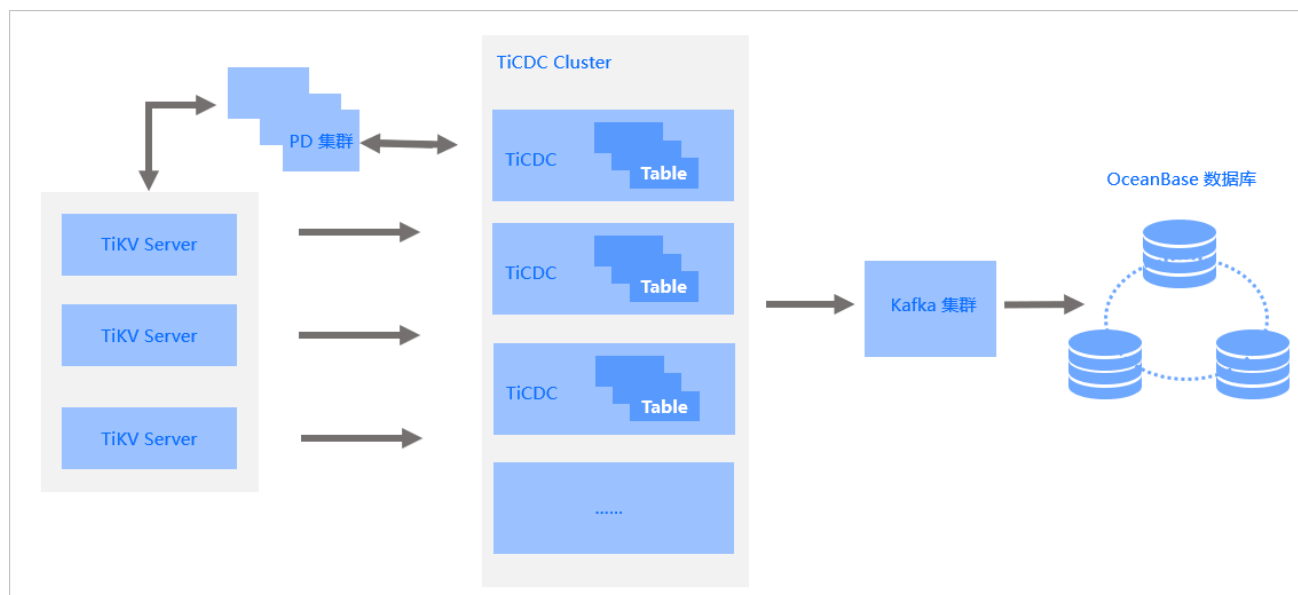
## 30 使用 OMS 从 TiDB 数据库迁移数据到 OceanBase 数据库 MySQL 租户

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 TiDB 数据库迁移数据到 OceanBase 数据库 MySQL 租户中的背景信息。

### 30.1 背景信息

OMS 支持创建源端 TiDB 数据库至目标端 OceanBase 数据库 MySQL 租户的数据迁移项目。您可以通过结构迁移、全量迁移和增量同步等，无缝迁移源端数据库的存量业务数据和增量数据至 OceanBase 数据库 MySQL 租户。

TiDB 数据库支持在线事务处理和在线分析处理（Hybrid Transactional and Analytical Processing, HTAP），是一款融合型分布式数据库产品。您需要部署 TiCDC 集群和 Kafka 集群来实现 TiDB 数据库至 OceanBase 数据库 MySQL 租户的增量数据同步。



TiCDC 是 TiDB 数据库的增量数据同步工具，通过 PD 集群（TiDB 集群的调度模块，通常由 3 个 PD 节点构成）来实现高可用。TiKV Server 是 TiDB 集群中的 TiKV 节点，它会以变更日志的方式主动发送变更数据至 TiCDC 集群。TiCDC 工具会通过多个 TiCDC 进程获取 TiKV 节点的数据并进行处理后，同步数据至 Kafka 集群。Kafka 集群会保存 TiCDC 工具转换的 TiDB 数据库的增量日志信息，以便 OMS 在执行增量数据同步时，从 Kafka 集群中获取相应数据并实时迁移数据至 OceanBase 数据库 MySQL 租户。如果您在新建 TiDB 数据源时，未绑定 Kafka 数据源，将无法进行增量同步。

### 30.2 相关文档

更多使用 OMS 从 TiDB 数据库迁移数据到 OceanBase 数据库 MySQL 租户的操作信息，请参见 [迁移 TiDB 数据库的数据至 OceanBase 数据库 MySQL 租户](#)。

## 31 使用 OMS 从 PostgreSQL 数据库迁移数据到 OceanBase 数据库 MySQL 租户

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 PostgreSQL 数据库迁移数据到 OceanBase 数据库 MySQL 租户中的背景信息。

### 31.1 背景信息

在 OMS 控制台创建从 PostgreSQL 数据库迁移数据至 OceanBase 数据库 MySQL 租户的数据迁移项目，您可以通过结构迁移、全量迁移和增量同步，无缝迁移源端数据库中的存量业务数据和增量数据至 OceanBase 数据库 MySQL 租户。

PostgreSQL 数据库支持单主库、单备库和主备库等模式。迁移 PostgreSQL 数据库的数据至 OceanBase 数据库 MySQL 租户时，不同类型的数据源支持的操作也不同。

类型	支持的操作
单主库	结构迁移 + 全量迁移 + 增量同步 + 全量校验 + 反向增量
单备库	结构迁移 + 全量迁移 + 全量校验
主备库	主库：支持增量同步 + 反向增量 备库：支持结构迁移 + 全量迁移 + 全量校验

### 31.2 相关文档

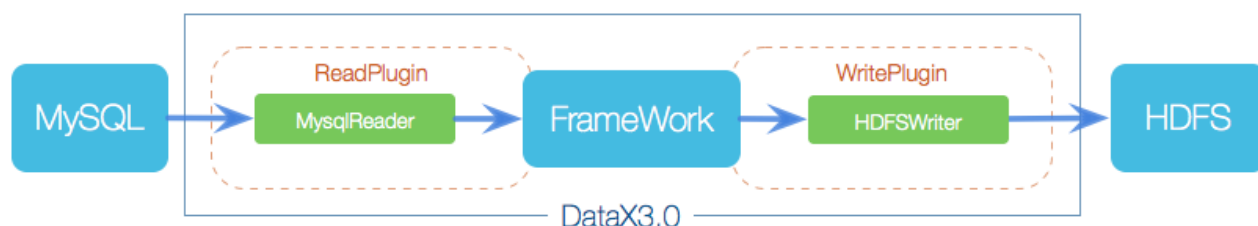
更多使用 OMS 从 PostgreSQL 数据库迁移数据到 OceanBase 数据库 MySQL 租户的操作信息，请参见 [迁移 PostgreSQL 数据库的数据至 OceanBase 数据库 MySQL 租户](#)。

## 32 使用 DataX 迁移 CSV 文件到 OceanBase 数据库

DataX 是阿里云 DataWorks 数据集成的开源版本，是阿里巴巴集团内被广泛使用的离线数据同步工具/平台。DataX 实现了包括 MySQL、Oracle、SQLserver、Postgre、HDFS、Hive、ADS、HBase、TableStore(OTS)、MaxCompute(ODPS)、Hologres、DRDS、OceanBase 等各种异构数据源之间高效的数据同步功能。

OceanBase 数据库社区版客户，可以在 [DataX 开源网站](#) 内下载源码，自行编译。编译时，可根据需要剔除在 `pom.xml` 中不用的数据库插件，否则编译出来的包会非常大。

### 32.1 框架设计



DataX 作为离线数据同步框架，采用 "Framework + Plugin" 模式构建。将数据源读取和写入抽象为 Reader/Writer 插件，纳入到整个同步框架中。

- Reader 作为数据采集模块，负责采集数据源的数据，将数据发送给 Framework。
- Writer 作为数据写入模块，负责不断向 Framework 获取数据，并将数据写入到目的端。
- Framework 用于连接 Reader 和 Writer，作为两者的数据传输通道，并处理缓冲、流控、并发、数据转换等核心技术问题。

DataX 以任务的形式迁移数据。每个任务只处理一个表，每个任务有一个 `json` 格式的配置文件，配置文件里包含 `reader` 和 `writer` 两部分。`reader` 和 `writer` 分别对 DataX 支持的数据库读写插件，例如，将 MySQL 表数据迁移到 OceanBase 数据库操作时，需要从 MySQL 读取数据写入 OceanBase 数据库，因此使用的插件为 MySQL 数据库的 `txtfilereader` 插件和 OceanBase 数据库的 `oceanbasev10writer` 插件来搭配完成。这里介绍下 `txtfilereader` 和 `oceanbasev10writer` 插件。

#### 32.1.1 txtfilereader 插件

`txtfilereader` 插件实现了从本地文件系统数据存储上读取数据的功能。在底层实现上，`txtfilereader` 获取本地文件数据，并转换为 DataX 传输协议传递给 Writer。

详细功能和参数说明请参考官方说明：[txtfilereader 插件](#)。

## 32.1.2 oceanbasev10writer 插件

`oceanbasev10writer` 插件实现了将数据写入到 OceanBase 数据库的目标表中的功能。在底层实现上，`oceanbasev10writer` 通过 Java 客户端（底层 MySQL JDBC 或 OceanBase Client）以 obproxy 远程连接 OceanBase 数据库，并执行相应的 insert sql 语句来将数据写入到 OceanBase 数据库，OceanBase 数据库内部会分批次提交入库。

实现原理方面，`oceanbasev10writer` 通过 DataX 框架获取 Reader 生成的协议数据，生成 insert 语句。在写入数据时，如果出现主键或唯一键冲突，OceanBase 数据库的 MySQL 租户可以通过 `replace` 模式来更新表中的所有字段；OceanBase 数据库的 Oracle 租户当前只能使用 Insert 方式。出于性能考虑，写入采用 batch 方式批量写，当行数累计到预定阈值时，才发起写入请求。

## 32.1.3 DataX 配置文件

配置文件示例：

```
{
  "job": {
    "content": [
      {
        "reader": {
          "name": "streamreader",
          "parameter": {
            "sliceRecordCount": 10,
            "column": [
              {
                "type": "long",
                "value": "10"
              },
              {
                "type": "string",
```

```
"value": "hello, 你好, 世界-DataX"
}
]
}
},
"writer": {
  "name": "streamwriter",
  "parameter": {
    "encoding": "UTF-8",
    "print": true
  }
}
}
},
"setting": {
  "speed": {
    "channel": 2
  }
}
}
}
```

### 32.1.3.1 注意

datax 仅迁移表数据，需要提前在目标端创建好对应的表对象结构。

将 json 配置文件放到 DataX 的目录 job 下，或者自定义路径。执行方法如下：

```
$bin/datax.py job/stream2stream.json
```

输出信息：

```
<.....>
```

```
2021-08-26 11:06:09.217 [job-0] INFO JobContainer - PerfTrace not enable!
```

```
2021-08-26 11:06:09.218 [job-0] INFO StandAloneJobContainerCommunicator - Total  
20 records, 380 bytes | Speed 38B/s, 2 records/s | Error 0 records, 0 bytes | All Task  
WaitWriterTime 0.000s | All Task WaitReaderTime 0.000s | Percentage 100.00%
```

```
2021-08-26 11:06:09.223 [job-0] INFO JobContainer -
```

任务启动时刻：2021-08-26 11:05:59

任务结束时刻：2021-08-26 11:06:09

任务总计耗时：10s

任务平均流量：38B/s

记录写入速度：2rec/s

读出记录总数：20

读写失败总数：0

DataX 任务执行结束会有个简单的任务报告，包含上述输出的平均流量、写入速度和读写失败总数等。

DataX 的 `job` 的参数 `settings` 可以指定速度参数和错误记录容忍度等。

```
"setting": {  
  "speed": {  
    "channel": 10  
  },  
  "errorLimit": {  
    "record": 10,  
    "percentage": 0.1  
  }  
}
```

参数说明：

- `errorLimit` 表示报错记录数的容忍度，超出这个限制后任务就中断退出。

- `channel` 是并发数，理论上并发越大，迁移性能越好。但实际操作中也要考虑源端的读压力、网络传输性能以及目标端写入性能。

## 32.2 环境准备

下载 tar 包地址：<http://datax-opensource.oss-cn-hangzhou.aliyuncs.com/datax.tar.gz>

解压安装文件：

```
tar zxvf datax.tar.gz
cd datax
```

目录如下：

```
$tree -L 1 --filelimit 30
.
├── bin
├── conf
├── job
├── lib
├── log
├── log_perf
├── plugin
├── script
└── tmp
```

安装文件中有以下几个目录需要了解：

目录名	说明
bin	可执行文件目录。该目录下的 datax.py 为 DataX 任务的启动脚本
conf	日志文件配置目录。该目录下存放 datax 与任务无关的配置文件
lib	运行时期依赖的包。该目录存放 DataX 运行所需要的全局 jar 文件
job	该目录下有一个用于测试验证 datax 安装的任务配置文件
log	日志文件目录。该目录下存放 datax 任务运行的日志；datax 运行时，默认会将日志输出到标准输出，同时写入到 log 目录下
plugin	插件文件目录。该目录下保存 DataX 支持的各种数据源插件

## 32.3 使用 DataX 迁移 CSV 文件到 OceanBase 示例

示例：将源端导出的 CSV 文件复制到目标端的 DataX 服务器上，然后导入到目标端 OceanBase 数据库中。

myjob.json 配置文件如下：

```
{
  "job": {
    "setting": {
      "speed": {
        "channel": 4
      },
      "errorLimit": {
        "record": 0,
        "percentage": 0.1
      }
    }
  },
}
```



```
"content": [  
  {  
    "reader": {  
      "name": "txtfilereader",  
      "parameter": {  
        "path": ["/tmp/tpcc/bmsql_oorder"],  
        "fileName": "bmsql_oorder",  
        "encoding": "UTF-8",  
        "column": ["*"],  
        "dateFormat": "yyyy-MM-dd hh:mm:ss" ,  
        "nullFormat": "\\N" ,  
        "fieldDelimiter": ","  
      }  
    },  
    "writer": {  
      "name": "oceanbasev10writer",  
      "parameter": {  
        "obWriteMode": "insert",  
        "column": ["*"],  
        "preSql": [  
          "truncate table bmsql_oorder"  
        ],  
        "connection": [  
          {  
            "jdbcUrl": "jdbc:oceanbase://127.0.0.1:2883/tpcc?",  
            "table": [  
              "bmsql_oorder"  
            ]  
          }  
        ]  
      }  
    }  
  ]  
}
```

```
],  
"username": "tpcc",  
"password": "*****",  
"writerThreadCount": 10,  
"batchSize": 1000,  
"memstoreThreshold": "0.9"  
}  
}  
}  
]  
}  
}
```

#### 参数说明

参数	描述
name	描述的是连接数据库的 reader 或 writer 对应的数据库插件的名称。其中 MySQL 的 reader 插件为：mysqlreader，OceanBase 的 writer 插件为 oceanbasev10writer。具体 reader 和 writer 的插件可以参考 datax 的文档： <a href="#">DataX 数据源指南</a> 。

jdbcUrl	<p>描述的是到连接的数据库的 JDBC 信息，使用 JSON 的数组描述，并支持一个库填写多个连接地址。您在 JSON 数组中填写一个 JDBC 连接即可。jdbcUrl 按照 MySQL 官方规范，并可以填写连接附件控制信息。具体请参见 <a href="#">MySQL 官方文档</a>。</p> <h3>32.3.3.1 注意</h3> <ul style="list-style-type: none"><li>• jdbcUrl 必须包含在 connection 配置单元中。</li><li>• OceanBase 数据库需要通过 obproxy 进行连接，端口默认 2883。</li><li>• writer 端的 jdbcUrl，无需在连接串两端加 []，reader 端的 jdbcUrl，必需在连接串两端加 []</li><li>• 必选：是</li><li>• 默认值：无</li></ul>
username	<p>数据源的用户名</p> <p>必选：是</p> <p>默认值：无</p>
password	<p>数据源指定用户名的密码</p> <p>必选：是</p> <p>默认值：无</p>
table	<p>所选取的需要同步的表。使用 JSON 的数组描述，因此支持多张表同时抽取。当配置为多张表时，用户自己需确保多张表是同一 schema 结构，MySQLReader 不予检查表是否同一逻辑表。</p> <h3>32.3.3.2 注意</h3> <p>table 必须包含在 connection 配置单元中。</p> <ul style="list-style-type: none"><li>• 必选：是</li><li>• 默认值：无</li></ul>

<p>column</p>	<p>所配置的表中需要同步的列名集合，使用 JSON 的数组描述字段信息。columns 不建议配置为 <code>['*']</code>，因为表结构发生变化时该配置也会发生变化。建议的配置方式是指定具体的列名。支持列裁剪，即列可以挑选部分列进行导出。支持列换序，即列可以不按照表 schema 信息进行导出。支持常量配置，用户需要按照 MySQL SQL 语法格式：<code>["id", "`table`", "1", "bazhen.csy", "null", "to_char(a + 1)", "2.3", "true"]</code>。</p> <h3>32.3.3.3 说明</h3> <ul style="list-style-type: none"><li>• <code>id</code> 为普通列名。</li><li>• <code>table</code> 为包含保留字的列名；</li><li>• <code>1</code> 为整形数字常量。</li><li>• <code>bazhen.csy</code> 为字符串常量。</li><li>• <code>null</code> 为空指针。</li><li>• <code>to_char(a + 1)</code> 为表达式。</li><li>• <code>2.3</code> 为浮点数。</li><li>• <code>true</code> 为布尔值。</li></ul> <p>• 必选：是</p> <p>• 默认值：无</p>
---------------	--

where

筛选条件，MySQLReader 根据指定的column、table、where 条件拼接 SQL，并根据这个 SQL 进行数据抽取。在实际业务场景中，往往会选择当天的数据进行同步，可以将where 条件指定为gmt\_create > \$bizdate。

### 32.3.3.4 注意

不可以将 where 条件指定为 limit 10，limit 不是 SQL 的合法 where 子句。where 条件可以有序地进行业务增量同步。如果不填写 where 语句，包括不提供 where 的 key 或者 value，DataX 均视作同步全量数据。

- 必选：否
- 默认值：无

配置 job 文件后，执行该 job。命令如下：

```
python datax.py ../job/myjob.json
```

## 32.4 更多信息

关于 DataX 的开源代码和更多信息，请参见 [DataX](#)。

## 33 使用 LOAD DATA 语句导入数据

OceanBase 数据库支持通过 `LOAD DATA` 命令加载外部文件的数据到数据库表中。

### 33.1 使用限制

带有触发器（Trigger）的表禁止使用 `LOAD DATA` 语句。

### 33.2 注意事项

OceanBase 数据库通过并行处理技术优化 `LOAD DATA` 的数据导入速率。该操作将数据分成多个子任务并行执行，每个子任务都视为独立的事务，执行顺序是不固定的。因此：

- 不能保证数据导入过程的全局原子性。
- 对于没有主键的表，数据写入顺序可能与原文件中的顺序不同。

### 33.3 使用场景

`LOAD DATA` 目前可以对 CSV 格式的文本文件进行导入，整个导入的过程如下：

#### 33.3.0.1 说明

OceanBase 数据库支持加载位于 OSS、服务器端（OBServer 节点）和客户端（本地）的数据文件。

##### 1. 解析文件。

OceanBase 会根据用户输入的文件名，读取文件中的数据，并且根据用户输入的并行度来决定并行或者串行解析输入文件中的数据。

##### 2. 分发数据。

由于 OceanBase 是分布式数据库系统，各个分区的数据可能分布在各个不同的 OBServer 节点上，`LOAD DATA` 会对解析出来的数据进行计算，决定数据需要被发送到哪个 OBServer 节点。

##### 3. 插入数据。

当目标 OBServer 节点收到了发送过来的数据之后，在本地执行 `INSERT` 操作把数据插入到对应的分区当中。

### 33.4 LOAD DATA 语法

有关 `LOAD DATA` 语法的详细信息，请参见 [LOAD DATA \(MySQL 模式\)](#) 和 [LOAD DATA \(Oracle 模式\)](#)。

## 33.4.1 获取 LOAD DATA 执行权限

在执行 `LOAD DATA` 语句之前，您需要先获得相应的权限。以下是授予执行权限的操作步骤：

1. 对用户进行授权 `FILE` 权限。

示例如下：

要为用户授予 `FILE` 权限，可以使用以下命令：

```
GRANT FILE ON *.* TO user_name;
```

其中，`user_name` 是需要执行 `LOAD DATA` 命令的用户。

2. 授予其他必要权限。

- MySQL 模式需要拥有对应表的 `INSERT` 权限。

示例如下：

要为用户授予 `INSERT` 权限，可以使用以下命令格式：

```
GRANT INSERT ON database_name.tbl_name TO user_name;
```

其中，`database_name` 是数据库名称，`tbl_name` 是表名，`user_name` 是需要执行 `LOAD DATA` 命令的用户。

- Oracle 模式需要拥有 `CREATE SESSION` 权限。

示例如下：

要为用户授予 `CREATE SESSION` 权限，可以使用以下命令格式：

```
GRANT CREATE SESSION TO user_name;
```

其中，`user_name` 是要授予权限的用户名。

## 33.5 示例

### 33.5.1.1 说明

OceanBase 数据库支持 Oracle 模式和 MySQL 模式两种模式。以下示例是在 MySQL 模式下，展示如何使用 `LOAD DATA` 语句。

## 33.5.2 从服务器端文件导入数据

1. 登录到要连接 OBDServer 节点所在的机器。

示例如下：

```
ssh admin@10.10.10.1
```

2. 在 `/home/admin/test_data` 目录下创建测试数据。

示例如下：

执行以下命令，编写一个名为 `student.sql` 的脚本。

```
vi student.sql
```

3. 进入编辑模式并添加测试数据。

示例如下：

按下 `i` 键或者 `Insert` 键进入 `vi` 编辑器的插入模式，在插入模式下添加以下内容。

```
1,"lin",98
2,"hei",90
3,"ali",95
```

4. 设置导入的文件路径。

### 33.5.2.2 注意

由于安全原因，设置系统变量 `secure_file_priv` 时，只能通过本地 Socket 连接数据库执行修改该全局变量的 SQL 语句。更多信息，请参见 [secure\\_file\\_priv](#)。

示例如下：

- a. 登录到要连接 OBDServer 节点所在的机器。

```
ssh admin@10.10.10.1
```

- b. 执行以下命令，通过本地 Unix Socket 连接方式连接租户 `mysql001`。

示例如下：



```
obclient -S /home/admin/oceanbase/run/sql.sock -uroot@mysql001 -p*****
```

c. 设置为文件所在目录为 `/`，表示没有限制，任意路径均可访问。

```
SET GLOBAL SECURE_FILE_PRIV = "/";
```

5. 重新连接数据库。

**示例如下：**

```
obclient -h127.0.0.1 -P2881 -utest_user001@mysql001 -p***** -A
```

6. 创建测试表。

**示例如下：**

执行下面 SQL 语句，创建测试表 `student`。

```
obclient [test]> CREATE TABLE student (id INT, name VARCHAR(50), score INT);
```

7. 使用 `LOAD DATA` 语句导入数据。

**示例如下：**

使用以下 `LOAD DATA` 语句将数据从文件加载到数据库表中，其中：

- 指定要加载的文件的完整路径和文件名为 `/home/admin/test_data/student.sql`。
- 指定要加载数据的目标表名为 `student`。
- 指定数据文件中的字段分隔符为逗号。
- 指定数据文件中的字段（字符类型）将使用双引号封闭。
- 指定数据文件中的行将使用换行符作为结束符。
- 指定要加载的数据文件中的列与目标表中的列的映射关系。数据文件中的第一列将映射到目标表的 `id` 列，第二列映射到 `name` 列，第三列映射到 `score` 列。

```
obclient [test]> LOAD DATA INFILE '/home/admin/test_data/student.sql'
INTO TABLE student
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
(id,name,score);
```

返回结果如下：

```
Query OK, 3 rows affected
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

8. 查看表信息。

示例如下：

```
obclient [test]> SELECT * FROM student;
```

返回结果如下：

```
+-----+-----+-----+
| id | name | score |
+-----+-----+-----+
| 1 | lin | 98 |
| 2 | hei | 90 |
| 3 | ali | 95 |
+-----+-----+-----+
3 rows in set
```

### 33.5.3 从客户端（本地）文件导入数据

使用以下语句，从本地文件导入数据至 OceanBase 数据库表中。

1. 在本地 `/home/admin/test_data` 目录下创建测试数据。

示例如下：

执行以下命令，编写一个名为 `test_tbl1.csv` 的脚本。

```
vi test_tbl1.csv
```

2. 进入编辑模式并添加测试数据。

示例如下：

按下 `i` 键或者 `Insert` 键进入 `vi` 编辑器的插入模式，在插入模式下添加以下内容。

```
1,11
2,22
3,33
```

### 3. 启动客户端。

#### 示例如下：

执行以下语句，使用 OBClient 命令行工具连接到 OceanBase 数据库。通过添加 `--local-infile` 参数启用从本地文件加载数据的功能。

```
obclient --local-infile -hxxx.xxx.xxx.xxx -P2881 -uroot@mysql001 -p***** -Dtest
```

## 33.5.3.3 注意

为了使用 `LOAD DATA LOCAL INFILE` 功能，请使用 V2.2.4 或之后的版本的 OBClient 客户端。如果您没有要求版本的 OBClient 客户端，也可以使用 MySQL 客户端来连接数据库。

### 4. 创建测试表。

#### 示例如下：

```
CREATE TABLE test_tbl1(col1 INT,col2 INT);
```

### 5. 在客户端中，执行 `LOAD DATA LOCAL INFILE` 语句来加载本地数据文件。

#### 示例如下：

```
obclient [test]> LOAD DATA LOCAL INFILE '/home/admin/test_data/test_tbl1.csv'
INTO TABLE test_tbl1 FIELDS TERMINATED BY ',';
```

返回结果如下：

```
Query OK, 3 rows affected
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

### 6. 查看表信息。

#### 示例如下：

```
obclient [test]> SELECT * FROM test_tbl1;
```

返回结果如下：

```
+-----+-----+
| col1 | col2 |
+-----+-----+
| 1 | 11 |
| 2 | 22 |
| 3 | 33 |
+-----+-----+
3 rows in set
```

## 33.6 异常处理

### 33.6.4 日志文件

如果导入的过程中出现了错误，出现错误的 `INSERT` 语句会被回滚，并且 `LOAD DATA` 语句会在 observer 进程安装路径的 `log` 子目录下产生名称为 `obloaddata.log.<XXXXXX>` 的日志文件。以下是一个日志文件的内容示例，日志中会包含 `LOAD DATA` 产生的任务的基本信息，包含租户名、输入文件名、目标表名、并行度、使用的 `LOAD DATA` 命令，并且以行为单位给出具体错误的信息。

```
Tenant name: mysql
File name: /home/admin/a.csv
Into table: `test`.`t`
Parallel: 1
Batch size: 1000
SQL trace: YD7A20BA65670-0005AADAAA3C****
Start time: 2020-07-29 21:08:13.073741
Load query:
load data infile '/home/admin/test.csv' into table t fields terminated by ',' lines
terminated by '\n'
Row ErrCode ErrMsg
```

1 1062 Duplicated primary key

2 1062 Duplicated primary key

## 33.7 相关文档

- 更多有关使用 `LOAD DATA` 语句旁路导入数据的信息，请参见 [旁路导入](#)。
- 更多有关连接数据库的详细信息，请参见 [连接方式概述](#)。
- 更多有关删除表的信息，请参见 [删除表](#)。

## 34 从 SQL 文件导入数据到 OceanBase 数据库

将 SQL 格式的数据文件导入到 OceanBase 数据库，可以采用命令行界面导入和工具导入两种方式。OceanBase 数据库支持多种工具导入 SQL 格式的数据文件，如：MySQLDumper、obloader 和 OceanBase 开发者工具（以下简称：ODC）等图形化界面工具。

本文将主要介绍使用命令行界面、obloader 和 ODC 从 SQL 文件导入数据到 OceanBase 数据库。

### 34.1 配置 SQL 文件信息

导入到 OceanBase 数据库 SQL 文件中的 SQL 语法必须符合 OceanBase 数据库的要求。

有关 OceanBase 数据库 SQL 语法的详细信息，请参见 [SQL 语法（MySQL 模式）](#) 和 [SQL 语法（Oracle 模式）](#)。

示例中使用的 SQL 文件信息如下：

```
[xxx@xxx /home/admin/test_sql]# cat test_tbl1.sql
DROP TABLE IF EXISTS test_data.test_tbl1;

CREATE TABLE IF NOT EXISTS test_data.test_tbl1(col1 INT,col2 VARCHAR(50),PRIMARY
KEY (col1));

INSERT INTO test_data.test_tbl1 VALUES(1,'test1');
INSERT INTO test_data.test_tbl1 VALUES(2,'test2');
INSERT INTO test_data.test_tbl1 VALUES(3,'test3');
INSERT INTO test_data.test_tbl1 VALUES(4,'test4');
INSERT INTO test_data.test_tbl1 VALUES(5,'test5');
```

### 34.2 使用命令行界面导入数据

1. 使用 SQL 文件所在的机器登录 OceanBase 数据库。

```
[xxx@xxx /home/admin]# obclient -hxxx.xxx.xxx.1 -P2881 -uroot@mysql001 -p -A
Enter password:
```

```
Welcome to the OceanBase. Commands end with ; or \g.  
Your OceanBase connection id is 3221709595  
Server version: OceanBase 4.0.0.0 (r100000302022111120-  
7cef93737c5cd03331b5f29130c6e80ac950d33b) (Built Nov 11 2022 20:38:33)  
  
Copyright (c) 2000, 2018, OceanBase and/or its affiliates. All rights reserved.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
obclient [(none)]>
```

## 2. 创建 Schema 库 test\_data 。

```
obclient [(none)]> CREATE DATABASE test_data;  
Query OK, 1 row affected
```

## 3. 使用 SOURCE 命令导入 SQL 文件数据。

```
obclient [(none)]> SOURCE /home/admin/test_tbl1.sql  
Query OK, 0 rows affected, 1 warning  
  
Query OK, 0 rows affected  
  
Query OK, 1 row affected  
  
Query OK, 1 row affected  
  
Query OK, 1 row affected  
  
Query OK, 1 row affected
```

```
Query OK, 1 row affected
```

#### 4. 查看导入数据信息。

```
obclient [test_data]> use test_data;
Database changed
obclient [test_data]> SHOW TABLES;
+-----+
| Tables_in_test_data |
+-----+
| test_tbl1 |
+-----+
1 row in set

obclient [test_data]> SELECT * FROM test_tbl1;
+-----+-----+
| col1 | col2 |
+-----+-----+
| 1 | test1 |
| 2 | test2 |
| 3 | test3 |
| 4 | test4 |
| 5 | test5 |
+-----+-----+
5 rows in set
```

## 34.3 使用 ODC 导入数据

ODC 对导入上传的文件，最大支持导出 2GB 的数据（压缩后），超出部分无法上传。如需导入大量数据，请使用导数工具 obloader。

下面内容将介绍使用 Web 版 ODC 导入数据。



## 1. 准备 ODC 环境。

ODC 分为 **Web 版** 和 **客户端版**，详细信息请参见 [Web 版部署概述](#) 和 [客户端版安装 ODC](#)。

## 2. 连接 OceanBase 数据库。

连接 OceanBase 数据库的详细信息，请参见 [创建个人连接](#)。

×

新建个人连接

数据库类型

物理库 ?

逻辑库 ?

所属区域 ?

独立部署/专有云

公有云

连接名称

请输入连接名称，30字以内

设置标签

智能解析 (可选)

粘贴连接串信息，自动识别连接信息，如：obclient -h 10.210.2.51 -P2883 -uroot@tenantname#clustername -p'obpasswd'

智能解析

连接模式 ?

Oracle

MySQL

连接地址

主机 IP

端口

集群名 (可选)

租户名

请输入主机地址

请输入端口

请输入集群名

请输入租户名

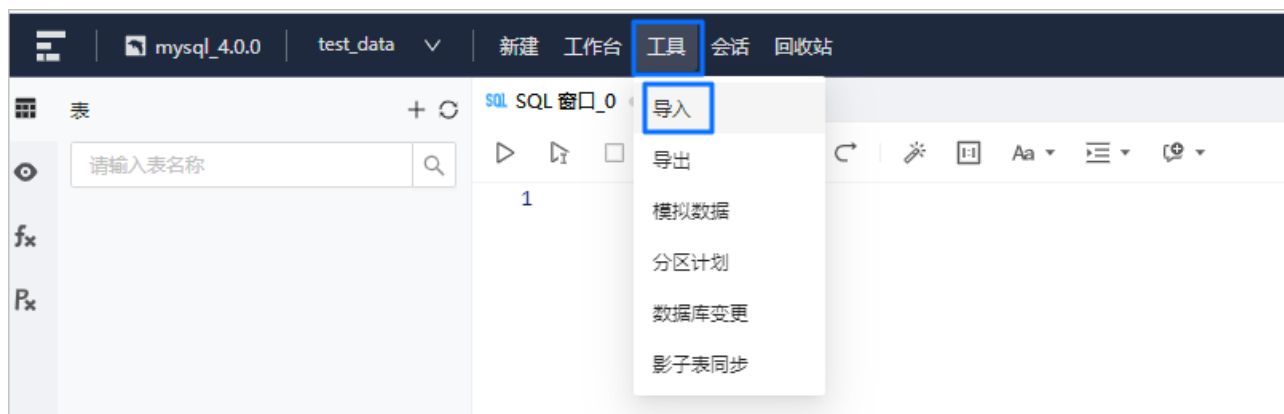
数据库账号



The image shows a database connection configuration form. It includes fields for '账号' (Account) and '密码' (Password), both with placeholder text '请输入账号' and '请输入密码'. Below these is a '测试连接' (Test Connection) button and a checked checkbox for '保存密码' (Save Password). There is also a field for '默认数据库/schema (可选)' (Default Database/schema (Optional)) with placeholder text '请填写默认数据库'. At the bottom, there is a 'SQL 查询超时' (SQL Query Timeout) field with a help icon. On the right side, there are three buttons: '复制连接串' (Copy Connection String), '取消' (Cancel), and '保存' (Save).

### 3. 进入导入设置面板。

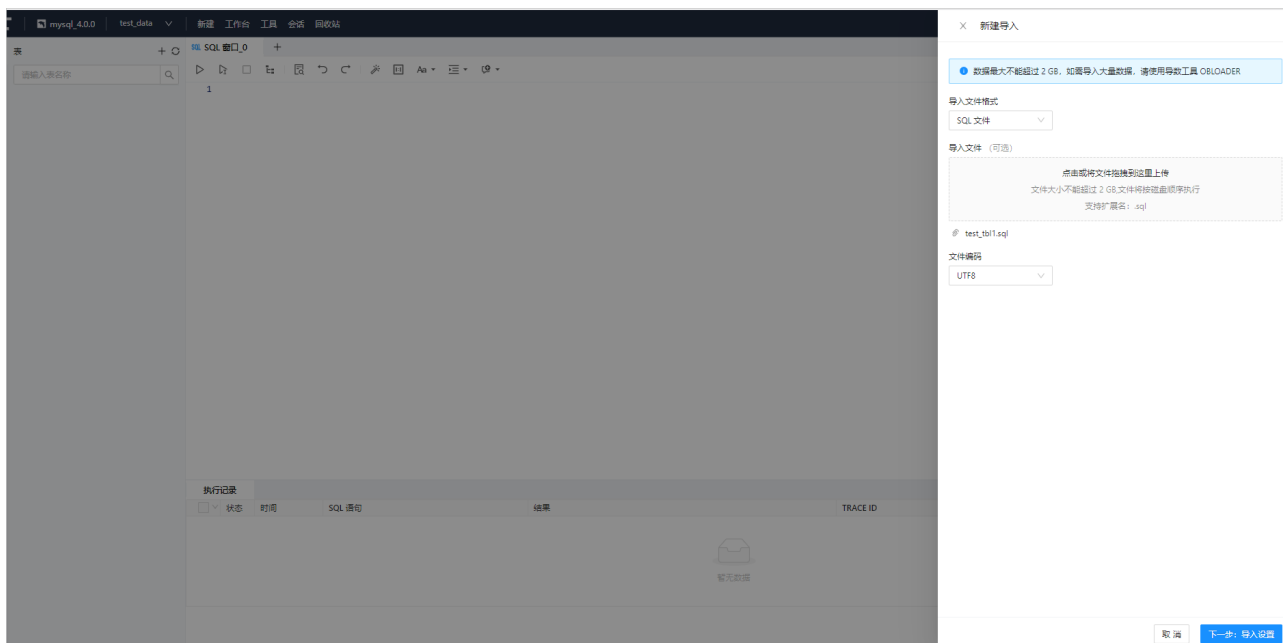
进入数据库连接后，单击顶部导航栏中的 **工具** 标签，在弹出的下拉菜单中单击 **导入** 标签进入导入设置面板。



### 4. 上传 SQL 文件。

如下图所示：

- 选择 **导入文件格式**：选择 **SQL 文件**。
- 上传 **导入文件**：单击文件池进入文件资源管理器以选择需要导入的文件，或者直接将文件拖入文件池中以完成文件上传。上传的文件格式需要与所选择的导入文件格式相同，SQL 文件支持上传 **.sql** 文件。
- 选择 **文件编码**：ODC 目前支持 **ASCII**、**ISO-8859-1**、**GB2312**、**GBK**、**GB18030**、**Unicode (UTF-8)**、**Unicode (UTF-16)**、**Unicode (UTF-32)** 和 **BIG5** 等编码格式。可在 **文件编码** 下拉框中选择需要的编码格式。
- 完成上述配置后，单击 **下一步：导入设置**。



## 5. 导入设置。

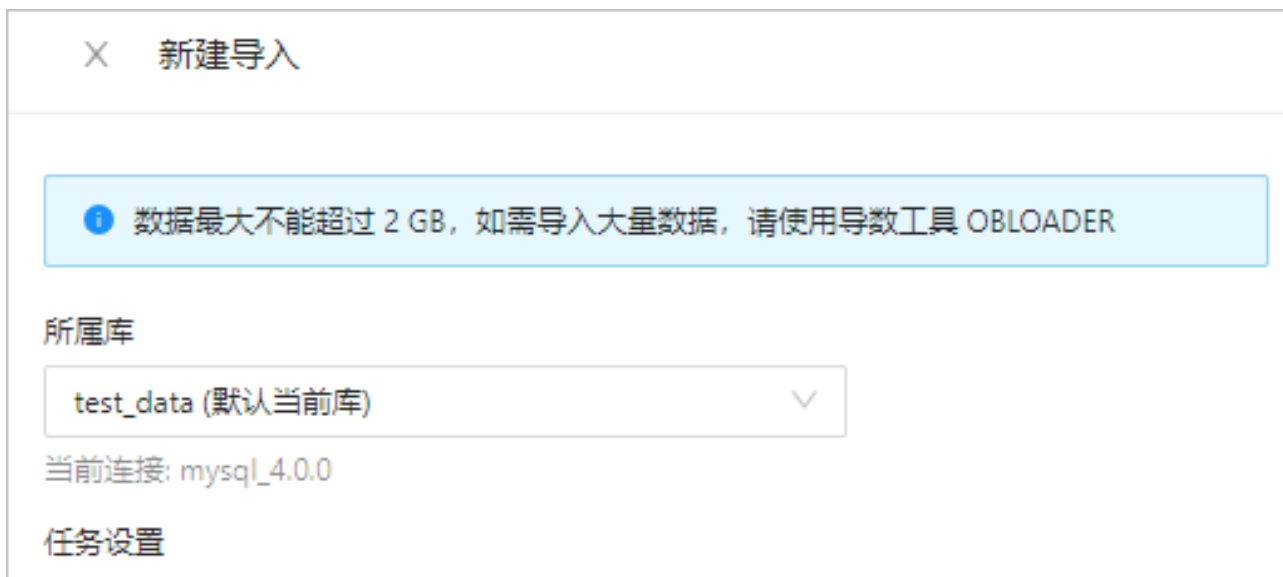
如下图所示：

- 选择 **所属库**：选择导入对象所在的数据库。该项下同时显示目标任务所在数据库的连接名称。
- 任务设置，选择 **执行方式**：支持选择 **立即执行** 和 **定时执行**。
- **sys 租户账号设置**：输入账户和密码信息后，单击密码框后的 **测试连接** 按钮测试账户信息是否正确。默认自动填入连接设置的账号，如连接失败，建议修改密码用于此次导出。

## 34.3.0.1 注意

sys 租户账号和密码 为集群租户下用户的账号和密码 (账号请勿填入"@sys#集群")。

- 完成上述配置后，单击 **提交**。



执行方式

☒ 立即执行 ☐ 定时执行

sys 租户账号设置

☒ 使用 sys 租户账号提升任务速度 

默认使用连接设置的账号，若连接失败，建议修改密码用于此次导入

账号

密码 (可选)

[测试连接](#) [修改密码](#)

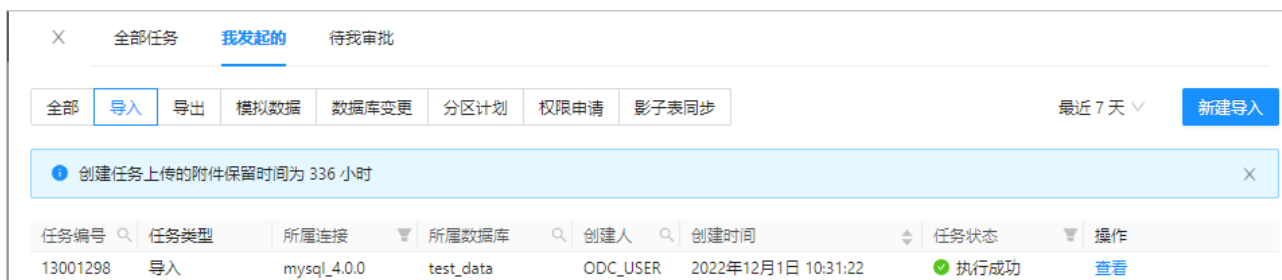
取消

上一步:上传文件

提交

6. 查看导入任务。

任务生成后会自动弹出 **任务中心** 面板，在任务中心可以查看任务信息，详情请参见 [导入任务](#)。



## 34.4 使用 obloader 导入数据

### 34.4.1 操作步骤

1. 设置 SQL 文件。
2. 准备 obloader 运行环境。详见 [准备环境](#) 和 [下载工具](#)。
3. 导入数据。

```
[xxx@xxx /ob-loader-dumper-3.3.2-SNAPSHOT/bin]
$ ./obloader -h <主机IP> -P <端#> -u <##> -p <密码> --sys-user <sys 租户下的 root 用户或 proxyro 用户> --sys-password <sys 租#下的账#密码> -c <集群> -t <租#> -D <Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

#### 34.4.1.1 说明

使用 OceanBase 物理节点的主机地址和端口导入数据时，无需指定 `-c` 选项。

有关 obloader 命令行选项的详细信息，请参见 [命令行选项](#)。

### 34.4.2 导入 SQL 文件数据示例

下表为示例中使用的数据库信息：

数据库信息	示例值
集群名	test4000
OceanBase DataBase Proxy (ODP) 主机地址	xxx.xxx.xxx.1
OceanBase DataBase Proxy (ODP) 端口号	2883
业务租户名称	mysql001
sys 租户下 root 用户的密码	*****
业务租户下的用户账号（要求读写权限）	obloader_user01
业务租户下的用户密码	*****
Schema 库名称	test_data

## 34.4.2.2 导入表结构

## 34.4.2.3 注意

- 导入表结构时，SQL 内容是 DDL，使用 `--ddl` 选项。要求 SQL 里不能有注释和 SET 开关语句等。
- SQL 文件的目录要在 `-f` 选项中的 `/data/TABLE` 下，例如：`/home/admin/test_sql/data/TABLE/test_tbl1-schema.sql`。
- SQL 文件的名称格式：`表名-schema.sql`。
- 导入数据时，需要指定 `--sys-user` 和 `--sys-password` 选项，obloader 获取表结构元数据信息。

场景描述：将 `/home/admin/test_sql/data/TABLE` 下的 SQL 数据文件，导入到集群 `test4000` 下的租户 `mysql001` 里的 Schema `test_data` 中。

示例语句如下：

### 1. 创建 SQL 文件。

```
[root@xxx /home/admin]# mkdir -p /home/admin/test_sql/data/TABLE
[root@xxx /home/admin]# cd /home/admin/test_sql/data/TABLE
[root@xxx /home/admin/test_sql/data/TABLE]# vi test_tbl1-schema.sql
[root@xxx /home/admin/test_sql/data/TABLE]# cat test_tbl1-schema.sql
```

```
CREATE TABLE IF NOT EXISTS test_tbl1(col1 INT,col2 VARCHAR(50),PRIMARY KEY
(col1));
```

2. 准备 obloader 运行环境。详见 [准备环境](#) 和 [下载工具](#)。
3. 导入表结构。

```
[root@xxx /home/admin/ob-loader-dumper-3.3.2-SNAPSHOT/bin]
$./obloader -h xxx.xxx.xxx.1 -P 2883 -u obloader_user01 -p ***** --sys-user root
--sys-password ***** -c test4000 -t mysql001 -D test_data --ddl --all -f /home
/admin/test_sql
2022-12-01 07:11:32 [INFO] Parsed args:
[--host] xxx.xxx.xxx.1
[--port] 2883
[--user] obloader_user01
[--tenant] mysql001
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
[--ddl] true
[--file-path] /home/admin/test_sql
[--all] true

2022-12-01 07:11:32 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-01 07:11:33 [INFO] The manifest file: "/home/admin/test_sql/data
/MANIFEST.bin" has been saved
2022-12-01 07:11:33 [INFO] Init writer thread pool finished
2022-12-01 07:11:33 [WARN] The object type : "SEQUENCE" doesn't exist in the -
schema.sql files
```

```
2022-12-01 07:11:33 [WARN] The object type : "TABLE_GROUP" doesn't exist in
the -schema.sql files
2022-12-01 07:11:33 [INFO] Start 128 schema file loader threads succeeded
2022-12-01 07:11:33 [INFO] No.1 sql of the file: "/home/admin/test_sql/data
/TABLE/test_tbl1-schema.sql" exec success . Elapsed: 36.94 ms
2022-12-01 07:11:33 [INFO] Load file: "test_tbl1-schema.sql" succeeded
2022-12-01 07:11:33 [WARN] The object type : "VIEW" doesn't exist in the -
schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "FUNCTION" doesn't exist in the -
schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "PROCEDURE" doesn't exist in the -
schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "TRIGGER" doesn't exist in the -
schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "PACKAGE" doesn't exist in the -
schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "TYPE" doesn't exist in the -
schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "PACKAGE_BODY" doesn't exist in
the -schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "TYPE_BODY" doesn't exist in the -
schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "SYNONYM" doesn't exist in the -
schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "PUBLIC_SYNONYM" doesn't exist
in the -schema.sql files
2022-12-01 07:11:33 [WARN] The object type : "FILE" doesn't exist in the -schema.
sql files
2022-12-01 07:11:34 [INFO] Close connection count: 1 of the DataSource. Key:
```



```
xxx.xxx.xxx.1_11532_332378361_test_data
2022-12-01 07:11:34 [INFO] Shutdown task context finished
2022-12-01 07:11:34 [INFO]
Finished Tasks: 1 Running Tasks: 0 Progress: 100.00%
2022-12-01 07:11:34 [INFO]

All Load Tasks Finished:

-----

-----

No.# | Type | Name | Count | Status
-----
-----

1 | TABLE | test_tbl1 | 1 -> 1 | SUCCESS
-----
-----

Total Count: 1 End Time: 2022-12-01 07:11:34

2022-12-01 07:11:34 [INFO] Load schema finished. Total Elapsed: 1.061 s
2022-12-01 07:11:34 [INFO] System exit 0
```

## 34.4.2.4 导入表数据

## 34.4.2.5 注意

- 导入表数据时，SQL 内容是 DML，使用 `--sql` 选项。SQL 文件中的内容仅包含 `INSERT` 语句，数据不换行。
- SQL 文件的目录要在 `-f` 选项中的 `data` 下，例如：`/home/admin/test_sql/data/test_tbl1.sql`。

- SQL 文件的名称格式：表名.sql。
- 数据库（Schema）下要存在 SQL 文件对应的表名。

场景描述：将 /home/admin/test\_sql/data/ SQL 数据文件，导入到集群 test4000 下的租户 mysql001 里的 Schema test\_data 中。

示例语句如下：

1. 创建 SQL 文件。

```
[root@xxx /home/admin]# mkdir -p /home/admin/test_sql/data
[root@xxx /home/admin]# cd /home/admin/test_sql/data
[root@xxx /home/admin/test_sql/data]# vi test_tbl1.sql
[root@xxx /home/admin/test_sql/data/TABLE]# cat test_tbl1.sql
INSERT INTO test_tbl1 VALUES(1,'test1');
INSERT INTO test_tbl1 VALUES(2,'test2');
INSERT INTO test_tbl1 VALUES(3,'test3');
INSERT INTO test_tbl1 VALUES(4,'test4');
INSERT INTO test_tbl1 VALUES(5,'test5');
```

2. 准备 obloader 运行环境。详见 [准备环境](#) 和 [下载工具](#)。
3. 导入表数据。

```
[root@xxx /home/admin/ob-loader-dumper-3.3.2-SNAPSHOT/bin]
$ ./obloader -h xxx.xxx.xxx.1 -P 2883 -u obloader_user01 -p ***** --sys-user root
--sys-password ***** -c test4000 -t mysql001 -D test_data --sql --all -f /home
/admin/test_sql
2022-12-01 07:17:39 [INFO] Parsed args:
[--host] xxx.xxx.xxx.1
[--port] 2883
[--user] obloader_user01
[--tenant] mysql001
[--cluster] test4000
[--password] *****
```

```
[--database] test_data
```

```
[--sys-user] root
```

```
[--sys-password] *****
```

```
[--sql] true
```

```
[--file-path] /home/admin/test_sql
```

```
[--all] true
```

```
2022-12-01 07:17:39 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
```

```
2022-12-01 07:17:40 [INFO] The manifest file: "/home/admin/test_sql/data
/MANIFEST.bin" has been saved
```

```
2022-12-01 07:17:40 [INFO] Query the column metadata for the table: "test_tbl1"
finished
```

```
2022-12-01 07:17:40 [WARN] File: "/home/admin/test_sql/data/TABLE/test_tbl1-
schema.sql" is unmatched on the suffix[.sql], ignore it
```

```
2022-12-01 07:17:40 [WARN] File: "/home/admin/test_sql/data/MANIFEST.bin" is
unmatched on the suffix[.sql], ignore it
```

```
2022-12-01 07:17:40 [INFO] Binding table: "test_tbl1" to the file: "/home/admin
/test_sql/data/test_tbl1.sql" finished
```

```
2022-12-01 07:17:40 [INFO] File: "/home/admin/test_sql/data/test_tbl1.sql" has
not been splitted. 205 < 67108864
```

```
2022-12-01 07:17:40 [INFO] Splitted 1 sql subfiles by 64.0 MB. Elapsed: 15.64 ms
```

```
2022-12-01 07:17:40 [INFO] Generate 1 subfiles finished
```

```
2022-12-01 07:17:40 [INFO] Ignore to clean any tables as --truncate-table or --
delete-from-table is not specified
```

```
2022-12-01 07:17:40 [ERROR] Invalid table entry: TableEntryKey [ cluster:
test4000, tenant: mysql001, database: test_data, table: test_tbl1 ], TableId:
-9223372036854775808, PartitionNum: 1, ReplicaNum: -9223372036854775808,
SchemaVersion: null, PartitionInfo: null
```

```
2022-12-01 07:17:40 [INFO] Query table entry and primary key for table:
"test_tbl1" finished. Remain: 0
2022-12-01 07:17:40 [INFO] Calculate leader: null of table: "test_tbl1", part: 0.
Remain: 0
2022-12-01 07:17:40 [INFO] Waiting to refresh observer load status .....
2022-12-01 07:17:40 [INFO] Refresh the observer load status success. Table:
"test_tbl1". Remain: 0
2022-12-01 07:17:40 [INFO] Refresh observer load status finished. Elapsed: 1.203
ms
2022-12-01 07:17:40 [INFO] Use c.l.d.PhasedBackoffWaitStrategy as available cpu
(s) is 64
2022-12-01 07:17:40 [INFO] Create 4096 slots for ring buffer finished. [0.0.0.0]
2022-12-01 07:17:40 [INFO] Start 128 database writer threads finished. [0.0.0.0]
2022-12-01 07:17:40 [INFO] Start 128 sql file reader threads succeeded
2022-12-01 07:17:40 [INFO] File: "/home/admin/test_sql/data/test_tbl1.sql" has
been parsed finished
2022-12-01 07:17:42 [INFO] Wait for the all the workers to drain of published
events then halt the workers
2022-12-01 07:17:42 [INFO] Close connection count: 4 of the DataSource. Key:
xxx.xxx.xxx.1_11532_332378361_test_data
2022-12-01 07:17:42 [INFO] Shutdown task context finished
2022-12-01 07:17:42 [INFO]
Finished Tasks: 1 Running Tasks: 0 Progress: 100.00%
2022-12-01 07:17:42 [INFO]

All Load Tasks Finished:
```

No.#	Type	Name	Count	Status
------	------	------	-------	--------

1	TABLE	test_tbl1	5 -> 5	SUCCESS
---	-------	-----------	--------	---------

Total Count: 5 End Time: 2022-12-01 07:17:42

2022-12-01 07:17:42 [INFO] Load record finished. Total Elapsed: 2.296 s

2022-12-01 07:17:42 [INFO] System exit 0

## 35 使用 OMS 从 OceanBase 数据库迁移数据到 OceanBase 数据库同类型租户

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）从 OceanBase 数据库迁移数据到 OceanBase 数据库同类型租户中的背景信息。

### 35.1 背景信息

您可以在 OMS 控制台创建从 OceanBase 数据库迁移数据至 OceanBase 数据库同类型租户的数据迁移项目，通过结构迁移、全量迁移和增量同步，无缝迁移源端数据库中的存量业务数据和增量数据至目标端数据库。

### 35.2 相关文档

更多使用 OMS 从 OceanBase 数据库迁移数据到 OceanBase 数据库同类型租户的操作信息，请参见 [OceanBase 数据库的单向数据迁移](#)。

## 36 使用 OMS 创建 OceanBase 数据库同类型租户容灾双活项目

本文将简单介绍使用 OceanBase 迁移服务（OceanBase Migration Service, OMS）创建 OceanBase 数据库同类型租户容灾双活项目的背景信息。

### 36.1 背景信息

随着越来越多的用户使用 OMS 进行数据迁移，OMS 需要解决的用户场景也越来越丰富。除目前广泛使用的单地域数据迁移、数据同步场景外，OMS 还支持在多个地域之间进行数据迁移，以及在异地城市之间进行数据同步或双活同步。

目前 OMS 支持的容灾双活场景如下：

- 同城数据迁移和数据同步
- 同城主备容灾
- 同城双活
- 异地数据迁移和数据同步
- 异地主备容灾
- 异地双活

主备容灾常见于有数据容灾需求的场景中。基于 OMS 创建主机房至备机房的数据同步项目（确保项目实时性），在主机房发生灾害、宕机等场景下，业务可以切换至备机房，避免产生业务影响。

但主备容灾容易造成备机房资源闲置浪费等问题。因此 OMS 提供了双活的解决方案，即同时由两个机房来承担实际业务流量。

OceanBase 数据库的容灾双活数据迁移项目无正向切换步骤，且容灾双活数据迁移项目仅允许在一条链路中打开 DDL 参数。

### 36.2 相关文档

更多使用 OMS 创建 OceanBase 数据库同类型租户容灾双活项目的操作信息，请参见 [OceanBase 数据库的容灾双活数据迁移](#)。

## 37 使用 OceanBase 导数工具在 OceanBase 集群 MySQL 租户间迁移数据

本文将介绍如何使用 OceanBase 导数工具从 OceanBase 集群 MySQL 租户迁移数据至 MySQL 租户。

### 37.1 OceanBase 导数工具简介

OceanBase 导数工具包括导出工具 obdumper 和导入工具 obloader。

- obdumper 是一款使用 Java 开发的客户端导出工具。可以使用该工具将 OceanBase 数据库中定义的对象和数据导出到文件中。有关 obdumper 的详细信息请参见 [obdumper 简介](#)。
- obloader 是一款使用 Java 开发的客户端导入工具。该工具提供了非常灵活的命令行选项，可在多种复杂的场景下，将定义和数据导入到 OceanBase 中。有关 obloader 的详细信息请参见 [obloader 简介](#)。

### 37.2 操作步骤

1. 准备 obdumper 和 obloader 运行环境。详见 [准备环境](#) 和 [下载工具](#)。
2. 创建存放导出数据的目录。
3. 导出表结构/数据。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h <主机IP> -P <端#> -u <##> -p <密码> [--sys-user <sys 租户下用户名> --sys-password <sys 租户下特定用户的密码>] -c <集群名称> -t <租#> -D <Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

#### 37.2.0.1 说明

- 使用 OceanBase 物理节点的主机地址和端口导出数据时，无需指定 -c 选项。
  - 有关 obdumper 命令行选项的详细信息，请参见 [命令行选项](#)。
4. 导入表结构/数据。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h <主机IP> -P <端#> -u <##> -p <密码> --sys-user <sys 租户下的 root
```



```
用户或 proxyro 用户> --sys-password <sys 租户下的账号#密码> -c <集群> -t <租户> -D
<Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

## 37.2.0.2 说明

- 使用 OceanBase 物理节点的主机地址和端口导入数据时，无需指定 `-c` 选项。
- 有关 obloader 命令行选项的详细信息，请参见 [命令行选项](#)。

## 37.3 示例

下表为示例中使用的数据库信息：

数据库信息	示例值
集群名称	test4000
OceanBase DataBase Proxy (ODP) 主机地址	xxx.xxx.xxx.xxx
OceanBase DataBase Proxy (ODP) 端口号	2883
sys 租户下 root 用户的密码	*****
源租户名称（MySQL 模式）	mysql001
租户 mysql001 的用户账号（要求读写权限）	obdumper_user01
租户 mysql001 下用户 obdumper_user01 的密码	*****
租户 mysql001 下 Schema 库名称	test_data
目标租户名称（MySQL 模式）	mysql002
租户 mysql002 的用户账号（要求读写权限）	obloader_user01
租户 mysql002 下用户 obloader_user01 的密码	*****
租户 mysql002 下 Schema 库名称	test_data

## 37.3.1 表结构迁移

使用 obdumper 和 obloader 进行表结构迁移时，需要指定 `--sys-user` 和 `--sys-password` 选项，获取表结构元数据信息。如果未指定该选项，导出/导入功能和性能可能会受到较大的影响。

### 37.3.1.1 说明

`--sys-user` 选项用于连接 `sys` 租户下拥有特定权限的用户。导出时如果未指定 `--sys-user` 选项，默认指定的是 `--sys-user root`。

## 37.3.1.2 导出 DDL 定义文件

场景描述：从集群 `test4000` 下的租户 `mysql001` 里的 Schema `test_data` 中，导出所有已支持的对象定义语句到 `/home/admin/test_migrate_data/ddl_data` 目录。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h xxx.xxx.xxx.xxx -P 2883 -u obdumper_user01 -p ***** -c test4000 -
t mysql001 -D test_data --sys-user root --sys-password ***** --ddl --all -f /home
/admin/test_migrate_data/ddl_data
2022-12-26 14:11:57 [INFO] Parsed args:
[--ddl] true
[--file-path] /home/admin/test_migrate_data/ddl_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obdumper_user01
[--tenant] mysql001
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
[--all] true

2022-12-26 14:11:57 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-26 14:11:57 [INFO] The manifest file: "/home/admin/test_migrate_data
/ddl_data/data/MANIFEST.bin" has been saved
```

```
2022-12-26 14:11:58 [INFO] Query column metadata for the table: "test_tbl1"
finished
2022-12-26 14:11:58 [INFO] Query column metadata for the table: "test_tbl2"
finished
2022-12-26 14:11:58 [WARN] No views are exist in the schema: "test_data"
2022-12-26 14:11:58 [WARN] No functions are exist in the schema: "test_data"
2022-12-26 14:11:58 [WARN] No procedures are exist in the schema: "test_data"
2022-12-26 14:11:58 [WARN] No table groups are exist in the tenant: "mysql001"
2022-12-26 14:11:58 [INFO] Generate 1 dump tasks finished. Total Elapsed: 3.926
ms
2022-12-26 14:11:58 [INFO] Start 3 schema dump threads for 1 dump tasks
finished.
2022-12-26 14:11:58 [INFO] Build direct com.alibaba.druid.pool.DruidDataSource
finished
2022-12-26 14:11:58 [INFO] No need to acquire DataSource for xxx@sys, as
observer is 4.0.0.0
2022-12-26 14:11:58 [INFO] Return the latest compatible version: 4.0.0.0 -> 4.0.0.0
2022-12-26 14:11:58 [INFO] Dump create objects success. DbType: OBMYSQL
Version: 4.0.0.0
2022-12-26 14:11:58 [INFO] ObMySQL(4.0.0.0) is older than 4.0 ? false
2022-12-26 14:11:58 [INFO] Load meta/obmysql/obmysql14x.xml, meta/obmysql
/obmysql22x.xml, meta/obmysql/obmysql2271.xml, meta/obmysql
/obmysql3230.xml, meta/obmysql/obmysql40x.xml succeeded
2022-12-26 14:11:58 [INFO] Query 0 dependencies elapsed 31.35 ms
2022-12-26 14:12:02 [INFO] ----- Finished Tasks: 0 Running Tasks: 1 Progress:
0.00% -----
2022-12-26 14:12:03 [INFO] Query table: "test_tbl1" attr finished. Remain: 0
2022-12-26 14:12:03 [INFO] Query table: "test_tbl2" attr finished. Remain: 0
2022-12-26 14:12:03 [INFO] Query 2 tables elapsed 5.721 s
```

```
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireTablespaceMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireSequenceMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireSynonymMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireTypeMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireTypeBodyMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquirePackageMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquirePackageBodyMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireTriggerMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireDatabaseLinkMapping()
2022-12-26 14:12:03 [INFO] Dump [TABLE] test_tbl1 to "/home/admin
/test_migrate_data/ddl_data/data/test_data/TABLE/test_tbl1-schema.sql "
finished
2022-12-26 14:12:03 [INFO] Dump [TABLE] test_tbl2 to "/home/admin
/test_migrate_data/ddl_data/data/test_data/TABLE/test_tbl2-schema.sql "
finished
2022-12-26 14:12:03 [INFO] No.1 It has dumped 2 TABLEs finished. Remain: 0
2022-12-26 14:12:03 [INFO] Total dumped 2 TABLEs finished. Elapsed: 5.818 s
2022-12-26 14:12:03 [INFO] Dump the ddl of schema: "test_data" finished
2022-12-26 14:12:04 [INFO] Close connection count: 24 of the DataSource. Key:
xxx.xxx.xxx.xxx_11532_2105466567_test_data
```

```

2022-12-26 14:12:04 [INFO] Shutdown task context finished
2022-12-26 14:12:04 [INFO] ----- Finished Tasks: 1 Running Tasks: 0 Progress:
100.00% -----
2022-12-26 14:12:04 [INFO]

All Dump Tasks Finished:

-----

-----

No.# | Type | Name | Count | Status
-----
-----

1 | TABLE | test_tbl1 | 1 | SUCCESS
2 | TABLE | test_tbl2 | 1 | SUCCESS
-----
-----

Total Count: 2 End Time: 2022-12-26 14:12:04

2022-12-26 14:12:04 [INFO] Dump schema finished. Total Elapsed: 6.076 s
2022-12-26 14:12:04 [INFO] Unnecessary to upload the data files to the remote
cloud storage service
2022-12-26 14:12:04 [INFO] System exit 0

```

- 导出目录结构如下。

```

[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$tree /home/admin/test_migrate_data/ddl_data
/home/admin/test_migrate_data/ddl_data
|—— data

```

```
| |—— CHECKPOINT.bin
| |—— MANIFEST.bin
| |—— test_data
| |—— TABLE
| |—— test_tbl1-schema.sql
| |—— test_tbl2-schema.sql
|—— logs
|—— ob-loader-dumper.error
|—— ob-loader-dumper.info
|—— ob-loader-dumper.warn
```

4 directories, 7 files

- 查看导出结果文件。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$cat /home/admin/test_migrate_data/ddl_data/data/test_data/TABLE/test_tbl1-
schema.sql
create table if not exists `test_tbl1` (
`col1` int(11) not null,
`col2` varchar(20),
`col3` int(11),
primary key (`col1`)
)
default charset=utf8mb4
default collate=utf8mb4_general_ci;
```

### 37.3.1.3 导入 DDL 定义文件

场景描述：将从集群 test4000 下的租户 mysql001 里的 Schema test\_data 导出到 /home/admin/test\_migrate\_data/ddl\_data 目录下所有已支持的定义信息，导入到集群 test4000 下的租户 mysql002 里的 Schema test\_data 中。

示例语句如下：

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h xxx.xxx.xxx.xxx -P 2883 -u obloader_user01 -p ***** --sys-user root --
sys-password ***** -c test4000 -t mysql002 -D test_data --ddl --all -f /home/admin
/test_migrate_data/ddl_data
2022-12-26 14:31:47 [INFO] Parsed args:
[--ddl] true
[--file-path] /home/admin/test_migrate_data/ddl_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obloader_user01
[--tenant] mysql002
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
[--all] true

2022-12-26 14:31:47 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-26 14:31:48 [INFO] The manifest file: "/home/admin/test_migrate_data
/ddl_data/data/MANIFEST.bin" has been saved
2022-12-26 14:31:48 [INFO] Init writer thread pool finished
2022-12-26 14:31:48 [WARN] The object type : "SEQUENCE" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "TABLE_GROUP" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [INFO] Start 128 schema file loader threads succeeded
```

```
2022-12-26 14:31:48 [INFO] No.1 sql of the file: "/home/admin/test_migrate_data
/ddl_data/data/test_data/TABLE/test_tbl2-schema.sql" exec success. Elapsed:
127.1 ms
2022-12-26 14:31:48 [INFO] Load file: "test_tbl2-schema.sql" succeeded
2022-12-26 14:31:48 [INFO] No.1 sql of the file: "/home/admin/test_migrate_data
/ddl_data/data/test_data/TABLE/test_tbl1-schema.sql" exec success. Elapsed:
177.2 ms
2022-12-26 14:31:48 [INFO] Load file: "test_tbl1-schema.sql" succeeded
2022-12-26 14:31:48 [WARN] The object type : "VIEW" doesn't exist in the -schema.
sql files
2022-12-26 14:31:48 [WARN] The object type : "FUNCTION" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "PROCEDURE" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "TRIGGER" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "PACKAGE" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "TYPE" doesn't exist in the -schema.
sql files
2022-12-26 14:31:48 [WARN] The object type : "PACKAGE_BODY" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "TYPE_BODY" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "SYNONYM" doesn't exist in the -
schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "PUBLIC_SYNONYM" doesn't exist in
the -schema.sql files
2022-12-26 14:31:48 [WARN] The object type : "FILE" doesn't exist in the -schema.sql
```



files

2022-12-26 14:31:49 [INFO] Close connection count: 1 of the DataSource. Key: xxx.

xxx.xxx.xxx\_11532\_528891866\_test\_data

2022-12-26 14:31:49 [INFO] Shutdown task context finished

2022-12-26 14:31:49 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress:

100.00% -----

2022-12-26 14:31:49 [INFO]

All Load Tasks Finished:

-----  
-----  
No.# | Type | Name | Count | Status  
-----

-----  
1 | TABLE | test\_tbl1 | 1 -> 1 | SUCCESS

2 | TABLE | test\_tbl2 | 1 -> 1 | SUCCESS  
-----

-----  
Total Count: 2 End Time: 2022-12-26 14:31:49

2022-12-26 14:31:49 [INFO] Load schema finished. Total Elapsed: 1.073 s

2022-12-26 14:31:49 [INFO] System exit 0

## 37.3.2 表数据迁移

以下将以导出 CSV 数据文件和导入 CSV 数据文件为例进行介绍使用 obdumper 和 obloader 进行表数据迁移。

## 37.3.2.4 导出 CSV 数据文件

文件类型定义：CSV 数据文件（后缀名 `.csv`）是逗号分隔值文件格式，CSV 数据文件以纯文本形式存储表格数据，可通过文本编辑器等工具或者 Excel 打开。

场景描述：从集群 `test4000` 下的租户 `mysql001` 里的 Schema `test_data` 中，导出所有已支持对象的数据到 `/home/admin/test_migrate_data/csv_data` 目录，数据格式为 CSV 格式。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h xxx.xxx.xxx.xxx -P 2883 -u obdumper_user01 -p ***** -c test4000 -
t mysql001 -D test_data --sys-user root --sys-password ***** --csv --table '*' -f
/home/admin/test_migrate_data/csv_data
2022-12-26 14:51:18 [INFO] Parsed args:
[--csv] true
[--file-path] /home/admin/test_migrate_data/csv_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obdumper_user01
[--tenant] mysql001
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
[--table] [*]

2022-12-26 14:51:18 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-26 14:51:18 [INFO] The manifest file: "/home/admin/test_migrate_data
/csv_data/data/MANIFEST.bin" has been saved
```

```
2022-12-26 14:51:18 [INFO] Query column metadata for the table: "test_tbl1"
finished
2022-12-26 14:51:18 [INFO] Query column metadata for the table: "test_tbl2"
finished
2022-12-26 14:51:18 [INFO] Query partition names for table: "test_tbl1" success.
(Non-partitioned)
2022-12-26 14:51:18 [INFO] Query partition names for table: "test_tbl2" success.
(Non-partitioned)
2022-12-26 14:51:18 [INFO] Query primary key for table: "test_tbl1" success.
Elapsed: 12.14 ms
2022-12-26 14:51:18 [INFO] Query primary key for table: "test_tbl2" success.
Elapsed: 12.24 ms
2022-12-26 14:51:18 [INFO] Query table entry for table: "test_tbl1" success.
Remain: 0. Elapsed: 2.868 ms
2022-12-26 14:51:18 [INFO] Query table entry for table: "test_tbl2" success.
Remain: 0. Elapsed: 2.868 ms
2022-12-26 14:51:18 [INFO] Query all table entries success. Total: 2. Elapsed:
61.20 ms
2022-12-26 14:51:18 [INFO] ....Splitting rows for non-partitioned table: "test_tbl1"
success. Batch: 1
2022-12-26 14:51:18 [INFO] ....Splitting rows for non-partitioned table: "test_tbl2"
success. Batch: 1
2022-12-26 14:51:18 [INFO] Split rows for non-partitioned table(with primary key):
"test_tbl2" success. Ranges: 1. Elapsed: 38.42 ms
2022-12-26 14:51:18 [INFO] Split rows for non-partitioned table(with primary key):
"test_tbl1" success. Ranges: 1. Elapsed: 38.57 ms
2022-12-26 14:51:18 [INFO] Generate 2 dump tasks finished. Total Elapsed: 53.46
ms
2022-12-26 14:51:18 [INFO] Start 128 record dump threads for 2 dump tasks
```

finished

2022-12-26 14:51:18 [INFO] Dump 5 rows test\_data.test\_tbl2 to "/home/admin/test\_migrate\_data/csv\_data/data/test\_data/TABLE/test\_tbl2.1.\*.csv" finished

2022-12-26 14:51:18 [INFO] Dump 5 rows test\_data.test\_tbl1 to "/home/admin/test\_migrate\_data/csv\_data/data/test\_data/TABLE/test\_tbl1.1.\*.csv" finished

2022-12-26 14:51:19 [INFO] Close connection count: 12 of the DataSource. Key: xxx.xxx.xxx.xxx\_11532\_2105466567\_test\_data

2022-12-26 14:51:20 [INFO] Shutdown task context finished

2022-12-26 14:51:20 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress: 100.00% -----

2022-12-26 14:51:20 [INFO]

All Dump Tasks Finished:

-----  
-----  
No.# | Type | Name | Count | Status  
-----  
-----

1 | TABLE | test\_tbl1 | 5 | SUCCESS

2 | TABLE | test\_tbl2 | 5 | SUCCESS  
-----  
-----

Total Count: 10 End Time: 2022-12-26 14:51:20

2022-12-26 14:51:20 [INFO] Unnecessary to merge the data files. As --file-name is missing

```
2022-12-26 14:51:20 [INFO] Dump record finished. Total Elapsed: 1.206 s
2022-12-26 14:51:20 [INFO] Unnecessary to upload the data files to the remote
cloud storage service
2022-12-26 14:51:20 [INFO] System exit 0
```

- 导出目录结构如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$tree /home/admin/test_migrate_data/csv_data
/home/admin/test_migrate_data/csv_data
|____ data
| |____ CHECKPOINT.bin
| |____ MANIFEST.bin
| |____ test_data
| |____ TABLE
| |____ test_tbl1.1.0.csv
| |____ test_tbl2.1.0.csv
|____ logs
|____ ob-loader-dumper.error
|____ ob-loader-dumper.info
|____ ob-loader-dumper.warn

4 directories, 7 files
```

- 查看导出结果文件。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$cat /home/admin/test_migrate_data/csv_data/data/test_data/TABLE/test_tbl1.
1.0.csv
'col1','col2','col3'
1,'China',86
2,'Taiwan',886
```

```
3,'Hong Kong',852
4,'Macao',853
5,'North Korea',850
```

## 37.3.2.5 导入 CSV 数据文件

场景描述：将从集群 test4000 下的租户 mysql001 里的 Schema test\_data 导出到 /home/admin/test\_migrate\_data/csv\_data 目录下所有已支持的 CSV 数据文件，导入到集群 test4000 下的租户 mysql002 里的 Schema test\_data 中。

示例语句如下：

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h xxx.xxx.xxx.xxx -P 2883 -u obloader_user01 -p ***** --sys-user root --
sys-password ***** -c test4000 -t mysql002 -D test_data --csv --table '*' -f /home
/admin/test_migrate_data/csv_data
2022-12-26 14:52:55 [INFO] Parsed args:
[--csv] true
[--file-path] /home/admin/test_migrate_data/csv_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obloader_user01
[--tenant] mysql002
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
[--table] [*]

2022-12-26 14:52:55 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
```

```
2022-12-26 14:52:55 [INFO] The manifest file: "/home/admin/test_migrate_data
/csv_data/data/MANIFEST.bin" has been saved
2022-12-26 14:52:55 [INFO] Query column metadata for the table: "test_tbl1" finished
2022-12-26 14:52:55 [INFO] Query column metadata for the table: "test_tbl2" finished
2022-12-26 14:52:55 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
/CHECKPOINT.bin" is unmatched on the suffix[.csv], ignore it
2022-12-26 14:52:55 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
/MANIFEST.bin" is unmatched on the suffix[.csv], ignore it
2022-12-26 14:52:55 [INFO] Binding table: "test_tbl2" to the file: "/home/admin
/test_migrate_data/csv_data/data/test_data/TABLE/test_tbl2.1.0.csv" finished
2022-12-26 14:52:55 [INFO] Binding table: "test_tbl1" to the file: "/home/admin
/test_migrate_data/csv_data/data/test_data/TABLE/test_tbl1.1.0.csv" finished
2022-12-26 14:52:55 [INFO] Splitted 2 csv subfiles by 64.0 MB. Elapsed: 19.43 ms
2022-12-26 14:52:55 [INFO] Generate 2 subfiles finished
2022-12-26 14:52:55 [INFO] Ignore to clean any tables as --truncate-table or --
delete-from-table is not specified
2022-12-26 14:52:55 [INFO] Query table entry and primary key for table: "test_tbl2"
finished. Remain: 1
2022-12-26 14:52:55 [INFO] Query table entry and primary key for table: "test_tbl1"
finished. Remain: 0
2022-12-26 14:52:55 [INFO] Query the leader location of "test_tbl2" finished.
Remain: 0
2022-12-26 14:52:55 [INFO] Query the leader location of "test_tbl1" finished.
Remain: 0
2022-12-26 14:52:55 [INFO] Calculate leader: xxx.xxx.xxx.xxx:2881 of table:
"test_tbl1", part: 0. Remain: 1
2022-12-26 14:52:55 [INFO] Calculate leader: xxx.xxx.xxx.xxx:2881 of table:
"test_tbl2", part: 0. Remain: 0
2022-12-26 14:52:55 [INFO] Waiting to refresh observer load status .....
```

```
2022-12-26 14:52:55 [INFO] Refresh observer load status success. Table: "test_tbl1".
Remain: 1
2022-12-26 14:52:55 [INFO] Refresh observer load status success. Table: "test_tbl2".
Remain: 0
2022-12-26 14:52:55 [INFO] Refresh observer load status finished. Elapsed: 29.35 ms
2022-12-26 14:52:55 [INFO] Use c.l.d.PhasedBackoffWaitStrategy as available cpu(s)
is 64
2022-12-26 14:52:55 [INFO] Create 4096 slots for ring buffer finished. [xxx.xxx.xxx.
xxx:2881]
2022-12-26 14:52:56 [INFO] Start 128 database writer threads finished. [xxx.xxx.xxx.
xxx:2881]
2022-12-26 14:52:56 [INFO] Start 128 csv file reader threads succeeded
2022-12-26 14:52:56 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
/test_data/TABLE/test_tbl1.1.0.csv" has been parsed finished
2022-12-26 14:52:56 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
/test_data/TABLE/test_tbl2.1.0.csv" has been parsed finished
2022-12-26 14:52:57 [INFO] Wait for the all the workers to drain of published events
then halt the workers
2022-12-26 14:52:57 [INFO] Close connection count: 36 of the DataSource. Key: xxx.
xxx.xxx.xxx_11532_528891866_test_data
2022-12-26 14:52:57 [INFO] Shutdown task context finished
2022-12-26 14:52:57 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress:
100.00% -----
2022-12-26 14:52:57 [INFO]

All Load Tasks Finished:
```



No.#	Type	Name	Count	Status
------	------	------	-------	--------

1	TABLE	test_tbl1	5 -> 5	SUCCESS
---	-------	-----------	--------	---------

2	TABLE	test_tbl2	5 -> 5	SUCCESS
---	-------	-----------	--------	---------

Total Count: 10 End Time: 2022-12-26 14:52:57

2022-12-26 14:52:57 [INFO] Load record finished. Total Elapsed: 1.973 s

2022-12-26 14:52:57 [INFO] System exit 0

## 38 使用 OceanBase 导数工具从 OceanBase 集群 MySQL 租户迁移数据到 Oracle 租户

本文将介绍如何使用 OceanBase 导数工具从 OceanBase 数据库 MySQL 租户迁移数据到 Oracle 租户。

### 38.1 OceanBase 导数工具简介

OceanBase 导数工具包括导出工具 obdumper 和导入工具 obloader。

- obdumper 是一款使用 Java 开发的客户端导出工具。可以使用该工具将 OceanBase 数据库中定义的对象和数据导出到文件中。有关 obdumper 的详细信息请参见 [obdumper 简介](#)。
- obloader 是一款使用 Java 开发的客户端导入工具。该工具提供了非常灵活的命令行选项，可在多种复杂的场景下，将定义和数据导入到 OceanBase 中。有关 obloader 的详细信息请参见 [obloader 简介](#)。

### 38.2 操作步骤

1. 准备 obdumper 和 obloader 运行环境。详见 [准备环境](#) 和 [下载工具](#)。
2. 创建存放导出数据的目录。
3. 导出表结构/数据。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h <主机IP> -P <端#> -u <##> -p <密码> [--sys-user <sys 租户下用户名> --sys-password <sys 租户下特定用户的密码>] -c <集群名称> -t <租#> -D <Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

#### 38.2.0.1 说明

- 使用 OceanBase 物理节点的主机地址和端口导出数据时，无需指定 `-c` 选项。
  - 有关 obdumper 命令行选项的详细信息，请参见 [命令行选项](#)。
4. 导入表结构/数据。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h <主机IP> -P <端#> -u <##> -p <密码> --sys-user <sys 租户下的 root
```

```
用户或 proxyro 用户> --sys-password <sys 租户下的账号#密码> -c <集群> -t <租户> -D
<Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

## 38.2.0.2 说明

- 使用 OceanBase 物理节点的主机地址和端口导入数据时，无需指定 `-c` 选项。
- 有关 obloader 命令行选项的详细信息，请参见 [命令行选项](#)。

## 38.3 导出和导入数据示例

下表为示例中使用的数据库信息：

数据库信息	示例值
集群名	test4000
OceanBase DataBase Proxy (ODP) 主机地址	xxx.xxx.xxx.xxx
OceanBase DataBase Proxy (ODP) 端口号	2883
sys 租户下 root 用户的密码	*****
源租户名称（MySQL 模式）	mysql001
租户 mysql001 的用户账号（要求读写权限）	obdumper_user01
租户 mysql001 下用户 obdumper_user01 的密码	*****
租户 mysql001 下 Schema 库名称	test_data
目标租户名称（Oracle 模式）	oracle001
租户 oracle001 的用户账号（要求读写权限）	obloader_user01
租户 oracle001 下用户 obloader_user01 的密码	*****
租户 oracle001 下 Schema 库名称	OBLOADER_USER01

### 38.3.1 表结构迁移

使用 obdumper 和 obloader 进行表结构迁移时，需要指定 `--sys-user` 和 `--sys-password` 选项，获取表结构元数据信息。如果未指定该选项，导出/导入功能和性能可能会受到较大的影响。

#### 38.3.1.1 说明

`--sys-user` 选项用于连接 `sys` 租户下拥有特定权限的用户。导出时如果未指定 `--sys-user` 选项，默认指定的是 `--sys-user root`。

## 38.3.1.2 导出 DDL 定义文件

场景描述：从集群 `test4000` 下的租户 `mysql001` 里的 Schema `test_data` 中，导出所有已支持的对象定义语句到 `/home/admin/test_migrate_data/ddl_data` 目录。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h xxx.xxx.xxx.xxx -P 2883 -u obdumper_user01 -p ***** -c test4000 -
t mysql001 -D test_data --sys-user root --sys-password ***** --ddl --all -f /home
/admin/test_migrate_data/ddl_data
2022-12-26 14:11:57 [INFO] Parsed args:
[--ddl] true
[--file-path] /home/admin/test_migrate_data/ddl_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obdumper_user01
[--tenant] mysql001
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
[--all] true

2022-12-26 14:11:57 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-26 14:11:57 [INFO] The manifest file: "/home/admin/test_migrate_data
/ddl_data/data/MANIFEST.bin" has been saved
```

```
2022-12-26 14:11:58 [INFO] Query column metadata for the table: "test_tbl1"
finished
2022-12-26 14:11:58 [INFO] Query column metadata for the table: "test_tbl2"
finished
2022-12-26 14:11:58 [WARN] No views are exist in the schema: "test_data"
2022-12-26 14:11:58 [WARN] No functions are exist in the schema: "test_data"
2022-12-26 14:11:58 [WARN] No procedures are exist in the schema: "test_data"
2022-12-26 14:11:58 [WARN] No table groups are exist in the tenant: "mysql001"
2022-12-26 14:11:58 [INFO] Generate 1 dump tasks finished. Total Elapsed: 3.926
ms
2022-12-26 14:11:58 [INFO] Start 3 schema dump threads for 1 dump tasks
finished.
2022-12-26 14:11:58 [INFO] Build direct com.alibaba.druid.pool.DruidDataSource
finished
2022-12-26 14:11:58 [INFO] No need to acquire DataSource for xxx@sys, as
observer is 4.0.0.0
2022-12-26 14:11:58 [INFO] Return the latest compatible version: 4.0.0.0 -> 4.0.0.0
2022-12-26 14:11:58 [INFO] Dump create objects success. DbType: OBMYSQL
Version: 4.0.0.0
2022-12-26 14:11:58 [INFO] ObMySQL(4.0.0.0) is older than 4.0 ? false
2022-12-26 14:11:58 [INFO] Load meta/obmysql/obmysql14x.xml, meta/obmysql
/obmysql22x.xml, meta/obmysql/obmysql2271.xml, meta/obmysql
/obmysql3230.xml, meta/obmysql/obmysql40x.xml succeeded
2022-12-26 14:11:58 [INFO] Query 0 dependencies elapsed 31.35 ms
2022-12-26 14:12:02 [INFO] ----- Finished Tasks: 0 Running Tasks: 1 Progress:
0.00% -----
2022-12-26 14:12:03 [INFO] Query table: "test_tbl1" attr finished. Remain: 0
2022-12-26 14:12:03 [INFO] Query table: "test_tbl2" attr finished. Remain: 0
2022-12-26 14:12:03 [INFO] Query 2 tables elapsed 5.721 s
```

```
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireTablespaceMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireSequenceMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireSynonymMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireTypeMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireTypeBodyMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquirePackageMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquirePackageBodyMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireTriggerMapping()
2022-12-26 14:12:03 [WARN] c.o.t.l.s.o.ObMySqlDatabase doesn't implement
acquireDatabaseLinkMapping()
2022-12-26 14:12:03 [INFO] Dump [TABLE] test_tbl1 to "/home/admin
/test_migrate_data/ddl_data/data/test_data/TABLE/test_tbl1-schema.sql "
finished
2022-12-26 14:12:03 [INFO] Dump [TABLE] test_tbl2 to "/home/admin
/test_migrate_data/ddl_data/data/test_data/TABLE/test_tbl2-schema.sql "
finished
2022-12-26 14:12:03 [INFO] No.1 It has dumped 2 TABLEs finished. Remain: 0
2022-12-26 14:12:03 [INFO] Total dumped 2 TABLEs finished. Elapsed: 5.818 s
2022-12-26 14:12:03 [INFO] Dump the ddl of schema: "test_data" finished
2022-12-26 14:12:04 [INFO] Close connection count: 24 of the DataSource. Key:
xxx.xxx.xxx.xxx_11532_2105466567_test_data
```

```
2022-12-26 14:12:04 [INFO] Shutdown task context finished
2022-12-26 14:12:04 [INFO] ----- Finished Tasks: 1 Running Tasks: 0 Progress:
100.00% -----
2022-12-26 14:12:04 [INFO]

All Dump Tasks Finished:

-----

-----

No.# | Type | Name | Count | Status
-----
-----

1 | TABLE | test_tbl1 | 1 | SUCCESS
2 | TABLE | test_tbl2 | 1 | SUCCESS
-----
-----

Total Count: 2 End Time: 2022-12-26 14:12:04

2022-12-26 14:12:04 [INFO] Dump schema finished. Total Elapsed: 6.076 s
2022-12-26 14:12:04 [INFO] Unnecessary to upload the data files to the remote
cloud storage service
2022-12-26 14:12:04 [INFO] System exit 0
```

- 导出目录结构如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$tree /home/admin/test_migrate_data/ddl_data
/home/admin/test_migrate_data/ddl_data
|—— data
```

```
| |—— CHECKPOINT.bin
| |—— MANIFEST.bin
| |—— test_data
| |—— TABLE
| |—— test_tbl1-schema.sql
| |—— test_tbl2-schema.sql
|—— logs
|—— ob-loader-dumper.error
|—— ob-loader-dumper.info
|—— ob-loader-dumper.warn
```

4 directories, 7 files

- 查看导出结果文件。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$cat /home/admin/test_migrate_data/ddl_data/data/test_data/TABLE/test_tbl1-
schema.sql
create table if not exists `test_tbl1` (
`col1` int(11) not null,
`col2` varchar(20),
`col3` int(11),
primary key (`col1`)
)
default charset=utf8mb4
default collate=utf8mb4_general_ci;
```

### 38.3.1.3 导入 DDL 定义文件

场景描述：将从集群 test4000 下的租户 mysql001 里的 Schema test\_data 导出到 /home/admin/test\_migrate\_data/ddl\_data 目录下所有已支持的定义信息，导入到集群 test4000 下的租户 oracle001 里的 Schema obloader\_user01 中。



## 38.3.1.4 注意

从 MySQL 租户里导出的 DDL 文件不符合 Oracle 租户的语法要求，所以从 MySQL 租户里导出的 DDL 文件先修改成符合 Oracle 租户的语法要求的 DDL 文件后，再导入 Oracle 租户中。

例如，从租户 `mysql001` 里导出的 DDL 文件，修改规则如下：

- 去掉 `if not exists` 关键字和单引号。
- 把数据类型修改为 Oracle 模式通用的数据类型：
  - 将 `int(11)` 改为 `number(11)`。
  - 将 `varchar(50)` 改为 `varchar2(50)`。
- 去掉默认字符序和字符集参数 `default charset=utf8mb4`、`default collate=utf8mb4_general_ci`。

有关 Oracle 模式数据类型的详细信息，请参见 [内建数据类型概述](#)。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h xxx.xxx.xxx.xxx -P 2883 -u obloader_user01 -p ***** --sys-user
root --sys-password ***** -c test4000 -t oracle001 -D obloader_user01 --ddl --all -
f /home/admin/test_migrate_data/ddl_data
2022-12-26 16:16:35 [INFO] Parsed args:
[--ddl] true
[--file-path] /home/admin/test_migrate_data/ddl_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obloader_user01
[--tenant] oracle001
[--cluster] test4000
[--password] *****
[--database] obloader_user01
[--sys-user] root
```

```
[--sys-password] *****
```

```
[--all] true
```

```
2022-12-26 16:16:35 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
```

```
2022-12-26 16:16:35 [INFO] The manifest file: "/home/admin/test_migrate_data
/ddl_data/data/MANIFEST.bin" has been saved
```

```
2022-12-26 16:16:35 [INFO] Init writer thread pool finished
```

```
2022-12-26 16:16:35 [WARN] The object type : "SEQUENCE" doesn't exist in the -
schema.sql files
```

```
2022-12-26 16:16:35 [WARN] The object type : "TABLE_GROUP" doesn't exist in the -
schema.sql files
```

```
2022-12-26 16:16:35 [INFO] Start 128 schema file loader threads succeeded
```

```
2022-12-26 16:16:35 [INFO] No.1 sql of the file: "/home/admin/test_migrate_data
/ddl_data/data/test_data/TABLE/test_tbl2-schema.sql" exec success. Elapsed:
66.25 ms
```

```
2022-12-26 16:16:35 [INFO] Load file: "test_tbl2-schema.sql" succeeded
```

```
2022-12-26 16:16:35 [INFO] No.1 sql of the file: "/home/admin/test_migrate_data
/ddl_data/data/test_data/TABLE/test_tbl1-schema.sql" exec success. Elapsed:
122.9 ms
```

```
2022-12-26 16:16:35 [INFO] Load file: "test_tbl1-schema.sql" succeeded
```

```
2022-12-26 16:16:35 [WARN] The object type : "VIEW" doesn't exist in the -schema.
sql files
```

```
2022-12-26 16:16:35 [WARN] The object type : "FUNCTION" doesn't exist in the -
schema.sql files
```

```
2022-12-26 16:16:35 [WARN] The object type : "PROCEDURE" doesn't exist in the -
schema.sql files
```

```
2022-12-26 16:16:35 [WARN] The object type : "TRIGGER" doesn't exist in the -
schema.sql files
```

```
2022-12-26 16:16:35 [WARN] The object type : "PACKAGE" doesn't exist in the -
schema.sql files
2022-12-26 16:16:35 [WARN] The object type : "TYPE" doesn't exist in the -schema.
sql files
2022-12-26 16:16:35 [WARN] The object type : "PACKAGE_BODY" doesn't exist in
the -schema.sql files
2022-12-26 16:16:35 [WARN] The object type : "TYPE_BODY" doesn't exist in the -
schema.sql files
2022-12-26 16:16:35 [WARN] The object type : "SYNONYM" doesn't exist in the -
schema.sql files
2022-12-26 16:16:35 [WARN] The object type : "PUBLIC_SYNONYM" doesn't exist in
the -schema.sql files
2022-12-26 16:16:35 [WARN] The object type : "FILE" doesn't exist in the -schema.
sql files
2022-12-26 16:16:36 [INFO] Close connection count: 1 of the DataSource. Key: xxx.
xxx.xxx.xxx_11532_1709388295_obloader_user01
2022-12-26 16:16:36 [INFO] Shutdown task context finished
2022-12-26 16:16:36 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress:
100.00% -----
2022-12-26 16:16:36 [INFO]

All Load Tasks Finished:

-----
-----
No.# | Type | Name | Count | Status
-----
-----
1 | TABLE | test_tbl1 | 1 -> 1 | SUCCESS
```

```
2 | TABLE | test_tbl2 | 1 -> 1 | SUCCESS
```

```
-----
```

```
-----
```

```
Total Count: 2 End Time: 2022-12-26 16:16:36
```

```
2022-12-26 16:16:36 [INFO] Load schema finished. Total Elapsed: 1.066 s
```

```
2022-12-26 16:16:36 [INFO] System exit 0
```

- 检查导入的表结构。

```
obclient [OBLOADER_USER01]> SELECT TABLE_NAME FROM USER_TABLES;
```

```
+-----+
```

```
| TABLE_NAME |
```

```
+-----+
```

```
| TEST_TBL2 |
```

```
| TEST_TBL1 |
```

```
+-----+
```

```
2 rows in set (0.034 sec)
```

```
obclient [OBLOADER_USER01]> SHOW CREATE TABLE test_tbl1\G
```

```
***** 1. row *****
```

```
TABLE: TEST_TBL1
```

```
CREATE TABLE: CREATE TABLE "TEST_TBL1" (
```

```
"COL1" NUMBER(11) CONSTRAINT "TEST_TBL1_OBNOTNULL_1672042595521895"
```

```
NOT NULL ENABLE,
```

```
"COL2" VARCHAR2(20),
```

```
"COL3" NUMBER(11),
```

```
CONSTRAINT "TEST_TBL1_OBPK_1672042595521915" PRIMARY KEY ("COL1")
```

```
) COMPRESS FOR ARCHIVE REPLICA_NUM = 3 BLOCK_SIZE = 16384
```

```
USE_BLOOM_FILTER = FALSE TABLET_SIZE = 134217728 PCTFREE = 0
1 row in set
```

## 38.3.2 表数据迁移

以下将以导出 CSV 数据文件和导入 CSV 数据文件为例进行介绍使用 obdumper 和 obloader 进行表数据迁移。

### 38.3.2.5 导出 CSV 数据文件

文件类型定义：CSV 数据文件（后缀名 .csv）是逗号分隔值文件格式，CSV 数据文件以纯文本形式存储表格数据，可通过文本编辑器等工具或者 Excel 打开。

场景描述：从集群 test4000 下的租户 mysql001 里的 Schema test\_data 中，导出所有已支持对象的数据到 /home/admin/test\_migrate\_data/csv\_data 目录，数据格式为 CSV 格式。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h xxx.xxx.xxx.xxx -P 2883 -u obdumper_user01 -p ***** -c test4000 -
t mysql001 -D test_data --sys-user root --sys-password ***** --csv --table '*' -f
/home/admin/test_migrate_data/csv_data
2022-12-26 14:51:18 [INFO] Parsed args:
[--csv] true
[--file-path] /home/admin/test_migrate_data/csv_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obdumper_user01
[--tenant] mysql001
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
```

```
[--table] [*]
```

```
2022-12-26 14:51:18 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
```

```
2022-12-26 14:51:18 [INFO] The manifest file: "/home/admin/test_migrate_data
/csv_data/data/MANIFEST.bin" has been saved
```

```
2022-12-26 14:51:18 [INFO] Query column metadata for the table: "test_tbl1"
finished
```

```
2022-12-26 14:51:18 [INFO] Query column metadata for the table: "test_tbl2"
finished
```

```
2022-12-26 14:51:18 [INFO] Query partition names for table: "test_tbl1" success.
(Non-partitioned)
```

```
2022-12-26 14:51:18 [INFO] Query partition names for table: "test_tbl2" success.
(Non-partitioned)
```

```
2022-12-26 14:51:18 [INFO] Query primary key for table: "test_tbl1" success.
Elapsed: 12.14 ms
```

```
2022-12-26 14:51:18 [INFO] Query primary key for table: "test_tbl2" success.
Elapsed: 12.24 ms
```

```
2022-12-26 14:51:18 [INFO] Query table entry for table: "test_tbl1" success.
Remain: 0. Elapsed: 2.868 ms
```

```
2022-12-26 14:51:18 [INFO] Query table entry for table: "test_tbl2" success.
Remain: 0. Elapsed: 2.868 ms
```

```
2022-12-26 14:51:18 [INFO] Query all table entries success. Total: 2. Elapsed:
61.20 ms
```

```
2022-12-26 14:51:18 [INFO] ....Splitting rows for non-partitioned table: "test_tbl1"
success. Batch: 1
```

```
2022-12-26 14:51:18 [INFO] ....Splitting rows for non-partitioned table: "test_tbl2"
success. Batch: 1
```

```
2022-12-26 14:51:18 [INFO] Split rows for non-partitioned table(with primary key):
```

```
"test_tbl2" success. Ranges: 1. Elapsed: 38.42 ms
2022-12-26 14:51:18 [INFO] Split rows for non-partitioned table(with primary key):
"test_tbl1" success. Ranges: 1. Elapsed: 38.57 ms
2022-12-26 14:51:18 [INFO] Generate 2 dump tasks finished. Total Elapsed: 53.46
ms
2022-12-26 14:51:18 [INFO] Start 128 record dump threads for 2 dump tasks
finished
2022-12-26 14:51:18 [INFO] Dump 5 rows test_data.test_tbl2 to "/home/admin
/test_migrate_data/csv_data/data/test_data/TABLE/test_tbl2.1.*.csv" finished
2022-12-26 14:51:18 [INFO] Dump 5 rows test_data.test_tbl1 to "/home/admin
/test_migrate_data/csv_data/data/test_data/TABLE/test_tbl1.1.*.csv" finished
2022-12-26 14:51:19 [INFO] Close connection count: 12 of the DataSource. Key:
xxx.xxx.xxx.xxx_11532_2105466567_test_data
2022-12-26 14:51:20 [INFO] Shutdown task context finished
2022-12-26 14:51:20 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress:
100.00% -----
2022-12-26 14:51:20 [INFO]
```

All Dump Tasks Finished:

```
-----
-----
No.# | Type | Name | Count | Status
-----
-----
```

```
1 | TABLE | test_tbl1 | 5 | SUCCESS
```

```
2 | TABLE | test_tbl2 | 5 | SUCCESS
-----
-----
```

```
Total Count: 10 End Time: 2022-12-26 14:51:20
```

```
2022-12-26 14:51:20 [INFO] Unnecessary to merge the data files. As --file-name is missing
```

```
2022-12-26 14:51:20 [INFO] Dump record finished. Total Elapsed: 1.206 s
```

```
2022-12-26 14:51:20 [INFO] Unnecessary to upload the data files to the remote cloud storage service
```

```
2022-12-26 14:51:20 [INFO] System exit 0
```

- 导出目录结构如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
```

```
$tree /home/admin/test_migrate_data/csv_data
```

```
/home/admin/test_migrate_data/csv_data
```

```
|—— data
```

```
| |—— CHECKPOINT.bin
```

```
| |—— MANIFEST.bin
```

```
| |—— test_data
```

```
| |—— TABLE
```

```
| |—— test_tbl1.1.0.csv
```

```
| |—— test_tbl2.1.0.csv
```

```
|—— logs
```

```
|—— ob-loader-dumper.error
```

```
|—— ob-loader-dumper.info
```

```
|—— ob-loader-dumper.warn
```

```
4 directories, 7 files
```

- 查看导出结果文件。



```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$cat /home/admin/test_migrate_data/csv_data/data/test_data/TABLE/test_tbl1.
1.0.csv
'col1','col2','col3'
1,'China',86
2,'Taiwan',886
3,'Hong Kong',852
4,'Macao',853
5,'North Korea',850
```

## 38.3.2.6 导入 CSV 数据文件

场景描述：将从集群 `test4000` 下的租户 `mysql001` 里的 Schema `test_data` 导出到 `/home/admin/test_migrate_data/csv_data` 目录下所有已支持的 CSV 数据文件，导入到集群 `test4000` 下的租户 `oracle001` 里的 Schema `OBLOADER_USER01` 中。

## 38.3.2.7 注意

- 需要将租户 `mysql001` 里的 Schema `test_data` 导出的目录 `/home/admin/test_migrate_data/csv_data/data/` 下的 `test_data` 名称修改为 `OBLOADER_USER01`。
- 将导出的 CSV 文件名中的表名中的英文字母改为大写。

示例语句如下：

```
bash [root@xxx /home/admin/test_migrate_data/csv_data/data]# mv test_data
OBLOADER_USER01 [root@xxx /home/admin/test_migrate_data/csv_data/data]# cd
OBLOADER_USER01/TABLE [root@xxx /home/admin/test_migrate_data/csv_data
/data/OBLOADER_USER01/TABLE]# ls test_tbl1.1.0.csv test_tbl2.1.0.csv [root@xxx
/home/admin/test_migrate_data/csv_data/data/OBLOADER_USER01/TABLE]# mv
test_tbl1.1.0.csv TEST_TBL1.1.0.csv [root@xxx /home/admin/test_migrate_data
/csv_data/data/OBLOADER_USER01/TABLE]# mv test_tbl2.1.0.csv TEST_TBL2.1.0.csv
<code></code>
```

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h xxx.xxx.xxx.xxx -P 2883 -u obloader_user01 -p ***** --sys-user
root --sys-password ***** -c test4000 -t oracle001 -D obloader_user01 --csv --
table '*' -f /home/admin/test_migrate_data/csv_data
2022-12-26 16:54:36 [INFO] Parsed args:
[--csv] true
[--file-path] /home/admin/test_migrate_data/csv_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obloader_user01
[--tenant] oracle001
[--cluster] test4000
[--password] *****
[--database] obloader_user01
[--sys-user] root
[--sys-password] *****
[--table] [*]

2022-12-26 16:54:36 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-26 16:54:36 [INFO] The manifest file: "/home/admin/test_migrate_data
/csv_data/data/MANIFEST.bin" has been saved
2022-12-26 16:54:36 [INFO] Query column metadata for the table: "TEST_TBL1"
finished
2022-12-26 16:54:36 [INFO] Query column metadata for the table: "TEST_TBL2"
finished
2022-12-26 16:54:36 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
/CHECKPOINT.bin" is unmatched on the suffix[.csv], ignore it
2022-12-26 16:54:36 [INFO] Binding table: "TEST_TBL2" to the file: "/home/admin
```

```
/test_migrate_data/csv_data/data/OBLOADER_USER01/TABLE/TEST_TBL2.1.0.csv"
finished
2022-12-26 16:54:36 [INFO] Binding table: "TEST_TBL1" to the file: "/home/admin
/test_migrate_data/csv_data/data/OBLOADER_USER01/TABLE/TEST_TBL1.1.0.csv"
finished
2022-12-26 16:54:36 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
/MANIFEST.bin" is unmatched on the suffix[.csv], ignore it
2022-12-26 16:54:36 [INFO] Splitted 2 csv subfiles by 64.0 MB. Elapsed: 19.73 ms
2022-12-26 16:54:36 [INFO] Generate 2 subfiles finished
2022-12-26 16:54:36 [INFO] Ignore to clean any tables as --truncate-table or --
delete-from-table is not specified
2022-12-26 16:54:37 [INFO] Query table entry and primary key for table:
"TEST_TBL1" finished. Remain: 0
2022-12-26 16:54:37 [INFO] Query table entry and primary key for table:
"TEST_TBL2" finished. Remain: 1
2022-12-26 16:54:37 [INFO] Query the leader location of "TEST_TBL2" finished.
Remain: 1
2022-12-26 16:54:37 [INFO] Query the leader location of "TEST_TBL1" finished.
Remain: 0
2022-12-26 16:54:37 [INFO] Calculate leader: xxx.xxx.xxx.xxx:2881 of table:
"TEST_TBL1", part: 0. Remain: 1
2022-12-26 16:54:37 [INFO] Calculate leader: xxx.xxx.xxx.xxx:2881 of table:
"TEST_TBL2", part: 0. Remain: 0
2022-12-26 16:54:37 [INFO] Waiting to refresh observer load status .....
2022-12-26 16:54:37 [INFO] Refresh observer load status success. Table:
"TEST_TBL2". Remain: 1
2022-12-26 16:54:37 [INFO] Refresh observer load status success. Table:
"TEST_TBL1". Remain: 0
2022-12-26 16:54:37 [INFO] Refresh observer load status finished. Elapsed: 31.20
```

ms

2022-12-26 16:54:37 [INFO] Use c.l.d.PhasedBackoffWaitStrategy as available cpu (s) is 64

2022-12-26 16:54:37 [INFO] Create 4096 slots for ring buffer finished. [xxx.xxx.xxx.xxx:2881]

2022-12-26 16:54:37 [INFO] Start 128 database writer threads finished. [xxx.xxx.xxx.xxx:2881]

2022-12-26 16:54:37 [INFO] Start 128 csv file reader threads succeeded

2022-12-26 16:54:37 [INFO] File: "/home/admin/test\_migrate\_data/csv\_data/data/OBLOADER\_USER01/TABLE/TEST\_TBL1.1.0.csv" has been parsed finished

2022-12-26 16:54:37 [INFO] File: "/home/admin/test\_migrate\_data/csv\_data/data/OBLOADER\_USER01/TABLE/TEST\_TBL2.1.0.csv" has been parsed finished

2022-12-26 16:54:38 [INFO] Wait for the all the workers to drain of published events then halt the workers

2022-12-26 16:54:38 [INFO] Close connection count: 37 of the DataSource. Key: xxx.xxx.xxx.xxx\_11532\_1709388295\_obloader\_user01

2022-12-26 16:54:38 [INFO] Shutdown task context finished

2022-12-26 16:54:38 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress: 100.00% -----

2022-12-26 16:54:38 [INFO]

All Load Tasks Finished:

-----				
-----				
No.#	Type	Name	Count	Status
-----				
-----				
1	TABLE	TEST_TBL2	5 -> 5	SUCCESS

```
2 | TABLE | TEST_TBL1 | 5 -> 5 | SUCCESS
```

```
-----  
-----  
  
Total Count: 10 End Time: 2022-12-26 16:54:38
```

```
2022-12-26 16:54:38 [INFO] Load record finished. Total Elapsed: 2.322 s
```

```
2022-12-26 16:54:38 [INFO] System exit 0
```

- 检查导入的表数据。

```
obclient [OBLOADER_USER01]> SELECT * FROM TEST_TBL1;
```

```
+-----+-----+-----+
```

```
| COL1 | COL2 | COL3 |
```

```
+-----+-----+-----+
```

```
| 1 | China | 86 |
```

```
| 2 | Taiwan | 886 |
```

```
| 3 | Hong Kong | 852 |
```

```
| 4 | Macao | 853 |
```

```
| 5 | North Korea | 850 |
```

```
+-----+-----+-----+
```

```
5 rows in set
```

## 39 使用 OceanBase 导数工具在 OceanBase 集群 Oracle 租户间迁移数据

本文将介绍如何使用 OceanBase 导数工具从 OceanBase 集群 Oracle 租户迁移数据至 Oracle 租户。

### 39.1 OceanBase 导数工具简介

OceanBase 导数工具包括导出工具 obdumper 和导入工具 obloader。

- obdumper 是一款使用 Java 开发的客户端导出工具。可以使用该工具将 OceanBase 数据库中定义的对象和数据导出到文件中。有关 obdumper 的详细信息请参见 [obdumper 简介](#)。
- obloader 是一款使用 Java 开发的客户端导入工具。该工具提供了非常灵活的命令行选项，可在多种复杂的场景下，将定义和数据导入到 OceanBase 中。有关 obloader 的详细信息请参见 [obloader 简介](#)。

### 39.2 操作步骤

1. 准备 obdumper 和 obloader 运行环境。详见 [准备环境](#) 和 [下载工具](#)。
2. 创建存放导出数据的目录。
3. 导出表结构/数据。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h <主机IP> -P <端#> -u <##> -p <密码> [--sys-user <sys 租户下用户名> --sys-password <sys 租户下特定用户的密码>] -c <集群名称> -t <租#> -D <Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

#### 39.2.0.1 说明

- 使用 OceanBase 物理节点的主机地址和端口导出数据时，无需指定 `-c` 选项。
  - 有关 obdumper 命令行选项的详细信息，请参见 [命令行选项](#)。
4. 导入表结构/数据。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h <主机IP> -P <端#> -u <##> -p <密码> --sys-user <sys 租户下的 root
```

```
用户或 proxyro 用户> --sys-password <sys 租户下的账号#密码> -c <集群> -t <租户> -D
<Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

## 39.2.0.2 说明

- 使用 OceanBase 物理节点的主机地址和端口导入数据时，无需指定 `-c` 选项。
- 有关 obloader 命令行选项的详细信息，请参见 [命令行选项](#)。

## 39.3 导出和导入数据示例

下表为示例中使用的数据库信息：

数据库信息	示例值
集群名	test4000
OceanBase DataBase Proxy (ODP) 主机地址	xxx.xxx.xxx.xxx
OceanBase DataBase Proxy (ODP) 端口号	2883
sys 租户下 root 用户的密码	*****
源租户名称（Oracle 模式）	oracle001
租户 oracle001 的用户账号（要求读写权限）	obdumper_user01
租户 oracle001 下用户 obdumper_user01 的密码	*****
租户 oracle001 下 Schema 库名称	OBDUMPER_USER01
目标租户名称（Oracle 模式）	oracle002
租户 oracle002 的用户账号（要求读写权限）	obloader_user01
租户 oracle002 下用户 obloader_user01 的密码	*****
租户 oracle002 下 Schema 库名称	OBLOADER_USER01

### 39.3.1 表结构迁移

使用 obdumper 和 obloader 进行表结构迁移时，需要指定 `--sys-user` 和 `--sys-password` 选项，获取表结构元数据信息。如果未指定该选项，导出/导入功能和性能可能会受到较大的影响。

#### 39.3.1.1 说明

`--sys-user` 选项用于连接 `sys` 租户下拥有特定权限的用户。导出时如果未指定 `--sys-user` 选项，默认指定的是 `--sys-user root`。

## 39.3.1.2 导出 DDL 定义文件

场景描述：从集群 `test4000` 下的租户 `oracle001` 里的 Schema `OBDUMPER_USER01` 中，导出所有已支持的对象定义语句到 `/home/admin/test_migrate_data/ddl_data` 目录。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h xxx.xxx.xxx.xxx -P 2883 -u obdumper_user01 -p ***** -c test4000 -
t oracle001 -D obdumper_user01 --sys-user root --sys-password ***** --ddl --all -
f /home/admin/test_migrate_data/ddl_data
2022-12-26 18:41:07 [INFO] Parsed args:
[--ddl] true
[--file-path] /home/admin/test_migrate_data/ddl_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obdumper_user01
[--tenant] oracle001
[--cluster] test4000
[--password] *****
[--database] obdumper_user01
[--sys-user] root
[--sys-password] *****
[--all] true

2022-12-26 18:41:07 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-26 18:41:07 [INFO] The manifest file: "/home/admin/test_migrate_data
```



```
/ddl_data/data/MANIFEST.bin" has been saved
2022-12-26 18:41:07 [INFO] Query column metadata for the table: "TEST_TBL1"
finished
2022-12-26 18:41:07 [INFO] Query column metadata for the table: "TEST_TBL2"
finished
2022-12-26 18:41:07 [INFO] Found 2 empty tables before dump out records.
Elapsed: 11.61 ms
2022-12-26 18:41:07 [WARN] No views are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No triggers are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No functions are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No procedures are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No types are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No type bodies are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No packages are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No package bodies are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No synonyms are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No public synonyms are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No sequences are exist in the schema:
"OBDUMPER_USER01"
```

```
2022-12-26 18:41:07 [WARN] No table groups are exist in the tenant: "oracle001"
2022-12-26 18:41:07 [INFO] Generate 1 dump tasks finished. Total Elapsed: 3.876
ms
2022-12-26 18:41:07 [INFO] Start 3 schema dump threads for 1 dump tasks
finished.
2022-12-26 18:41:07 [INFO] Build direct com.alibaba.druid.pool.DruidDataSource
finished
2022-12-26 18:41:07 [INFO] No need to acquire DataSource for xxx@sys, as
observer is 4.0.0.0
2022-12-26 18:41:07 [INFO] Return the latest compatible version: 4.0.0.0 -> 4.0.0.0
2022-12-26 18:41:07 [INFO] Dump create objects success. DbType: OBORACLE
Version: 4.0.0.0
2022-12-26 18:41:07 [INFO] ObOracle(4.0.0.0) is older than 4.0.0.0 ? false
2022-12-26 18:41:07 [INFO] Load meta/oboracle/oboracle22x.xml, meta/oboracle
/oboracle40x.xml succeeded
2022-12-26 18:41:07 [INFO] Query 0 dependencies elapsed 41.82 ms
2022-12-26 18:41:08 [INFO] Query table: "TEST_TBL1" attr finished. Remain: 0
2022-12-26 18:41:08 [INFO] Query table: "TEST_TBL2" attr finished. Remain: 0
2022-12-26 18:41:08 [INFO] Query 2 tables elapsed 285.4 ms
2022-12-26 18:41:08 [INFO] Dump [TABLE] TEST_TBL1 to "/home/admin
/test_migrate_data/ddl_data/data/OBDUMPER_USER01/TABLE/TEST_TBL1-
schema.sql " finished
2022-12-26 18:41:08 [INFO] Dump [TABLE] TEST_TBL2 to "/home/admin
/test_migrate_data/ddl_data/data/OBDUMPER_USER01/TABLE/TEST_TBL2-
schema.sql " finished
2022-12-26 18:41:08 [INFO] No.1 It has dumped 2 TABLEs finished. Remain: 0
2022-12-26 18:41:08 [INFO] Total dumped 2 TABLEs finished. Elapsed: 376.1 ms
2022-12-26 18:41:08 [INFO] Dump the ddl of schema: "OBDUMPER_USER01"
finished
```

```
2022-12-26 18:41:08 [INFO] Close connection count: 19 of the DataSource. Key:
xxx.xxx.xxx.xxx_11532_840547833_obdumper_user01
2022-12-26 18:41:08 [INFO] Shutdown task context finished
2022-12-26 18:41:08 [INFO] ----- Finished Tasks: 1 Running Tasks: 0 Progress:
100.00% -----
2022-12-26 18:41:08 [INFO]
```

All Dump Tasks Finished:

```
-----
-----
No.# | Type | Name | Count | Status
-----
-----
```

```
1 | TABLE | TEST_TBL2 | 1 | SUCCESS
```

```
2 | TABLE | TEST_TBL1 | 1 | SUCCESS
-----
-----
```

Total Count: 2 End Time: 2022-12-26 18:41:08

```
2022-12-26 18:41:08 [INFO] Dump schema finished. Total Elapsed: 1.070 s
```

```
2022-12-26 18:41:08 [INFO] Unnecessary to upload the data files to the remote
cloud storage service
```

```
2022-12-26 18:41:08 [INFO] System exit 0
```

- 导出目录结构如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$tree /home/admin/test_migrate_data/ddl_data
```

```
/home/admin/test_migrate_data/ddl_data
```

```
|—— data
| |—— CHECKPOINT.bin
| |—— MANIFEST.bin
| |—— OBDUMPER_USER01
| |—— TABLE
| |—— TEST_TBL1-schema.sql
| |—— TEST_TBL2-schema.sql
|—— logs
|—— ob-loader-dumper.error
|—— ob-loader-dumper.info
|—— ob-loader-dumper.warn
```

```
4 directories, 7 files
```

- 查看导出结果文件。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$cat /home/admin/test_migrate_data/ddl_data/data/OBDUMPER_USER01/TABLE
/TEST_TBL1-schema.sql
CREATE TABLE "TEST_TBL1" (
"COL1" NUMBER(11,0) NOT NULL,
"COL2" VARCHAR2(20 BYTE),
"COL3" NUMBER(11,0),
CONSTRAINT "TEST_TBL1_OBP_1672051192536124" PRIMARY KEY ("COL1")
);
```

### 39.3.1.3 导入 DDL 定义文件

场景描述：将从集群 test4000 下的租户 oracle001 里的 Schema OBDUMPER\_USER01 导出到 /home/admin/test\_migrate\_data/ddl\_data 目录下所有已支持的定义信息，导入到集群 test4000 下的租户 oracle002 里的 Schema OBLOADER\_USER01 中。

- 示例语句如下：

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h xxx.xxx.xxx.xxx -P 2883 -u obloader_user01 -p ***** --sys-user
root --sys-password ***** -c test4000 -t oracle002 -D obloader_user01 --ddl --all -
f /home/admin/test_migrate_data/ddl_data
2022-12-26 18:48:20 [INFO] Parsed args:
[--ddl] true
[--file-path] /home/admin/test_migrate_data/ddl_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obloader_user01
[--tenant] oracle002
[--cluster] test4000
[--password] *****
[--database] obloader_user01
[--sys-user] root
[--sys-password] *****
[--all] true

2022-12-26 18:48:20 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-26 18:48:20 [INFO] The manifest file: "/home/admin/test_migrate_data
/ddl_data/data/MANIFEST.bin" has been saved
2022-12-26 18:48:20 [INFO] Init writer thread pool finished
2022-12-26 18:48:20 [WARN] The object type : "SEQUENCE" doesn't exist in the -
schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "TABLE_GROUP" doesn't exist in the -
schema.sql files
2022-12-26 18:48:20 [INFO] Start 128 schema file loader threads succeeded
```

```
2022-12-26 18:48:20 [INFO] No.1 sql of the file: "/home/admin/test_migrate_data
/ddl_data/data/OBDUMPER_USER01/TABLE/TEST_TBL1-schema.sql" exec success.
Elapsed: 124.9 ms
2022-12-26 18:48:20 [INFO] Load file: "TEST_TBL1-schema.sql" succeeded
2022-12-26 18:48:20 [INFO] No.1 sql of the file: "/home/admin/test_migrate_data
/ddl_data/data/OBDUMPER_USER01/TABLE/TEST_TBL2-schema.sql" exec success.
Elapsed: 182.3 ms
2022-12-26 18:48:20 [INFO] Load file: "TEST_TBL2-schema.sql" succeeded
2022-12-26 18:48:20 [WARN] The object type : "VIEW" doesn't exist in the -schema.
sql files
2022-12-26 18:48:20 [WARN] The object type : "FUNCTION" doesn't exist in the -
schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "PROCEDURE" doesn't exist in the -
schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "TRIGGER" doesn't exist in the -
schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "PACKAGE" doesn't exist in the -
schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "TYPE" doesn't exist in the -schema.
sql files
2022-12-26 18:48:20 [WARN] The object type : "PACKAGE_BODY" doesn't exist in
the -schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "TYPE_BODY" doesn't exist in the -
schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "SYNONYM" doesn't exist in the -
schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "PUBLIC_SYNONYM" doesn't exist in
the -schema.sql files
2022-12-26 18:48:20 [WARN] The object type : "FILE" doesn't exist in the -schema.
```

sql files

2022-12-26 18:48:21 [INFO] Close connection count: 1 of the DataSource. Key: xxx.  
xxx.xxx.xxx\_11532\_1905901800\_obloader\_user01

2022-12-26 18:48:21 [INFO] Shutdown task context finished

2022-12-26 18:48:21 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress:  
100.00% -----

2022-12-26 18:48:21 [INFO]

All Load Tasks Finished:

-----  
-----  
No.# | Type | Name | Count | Status  
-----  
-----

1 | TABLE | TEST\_TBL2 | 1 -> 1 | SUCCESS

2 | TABLE | TEST\_TBL1 | 1 -> 1 | SUCCESS  
-----  
-----

Total Count: 2 End Time: 2022-12-26 18:48:21

2022-12-26 18:48:21 [INFO] Load schema finished. Total Elapsed: 1.064 s

2022-12-26 18:48:21 [INFO] System exit 0

- 检查导入的表结构。

obclient [OBLOADER\_USER01]> SHOW CREATE TABLE test\_tbl1\G

\*\*\*\*\* 1. row \*\*\*\*\*

TABLE: TEST\_TBL1

```
CREATE TABLE: CREATE TABLE "TEST_TBL1" (  
  "COL1" NUMBER(11) CONSTRAINT "TEST_TBL1_OBNOTNULL_1672051700809789"  
  NOT NULL ENABLE,  
  "COL2" VARCHAR2(20),  
  "COL3" NUMBER(11),  
  CONSTRAINT "TEST_TBL1_OBPK_1672051192536124" PRIMARY KEY ("COL1")  
) COMPRESS FOR ARCHIVE REPLICA_NUM = 3 BLOCK_SIZE = 16384  
  USE_BLOOM_FILTER = FALSE TABLET_SIZE = 134217728 PCTFREE = 0  
1 row in set
```

## 39.3.2 表数据迁移

以下将以导出 CSV 数据文件和导入 CSV 数据文件为例进行介绍使用 obdumper 和 obloader 进行表数据迁移。

### 39.3.2.4 导出 CSV 数据文件

文件类型定义：CSV 数据文件（后缀名 .csv）是逗号分隔值文件格式，CSV 数据文件以纯文本形式存储表格数据，可通过文本编辑器等工具或者 Excel 打开。

场景描述：从集群 test4000 下的租户 oracle001 里的 Schema OBDUMPER\_USER01 中，导出所有已支持对象的数据到 /home/admin/test\_migrate\_data/csv\_data 目录，数据格式为 CSV 格式。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]  
$./obdumper -h xxx.xxx.xxx.xxx -P 2883 -u obdumper_user01 -p ***** -c test4000 -  
t oracle001 -D obdumper_user01 --sys-user root --sys-password ***** --csv --  
table '*' -f /home/admin/test_migrate_data/csv_data  
2022-12-27 10:02:20 [INFO] Parsed args:  
[--csv] true  
[--file-path] /home/admin/test_migrate_data/csv_data  
[--host] xxx.xxx.xxx.xxx  
[--port] 2883
```



```
[--user] obdumper_user01
```

```
[--tenant] oracle001
```

```
[--cluster] test4000
```

```
[--password] *****
```

```
[--database] obdumper_user01
```

```
[--sys-user] root
```

```
[--sys-password] *****
```

```
[--table] [*]
```

```
2022-12-27 10:02:20 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
```

```
2022-12-27 10:02:21 [INFO] The manifest file: "/home/admin/test_migrate_data
/csv_data/data/MANIFEST.bin" has been saved
```

```
2022-12-27 10:02:21 [INFO] Query column metadata for the table: "TEST_TBL1"
finished
```

```
2022-12-27 10:02:21 [INFO] Query column metadata for the table: "TEST_TBL2"
finished
```

```
2022-12-27 10:02:21 [INFO] Query partition names for table: "TEST_TBL1" success.
(Non-partitioned)
```

```
2022-12-27 10:02:21 [INFO] Query partition names for table: "TEST_TBL2" success.
(Non-partitioned)
```

```
2022-12-27 10:02:21 [INFO] Query primary key for table: "TEST_TBL1" success.
Elapsed: 160.8 ms
```

```
2022-12-27 10:02:21 [INFO] Query primary key for table: "TEST_TBL2" success.
Elapsed: 160.8 ms
```

```
2022-12-27 10:02:21 [INFO] Query table entry for table: "TEST_TBL2" success.
Remain: 0. Elapsed: 4.120 ms
```

```
2022-12-27 10:02:21 [INFO] Query table entry for table: "TEST_TBL1" success.
Remain: 0. Elapsed: 4.203 ms
```

```
2022-12-27 10:02:21 [INFO] Query all table entries success. Total: 2. Elapsed:
208.0 ms
2022-12-27 10:02:21 [INFO] ....Splitting rows for non-partitioned table:
"TEST_TBL2" success. Batch: 1
2022-12-27 10:02:21 [INFO] ....Splitting rows for non-partitioned table:
"TEST_TBL1" success. Batch: 1
2022-12-27 10:02:21 [INFO] ....Splitting rows for non-partitioned table:
"TEST_TBL2" success. Batch: 2
2022-12-27 10:02:21 [INFO] ....Splitting rows for non-partitioned table:
"TEST_TBL1" success. Batch: 2
2022-12-27 10:02:21 [INFO] Split rows for non-partitioned table(with primary key):
"TEST_TBL2" success. Ranges: 2. Elapsed: 48.60 ms
2022-12-27 10:02:21 [INFO] Split rows for non-partitioned table(with primary key):
"TEST_TBL1" success. Ranges: 2. Elapsed: 48.60 ms
2022-12-27 10:02:21 [INFO] Generate 4 dump tasks finished. Total Elapsed: 62.72
ms
2022-12-27 10:02:21 [INFO] Start 128 record dump threads for 4 dump tasks
finished
2022-12-27 10:02:21 [INFO] Dump 5 rows OBDUMPER_USER01.TEST_TBL2 to "/home
/admin/test_migrate_data/csv_data/data/OBDUMPER_USER01/TABLE/TEST_TBL2.
1.*.csv" finished
2022-12-27 10:02:21 [INFO] Dump 5 rows OBDUMPER_USER01.TEST_TBL1 to "/home
/admin/test_migrate_data/csv_data/data/OBDUMPER_USER01/TABLE/TEST_TBL1.
1.*.csv" finished
2022-12-27 10:02:22 [INFO] Close connection count: 12 of the DataSource. Key:
xxx.xxx.xxx.xxx_11532_840547833_obdumper_user01
2022-12-27 10:02:22 [INFO] Shutdown task context finished
2022-12-27 10:02:22 [INFO] ----- Finished Tasks: 4 Running Tasks: 0 Progress:
100.00% -----
```

```
2022-12-27 10:02:22 [INFO]
```

```
All Dump Tasks Finished:
```

```
-----  
-----  
No.# | Type | Name | Count | Status  
-----  
-----
```

```
1 | TABLE | TEST_TBL2 | 5 | SUCCESS
```

```
2 | TABLE | TEST_TBL1 | 5 | SUCCESS  
-----  
-----
```

```
Total Count: 10 End Time: 2022-12-27 10:02:22
```

```
2022-12-27 10:02:22 [INFO] Unnecessary to merge the data files. As --file-name is  
missing
```

```
2022-12-27 10:02:22 [INFO] Dump record finished. Total Elapsed: 1.356 s
```

```
2022-12-27 10:02:22 [INFO] Unnecessary to upload the data files to the remote  
cloud storage service
```

```
2022-12-27 10:02:22 [INFO] System exit 0
```

- 导出目录结构如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
```

```
$tree /home/admin/test_migrate_data/csv_data
```

```
/home/admin/test_migrate_data/csv_data
```

```
|—— data
```

```
| |—— CHECKPOINT.bin
```

```
| |—— MANIFEST.bin
| |—— OBDUMPER_USER01
| |—— TABLE
| |—— TEST_TBL1.1.0.csv
| |—— TEST_TBL2.1.0.csv
|—— logs
|—— ob-loader-dumper.error
|—— ob-loader-dumper.info
|—— ob-loader-dumper.warn
```

4 directories, 7 files

- 查看导出结果文件。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$cat /home/admin/test_migrate_data/csv_data/data/OBDUMPER_USER01/TABLE
/TEST_TBL1.1.0.csv
'COL1','COL2','COL3'
1,'China',86
2,'Taiwan',886
3,'Hong Kong',852
4,'Macao',853
5,'North Korea',850
```

### 39.3.2.5 导入 CSV 数据文件

场景描述：将从集群 test4000 下的租户 oracle001 里的 Schema OBDUMPER\_USER01 导出到 /home/admin/test\_migrate\_data/csv\_data 目录下所有已支持的 CSV 数据文件，导入到集群 test4000 下的租户 oracle002 里的 Schema OBLOADER\_USER01 中。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h xxx.xxx.xxx.xxx -P 2883 -u obloader_user01 -p ***** --sys-user
root --sys-password ***** -c test4000 -t oracle002 -D obloader_user01 --csv --
table '*' -f /home/admin/test_migrate_data/csv_data
2022-12-27 10:10:36 [INFO] Parsed args:
[--csv] true
[--file-path] /home/admin/test_migrate_data/csv_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obloader_user01
[--tenant] oracle002
[--cluster] test4000
[--password] *****
[--database] obloader_user01
[--sys-user] root
[--sys-password] *****
[--table] [*]

2022-12-27 10:10:36 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-27 10:10:37 [INFO] The manifest file: "/home/admin/test_migrate_data
/csv_data/data/MANIFEST.bin" has been saved
2022-12-27 10:10:37 [INFO] Query column metadata for the table: "TEST_TBL1"
finished
2022-12-27 10:10:37 [INFO] Query column metadata for the table: "TEST_TBL2"
finished
2022-12-27 10:10:37 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
/CHECKPOINT.bin" is unmatched on the suffix[.csv], ignore it
2022-12-27 10:10:37 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
```

```
/MANIFEST.bin" is unmatched on the suffix[.csv], ignore it
2022-12-27 10:10:37 [INFO] Binding table: "TEST_TBL2" to the file: "/home/admin
/test_migrate_data/csv_data/data/OBDUMPER_USER01/TABLE/TEST_TBL2.1.0.csv"
finished
2022-12-27 10:10:37 [INFO] Binding table: "TEST_TBL1" to the file: "/home/admin
/test_migrate_data/csv_data/data/OBDUMPER_USER01/TABLE/TEST_TBL1.1.0.csv"
finished
2022-12-27 10:10:37 [INFO] Splitted 2 csv subfiles by 64.0 MB. Elapsed: 19.13 ms
2022-12-27 10:10:37 [INFO] Generate 2 subfiles finished
2022-12-27 10:10:37 [INFO] Ignore to clean any tables as --truncate-table or --
delete-from-table is not specified
2022-12-27 10:10:37 [INFO] Query table entry and primary key for table:
"TEST_TBL2" finished. Remain: 1
2022-12-27 10:10:37 [INFO] Query table entry and primary key for table:
"TEST_TBL1" finished. Remain: 0
2022-12-27 10:10:37 [INFO] Query the leader location of "TEST_TBL1" finished.
Remain: 0
2022-12-27 10:10:37 [INFO] Query the leader location of "TEST_TBL2" finished.
Remain: 0
2022-12-27 10:10:37 [INFO] Calculate leader: xxx.xxx.xxx.xxx:2881 of table:
"TEST_TBL2", part: 0. Remain: 0
2022-12-27 10:10:37 [INFO] Calculate leader: xxx.xxx.xxx.xxx:2881 of table:
"TEST_TBL1", part: 0. Remain: 1
2022-12-27 10:10:37 [INFO] Waiting to refresh observer load status .....
2022-12-27 10:10:38 [INFO] Refresh observer load status success. Table:
"TEST_TBL2". Remain: 1
2022-12-27 10:10:38 [INFO] Refresh observer load status success. Table:
"TEST_TBL1". Remain: 0
2022-12-27 10:10:38 [INFO] Refresh observer load status finished. Elapsed: 36.97
```

ms

2022-12-27 10:10:38 [INFO] Use c.l.d.PhasedBackoffWaitStrategy as available cpu (s) is 64

2022-12-27 10:10:38 [INFO] Create 4096 slots for ring buffer finished. [xxx.xxx.xxx.xxx:2881]

2022-12-27 10:10:38 [INFO] Start 128 database writer threads finished. [xxx.xxx.xxx.xxx:2881]

2022-12-27 10:10:38 [INFO] Start 128 csv file reader threads succeeded

2022-12-27 10:10:38 [INFO] File: "/home/admin/test\_migrate\_data/csv\_data/data/OBDUMPER\_USER01/TABLE/TEST\_TBL2.1.0.csv" has been parsed finished

2022-12-27 10:10:38 [INFO] File: "/home/admin/test\_migrate\_data/csv\_data/data/OBDUMPER\_USER01/TABLE/TEST\_TBL1.1.0.csv" has been parsed finished

2022-12-27 10:10:39 [INFO] Wait for the all the workers to drain of published events then halt the workers

2022-12-27 10:10:39 [INFO] Close connection count: 37 of the DataSource. Key: xxx.xxx.xxx.xxx\_11532\_1905901800\_obloader\_user01

2022-12-27 10:10:39 [INFO] Shutdown task context finished

2022-12-27 10:10:39 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress: 100.00% -----

2022-12-27 10:10:39 [INFO]

All Load Tasks Finished:

-----  
-----  
No.# | Type | Name | Count | Status  
-----  
-----

1 | TABLE | TEST\_TBL2 | 5 -> 5 | SUCCESS

```
2 | TABLE | TEST_TBL1 | 5 -> 5 | SUCCESS
```

```
-----  
-----  
  
Total Count: 10 End Time: 2022-12-27 10:10:39
```

```
2022-12-27 10:10:39 [INFO] Load record finished. Total Elapsed: 2.583 s
```

```
2022-12-27 10:10:39 [INFO] System exit 0
```

- 检查导入的表数据。

```
obclient [OBLOADER_USER01]> SELECT * FROM test_tbl1;
```

```
+-----+-----+-----+
```

```
| COL1 | COL2 | COL3 |
```

```
+-----+-----+-----+
```

```
| 1 | China | 86 |
```

```
| 2 | Taiwan | 886 |
```

```
| 3 | Hong Kong | 852 |
```

```
| 4 | Macao | 853 |
```

```
| 5 | North Korea | 850 |
```

```
+-----+-----+-----+
```

```
5 rows in set
```



## 40 使用 OceanBase 导数工具从 OceanBase 集群 Oracle 租户迁移数据到 MySQL 租户

本文将介绍如何使用 OceanBase 导数工具从 OceanBase 数据库 Oracle 租户迁移数据到 MySQL 租户。

### 40.1 OceanBase 导数工具简介

OceanBase 导数工具包括导出工具 obdumper 和导入工具 obloader。

- obdumper 是一款使用 Java 开发的客户端导出工具。可以使用该工具将 OceanBase 数据库中定义的对象和数据导出到文件中。有关 obdumper 的详细信息请参见 [obdumper 简介](#)。
- obloader 是一款使用 Java 开发的客户端导入工具。该工具提供了非常灵活的命令行选项，可在多种复杂的场景下，将定义和数据导入到 OceanBase 中。有关 obloader 的详细信息请参见 [obloader 简介](#)。

### 40.2 操作步骤

1. 准备 obdumper 和 obloader 运行环境。详见 [准备环境](#) 和 [下载工具](#)。
2. 创建存放导出数据的目录。
3. 导出表结构/数据。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h <主机IP> -P <端#> -u <##> -p <密码> [--sys-user <sys 租户下用户名> --sys-password <sys 租户下特定用户的密码>] -c <集群名称> -t <租#> -D <Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

#### 40.2.0.1 说明

- 使用 OceanBase 物理节点的主机地址和端口导出数据时，无需指定 `-c` 选项。
  - 有关 obdumper 命令行选项的详细信息，请参见 [命令行选项](#)。
4. 导入表结构/数据。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h <主机IP> -P <端#> -u <##> -p <密码> --sys-user <sys 租户下的 root
```

```
用户或 proxyro 用户> --sys-password <sys 租户下的账号#密码> -c <集群> -t <租户> -D
<Schema 库名> [--ddl] [--csv|--sql] [--all|--table '表名'] -f <数据#件或者#录>
```

## 40.2.0.2 说明

- 使用 OceanBase 物理节点的主机地址和端口导入数据时，无需指定 `-c` 选项。
- 有关 obloader 命令行选项的详细信息，请参见 [命令行选项](#)。

## 40.3 导出和导入数据示例

下表为示例中使用的数据库信息：

数据库信息	示例值
集群名	test4000
OceanBase DataBase Proxy (ODP) 主机地址	xxx.xxx.xxx.xxx
OceanBase DataBase Proxy (ODP) 端口号	2883
sys 租户下 root 用户的密码	*****
源租户名称（Oracle 模式）	oracle001
租户 oracle001 的用户账号（要求读写权限）	obdumper_user01
租户 oracle001 下用户 obdumper_user01 的密码	*****
租户 oracle001 下 Schema 库名称	OBDUMPER_USER01
目标租户名称（MySQL 模式）	mysql001
租户 mysql001 的用户账号（要求读写权限）	obloader_user01
租户 mysql001 下用户 obloader_user01 的密码	*****
租户 mysql001 下 Schema 库名称	test_data

### 40.3.1 表结构迁移

使用 obdumper 和 obloader 进行表结构迁移时，需要指定 `--sys-user` 和 `--sys-password` 选项，获取表结构元数据信息。如果未指定该选项，导出/导入功能和性能可能会受到较大的影响。

#### 40.3.1.1 说明

`--sys-user` 选项用于连接 `sys` 租户下拥有特定权限的用户。导出时如果未指定 `--sys-user` 选项，默认指定的是 `--sys-user root`。

## 40.3.1.2 导出 DDL 定义文件

场景描述：从集群 `test4000` 下的租户 `oracle001` 里的 Schema `OBDUMPER_USER01` 中，导出所有已支持的对象定义语句到 `/home/admin/test_migrate_data/ddl_data` 目录。

● 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h xxx.xxx.xxx.xxx -P 2883 -u obdumper_user01 -p ***** -c test4000 -
t oracle001 -D obdumper_user01 --sys-user root --sys-password ***** --ddl --all -
f /home/admin/test_migrate_data/ddl_data
2022-12-26 18:41:07 [INFO] Parsed args:
[--ddl] true
[--file-path] /home/admin/test_migrate_data/ddl_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obdumper_user01
[--tenant] oracle001
[--cluster] test4000
[--password] *****
[--database] obdumper_user01
[--sys-user] root
[--sys-password] *****
[--all] true

2022-12-26 18:41:07 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-26 18:41:07 [INFO] The manifest file: "/home/admin/test_migrate_data
```

```
/ddl_data/data/MANIFEST.bin" has been saved
2022-12-26 18:41:07 [INFO] Query column metadata for the table: "TEST_TBL1"
finished
2022-12-26 18:41:07 [INFO] Query column metadata for the table: "TEST_TBL2"
finished
2022-12-26 18:41:07 [INFO] Found 2 empty tables before dump out records.
Elapsed: 11.61 ms
2022-12-26 18:41:07 [WARN] No views are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No triggers are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No functions are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No procedures are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No types are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No type bodies are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No packages are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No package bodies are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No synonyms are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No public synonyms are exist in the schema:
"OBDUMPER_USER01"
2022-12-26 18:41:07 [WARN] No sequences are exist in the schema:
"OBDUMPER_USER01"
```

```
2022-12-26 18:41:07 [WARN] No table groups are exist in the tenant: "oracle001"
2022-12-26 18:41:07 [INFO] Generate 1 dump tasks finished. Total Elapsed: 3.876
ms
2022-12-26 18:41:07 [INFO] Start 3 schema dump threads for 1 dump tasks
finished.
2022-12-26 18:41:07 [INFO] Build direct com.alibaba.druid.pool.DruidDataSource
finished
2022-12-26 18:41:07 [INFO] No need to acquire DataSource for xxx@sys, as
observer is 4.0.0.0
2022-12-26 18:41:07 [INFO] Return the latest compatible version: 4.0.0.0 -> 4.0.0.0
2022-12-26 18:41:07 [INFO] Dump create objects success. DbType: OBORACLE
Version: 4.0.0.0
2022-12-26 18:41:07 [INFO] ObOracle(4.0.0.0) is older than 4.0.0.0 ? false
2022-12-26 18:41:07 [INFO] Load meta/oboracle/oboracle22x.xml, meta/oboracle
/oboracle40x.xml succeeded
2022-12-26 18:41:07 [INFO] Query 0 dependencies elapsed 41.82 ms
2022-12-26 18:41:08 [INFO] Query table: "TEST_TBL1" attr finished. Remain: 0
2022-12-26 18:41:08 [INFO] Query table: "TEST_TBL2" attr finished. Remain: 0
2022-12-26 18:41:08 [INFO] Query 2 tables elapsed 285.4 ms
2022-12-26 18:41:08 [INFO] Dump [TABLE] TEST_TBL1 to "/home/admin
/test_migrate_data/ddl_data/data/OBDUMPER_USER01/TABLE/TEST_TBL1-
schema.sql " finished
2022-12-26 18:41:08 [INFO] Dump [TABLE] TEST_TBL2 to "/home/admin
/test_migrate_data/ddl_data/data/OBDUMPER_USER01/TABLE/TEST_TBL2-
schema.sql " finished
2022-12-26 18:41:08 [INFO] No.1 It has dumped 2 TABLEs finished. Remain: 0
2022-12-26 18:41:08 [INFO] Total dumped 2 TABLEs finished. Elapsed: 376.1 ms
2022-12-26 18:41:08 [INFO] Dump the ddl of schema: "OBDUMPER_USER01"
finished
```

```
2022-12-26 18:41:08 [INFO] Close connection count: 19 of the DataSource. Key:
xxx.xxx.xxx.xxx_11532_840547833_obdumper_user01
2022-12-26 18:41:08 [INFO] Shutdown task context finished
2022-12-26 18:41:08 [INFO] ----- Finished Tasks: 1 Running Tasks: 0 Progress:
100.00% -----
2022-12-26 18:41:08 [INFO]
```

All Dump Tasks Finished:

```
-----
-----
No.# | Type | Name | Count | Status
-----
-----
```

```
1 | TABLE | TEST_TBL2 | 1 | SUCCESS
```

```
2 | TABLE | TEST_TBL1 | 1 | SUCCESS
-----
-----
```

Total Count: 2 End Time: 2022-12-26 18:41:08

```
2022-12-26 18:41:08 [INFO] Dump schema finished. Total Elapsed: 1.070 s
```

```
2022-12-26 18:41:08 [INFO] Unnecessary to upload the data files to the remote
cloud storage service
```

```
2022-12-26 18:41:08 [INFO] System exit 0
```

- 导出目录结构如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$tree /home/admin/test_migrate_data/ddl_data
```

```
/home/admin/test_migrate_data/ddl_data
```

```
|—— data
| |—— CHECKPOINT.bin
| |—— MANIFEST.bin
| |—— OBDUMPER_USER01
| |—— TABLE
| |—— TEST_TBL1-schema.sql
| |—— TEST_TBL2-schema.sql
|—— logs
|—— ob-loader-dumper.error
|—— ob-loader-dumper.info
|—— ob-loader-dumper.warn
```

```
4 directories, 7 files
```

- 查看导出结果文件。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$cat /home/admin/test_migrate_data/ddl_data/data/OBDUMPER_USER01/TABLE
/TEST_TBL1-schema.sql
CREATE TABLE "TEST_TBL1" (
"COL1" NUMBER(11,0) NOT NULL,
"COL2" VARCHAR2(20 BYTE),
"COL3" NUMBER(11,0),
CONSTRAINT "TEST_TBL1_OBP_1672051192536124" PRIMARY KEY ("COL1")
);
```

### 40.3.1.3 导入 DDL 定义文件

场景描述：将从集群 test4000 下的租户 oracle001 里的 Schema OBDUMPER\_USER01 导出到 /home/admin/test\_migrate\_data/ddl\_data 目录下所有已支持的定义信息，导入到集群 test4000 下的租户 mysql001 里的 Schema test\_data 中。

## 40.3.1.4 注意

从 Oracle 租户里导出的 DDL 文件不符合 MySQL 租户的语法要求，所以从 Oracle 租户里导出的 DDL 文件先修改成符合 MySQL 租户的语法要求的 DDL 文件后，再导入 MySQL 租户中。

例如，从租户 `oracle001` 里导出的 DDL 文件，修改规则如下：

- 把数据类型修改为 MySQL 模式通用的数据类型：
  - 将 `NUMBER(11,0)` 改为 `int(11)`。
  - 将 `VARCHAR2(20 BYTE)` 改为 `varchar(20)`。
- 去掉 `CONSTRAINT "TEST_TBL1_OBPK_1672051192536124"` 参数。
- 去所有掉双引号。

有关 MySQL 模式数据类型的详细信息，请参见 [数据类型概述](#)。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h xxx.xxx.xxx.xxx -P 2883 -u obloader_user01 -p ***** --sys-user
root --sys-password ***** -c test4000 -t mysql001 -D test_data --ddl --all -f /home
/admin/test_migrate_data/ddl_data
2022-12-27 11:00:35 [INFO] Parsed args:
[--ddl] true
[--file-path] /home/admin/test_migrate_data/ddl_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obloader_user01
[--tenant] mysql001
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
```



```
[--all] true
```

```
2022-12-27 11:00:35 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
```

```
2022-12-27 11:00:35 [INFO] The manifest file: "/home/admin/test_migrate_data
/ddl_data/data/MANIFEST.bin" has been saved
```

```
2022-12-27 11:00:35 [INFO] Init writer thread pool finished
```

```
2022-12-27 11:00:35 [WARN] The object type : "SEQUENCE" doesn't exist in the -
schema.sql files
```

```
2022-12-27 11:00:35 [WARN] The object type : "TABLE_GROUP" doesn't exist in the -
schema.sql files
```

```
2022-12-27 11:00:35 [INFO] Start 128 schema file loader threads succeeded
```

```
2022-12-27 11:00:36 [INFO] No.1 sql of the file: "/home/admin/test_migrate_data
/ddl_data/data/OBDUMPER_USER01/TABLE/TEST_TBL1-schema.sql" exec success.
Elapsed: 70.51 ms
```

```
2022-12-27 11:00:36 [INFO] Load file: "TEST_TBL1-schema.sql" succeeded
```

```
2022-12-27 11:00:36 [INFO] No.1 sql of the file: "/home/admin/test_migrate_data
/ddl_data/data/OBDUMPER_USER01/TABLE/TEST_TBL2-schema.sql" exec success.
Elapsed: 118.7 ms
```

```
2022-12-27 11:00:36 [INFO] Load file: "TEST_TBL2-schema.sql" succeeded
```

```
2022-12-27 11:00:36 [WARN] The object type : "VIEW" doesn't exist in the -schema.
sql files
```

```
2022-12-27 11:00:36 [WARN] The object type : "FUNCTION" doesn't exist in the -
schema.sql files
```

```
2022-12-27 11:00:36 [WARN] The object type : "PROCEDURE" doesn't exist in the -
schema.sql files
```

```
2022-12-27 11:00:36 [WARN] The object type : "TRIGGER" doesn't exist in the -
schema.sql files
```

```
2022-12-27 11:00:36 [WARN] The object type : "PACKAGE" doesn't exist in the -
```

schema.sql files

2022-12-27 11:00:36 [WARN] The object type : "TYPE" doesn't exist in the -schema.sql files

2022-12-27 11:00:36 [WARN] The object type : "PACKAGE\_BODY" doesn't exist in the -schema.sql files

2022-12-27 11:00:36 [WARN] The object type : "TYPE\_BODY" doesn't exist in the -schema.sql files

2022-12-27 11:00:36 [WARN] The object type : "SYNONYM" doesn't exist in the -schema.sql files

2022-12-27 11:00:36 [WARN] The object type : "PUBLIC\_SYNONYM" doesn't exist in the -schema.sql files

2022-12-27 11:00:36 [WARN] The object type : "FILE" doesn't exist in the -schema.sql files

2022-12-27 11:00:37 [INFO] Close connection count: 1 of the DataSource. Key: xxx.xxx.xxx\_11532\_332378361\_test\_data

2022-12-27 11:00:37 [INFO] Shutdown task context finished

2022-12-27 11:00:37 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress: 100.00% -----

2022-12-27 11:00:37 [INFO]

All Load Tasks Finished:

-----  
-----  
No.# | Type | Name | Count | Status  
-----  
-----

1 | TABLE | TEST\_TBL2 | 1 -> 1 | SUCCESS

2 | TABLE | TEST\_TBL1 | 1 -> 1 | SUCCESS

```
-----  
-----  
  
Total Count: 2 End Time: 2022-12-27 11:00:37
```

```
2022-12-27 11:00:37 [INFO] Load schema finished. Total Elapsed: 1.073 s
```

```
2022-12-27 11:00:37 [INFO] System exit 0
```

- 检查导入的表结构。

```
obclient [test_data]> SHOW CREATE TABLE test_tbl1\G
```

```
***** 1. row *****
```

```
Table: test_tbl1
```

```
Create Table: CREATE TABLE `test_tbl1` (
```

```
`COL1` int(11) NOT NULL,
```

```
`COL2` varchar(20) DEFAULT NULL,
```

```
`COL3` int(11) DEFAULT NULL,
```

```
PRIMARY KEY (`COL1`)
```

```
) DEFAULT CHARSET = utf8mb4 ROW_FORMAT = DYNAMIC COMPRESSION = 'zstd_1.
```

```
3.8' REPLICA_NUM = 3 BLOCK_SIZE = 16384 USE_BLOOM_FILTER = FALSE
```

```
TABLET_SIZE = 134217728 PCTFREE = 0
```

```
1 row in set
```

## 40.3.2 表数据迁移

以下将以导出 CSV 数据文件和导入 CSV 数据文件为例进行介绍使用 obdumper 和 obloader 进行表数据迁移。

### 40.3.2.5 导出 CSV 数据文件

文件类型定义：CSV 数据文件（后缀名 `.csv`）是逗号分隔值文件格式，CSV 数据文件以纯文本形式存储表格数据，可通过文本编辑器等工具或者 Excel 打开。

场景描述：从集群 `test4000` 下的租户 `oracle001` 里的 Schema `OBDUMPER_USER01` 中，导出所有已支持对象的数据到 `/home/admin/test_migrate_data/csv_data` 目录，数据格式为 CSV 格式。

- 示例语句如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obdumper -h xxx.xxx.xxx.xxx -P 2883 -u obdumper_user01 -p ***** -c test4000 -
t oracle001 -D obdumper_user01 --sys-user root --sys-password ***** --csv --
table '*' -f /home/admin/test_migrate_data/csv_data
2022-12-27 10:02:20 [INFO] Parsed args:
[--csv] true
[--file-path] /home/admin/test_migrate_data/csv_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obdumper_user01
[--tenant] oracle001
[--cluster] test4000
[--password] *****
[--database] obdumper_user01
[--sys-user] root
[--sys-password] *****
[--table] [*]

2022-12-27 10:02:20 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-27 10:02:21 [INFO] The manifest file: "/home/admin/test_migrate_data
/csv_data/data/MANIFEST.bin" has been saved
2022-12-27 10:02:21 [INFO] Query column metadata for the table: "TEST_TBL1"
finished
2022-12-27 10:02:21 [INFO] Query column metadata for the table: "TEST_TBL2"
```

finished

2022-12-27 10:02:21 [INFO] Query partition names for table: "TEST\_TBL1" success.  
(Non-partitioned)

2022-12-27 10:02:21 [INFO] Query partition names for table: "TEST\_TBL2" success.  
(Non-partitioned)

2022-12-27 10:02:21 [INFO] Query primary key for table: "TEST\_TBL1" success.  
Elapsed: 160.8 ms

2022-12-27 10:02:21 [INFO] Query primary key for table: "TEST\_TBL2" success.  
Elapsed: 160.8 ms

2022-12-27 10:02:21 [INFO] Query table entry for table: "TEST\_TBL2" success.  
Remain: 0. Elapsed: 4.120 ms

2022-12-27 10:02:21 [INFO] Query table entry for table: "TEST\_TBL1" success.  
Remain: 0. Elapsed: 4.203 ms

2022-12-27 10:02:21 [INFO] Query all table entries success. Total: 2. Elapsed:  
208.0 ms

2022-12-27 10:02:21 [INFO] ....Splitting rows for non-partitioned table:  
"TEST\_TBL2" success. Batch: 1

2022-12-27 10:02:21 [INFO] ....Splitting rows for non-partitioned table:  
"TEST\_TBL1" success. Batch: 1

2022-12-27 10:02:21 [INFO] ....Splitting rows for non-partitioned table:  
"TEST\_TBL2" success. Batch: 2

2022-12-27 10:02:21 [INFO] ....Splitting rows for non-partitioned table:  
"TEST\_TBL1" success. Batch: 2

2022-12-27 10:02:21 [INFO] Split rows for non-partitioned table(with primary key):  
"TEST\_TBL2" success. Ranges: 2. Elapsed: 48.60 ms

2022-12-27 10:02:21 [INFO] Split rows for non-partitioned table(with primary key):  
"TEST\_TBL1" success. Ranges: 2. Elapsed: 48.60 ms

2022-12-27 10:02:21 [INFO] Generate 4 dump tasks finished. Total Elapsed: 62.72  
ms

```
2022-12-27 10:02:21 [INFO] Start 128 record dump threads for 4 dump tasks
finished
2022-12-27 10:02:21 [INFO] Dump 5 rows OBDUMPER_USER01.TEST_TBL2 to "/home
/admin/test_migrate_data/csv_data/data/OBDUMPER_USER01/TABLE/TEST_TBL2.
1.*.csv" finished
2022-12-27 10:02:21 [INFO] Dump 5 rows OBDUMPER_USER01.TEST_TBL1 to "/home
/admin/test_migrate_data/csv_data/data/OBDUMPER_USER01/TABLE/TEST_TBL1.
1.*.csv" finished
2022-12-27 10:02:22 [INFO] Close connection count: 12 of the DataSource. Key:
xxx.xxx.xxx.xxx_11532_840547833_obdumper_user01
2022-12-27 10:02:22 [INFO] Shutdown task context finished
2022-12-27 10:02:22 [INFO] ----- Finished Tasks: 4 Running Tasks: 0 Progress:
100.00% -----
2022-12-27 10:02:22 [INFO]
```

All Dump Tasks Finished:

```
-----
-----
No.# | Type | Name | Count | Status
-----
-----
```

```
1 | TABLE | TEST_TBL2 | 5 | SUCCESS
```

```
2 | TABLE | TEST_TBL1 | 5 | SUCCESS
-----
-----
```

Total Count: 10 End Time: 2022-12-27 10:02:22

```
2022-12-27 10:02:22 [INFO] Unnecessary to merge the data files. As --file-name is
missing
2022-12-27 10:02:22 [INFO] Dump record finished. Total Elapsed: 1.356 s
2022-12-27 10:02:22 [INFO] Unnecessary to upload the data files to the remote
cloud storage service
2022-12-27 10:02:22 [INFO] System exit 0
```

- 导出目录结构如下。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$tree /home/admin/test_migrate_data/csv_data
/home/admin/test_migrate_data/csv_data
|—— data
| |—— CHECKPOINT.bin
| |—— MANIFEST.bin
| |—— OBDUMPER_USER01
| |—— TABLE
| |—— TEST_TBL1.1.0.csv
| |—— TEST_TBL2.1.0.csv
|—— logs
|—— ob-loader-dumper.error
|—— ob-loader-dumper.info
|—— ob-loader-dumper.warn

4 directories, 7 files
```

- 查看导出结果文件。

```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$cat /home/admin/test_migrate_data/csv_data/data/OBDUMPER_USER01/TABLE
/TEST_TBL1.1.0.csv
```

```
'COL1','COL2','COL3'
1,'China',86
2,'Taiwan',886
3,'Hong Kong',852
4,'Macao',853
5,'North Korea',850
```

## 40.3.2.6 导入 CSV 数据文件

场景描述：将从集群 test4000 下的租户 oracle001 里的 Schema OBDUMPER\_USER01 导出到 /home/admin/test\_migrate\_data/csv\_data 目录下所有已支持的 CSV 数据文件，导入到集群 test4000 下的租户 mysql001 里的 Schema test\_data 中。

## 40.3.2.7 注意

- 需要将租户 oracle001 里的 Schema OBDUMPER\_USER01 导出的目录 /home/admin/test\_migrate\_data/csv\_data/data/ 下的 OBDUMPER\_USER01 名称修改为 test\_data。
- 将导出的 CSV 文件名中的表名中的英文字母改为小写。

示例语句如下：

```
[root@xxx /home/admin/test_migrate_data/csv_data/data]# mv
OBDUMPER_USER01 test_data
[root@xxx /home/admin/test_migrate_data/csv_data/data]# cd test_data/TABLE
[root@xxx /home/admin/test_migrate_data/csv_data/data/test_data/TABLE]# ls
TEST_TBL1.1.0.csv TEST_TBL2.1.0.csv
[root@xxx /home/admin/test_migrate_data/csv_data/data/test_data/TABLE]# mv
TEST_TBL1.1.0.csv test_tbl1.1.0.csv
[root@xxx /home/admin/test_migrate_data/csv_data/data/test_data/TABLE]# mv
TEST_TBL2.1.0.csv test_tbl2.1.0.csv
```

- 示例语句如下。



```
[xxx@xxx /ob-loader-dumper-4.0.0-RELEASE/bin]
$./obloader -h xxx.xxx.xxx.xxx -P 2883 -u obloader_user01 -p ***** --sys-user
root --sys-password ***** -c test4000 -t mysql001 -D test_data --csv --table '*' -f
/home/admin/test_migrate_data/csv_data
2022-12-27 11:17:21 [INFO] Parsed args:
[--csv] true
[--file-path] /home/admin/test_migrate_data/csv_data
[--host] xxx.xxx.xxx.xxx
[--port] 2883
[--user] obloader_user01
[--tenant] mysql001
[--cluster] test4000
[--password] *****
[--database] test_data
[--sys-user] root
[--sys-password] *****
[--table] [*]

2022-12-27 11:17:22 [INFO] Load jdbc driver class: "com.oceanbase.jdbc.Driver"
finished
2022-12-27 11:17:22 [INFO] The manifest file: "/home/admin/test_migrate_data
/csv_data/data/MANIFEST.bin" has been saved
2022-12-27 11:17:22 [INFO] Query column metadata for the table: "test_tbl1"
finished
2022-12-27 11:17:22 [INFO] Query column metadata for the table: "test_tbl2"
finished
2022-12-27 11:17:22 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
/CHECKPOINT.bin" is unmatched on the suffix[.csv], ignore it
2022-12-27 11:17:22 [INFO] File: "/home/admin/test_migrate_data/csv_data/data
```

```
/MANIFEST.bin" is unmatched on the suffix[.csv], ignore it
2022-12-27 11:17:22 [INFO] Binding table: "test_tbl2" to the file: "/home/admin
/test_migrate_data/csv_data/data/test_data/TABLE/test_tbl2.1.0.csv" finished
2022-12-27 11:17:22 [INFO] Binding table: "test_tbl1" to the file: "/home/admin
/test_migrate_data/csv_data/data/test_data/TABLE/test_tbl1.1.0.csv" finished
2022-12-27 11:17:22 [INFO] Splitted 2 csv subfiles by 64.0 MB. Elapsed: 21.68 ms
2022-12-27 11:17:22 [INFO] Generate 2 subfiles finished
2022-12-27 11:17:22 [INFO] Ignore to clean any tables as --truncate-table or --
delete-from-table is not specified
2022-12-27 11:17:22 [INFO] Query table entry and primary key for table:
"test_tbl1" finished. Remain: 1
2022-12-27 11:17:22 [INFO] Query table entry and primary key for table:
"test_tbl2" finished. Remain: 0
2022-12-27 11:17:22 [INFO] Query the leader location of "test_tbl1" finished.
Remain: 0
2022-12-27 11:17:22 [INFO] Query the leader location of "test_tbl2" finished.
Remain: 0
2022-12-27 11:17:22 [INFO] Calculate leader: xxx.xxx.xxx.xxx:2881 of table:
"test_tbl1", part: 0. Remain: 1
2022-12-27 11:17:22 [INFO] Calculate leader: xxx.xxx.xxx.xxx:2881 of table:
"test_tbl2", part: 0. Remain: 0
2022-12-27 11:17:22 [INFO] Waiting to refresh observer load status .....
2022-12-27 11:17:22 [INFO] Refresh observer load status success. Table:
"test_tbl1". Remain: 1
2022-12-27 11:17:22 [INFO] Refresh observer load status success. Table:
"test_tbl2". Remain: 0
2022-12-27 11:17:22 [INFO] Refresh observer load status finished. Elapsed: 29.40
ms
2022-12-27 11:17:22 [INFO] Use c.l.d.PhasedBackoffWaitStrategy as available cpu
```

(s) is 64

2022-12-27 11:17:22 [INFO] Create 4096 slots for ring buffer finished. [xxx.xxx.xxx.xxx:2881]

2022-12-27 11:17:22 [INFO] Start 128 database writer threads finished. [xxx.xxx.xxx.xxx:2881]

2022-12-27 11:17:22 [INFO] Start 128 csv file reader threads succeeded

2022-12-27 11:17:22 [INFO] File: "/home/admin/test\_migrate\_data/csv\_data/data/test\_data/TABLE/test\_tbl2.1.0.csv" has been parsed finished

2022-12-27 11:17:22 [INFO] File: "/home/admin/test\_migrate\_data/csv\_data/data/test\_data/TABLE/test\_tbl1.1.0.csv" has been parsed finished

2022-12-27 11:17:24 [INFO] Wait for the all the workers to drain of published events then halt the workers

2022-12-27 11:17:24 [INFO] Close connection count: 36 of the DataSource. Key: xxx.xxx.xxx.xxx\_11532\_332378361\_test\_data

2022-12-27 11:17:24 [INFO] Shutdown task context finished

2022-12-27 11:17:24 [INFO] ----- Finished Tasks: 2 Running Tasks: 0 Progress: 100.00% -----

2022-12-27 11:17:24 [INFO]

All Load Tasks Finished:

-----  
-----  
No.# | Type | Name | Count | Status  
-----  
-----

1 | TABLE | test\_tbl1 | 5 -> 5 | SUCCESS

2 | TABLE | test\_tbl2 | 5 -> 5 | SUCCESS  
-----

```
-----  
  
Total Count: 10 End Time: 2022-12-27 11:17:24
```

```
2022-12-27 11:17:24 [INFO] Load record finished. Total Elapsed: 1.950 s
```

```
2022-12-27 11:17:24 [INFO] System exit 0
```

- 检查导入的表数据。

```
obclient [test_data]> SELECT * FROM test_tbl1;
```

```
+-----+-----+-----+
```

```
| COL1 | COL2 | COL3 |
```

```
+-----+-----+-----+
```

```
| 1 | China | 86 |
```

```
| 2 | Taiwan | 886 |
```

```
| 3 | Hong Kong | 852 |
```

```
| 4 | Macao | 853 |
```

```
| 5 | North Korea | 850 |
```

```
+-----+-----+-----+
```

```
5 rows in set
```

## 41 表与表之间的数据迁移

本文将主要介绍使用 SQL 语句实现表与表之间的数据迁移。

### 41.1 表与表之间数据迁移

#### 41.1.1 语法 1

```
INSERT INTO target_table_name[(target_col_name[, target_col_name] ...)]
SELECT [(source_col_name[, source_col_name] ...)]
FROM source_table_name
[WHERE expr];
```

参数解释：

参数	描述
target_table_name	数据迁移的目标表。
target_col_name	目标表的列名称。如果要更新目标表的全部列数据，可以不填写列名称。
source_col_name	源表的列名称。选择需要迁移的列，如果要选中全部数据可以用 * 表示。 <h3>41.1.1.1 注意</h3> <p>选择的列数量需要与目的表中的列数量保持一致。</p>
source_table_name	数据迁移的源表。
WHERE expr	迁移数据的筛选条件，不填表示迁移 <code>SELECT</code> 选中的所有行记录。

#### 41.1.2 语法 2

##### 41.1.2.2 功能适用性

`MERGE` 语句暂时只支持 OceanBase 数据库 Oracle 模式。

```
MERGE INTO target_table_name
USING source_table_name
ON (expr)
WHEN NOT MATCHED THEN INSERT VALUES((source_col_name[, source_col_name] ...))
[WHERE expr];
```

参数解释：

参数	描述
target_table_name	数据迁移的目标表。
source_table_name	数据迁移的源表。
ON (expr)	源表和目标表的连接条件。
source_col_name	源表的列名称。选择需要迁移的列，如果要选中全部数据可以用 * 表示。  <h3>41.1.2.3 注意</h3> <p>选择的列数量需要与目标表中的列数量保持一致，并且迁移的数据要符合目标表的数据类型。</p>
WHERE expr	迁移数据的筛选条件。

### 41.1.3 示例 1

将表 tbl1 中符合条件（ age > 10 ）的数据插入表 tbl2 中。

```
obclient [test]> SELECT * FROM tbl1;
+-----+-----+-----+
| id | name | age |
+-----+-----+-----+
| 1 | ab | 8 |
| 2 | bc | 18 |
| 3 | cd | 14 |
| 4 | de | 19 |
| 5 | ef | 6 |
```

```
| 6 | fg | 15 |
```

```
+-----+-----+-----+
```

6 rows in set

```
obclient [test]> DESC tbl2;
```

```
+-----+-----+-----+-----+-----+-----+
```

```
| Field | Type | Null | Key | Default | Extra |
```

```
+-----+-----+-----+-----+-----+-----+
```

```
| col1 | int(11) | YES | | NULL | |
```

```
| col2 | int(11) | YES | | NULL | |
```

```
+-----+-----+-----+-----+-----+-----+
```

2 rows in set

```
obclient [test]> SELECT * FROM tbl2;
```

Empty set

```
obclient [test]> INSERT INTO tbl2 SELECT id,age FROM tbl1 WHERE age > 10;
```

Query OK, 4 rows affected

Records: 4 Duplicates: 0 Warnings: 0

```
obclient [test]> SELECT * FROM tbl2;
```

```
+-----+-----+
```

```
| col1 | col2 |
```

```
+-----+-----+
```

```
| 2 | 18 |
```

```
| 3 | 14 |
```

```
| 4 | 19 |
```

```
| 6 | 15 |
```

```
+-----+-----+
```

```
4 rows in set
```

## 41.1.4 示例 2

如果在表 `tbl1` 中找不到与表 `tbl2` 取值相同的行（`tbl1.id=tbl2.col1 and tbl1.age=tbl2.col2`），那么将 `tbl1` 中的这行插入到 `tbl2` 中，且只插入满足 `tbl1.age < 10` 的行。

```
obclient [SYS]> SELECT * FROM tbl1;
```

```
+-----+-----+-----+
```

```
| ID | NAME | AGE |
```

```
+-----+-----+-----+
```

```
| 1 | ab | 8 |
```

```
| 2 | bc | 18 |
```

```
| 3 | cd | 14 |
```

```
| 4 | de | 19 |
```

```
| 5 | ef | 6 |
```

```
| 6 | fg | 15 |
```

```
+-----+-----+-----+
```

```
6 rows in set
```

```
obclient [SYS]> SELECT * FROM tbl2;
```

```
Empty set
```

```
obclient [SYS]> MERGE INTO tbl2
```

```
USING tbl1
```

```
ON (tbl1.id=tbl2.col1 and tbl1.age=tbl2.col2)
```

```
WHEN NOT MATCHED THEN INSERT VALUES(tbl1.id,tbl1.age)
```

```
WHERE tbl1.age < 10;
```

```
Query OK, 2 rows affected
```



```
obclient [SYS]> SELECT * FROM tbl2;
```

```
+-----+-----+
```

```
| COL1 | COL2 |
```

```
+-----+-----+
```

```
| 1 | 8 |
```

```
| 5 | 6 |
```

```
+-----+-----+
```

```
2 rows in set
```

## 41.2 相关文档

- [INSERT（MySQL 模式）](#)
- [INSERT（Oracle 模式）](#)
- [MERGE（Oracle 模式）](#)

## 42 资源单元迁移

集群单个 Zone 中多台 OBCServer 节点之间调整资源分布不均衡时，可以调整租户使用的资源分布，以便集群资源使用均衡。

本文将主要介绍使用 SQL 语句实现资源单元的迁移。

### 42.0.0.1 功能适用性

ALTER SYSTEM MIGRATE UNIT 语句仅支持在 sys 租户执行。

## 42.1 语法

```
ALTER SYSTEM MIGRATE UNIT [=] unit_id  
DESTINATION [=] ip_port;
```

参数解释：

参数	描述
unit_id	Unit 编号。
ip_port	将 Unit 迁移到的目标节点地址。

## 42.2 示例

### 1. 查看资源单元分布。

执行下面 SQL 语句，查看租户 mysql001 的资源单元在各 zone 的分布情况。

```
SELECT t.TENANT_NAME,u.ZONE,u.UNIT_ID,u.SVR_IP FROM oceanbase.  
DBA_OB_TENANTS t,oceanbase.DBA_OB_UNITS u WHERE t.TENANT_ID=u.  
TENANT_ID AND t.TENANT_NAME='mysql001';
```

返回结果如下：

```
+-----+-----+-----+-----+  
| TENANT_NAME | ZONE | UNIT_ID | SVR_IP |  
+-----+-----+-----+-----+  
| mysql001 | zone1 | 1002 | 10.10.10.1 |
```

```
| mysql001 | zone2 | 1004 | 10.10.10.2 |
| mysql001 | zone3 | 1006 | 10.10.10.3 |
+-----+-----+-----+-----+
3 rows in set
```

## 2. 启动资源单元迁移。

执行下面 SQL 语句，将租户 `mysql001` 的 `zone3` 中的资源单元从 `10.10.10.3:2882` 迁移至 `10.10.10.4:2882`。

```
ALTER SYSTEM MIGRATE UNIT = 1006 DESTINATION = '10.10.10.4:2882';
```

## 3. 查看迁移情况。

执行下面 SQL 语句，查看租户 `mysql001` 的资源单元迁移后在各 `zone` 的分布情况。

```
SELECT t.TENANT_NAME,u.ZONE,u.UNIT_ID,u.SVR_IP FROM oceanbase.
DBA_OB_TENANTS t,oceanbase.DBA_OB_UNITS u WHERE t.TENANT_ID=u.
TENANT_ID AND t.TENANT_NAME='mysql001';
```

返回结果如下：

```
+-----+-----+-----+-----+
| TENANT_NAME | ZONE | UNIT_ID | SVR_IP |
+-----+-----+-----+-----+
| mysql001 | zone1 | 1002 | 10.10.10.1 |
| mysql001 | zone2 | 1004 | 10.10.10.2 |
| mysql001 | zone3 | 1006 | 10.10.10.4 |
+-----+-----+-----+-----+
3 rows in set
```

## 42.3 相关文档

- 资源单元迁移语法详细信息，请参见 [MIGRATE UNIT](#)。
- 取消资源单元迁移详细信息，请参见 [CANCEL MIGRATE UNIT](#)。

## 43 使用 OUTFILE 语句导出数据

`SELECT INTO OUTFILE` 语句常用的一种数据导出方式。`SELECT INTO OUTFILE` 语句能够对需要导出的字段做出限制，这很好的满足了某些不需要导出主键字段的场景。配合 `LOAD DATA INFILE` 语句导入数据，是一种很便利的数据导入导出方式。

### 43.1 背景信息

OceanBase 数据库兼容这一个语法。

模式	建议使用的 OceanBase 数据库版本	建议使用的客户端
MySQL 模式	V2.2.40 及以上	MySQL Client、OBClient
Oracle 模式	V2.2.40 及以上	OBClient

#### 43.1.0.1 注意

客户端需要直连 OceanBase 数据库实例以做导入导出操作。

### 43.2 权限要求

- 在 MySQL 租户下执行 `SELECT INTO` 语句，需要拥有 `FILE` 权限和对应表的 `SELECT` 权限。如果需要为用户授予 `FILE` 权限，可以使用以下命令格式：

```
GRANT FILE ON *.* TO user_name;
```

其中，`user_name` 是需要执行 `SELECT INTO` 命令的用户。有关 OceanBase 数据库权限的详细介绍，请参见 [MySQL 模式下的权限分类](#)。

- 在 Oracle 租户下执行 `SELECT INTO` 语句，需要拥有对应表的 `SELECT` 权限。有关 OceanBase 数据库权限的详细介绍，参见 [Oracle 模式下的权限分类](#)。

### 43.3 语法

```
SELECT [/+parallel(N)*/] column_list_option  
INTO {OUTFILE 'file_name' [PARTITION BY part_expr] [{CHARSET | CHARACTER SET}  
charset_name] [field_opt] [line_opt] [file_opt]}
```

```
| DUMPFILE 'file_name'  
| into_var_list}  
FROM table_name_list  
[WHERE where_conditions]  
[GROUP BY group_by_list [HAVING having_search_conditions]]  
[ORDER BY order_expression_list];
```

column\_list\_option:

column\_name [, column\_name ...]

field\_opt:

{COLUMNS | FIELDS} field\_term\_list

field\_term\_list:

field\_term [, field\_term ...]

field\_term:

{[OPTIONALLY] ENCLOSED | TERMINATED | ESCAPED} BY string

line\_opt:

LINES line\_term\_list

line\_term\_list:

line\_term [, line\_term ...]

line\_term:

{STARTING | TERMINATED} BY string

file\_opt:

```
file_option [, file_option ...]
```

file\_option:

SINGLE [=] {TRUE | FALSE}

| MAX\_FILE\_SIZE [=] {int | string}

| BUFFER\_SIZE [=] {int | string}

## 43.4 参数解释

参数	描述
parallel(N)	可选项，指定执行语句的并行度。
column_list_option	表示导出的列选项。如果要选中全部数据可以用 * 表示。 column_name：列名称。更多查询语句列选项的信息，参见 <a href="#">SIMPLE SELECT</a> 。

file\_name

用于指定导出文件的路径和文件名。file\_name 有以下格式：

- 将导出文件保存在 OBCloud 节点：/ \$PATH / \$FILENAME。
- 将导出文件保存在 OSS 上：oss:// \$PATH / \$FILENAME / ? host = \$HOST & access\_id = \$ACCESS\_ID & access\_key = \$ACCESSKEY。

参数解释如下：

- \$PATH：指定要保存导出文件的路径。
  - 导出到 OBCloud 节点中就是指定导出文件在 OBCloud 节点的路径。
  - 导出到 OSS 上就是指定存储桶中的文件路径。
- \$FILENAME：指定要导出文件的名称。当 SINGLE = FALSE 时表示导出文件的前缀，不指定时会生成默认的前缀 data，系统自动生成后缀。
- \$HOST：指定 OSS 服务的主机名或 CDN 加速的域名，即要访问的 OSS 服务的地址。
- \$ACCESS\_ID：指定访问 OSS 服务所需的 Access Key ID，用于身份验证。
- \$ACCESSKEY：指定了访问 OSS 服务所需的 Access Key Secret，用于身份验证。

## 43.4.0.1 说明

由于阿里云 OSS 有文件大小的限制，对于超过 5 GB 的文件，导出到 OSS 时会被拆分成多个文件，每个文件小于 5 GB。

PARTITION BY part_expr	<h2>43.4.0.2 说明</h2> <p>对于 OceanBase 数据库 V4.3.2 版本，从 V4.3.2 BP1 版本开始支持控制导出数据的分区方式。</p> <p>可选项，用于控制导出数据的分区方式，part_expr 的值作为导出路径的一部分，对每行数据计算 part_expr 的值，part_expr 取值相同的行属于同一个分区，将导出到同一个目录中。</p> <h2>43.4.0.3 注意</h2> <ul style="list-style-type: none"> <li>当数据按分区导出数据时，要求 SINGLE = FALSE，即允许导出多文件。</li> <li>当前按分区导出数据仅支持导入到 OSS 上。</li> </ul>
CHARSET   CHARACTER SET charset_name	可选项，指定导出到外部文件的字符集。charset_name 表示字符集的名称。
field_opt	可选项，导出字段格式选项。指定输出文件中各个字段的格式，通过 FIELDS 或 COLUMNS 子句来指定。详细介绍可参见下文 <a href="#">field_term</a> 。
line_opt	可选项，导出数据行的开始和结束符选项。指定输出文件中每一行的开始和结束字符，通过 LINES 子句设置。详细介绍可参见下文 <a href="#">line_term</a> 。
file_opt	可选项，控制是否导出到多个文件和导出到多文件时单个文件的大小。详细介绍可参见下文 <a href="#">file_option</a> 。
FROM table_name_list	指定选择数据的对象。
WHERE where_conditions	可选项，指定筛选条件，查询结果中仅包含满足条件的数据。更多查询语句的筛选信息，参见 <a href="#">SIMPLE SELECT</a> 。
GROUP BY group_by_list	<p>可选项，指定分组的字段，通常与聚合函数配合使用。</p> <h2>43.4.0.4 说明</h2> <p>SELECT 子句后面的所有列中，没有使用聚合函数的列，必须出现在 GROUP BY 子句后面。</p>



HAVING having_search_conditions	可选项，筛选分组后的各组数据。HAVING 子句与 WHERE 子句类似，但是 HAVING 子句可以使用累计函数（如 SUM、AVG 等）。
ORDER BY order_expression_list	可选项，指定结果集按照一个列或者多个列用来 ASC 或 DESC 显示查询结果。不指定 ASC 或者 DESC 时，默认为 ASC。 <ul style="list-style-type: none"><li>• ASC：表示升序。</li><li>• DESC：表示降序。</li></ul>

## 43.4.1 field\_term

- [OPTIONALLY] ENCLOSED BY string：用来指定包裹字段值的符号，默认没有引用符号。例如，ENCLOSED BY '"' 表示字符值放在双引号之间。如果使用了 OPTIONALLY 关键字，则仅对字符串类型的值使用指定字符包裹。
- TERMINATED BY string：用来指定字段值之间的符号。例如，TERMINATED BY ',' 指定了逗号作为两个字段值之间的标志。
- ESCAPED BY string：用来指定转义字符，以便处理特殊字符或解析特殊格式的数据。默认的转义字符是反斜杠（\）。

## 43.4.2 line\_term

- STARTING BY string：指定每一行开始的字符，默认没有起始符号。
- TERMINATED BY string：指定每一行的结束字符，默认使用换行符。例如，... LINES TERMINATED BY '\n' ... 表示一行将以换行符作为结束标志。

## 43.4.3 file\_option

- SINGLE [=] {TRUE | FALSE}：用于控制将数据导出到单个文件或多个文件。
  - SINGLE [=] TRUE：默认值，表示只能导出到单个文件。
  - SINGLE [=] FALSE：表示可以导出到多个文件。

### 43.4.3.5 注意

当并行度大于 1 且 `SINGLE = FALSE` 时，可以导出到多个文件，达到并行读并行写和提高导出速度的效果。

- `MAX_FILE_SIZE [=] {int | string}`：用于控制导出时单个文件的大小，仅在 `SINGLE = FALSE` 时生效。
- `BUFFER_SIZE [=] {int | string}`：用于控制导出时每个线程为每个分区专门申请的内存大小（不分区可视为单个分区），默认取值为 1 MB。

## 43.4.3.6 说明

- `BUFFER_SIZE` 用于导出性能调优，当机器内存充足且希望提高导出效率时，可设置一个较大的值（例如 4 MB），当机器内存不足时，可设置一个较小的值（例如 4 KB。设置为 0 时，表示单个线程中所有分区都使用一块公共内存）。
- 对于 OceanBase 数据库 V4.3.2 版本，从 V4.3.2 BP1 版本开始支持 `BUFFER_SIZE` 参数。

## 43.5 示例

### 43.5.4 导出数据文件到本地

1. 设置导出的文件路径。

要导出文件，需要先设置系统变量 `secure_file_priv`，配置导出文件可以访问的路径。

#### 43.5.4.1 注意

由于安全原因，设置系统变量 `secure_file_priv` 时，只能通过本地 Socket 连接数据库执行修改该全局变量的 SQL 语句。更多信息，请参见 [secure\\_file\\_priv](#)。

- a. 登录到要连接 OceanBase 数据库的 OBServer 节点。

```
ssh admin@xxx.xxx.xxx.xxx
```

- b. 执行以下命令，通过本地 Unix Socket 连接方式连接租户 `oracle001`。

```
obclient -S /home/admin/oceanbase/run/sql.sock -usys@oracle001 -p*****
```

- c. 设置导出路径为 `/home/admin/test_data`。

```
SET GLOBAL secure_file_priv = "/home/admin/test_data";
```

d. 退出登录。

2. 重新连接数据库后，使用 `SELECT INTO OUTFILE` 语句导出数据。指定逗号作为两个字段值之间的标志；对字符串类型的值使用 `"` 字符包裹；使用换行符作为结束标志。

- 串行写单个文件，指定文件名为 `test_tbl1.csv`。

```
SELECT /*+parallel(2)*/ *  
INTO OUTFILE '/home/admin/test_data/test_tbl1.csv'  
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'  
LINES TERMINATED BY '\n'  
FROM test_tbl1;
```

返回结果如下：

```
Query OK, 9 rows affected
```

- 并行写多个文件，不指定文件名，并且每个文件的大小不超过 4MB。

```
SELECT /*+parallel(2)*/ *  
INTO OUTFILE '/home/admin/test_data/'  
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'  
LINES TERMINATED BY '\n'  
SINGLE = FALSE MAX_FILE_SIZE = '4MB'  
FROM test_tbl1;
```

返回结果如下：

```
Query OK, 9 rows affected
```

- 并行写多个文件，指定文件名的前缀为 `dd2024`，并且每个文件的大小不超过 4MB。

```
SELECT /*+parallel(2)*/ *  
INTO OUTFILE '/home/admin/test_data/dd2024'  
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'  
LINES TERMINATED BY '\n'
```

```
SINGLE = FALSE MAX_FILE_SIZE = '4MB'  
FROM test_tbl1;
```

返回结果如下：

```
Query OK, 9 rows affected
```

## 43.5.4.2 说明

- 当多个导出任务同时导出到相同路径时，可能出现报错、只导出一部分数据等问题。可以通过合理设置导出路径规避，例如：

```
SELECT /*+parallel(2)*/ * INTO OUTFILE 'test/data' SINGLE = FALSE FROM t1; 和  
SELECT /*+parallel(2)*/ * INTO OUTFILE 'test/data' SINGLE = FALSE FROM t2; 同时  
执行时可能由于导出文件名相同而报错，建议将导出路径设置为 test/data1 和 test  
/data2。
```

- 当 `SINGLE = FALSE`，且导出因为 `file already exist` 等原因失败后，可以清除导出目录下所有与导出目标具有相同前缀的文件，或者删除导出目录再重建，然后再次执行导出操作。例如：

```
SELECT /*+parallel(2)*/ * INTO OUTFILE 'test/data' SINGLE = FALSE FROM t1; 失败  
后，可以删除 test 目录下所有 data 前缀的文件，或者直接删除 test 目录再重建，然后  
再次尝试执行导出操作。
```

- 登录机器，在 OBCServer 节点的 `/home/admin/test_data` 目录下查看导出的文件信息。

```
[xxx@xxx /home/admin/test_data]# ls
```

返回结果如下：

```
data_0_0_0 data_0_1_0 dd2024_0_0_0 dd2024_0_1_0 test_tbl1.csv
```

其中，`test_tbl1.csv` 是串行写单个文件示例导出的文件名；`data_0_0_0` 和 `data_0_1_0` 是并行写多个文件，不指定文件名示例导出的文件名；`dd2024_0_0_0` 和 `dd2024_0_1_0` 是并行写多个文件，指定文件名的前缀为 `dd2024` 示例导出的文件名。

## 43.5.5 导出数据文件到 OSS

使用 `SELECT INTO OUTFILE` 语句从 `test_tbl2` 表中按分区导出数据到指定的 OSS 存储位置。分区依据是 `col1` 和 `col2` 列的组合，相同的行属于同一个分区，将导出到同一个目录中。

```
SELECT /*+parallel(3)*/ *  
INTO OUTFILE 'oss://$DATA_FOLDER_NAME/?  
host=$OSS_HOST&access_id=$OSS_ACCESS_ID&access_key=$OSS_ACCESS_KEY'  
PARTITION BY CONCAT(col1,'/',col2)  
SINGLE = FALSE BUFFER_SIZE = '2MB'  
FROM test_tbl2;
```

存储位置由 `$DATA_FOLDER_NAME` 变量指定，同时需要提供 OSS 的主机地址、访问 ID 和访问密钥。

## 43.6 更多信息

通过 `SELECT INTO OUTFILE` 方法导出的文件，可以通过 `LOAD DATA` 语句进行导入，详细方法，请参考 [使用 LOAD DATA 导入数据](#)。

## 44 旁路导入概述

OceanBase 数据库支持旁路导入的方式向数据库插入数据，即 OceanBase 数据库支持向 `data` 文件中直接写入数据的功能。旁路导入可以绕过 SQL 层的接口，直接在 `data` 文件中直接分配空间并插入数据，从而提高数据导入的效率。

### 44.0.0.1 注意

不建议在执行旁路导入任务过程中执行升级，可能导致旁路导入任务失败。

## 44.1 功能概述

OceanBase 数据库支持两种方式的旁路导入：全量旁路导入和增量旁路导入。

`LOAD DATA` 语句、`INSERT INTO SELECT` 语句以及 `CREATE TABLE AS SELECT` 语句导入数据时支持旁路导入，可以通过对单个导入任务加 Hint 指定数据的导入方式为旁路导入，也可以通过全局配置项 [default\\_load\\_mode](#) 来指定数据的导入方式为旁路导入。

#### ● 全量旁路导入

- 全量旁路导入用于将完整的数据集一次性直接写入到数据库的数据文件中。该方法能够绕过 SQL 层的接口，在数据文件中直接分配空间并插入数据，从而提高数据导入的效率。
- 全量旁路导入通常用于初始化数据库、数据迁移或在需要快速载入大量数据时使用。

#### ● 增量旁路导入

- 增量旁路导入用于在已有大量数据的情况下，直接将新增的数据写入数据库的数据文件，无需通过 SQL 接口。该方法能够绕过 SQL 层的数据处理过程，直接将新增数据写入数据文件，从而提高数据的写入效率。
- 增量旁路导入通常用于高吞吐量的数据写入场景，例如大规模的实时数据采集、日志写入等场景。

## 44.2 使用场景

旁路导入功能可以在下面的场景中使用：

- 数据迁移与同步。对于数据迁移和同步，通常需要将大量的各种格式的数据从不同的数据源向 OceanBase 数据库进行迁移，传统的 SQL 接口性能可能在时效性上无法得到满足。
- 传统 ETL。当数据在源端进行了抽取和转化之后，在装载到目标端时，通常需要在短时间内加载大量的数据，使用旁路导入技术，会提升数据导入的性能。而对于 ELT 技术，在装载数据的过程中，也可以通过旁路导入技术提高效率。
- 从文本文件或者其他数据源向 OceanBase 数据库加载数据，利用旁路导入技术，也能够提升加载数据效率。

## 44.3 相关文档

- [全量旁路导入](#)
- [增量旁路导入](#)

## 45 全量旁路导入

本文介绍如何通过 `LOAD DATA` 语句、`INSERT INTO SELECT` 语句和 `CREATE TABLE AS SELECT` 语句实现全量旁路导入。

### 45.1 注意事项

在使用全量旁路导入时，有以下注意事项：

1. OceanBase 数据库从 V4.3.0 版本开始支持 LOB 数据类型的旁路导入。
2. 旁路导入期间会加表锁，表无法被写入其他数据，整个过程中表是只读的。
3. 当导入的数据量小于 MemTable 时，旁路导入没有优势。因为旁路导入写磁盘，比不过只写内存不需要转储或合并的场景。
4. 全量旁路导入适用于大表首次导入、10 GB~TB 级别的数据迁移以及 CPU 和内存都不是特别充裕的场景，因为旁路导入的执行路径短，可以省 CPU 开销。
5. `LOAD DATA` 语句支持在多行事务中运行，但由于其属于 DDL 操作，因此在执行时会主动提交之前的事务。
6. 使用 `INSERT INTO SELECT` 语句旁路导入数据时，只支持 PDML (Parallel Data Manipulation Language，并行数据操纵语言)，非 PDML 不能用旁路导入。

### 45.2 使用 `LOAD DATA` 语句旁路导入数据

`LOAD DATA` 语句通过使用 `append/direct()` Hint 来执行旁路导入，在未指定 Hint 时能基于配置项 [default\\_load\\_mode](#) 确定导入数据的行为。

#### 45.2.0.1 注意

OceanBase 数据库通过并行处理技术优化 `LOAD DATA` 的数据导入速率。该操作将数据分成多个子任务并行执行，每个子任务都视为独立的事务，执行顺序是不固定的。因此对于没有主键的表，数据写入顺序可能与原文件中的顺序不同。

#### 45.2.1 使用限制

- 在导入过程中无法同时执行两个写操作语句（即不能同时写一个表），因为导入过程中会先加表锁，并且整个导入过程中只能进行读操作。
- 不支持在触发器 (Trigger) 使用。
- 不支持含有生成列的表（某些索引会产生隐藏生成列，例如 `KEY idx_c2 (c2(16)) GLOBAL`）。



- 不支持单行超过 2M 的数据导入。
- 不支持 Liboblog 和闪回查询（Flashback Query）。

## 45.2.2 使用语法

```
LOAD DATA /*+ [APPEND | DIRECT(need_sort,max_error,'full')] parallel(N) */  
[REMOTE_OSS | LOCAL] INFILE 'file_name' INTO TABLE table_name [COMPRESSION]...
```

更多 `LOAD DATA` 语法的信息，请参见 [LOAD DATA](#)。

参数解释：

参数	描述
APPEND   DIRECT()	<p>使用 Hint 启用旁路导入功能。</p> <ul style="list-style-type: none"><li>• APPEND Hint 默认等同于使用的 <code>DIRECT(true, 0)</code>，同时可以实现在线收集统计信息（<code>GATHER_OPTIMIZER_STATISTICS</code> Hint）的功能。</li><li>• direct() 参数解释如下：<ul style="list-style-type: none"><li>■ need_sort：表示写入的数据是否需要排序，值为 bool 类型：<ul style="list-style-type: none"><li>■ true：表示需要排序。</li><li>■ false：表示不需要排序。</li></ul></li><li>■ max_error：表示最大容忍的错误行数。值为 INT 类型，超过这个数值导入任务执行会失败。</li><li>■ full：表示全量旁路导入，可选项，取值须使用英文单引号包起来。</li></ul></li></ul>
parallel(N)	加载数据的并行度，必填项，默认为 4。

REMOTE\_OSS | LOCAL

可选项，

- REMOTE\_OSS 用于指定是否从 OSS 文件系统读取数据。

## 45.2.2.2 注意

如果使用了此参数，file\_name 必须是一个 OSS 的地址。

- LOCAL 用于指定是否从客户端的本地文件系统读取数据。如果不使用 LOCAL 参数，那么将从服务器端（OBServer 节点）的文件系统读取数据。

file_name	<p>指定输入文件的路径和文件名。file_name 有以下格式：</p> <ul style="list-style-type: none"><li>• 导入文件在 OBServer 节点或客户端：/\$PATH/\$FILENAME。</li><li>• 导入文件在 OSS 上：oss://\$PATH/\$FILENAME/?host=\$HOST&amp;access_id=\$ACCESS_ID&amp;access_key=\$ACCESSKEY。</li></ul> <p>参数解释如下：</p> <ul style="list-style-type: none"><li>• \$PATH：指定了存储桶中的文件路径，表示文件所在的目录。</li><li>• \$FILENAME：指定了文件的名称，表示要访问的具体文件。</li><li>• \$HOST：指定了 OSS 服务的主机名或 CDN 加速的域名，即要访问的 OSS 服务的地址。</li><li>• \$ACCESS_ID：指定了访问 OSS 服务所需的 Access Key ID，用于身份验证。</li><li>• \$ACCESSKEY：指定了访问 OSS 服务所需的 Access Key Secret，用于身份验证。</li></ul> <h3>45.2.2.3 说明</h3> <p>在导入 OSS 上的文件时，需要确保以下信息：</p> <ul style="list-style-type: none"><li>• 确保访问 OSS 存储桶和文件的权限。你需要拥有足够的权限来读取指定的存储桶和文件。这通常需要在 OSS 控制台或通过 OSS API 设置访问权限，并将访问密钥（Access Key ID 和 Access Key Secret）配置为具有适当权限的凭据。</li><li>• 确保数据库服务器可以通过网络连接到指定的 \$HOST 地址，以访问 OSS 服务。如果使用的是 OSS 服务的 CDN 加速域名，还需要确保 CDN 配置正确，并且网络连接正常。</li></ul>
table_name	导入数据的表的名称，支持指定表任意列数目。

COMPRESSION

指定压缩文件格式，取值有

- **AUTO**: 根据文件名后缀自动探测压缩算法。

在使用 **AUTO** 参数时，不同的后缀名有对应的压缩格式。

- **.gz**: 对应 GZIP 压缩文件。
- **.deflate**: 对应 DEFLATE 压缩文件。
- **.zst/.zstd**: 对应 ZSTD 压缩文件。

- **NONE**: 表示文件没有压缩。

- **GZIP**: GZIP 压缩文件。

- **DEFLATE**: 不带元数据的GZIP压缩文件。

- **ZSTD**: ZSTD 压缩文件。

可以明确文件的压缩格式或者让程序根据文件名后缀来探测压缩格式。

## 45.2.3 使用示例

### 45.2.3.4 说明

下面的示例是从服务器端文件导入数据的方法。OceanBase 数据库 **LOAD DATA** 语句旁路导入数据还支持加载本地文件（**LOCAL INFILE**）的方式。更多有关 **LOAD DATA LOCAL INFILE** 的示例信息，请参见 [使用 LOAD DATA 语句导入数据](#)

MySQL 模式

Oracle 模式

1. 登录到需要连接 OBDServer 节点所在的机器上，在 **/home/admin** 目录下创建测试数据。

### 45.2.3.5 说明

OceanBase 数据库中的 **LOAD DATA** 语句仅支持加载 OBDServer 节点本地的输入文件。因此，需要在导入之前将文件拷贝到某台 OBDServer 上。

- a. 进入 OBDServer 节点所在的机器。

```
[xxx@xxx /home/admin]# ssh admin@10.10.10.1
```

- b. 创建测试数据 **tbl1.csv**。

```
[admin@xxx /home/admin]# vi tbl1.csv
1,11
2,22
3,33
```

## 2. 设置导入的文件路径。

设置系统变量 `secure_file_priv`，配置导入或导出文件时可以访问的路径。

### 45.2.3.6 注意

由于安全原因，设置系统变量 `secure_file_priv` 时，只能通过本地 Socket 连接数据库执行修改该全局变量的 SQL 语句。更多信息，请参见 [secure\\_file\\_priv](#)。

#### a. 登录到需要连接 OBCServer 节点所在的机器上。

```
[xxx@xxx /home/admin]# ssh admin@10.10.10.1
```

#### b. 执行以下命令，通过本地 Unix Socket 连接方式连接租户 `mysql001`。

```
obclient -S /home/admin/oceanbase/run/sql.sock -uroot@mysql001 -p*****
```

#### c. 设置导入路径为 `/home/admin`。

```
obclient [(none)]> SET GLOBAL secure_file_priv = "/home/admin";
Query OK, 0 rows affected
```

## 3. 重新连接数据库后，使用 `LOAD /*+ DIRECT */ DATA` 语句导入数据。

#### a. 创建表 `tbl2`。

```
obclient [test]> CREATE TABLE tbl2(col1 INT PRIMARY KEY,col2 INT);
Query OK, 0 rows affected
```

#### b. 查询表 `tbl2` 是否有数据，此时显示表为空。

```
obclient [test]> SELECT * FROM tbl2;
Empty set
```

c. 使用旁路导入将 `tbl1.csv` 文件中的数据导入到表 `tbl2`。

- 指定表 `tbl2` 的所有列导入数据。

```
obclient [test]> LOAD DATA /*+ direct(true,1024,'full') parallel(16) */ INFILE '
/home/admin/tbl1.csv' INTO TABLE tbl2 FIELDS TERMINATED BY ',';
Query OK, 3 rows affected
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

- 指定表 `tbl2` 的指定任意列导入数据。例如指定 `col1` , `col2` 列。

```
obclient [test]> LOAD DATA /*+ direct(true,1024,'full') parallel(16) */ INFILE '
/home/admin/tbl1.csv' INTO TABLE tbl2 FIELDS TERMINATED BY ','(col1,col2);
Query OK, 3 rows affected
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

- 使用配置项 `default_load_mode` 导入数据。

- a. 设置 `default_load_mode` 的值为 `FULL_DIRECT_WRITE`。

```
obclient [test]> ALTER SYSTEM SET default_load_mode
='FULL_DIRECT_WRITE';
```

- b. 不指定 `LOAD DATA` 语句的 Hint。

```
obclient [test]> LOAD DATA INFILE '/home/admin/tbl1.csv' INTO TABLE
tbl2 FIELDS TERMINATED BY ',';
```

d. 验证表 `tbl2` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl2;
```

查询结果如下：

```
+-----+-----+
| col1 | col2 |
+-----+-----+
```

```
| 1 | 11 |  
| 2 | 22 |  
| 3 | 33 |  
+-----+-----+  
3 rows in set
```

结果显示表 `tbl2` 中已导入数据。

1. 登录到需要连接 OBDServer 节点所在的机器上，在 `/home/admin` 目录下创建测试数据 `tbl1`。

## 45.2.3.7 说明

OceanBase 数据库中的 `LOAD DATA` 语句仅支持加载 OBDServer 节点本地的输入文件。因此，需要在导入之前将文件拷贝到某台 OBDServer 上。

- a. 进入 OBDServer 节点所在的机器。

```
[xxx@xxx /home/admin]# ssh admin@10.10.10.1
```

- b. 创建测试数据 `tbl1.csv`。

```
[admin@xxx /home/admin]# vi tbl1.csv  
1,11  
2,22  
3,33
```

2. 设置导入的文件路径。

设置系统变量 `secure_file_priv`，配置导入或导出文件时可以访问的路径。

## 45.2.3.8 注意

由于安全原因，设置系统变量 `secure_file_priv` 时，只能通过本地 Socket 连接数据库执行修改该全局变量的 SQL 语句。更多信息，请参见 [secure\\_file\\_priv](#)。

- a. 登录到需要连接 OBDServer 节点所在的机器上。

```
[xxx@xxx /home/admin]# ssh admin@10.10.10.1
```

- b. 执行以下命令，通过本地 Unix Socket 连接方式连接租户 `oracle001`。

```
obclient -S /home/admin/oceanbase/run/sql.sock -usys@oracle001 -p*****
```

- c. 设置导入路径为 `/home/admin`。

```
obclient [(none)]> SET GLOBAL secure_file_priv = "/home/admin";  
Query OK, 0 rows affected
```

3. 重新连接数据库后，使用 `LOAD /*+ DIRECT */ DATA` 语句导入数据。

- a. 创建表 `tbl2`。

```
obclient [SYS]> CREATE TABLE tbl2(col1 INT PRIMARY KEY,col2 INT);  
Query OK, 0 rows affected
```

- b. 查询表 `tbl2` 是否有数据，此时显示表为空。

```
obclient [SYS]> SELECT * FROM tbl2;  
Empty set
```

- c. 使用旁路导入将 `tbl1.csv` 文件中的数据导入到表 `tbl2`。

- 指定表 `tbl2` 的所有列导入数据。

```
obclient [SYS]> LOAD DATA /*+ direct(true,1024,'full') parallel(16) */ INFILE '  
/home/admin/tbl1.csv' INTO TABLE tbl2 FIELDS TERMINATED BY ',';  
Query OK, 3 rows affected  
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

- 指定表 `tbl2` 的指定任意列导入数据。例如指定 `col1`，`col2` 列。

```
obclient [SYS]> LOAD DATA /*+ direct(true,1024,'full') parallel(16) */ INFILE '  
/home/admin/tbl1.csv' INTO TABLE tbl2 FIELDS TERMINATED BY ','(col1,col2);  
Query OK, 3 rows affected  
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```



- 使用配置项 `default_load_mode` 导入数据。

- a. 设置 `default_load_mode` 的值为 `FULL_DIRECT_WRITE`。

```
obclient [SYS]> ALTER SYSTEM SET default_load_mode
='FULL_DIRECT_WRITE';
```

- b. 不指定 `LOAD DATA` 语句的 Hint。

```
obclient [SYS]> LOAD DATA INFILE '/home/admin/tbl1.csv' INTO TABLE
tbl2 FIELDS TERMINATED BY ',';
```

- d. 验证表 `tbl2` 中是否已导入数据。

```
obclient [SYS]> SELECT * FROM tbl2;
```

查询结果如下：

```
+-----+-----+
| col1 | col2 |
+-----+-----+
| 1 | 11 |
| 2 | 22 |
| 3 | 33 |
+-----+-----+
3 rows in set
```

结果显示表 `tbl2` 中已导入数据。

## 45.3 使用 INSERT INTO SELECT 语句旁路导入数据

`INSERT INTO SELECT` 语句通过 Hint 使用 `append/direct()` 加上 `enable_parallel_dml` 来执行旁路导入，在未指定 Hint 时能基于配置项 [default\\_load\\_mode](#) 确定导入数据的行为。

## 45.3.4 使用限制

- 只支持 PDML（Parallel Data Manipulation Language，并行数据操纵语言），非 PDML 不能用旁路导入。有关并行 DML 的更多信息，参见 [并行 DML](#)。
- 在导入过程中无法同时执行两个写操作语句（即不能同时写一个表），因为导入过程中会先加表锁，并且整个导入过程中只能进行读操作。
- 旁路导入属于 DDL 语句，无法在多行事务（包含多个操作的事务）中执行。
  - 不能在 Begin 中执行。
  - Autocommit 必须设置为 1。
- 不支持在触发器（Trigger）使用。
- 不支持含有生成列的表（某些索引会产生隐藏生成列，例如 KEY idx\_c2 (c2(16)) GLOBAL）。
- 不支持 Liboblog 和闪回查询（Flashback Query）。

## 45.3.5 使用语法

```
INSERT /*+ [APPEND |DIRECT(need_sort,max_error,'full')] enable_parallel_dml parallel
(N) */ INTO table_name select_sentence
```

更多 INSERT INTO 语法的信息，请参见 [INSERT（MySQL 模式）](#) 和 [INSERT（Oracle 模式）](#)。

## 参数解释：

参数	描述
APPEND   DIRECT()	<p>使用 Hint 启用旁路导入功能。</p> <ul style="list-style-type: none"> <li>APPEND Hint 默认等同于使用的 DIRECT(true, 0)，同时可以实现在线收集统计信息（GATHER_OPTIMIZER_STATISTICS Hint）的功能。</li> <li>DIRECT() 参数解释如下： <ul style="list-style-type: none"> <li>need_sort：表示写入的数据是否需要排序，值为 bool 类型： <ul style="list-style-type: none"> <li>true：表示需要排序。</li> <li>false：表示不需要排序。</li> </ul> </li> <li>max_error：表示最大容忍的错误行数。值为 INT 类型，超过这个数值导入任务执行会失败。</li> <li>full：表示全量旁路导入，可选项，取值须使用英文单引号包起来。</li> </ul> </li> </ul>
enable_parallel_dml	<p>加载数据的并行度。</p> <h3>45.3.5.1 说明</h3> <p>一般情况下，enable_parallel_dml Hint 和 parallel Hint 必须配合使用才能开启并行 DML。不过，当目标表的 Schema 上指定了表级别的并行度时，仅需指定 enable_parallel_dml Hint。</p>
parallel(N)	<p>加载数据的并行度，必填项，取值是大于 1 的整数。</p>

## 45.3.6 使用示例

MySQL 模式

Oracle 模式

使用旁路导入将表 tbl2 中的部分数据导入到 tbl1 中。

1. 查询表 tbl1 是否有数据，此时显示表为空。

```
obclient [test]> SELECT * FROM tbl1;  
Empty set
```

2. 查询表 `tbl2` 是否有数据。

```
obclient [test]> SELECT * FROM tbl2;
```

查询到表 `tbl2` 有数据。

```
+-----+-----+-----+  
| col1 | col2 | col3 |  
+-----+-----+-----+  
| 1 | a1 | 11 |  
| 2 | a2 | 22 |  
| 3 | a3 | 33 |  
+-----+-----+-----+  
3 rows in set
```

3. 使用旁路导入将表 `tbl2` 中的数据导入到表 `tbl1`。

- 指定 `INSERT INTO SELECT` 语句的 Hint。

```
obclient [test]> INSERT /*+ DIRECT(true, 0, 'full') enable_parallel_dml parallel  
(16) */ INTO tbl1 SELECT t2.col1,t2.col3 FROM tbl2 t2;  
Query OK, 3 rows affected  
Records: 3 Duplicates: 0 Warnings: 0
```

- 不指定 `INSERT INTO SELECT` 语句的 Hint。

设置配置项 `default_load_mode` 的值为 `FULL_DIRECT_WRITE`。

```
obclient [test]> ALTER SYSTEM SET default_load_mode = 'FULL_DIRECT_WRITE';
```

```
obclient [test]> INSERT INTO tbl1 SELECT t2.col1,t2.col3 FROM tbl2 t2;
```

4. 验证表 `tbl1` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl1;
```

查询结果如下：

```
+-----+-----+
| col1 | col2 |
+-----+-----+
| 1 | 11 |
| 2 | 22 |
| 3 | 33 |
+-----+-----+
3 rows in set
```

结果显示表 `tbl1` 中已导入数据。

5. (可选)在 `EXPLAIN EXTENDED` 语句的返回结果的 `Note` 中，查看是否通过旁路导入写入的数据。

```
obclient [test]> EXPLAIN EXTENDED INSERT /*+ direct(true, 0, 'full')
enable_parallel_dml parallel(16) */ INTO tbl1 SELECT t2.col1,t2.col3 FROM tbl2 t2;
```

返回结果如下：

```
+-----+-----+
| Query Plan |
+-----+-----+
| =====
===== |
|ID|OPERATOR |NAME |EST.ROWS|EST.TIME(us)||
|-----|
|0 |PX COORDINATOR ||3 |27 ||
|1 |  └─EXCHANGE OUT DISTR |:EX10001 |3 |27 ||
```

```

|2|  └─INSERT |3|26 || |
|3|  └─EXCHANGE IN DISTR |3|1 ||
|4|  └─EXCHANGE OUT DISTR (RANDOM):EX10000 |3|1 ||
|5|  └─SUBPLAN SCAN |ANONYMOUS_VIEW1|3|1 ||
|6|  └─PX BLOCK ITERATOR |3|1 ||
|7|  └─TABLE FULL SCAN |t2 |3|1 ||

=====

===== |
| Outputs & filters: |
| ----- |
| 0 - output(nil), filter(nil), rowset=16 |
| 1 - output(nil), filter(nil), rowset=16 |
| dop=16 |
| 2 - output(nil), filter(nil) |
| columns([{tbl1: ({tbl1: (tbl1.__pk_increment(0x7efa63627790), tbl1.col1
(0x7efa63611980), tbl1.col3(0x7efa63611dc0))})})], partitions(p0), |
| column_values([T_HIDDEN_PK(0x7efa63627bd0)], [column_conv(INT,PS:(11,0),
NULL,ANONYMOUS_VIEW1.col1(0x7efa63626f10))(0x7efa63627ff0)], [column_conv
(INT, |
| PS:(11,0),NULL,ANONYMOUS_VIEW1.col3(0x7efa63627350))(0x7efa6362ff20)]) |
| 3 - output([T_HIDDEN_PK(0x7efa63627bd0)], [ANONYMOUS_VIEW1.col1
(0x7efa63626f10)], [ANONYMOUS_VIEW1.col3(0x7efa63627350)]), filter(nil),
rowset=16 |
| 4 - output([T_HIDDEN_PK(0x7efa63627bd0)], [ANONYMOUS_VIEW1.col1
(0x7efa63626f10)], [ANONYMOUS_VIEW1.col3(0x7efa63627350)]), filter(nil),
rowset=16 |
| dop=16 |
| 5 - output([ANONYMOUS_VIEW1.col1(0x7efa63626f10)], [ANONYMOUS_VIEW1.
col3(0x7efa63627350)]), filter(nil), rowset=16 |

```

```

| access([ANONYMOUS_VIEW1.col1(0x7efa63626f10)], [ANONYMOUS_VIEW1.col3
(0x7efa63627350)]) |
| 6 - output([t2.col1(0x7efa63625ed0)], [t2.col3(0x7efa63626780)]), filter(nil),
rowset=16 |
| 7 - output([t2.col1(0x7efa63625ed0)], [t2.col3(0x7efa63626780)]), filter(nil),
rowset=16 |
| access([t2.col1(0x7efa63625ed0)], [t2.col3(0x7efa63626780)]), partitions(p0) |
| is_index_back=false, is_global_index=false, |
| range_key([t2.__pk_increment(0x7efa63656410)]), range(MIN ; MAX)always true |
| Used Hint: |
| ----- |
| /*+ |
| |
| USE_PLAN_CACHE( NONE ) |
| PARALLEL(16) |
| ENABLE_PARALLEL_DML |
| DIRECT(TRUE, 0, 'FULL') |
| */ |
| Qb name trace: |
| ----- |
| stmt_id:0, stmt_type:T_EXPLAIN |
| stmt_id:1, INS$1 |
| stmt_id:2, SEL$1 |
| Outline Data: |
| ----- |
| /*+ |
| BEGIN_OUTLINE_DATA |
| PARALLEL(@"SEL$1" "t2"@"SEL$1" 16) |
| FULL(@"SEL$1" "t2"@"SEL$1") |

```

```
| USE_PLAN_CACHE( NONE ) |
| PARALLEL(16) |
| ENABLE_PARALLEL_DML |
| OPTIMIZER_FEATURES_ENABLE('4.3.3.0') |
| DIRECT(TRUE, 0, 'FULL') |
| END_OUTLINE_DATA |
| */ |
| Optimization Info: |
| ----- |
| t2: |
| table_rows:3 |
| physical_range_rows:3 |
| logical_range_rows:3 |
| index_back_rows:0 |
| output_rows:3 |
| table_dop:16 |
| dop_method:Global DOP |
| available_index_name:[tbl2] |
| stats info:[version=0, is_locked=0, is_expired=0] |
| dynamic sampling level:0 |
| estimation method:[DEFAULT, STORAGE] |
| Plan Type: |
| DISTRIBUTED |
| Note: |
| Degree of Parallelism is 16 because of hint |
| Direct-mode is enabled in insert into select |
+-----+
+-----+
77 rows in set (0.009 sec)
```



使用旁路导入将表 `tbl4` 中的部分数据导入到 `tbl3` 中。

1. 查询表 `tbl3` 是否有数据，此时显示表为空。

```
obclient [test]> SELECT * FROM tbl3;  
Empty set
```

2. 查询表 `tbl4` 是否有数据。

```
obclient [test]> SELECT * FROM tbl4;
```

查询到表 `tbl4` 有数据。

```
+-----+-----+-----+  
| COL1 | COL2 | COL3 |  
+-----+-----+-----+  
| 1 | a1 | 11 |  
| 2 | a2 | 22 |  
| 3 | a3 | 33 |  
+-----+-----+-----+  
3 rows in set (0.000 sec)
```

3. 使用旁路导入将表 `tbl4` 中的数据导入到表 `tbl3`。

- 指定 `INSERT INTO SELECT` 语句的 Hint。

```
obclient [test]> INSERT /*+ direct(true, 0, 'full') enable_parallel_dml parallel(16)  
*/ INTO tbl3 SELECT t2.col1, t2.col3 FROM tbl4 t2 WHERE ROWNUM <= 10000;  
Query OK, 3 rows affected  
Records: 3 Duplicates: 0 Warnings: 0
```

- 不指定 `INSERT INTO SELECT` 语句的 Hint。

设置配置项 `default_load_mode` 的值为 `FULL_DIRECT_WRITE`。

```
obclient [test]> ALTER SYSTEM SET default_load_mode ='FULL_DIRECT_WRITE';
```

```
obclient [test]> INSERT INTO tbl3 SELECT t2.col1, t2.col3 FROM tbl4 t2 WHERE
ROWNUM <= 10000;
```

4. 验证表 `tbl3` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl3;
```

查询结果如下：

```
+-----+-----+
| col1 | col3 |
+-----+-----+
| 1 | 11 |
| 2 | 22 |
| 3 | 33 |
+-----+-----+
3 rows in set
```

结果显示表 `tbl3` 中已导入数据。

5. (可选)在 `EXPLAIN EXTENDED` 语句的返回结果的 `Note` 中，查看是否通过旁路导入写入的数据。

```
obclient [test]> EXPLAIN EXTENDED INSERT /*+ direct(true, 0, 'full')
enable_parallel_dml parallel(16) */ INTO tbl3 SELECT t2.col1,t2.col3 FROM tbl4 t2;
```

返回结果如下：

```
+-----+-----+
| Query Plan |
+-----+-----+
+-----+-----+
```

```

=====
===== |
|ID|OPERATOR |NAME |EST.ROWS|EST.TIME(us)| |
|-----|
|0|OPTIMIZER STATS MERGE ||3|29||
|1| └─PX COORDINATOR ||3|29||
|2| └─EXCHANGE OUT DISTR |:EX10001|3|27||
|3| └─INSERT ||3|27||
|4| └─OPTIMIZER STATS GATHER ||3|1||
|5| └─EXCHANGE IN DISTR ||3|1||
|6| └─EXCHANGE OUT DISTR (RANDOM) |:EX10000|3|1||
|7| └─SUBPLAN SCAN |ANONYMOUS_VIEW1|3|1||
|8| └─PX BLOCK ITERATOR ||3|1||
|9| └─COLUMN TABLE FULL SCAN|T2|3|1||
=====
===== |
| Outputs & filters: |
|-----|
| 0 - output(nil), filter(nil), rowset=16 |
| 1 - output([column_conv(NUMBER,PS:(-1,0),NULL,ANONYMOUS_VIEW1.COL1
(0x7ef8f6027720))(0x7ef8f60287f0)], [column_conv(NUMBER,PS:(-1,0),NULL,
ANONYMOUS_VIEW1.COL3(0x7ef8f6027b60))(0x7ef8f6030710)]), filter(nil),
rowset=16 |
| 2 - output([column_conv(NUMBER,PS:(-1,0),NULL,ANONYMOUS_VIEW1.COL1
(0x7ef8f6027720))(0x7ef8f60287f0)], [column_conv(NUMBER,PS:(-1,0),NULL,
ANONYMOUS_VIEW1.COL3(0x7ef8f6027b60))(0x7ef8f6030710)]), filter(nil),
rowset=16 |
| dop=16 |
| 3 - output([column_conv(NUMBER,PS:(-1,0),NULL,ANONYMOUS_VIEW1.COL1

```

```

(0x7ef8f6027720))(0x7ef8f60287f0)], [column_conv(NUMBER,PS:(-1,0),NULL,
ANONYMOUS_VIEW1.COL3(0x7ef8f6027b60))(0x7ef8f6030710))], filter(nil) |
| columns([TBL3: ({TBL3: (TBL3.__pk_increment(0x7ef8f6027fa0), TBL3.COL1
(0x7ef8f6011d50), TBL3.COL3(0x7ef8f6012190))})), partitions(p0), |
| column_values([T_HIDDEN_PK(0x7ef8f60283e0)], [column_conv(NUMBER,PS:
(-1,0),NULL,ANONYMOUS_VIEW1.COL1(0x7ef8f6027720))(0x7ef8f60287f0)],
[column_conv(NUMBER, |
| PS:(-1,0),NULL,ANONYMOUS_VIEW1.COL3(0x7ef8f6027b60))(0x7ef8f6030710))]) |
| 4 - output([column_conv(NUMBER,PS:(-1,0),NULL,ANONYMOUS_VIEW1.COL1
(0x7ef8f6027720))(0x7ef8f60287f0)], [column_conv(NUMBER,PS:(-1,0),NULL,
ANONYMOUS_VIEW1.COL3(0x7ef8f6027b60))(0x7ef8f6030710)], |
| [T_HIDDEN_PK(0x7ef8f60283e0)]), filter(nil), rowset=16 |
| 5 - output([T_HIDDEN_PK(0x7ef8f60283e0)], [ANONYMOUS_VIEW1.COL1
(0x7ef8f6027720)], [ANONYMOUS_VIEW1.COL3(0x7ef8f6027b60)]), filter(nil),
rowset=16 |
| 6 - output([T_HIDDEN_PK(0x7ef8f60283e0)], [ANONYMOUS_VIEW1.COL1
(0x7ef8f6027720)], [ANONYMOUS_VIEW1.COL3(0x7ef8f6027b60)]), filter(nil),
rowset=16 |
| dop=16 |
| 7 - output([ANONYMOUS_VIEW1.COL1(0x7ef8f6027720)], [ANONYMOUS_VIEW1.
COL3(0x7ef8f6027b60)]), filter(nil), rowset=16 |
| access([ANONYMOUS_VIEW1.COL1(0x7ef8f6027720)], [ANONYMOUS_VIEW1.COL3
(0x7ef8f6027b60)]) |
| 8 - output([T2.COL1(0x7ef8f60264c0)], [T2.COL3(0x7ef8f6026f90)]), filter(nil),
rowset=16 |
| 9 - output([T2.COL1(0x7ef8f60264c0)], [T2.COL3(0x7ef8f6026f90)]), filter(nil),
rowset=16 |
| access([T2.COL1(0x7ef8f60264c0)], [T2.COL3(0x7ef8f6026f90)]), partitions(p0) |
| is_index_back=false, is_global_index=false, |

```

```

| range_key([T2.__pk_increment(0x7ef8f60584a0)]), range(MIN ; MAX)always true |
| Used Hint: |
| ----- |
| /*+ |
| |
| USE_PLAN_CACHE( NONE ) |
| PARALLEL(16) |
| ENABLE_PARALLEL_DML |
| DIRECT(TRUE, 0, 'FULL') |
| */ |
| Qb name trace: |
| ----- |
| stmt_id:0, stmt_type:T_EXPLAIN |
| stmt_id:1, INS$1 |
| stmt_id:2, SEL$1 |
| Outline Data: |
| ----- |
| /*+ |
| BEGIN_OUTLINE_DATA |
| PARALLEL(@"SEL$1" "T2"@"SEL$1" 16) |
| FULL(@"SEL$1" "T2"@"SEL$1") |
| USE_COLUMN_TABLE(@"SEL$1" "T2"@"SEL$1") |
| USE_PLAN_CACHE( NONE ) |
| PARALLEL(16) |
| ENABLE_PARALLEL_DML |
| OPTIMIZER_FEATURES_ENABLE('4.3.3.0') |
| DIRECT(TRUE, 0, 'FULL') |
| END_OUTLINE_DATA |
| */ |

```

```
| Optimization Info: |
| ----- |
| T2: |
| table_rows:3 |
| physical_range_rows:3 |
| logical_range_rows:3 |
| index_back_rows:0 |
| output_rows:3 |
| table_dop:16 |
| dop_method:Global DOP |
| available_index_name:[TBL4] |
| stats info:[version=0, is_locked=0, is_expired=0] |
| dynamic sampling level:0 |
| estimation method:[DEFAULT, STORAGE] |
| Plan Type: |
| DISTRIBUTED |
| Note: |
| Degree of Parallelism is 16 because of hint |
+-----+
+-----+
+-----+
82 rows in set (0.006 sec)
```

## 45.4 使用 CREATE TABLE AS SELECT 语句旁路导入数据

CREATE TABLE AS SELECT 语句通过设置 `DIRECT()` Hint 来指定旁路导入的导入方式，在未指定 Hint 时能基于配置项 [default\\_load\\_mode](#) 确定导入数据的行为。

### 45.4.7 使用语法

```
CREATE /*+ [APPEND | DIRECT(need_sort,max_error,load_type)] parallel(N) */ TABLE
table_name [AS] select_sentence
```

更多 `CREATE TABLE` 语法的信息，请参见 [CREATE TABLE（MySQL 模式）](#) 和 [CREATE TABLE（Oracle 模式）](#)。

参数解释：

参数	描述
APPEND   DIRECT()	<p>使用 Hint 启用旁路导入功能。</p> <ul style="list-style-type: none"> <li>● APPEND Hint 默认等同于使用的 <code>DIRECT(true, 0)</code>，同时可以实现在线收集统计信息（<code>GATHER_OPTIMIZER_STATISTICS</code> Hint）的功能。</li> <li>● DIRECT() 参数解释如下： <ul style="list-style-type: none"> <li>■ need_sort：表示写入的数据是否需要排序，值为 bool 类型： <ul style="list-style-type: none"> <li>■ true：表示需要排序。</li> <li>■ false：表示不需要排序。</li> </ul> </li> <li>■ max_error：表示最大容忍的错误行数。值为 INT 类型，超过这个数值导入任务执行会失败。</li> <li>■ full：表示全量旁路导入，可选项，取值须使用英文单引号包起来。</li> </ul> </li> </ul>
parallel(N)	加载数据的并行度，必填项，取值是大于 1 的整数。

## 45.4.8 使用示例

MySQL 模式

Oracle 模式

使用旁路导入将表 `tbl1` 中的数据导入到其他表中。

1. 创建表 `tbl1`。

```
obclient [test]> CREATE TABLE tbl1(c1 int);
```

2. 插入数据到表 `tbl1`。

```
obclient [test]> INSERT INTO tbl1 VALUES (1),(2),(3);
```

3. 查询表 `tbl1` 中的数据。

```
obclient [test]> SELECT * FROM tbl1;
```

查询结果如下：

```
+-----+
| c1 |
+-----+
| 1 |
| 2 |
| 3 |
+-----+
3 rows in set (0.027 sec)
```

4. 使用旁路导入将表 `tbl1` 中数据导入到表 `tbl2`。

- 指定 `CREATE TABLE AS SELECT` 语句的 Hint。

- 使用 `APPEND` Hint 旁路导入数据。

```
obclient [test]> CREATE /*+ append parallel(4) */ TABLE tbl2 AS SELECT * FROM
tbl1;
```

- 使用 `DIRECT` Hint 旁路导入数据。

```
obclient [test]> CREATE /*+ direct(true, 0, 'full') parallel(4) */ TABLE tbl2 AS
SELECT * FROM tbl1;
```

- 不指定 `CREATE TABLE AS SELECT` 语句的 Hint。

设置配置项 `default_load_mode` 的值为 `FULL_DIRECT_WRITE`。

```
obclient [test]> ALTER SYSTEM SET default_load_mode ='FULL_DIRECT_WRITE';
```



```
obclient [test]> CREATE TABLE tbl2 AS SELECT * FROM tbl1;
```

5. 验证表 `tbl2` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl2;
```

查询结果如下：

```
+-----+  
| c1 |  
+-----+  
| 1 |  
| 3 |  
| 2 |  
+-----+  
3 rows in set (0.050 sec)
```

结果显示表 `tbl2` 中已导入数据。

使用旁路导入将表 `tbl1` 中的数据导入到其他表中。

1. 创建表 `tbl1` 。

```
obclient [SYS]> CREATE TABLE tbl1(c1 int);
```

2. 插入数据到表 `tbl1` 。

```
obclient [SYS]> INSERT INTO tbl1 VALUES (1),(2),(3);
```

3. 查询表 `tbl1` 中的数据。

```
obclient [SYS]> SELECT * FROM tbl1;
```

查询结果如下：

```
+-----+  
| C1 |
```

```
+-----+
| 1 |
| 2 |
| 3 |
+-----+
3 rows in set (0.045 sec)
```

#### 4. 使用旁路导入将表 `tbl1` 中数据导入到表 `tbl2` 。

- 指定 `CREATE TABLE AS SELECT` 语句的 Hint。

- 使用 `APPEND` Hint 旁路导入数据。

```
obclient [SYS]> CREATE /*+ append parallel(4) */ TABLE tbl2 AS SELECT * FROM
tbl1;
```

- 使用 `DIRECT` Hint 旁路导入数据。

```
obclient [SYS]> CREATE /*+ direct(true, 0, 'full') parallel(4) */ TABLE tbl2 AS
SELECT * FROM tbl1;
```

- 不指定 `CREATE TABLE AS SELECT` 语句的 Hint。

设置 `default_load_mode` 的值为 `FULL_DIRECT_WRITE` 。

```
obclient [SYS]> ALTER SYSTEM SET default_load_mode ='FULL_DIRECT_WRITE';
```

```
obclient [SYS]> CREATE TABLE tbl2 AS SELECT * FROM tbl1;
```

#### 5. 验证表 `tbl2` 中是否已导入数据。

```
obclient [SYS]> SELECT * FROM tbl2;
```

查询结果如下：

```
+-----+
| C1 |
+-----+
```

```
| 3 |  
| 2 |  
| 1 |  
+-----+  
3 rows in set (0.013 sec)
```

结果显示表 `tbl2` 中已导入数据。

## 45.5 相关文档

- [表与表之间的数据迁移](#)
- [SQL 执行计划简介](#)
- [连接方式概述](#)
- [删除表](#)

## 46 增量旁路导入

当表中已有数据，想要导入增量数据时，可选择增量旁路导入功能来完成。虽然通过全量旁路导入能够导入增量数据，但是通过全量旁路导入来导入增量数据的过程将重写所有的原始数据，导入性能不佳。增量旁路导入功能与全量旁路导入不同的是，导入流程将只会操作增量数据，能够保证导入性能。

本文介绍如何通过 `LOAD DATA` 语句、`INSERT INTO SELECT` 语句和 `CREATE TABLE AS SELECT` 语句实现增量旁路导入。

### 46.1 注意事项

在使用旁路导入时，有以下注意事项：

1. 增量旁路导入的数据会触发转储，对于数据量较小且能在分钟内完成导入的情况，不建议使用增量旁路导入。
2. `LOAD DATA` 语句支持在多行事务中运行，但由于其属于 DDL 操作，因此在执行时会主动提交之前的事务。
3. 使用 `INSERT INTO SELECT` 语句旁路导入数据时，只支持 PDML（Parallel Data Manipulation Language，并行数据操纵语言），非 PDML 不能用旁路导入。

### 46.2 使用 `LOAD DATA` 语句旁路导入数据

`LOAD DATA` 语句通过使用 `DIRECT()` Hint 来执行增量旁路导入，在未指定 Hint 时能基于配置项 [default\\_load\\_mode](#) 确定导入数据的行为。

#### 46.2.1 使用限制

- 在导入过程中无法同时执行两个写操作语句（即不能同时写一个表），因为导入过程中会先加表锁，并且整个导入过程中只能进行读操作。
- 不支持在触发器（Trigger）使用。
- 不支持含有生成列的表（某些索引会产生隐藏生成列，例如 `KEY idx_c2 (c2(16)) GLOBAL`）。
- 不支持单行超过 2M 的数据导入。
- 不支持 Liboblog 和闪回查询（Flashback Query）。
- 有索引（不包括主键）的表不支持增量旁路导入。
- 有外键的表不支持增量旁路导入。

## 46.2.2 使用语法

```
LOAD DATA /*+ [DIRECT(need_sort,max_error,{inc|inc_replace})] parallel(N) */  
[REMOTE_OSS | LOCAL] INFILE 'file_name' INTO TABLE table_name [COMPRESSION]...
```

更多 `LOAD DATA` 语法的信息，请参见 [LOAD DATA](#)。

参数解释：

参数	描述
DIRECT()	<p>使用 Hint 启用旁路导入功能。DIRECT() 参数解释如下：</p> <ul style="list-style-type: none"><li>• <code>need_sort</code>：表示写入的数据是否需要排序，值为 <code>bool</code> 类型：<ul style="list-style-type: none"><li>■ <code>true</code>：表示需要排序。</li><li>■ <code>false</code>：表示不需要排序。</li></ul></li><li>• <code>max_error</code>：表示最大容忍的错误行数。值为 <code>INT</code> 类型，超过这个数值导入任务执行会失败。</li><li>• <code>inc inc_replace</code>：表示增量旁路导入的两种模式，取值须使用英文单引号包起来。<ul style="list-style-type: none"><li>■ <code>inc</code>：表示增量导入，检查主键是否重复，支持 <code>INSERT</code> 和 <code>IGNORE</code> 语义。</li><li>■ <code>inc_replace</code>：表示增量导入，但不检查主键是否重复，相当于 <code>REPLACE</code> 语义的增量导入。</li></ul></li></ul> <h3>46.2.2.1 注意</h3> <p>当 <code>load_mode</code> 取值为 <code>inc_replace</code> 时，<code>LOAD DATA</code> 语句中不允许有 <code>REPLACE</code> 或 <code>IGNORE</code> 关键字。</p>
parallel(N)	加载数据的并行度，必填项，默认值为 4。

REMOTE\_OSS | LOCAL

- 可选项，
- REMOTE\_OSS 用于指定是否从 OSS 文件系统读取数据。

### 46.2.2.2 注意

如果使用了此参数，file\_name 必须是一个 OSS 的地址。

- LOCAL 用于指定是否从客户端的本地文件系统读取数据。如果不使用 LOCAL 参数，那么将从服务器端（OBServer 节点）的文件系统读取数据。

file_name	<p>指定输入文件的路径和文件名。file_name 有以下格式：</p> <ul style="list-style-type: none"><li>• 导入文件在 OBServer 节点或客户端：/\$PATH/\$FILENAME。</li><li>• 导入文件在 OSS 上：oss://\$PATH/\$FILENAME/?host=\$HOST&amp;access_id=\$ACCESS_ID&amp;access_key=\$ACCESSKEY。</li></ul> <p>参数解释如下：</p> <ul style="list-style-type: none"><li>• \$PATH：指定了存储桶中的文件路径，表示文件所在的目录。</li><li>• \$FILENAME：指定了文件的名称，表示要访问的具体文件。</li><li>• \$HOST：指定了 OSS 服务的主机名或 CDN 加速的域名，即要访问的 OSS 服务的地址。</li><li>• \$ACCESS_ID：指定了访问 OSS 服务所需的 Access Key ID，用于身份验证。</li><li>• \$ACCESSKEY：指定了访问 OSS 服务所需的 Access Key Secret，用于身份验证。</li></ul> <h3>46.2.2.3 说明</h3> <p>在导入 OSS 上的文件时，需要确保以下信息：</p> <ul style="list-style-type: none"><li>• 确保访问 OSS 存储桶和文件的权限。你需要拥有足够的权限来读取指定的存储桶和文件。这通常需要在 OSS 控制台或通过 OSS API 设置访问权限，并将访问密钥（Access Key ID 和 Access Key Secret）配置为具有适当权限的凭据。</li><li>• 确保数据库服务器可以通过网络连接到指定的 \$HOST 地址，以访问 OSS 服务。如果使用的是 OSS 服务的 CDN 加速域名，还需要确保 CDN 配置正确，并且网络连接正常。</li></ul>
table_name	导入数据的表的名称，支持指定表任意列数目。

COMPRESSION

指定压缩文件格式，取值有

- **AUTO**: 根据文件名后缀自动探测压缩算法。

在使用 **AUTO** 参数时，不同的后缀名有对应的压缩格式。

- **.gz**: 对应 GZIP 压缩文件。
- **.deflate**: 对应 DEFLATE 压缩文件。
- **.zst/.zstd**: 对应 ZSTD 压缩文件。

- **NONE**: 表示文件没有压缩。

- **GZIP**: GZIP 压缩文件。

- **DEFLATE**: 不带元数据的GZIP压缩文件。

- **ZSTD**: ZSTD 压缩文件。

可以明确文件的压缩格式或者让程序根据文件名后缀来探测压缩格式。

## 46.2.2.4 说明

LOAD DATA 语句中增量旁路导入和全量旁路导入一样，也支持通配符方式多文件导入数据。

## 46.2.3 使用示例

LOAD DATA 语句中增量旁路导入的操作步骤与全量旁路导入的步骤一样，只需要将 **full** 字段的取值替换为 **inc** 或 **inc\_replace** 即可。

## 46.2.3.5 说明

下面的示例是从服务器端文件导入数据的方法。OceanBase 数据库 **LOAD DATA** 语句旁路导入数据还支持加载本地文件（**LOCAL INFILE**）的方式。更多有关 **LOAD DATA LOCAL INFILE** 的示例信息，请参见 [使用 LOAD DATA 语句导入数据](#)

MySQL 模式

Oracle 模式

1. 登录到需要连接 OBCServer 节点所在的机器上，在 **/home/admin** 目录下创建测试数据 **tbl1**。

## 46.2.3.6 说明



OceanBase 数据库中的 `LOAD DATA` 语句仅支持加载 OServer 节点本地的输入文件。因此，需要在导入之前将文件拷贝到某台 OServer 上。

```
[xxx@xxx /home/admin]# ssh admin@10.10.10.1
```

```
[admin@xxx /home/admin]# vi tbl1.csv
```

```
1,11
```

```
2,22
```

```
3,33
```

## 2. 设置导入的文件路径。

设置系统变量 `secure_file_priv`，配置导入或导出文件时可以访问的路径。

### 46.2.3.7 注意

由于安全原因，设置系统变量 `secure_file_priv` 时，只能通过本地 Socket 连接数据库执行修改该全局变量的 SQL 语句。更多信息，请参见 [secure\\_file\\_priv](#)。

#### a. 登录到需要连接 OServer 节点所在的机器上。

```
[xxx@xxx /home/admin]# ssh admin@10.10.10.1
```

#### b. 执行以下命令，通过本地 Unix Socket 连接方式连接租户 `mysql001`。

```
obclient -S /home/admin/oceanbase/run/sql.sock -uroot@mysql001 -p*****
```

#### c. 设置导入路径为 `/home/admin`。

```
obclient [(none)]> SET GLOBAL secure_file_priv = "/home/admin";  
Query OK, 0 rows affected
```

## 3. 重新连接数据库后，使用 `LOAD /*+ DIRECT */ DATA` 语句导入数据。

#### a. 创建表 `tbl1`。

```
obclient [test]> CREATE TABLE tbl1(col1 INT PRIMARY KEY,col2 INT);  
Query OK, 0 rows affected
```

#### b. 查询表 `tbl1` 是否有数据，此时显示表为空。

```
obclient [test]> SELECT * FROM tbl1;  
Empty set
```

c. 使用旁路导入将 `tbl1.csv` 文件中的数据导入到表 `tbl1`。

- 指定表 `tbl1` 的所有列导入数据。

```
obclient [test]> LOAD DATA /*+ direct(true,1024,'inc_replace') parallel(16) */  
INFILE '/home/admin/tbl1.csv' INTO TABLE tbl1 FIELDS TERMINATED BY ',';  
Query OK, 3 rows affected  
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

- 指定表 `tbl1` 的指定任意列导入数据。例如指定 `col1` , `col2` 列。

```
obclient [test]> LOAD DATA /*+ direct(true,1024,'inc_replace') parallel(16) */  
INFILE '/home/admin/tbl1.csv' INTO TABLE tbl1 FIELDS TERMINATED BY  
'',(col1,col2);  
Query OK, 3 rows affected  
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

- 使用配置项 `default_load_mode` 导入数据。

a. 设置 `default_load_mode` 的值为 `INC_DIRECT_WRITE` 或 `INC_REPLACE_DIRECT_WRITE`。

```
obclient [test]> ALTER SYSTEM SET default_load_mode  
='INC_DIRECT_WRITE';
```

b. 不指定 `LOAD DATA` 语句的 Hint。

```
obclient [test]> LOAD DATA INFILE '/home/admin/tbl1.csv' INTO TABLE  
tbl2 FIELDS TERMINATED BY ',';
```

d. 验证表 `tbl1` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl1;
```

查询结果如下：

```
+-----+-----+
| col1 | col2 |
+-----+-----+
| 1 | 11 |
| 2 | 22 |
| 3 | 33 |
+-----+-----+
3 rows in set
```

结果显示表 `tbl2` 中已导入数据。

1. 登录到需要连接 OBDServer 节点所在的机器上，在 `/home/admin` 目录下创建测试数据 `tbl1`。

## 46.2.3.8 说明

OceanBase 数据库中的 `LOAD DATA` 语句仅支持加载 OBDServer 节点本地的输入文件。因此，需要在导入之前将文件拷贝到某台 OBDServer 上。

```
[xxx@xxx /home/admin]# ssh admin@10.10.10.1
```

```
[admin@xxx /home/admin]# vi tbl1.csv
```

```
1,11
```

```
2,22
```

```
3,33
```

2. 设置导入的文件路径。

设置系统变量 `secure_file_priv`，配置导入或导出文件时可以访问的路径。

## 46.2.3.9 注意

由于安全原因，设置系统变量 `secure_file_priv` 时，只能通过本地 Socket 连接数据库执行修改该全局变量的 SQL 语句。更多信息，请参见 [secure\\_file\\_priv](#)。

- a. 登录到需要连接 OBDServer 节点所在的机器上。

```
[xxx@xxx /home/admin]# ssh admin@10.10.10.1
```

- b. 执行以下命令，通过本地 Unix Socket 连接方式连接租户 `oracle001`。

```
obclient -S /home/admin/oceanbase/run/sql.sock -usys@oracle001 -p*****
```

- c. 设置导入路径为 `/home/admin`。

```
obclient [(none)]> SET GLOBAL secure_file_priv = "/home/admin";  
Query OK, 0 rows affected
```

3. 重新连接数据库后，使用 `LOAD /*+ DIRECT */ DATA` 语句导入数据。

- a. 创建表 `tbl2`。

```
obclient [test]> CREATE TABLE tbl2(col1 INT PRIMARY KEY,col2 INT);  
Query OK, 0 rows affected
```

- b. 查询表 `tbl2` 是否有数据，此时显示表为空。

```
obclient [test]> SELECT * FROM tbl2;  
Empty set
```

- c. 使用旁路导入将 `tbl1.csv` 文件中的数据导入到表 `tbl2`。

- 指定表 `tbl2` 的所有列导入数据。

```
obclient [test]> LOAD DATA /*+ direct(true,1024,'inc_replace') parallel(16) */  
INFILE '/home/admin/tbl1.csv' INTO TABLE tbl2 FIELDS TERMINATED BY ',';  
Query OK, 3 rows affected  
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

- 指定表 `tbl2` 的指定任意列导入数据。例如指定 `col1`，`col2` 列。

```
obclient [test]> LOAD DATA /*+ direct(true,1024,'inc_replace') parallel(16) */  
INFILE '/home/admin/tbl1.csv' INTO TABLE tbl2 FIELDS TERMINATED BY  
'',(col1,col2);
```

```
Query OK, 3 rows affected
```

```
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0
```

- 使用配置项 `default_load_mode` 导入数据。

- a. 设置 `default_load_mode` 的值为 `INC_DIRECT_WRITE` 或 `INC_REPLACE_DIRECT_WRITE`。

```
obclient [test]> ALTER SYSTEM SET default_load_mode
='INC_DIRECT_WRITE';
```

- b. 不指定 `LOAD DATA` 语句的 Hint。

```
obclient [test]> LOAD DATA INFILE '/home/admin/tbl1.csv' INTO TABLE
tbl2 FIELDS TERMINATED BY ',';
```

- d. 验证表 `tbl2` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl2;
```

查询结果如下：

```
+-----+-----+
| col1 | col2 |
+-----+-----+
| 1 | 11 |
| 2 | 22 |
| 3 | 33 |
+-----+-----+
3 rows in set
```

结果显示表 `tbl2` 中已导入数据。

## 46.3 使用 INSERT INTO SELECT 语句旁路导入数据

`INSERT INTO SELECT` 语句通过使用 `direct()` 加上 `enable_parallel_dml` Hint 来走旁路导入，在未指定 Hint 时能基于配置项 [default\\_load\\_mode](#) 确定导入数据的行为。

## 46.3.4 使用限制

- 只支持 PDML（Parallel Data Manipulation Language，并行数据操纵语言），非 PDML 不能用旁路导入。有关并行 DML 的更多信息，参见 [并行 DML](#)。
- 在导入过程中无法同时执行两个写操作语句（即不能同时写一个表），因为导入过程中会先加表锁，并且整个导入过程中只能进行读操作。
- 不支持在触发器（Trigger）使用。
- 不支持含有生成列的表（某些索引会产生隐藏生成列，例如 `KEY idx_c2 (c2(16)) GLOBAL`）。
- 不支持 Liboblog 和闪回查询（Flashback Query）。
- 有索引（不包括主键）的表不支持增量旁路导入。
- 有外键的表不支持增量旁路导入。

## 46.3.5 使用语法

```
INSERT /*+ [DIRECT(need_sort,max_error,{'inc'|'inc_replace'})] enable_parallel_dml
parallel(N) */ INTO table_name select_sentence
```

更多 `INSERT INTO` 语法的信息，请参见 [INSERT \(MySQL 模式\)](#) 和 [INSERT \(Oracle 模式\)](#)。

## 参数解释：

参数	描述
DIRECT()	<p>使用 Hint 启用旁路导入功能。DIRECT() 参数解释如下：</p> <ul style="list-style-type: none"> <li>• <code>need_sort</code>：表示写入的数据是否需要排序，值为 <code>bool</code> 类型： <ul style="list-style-type: none"> <li>■ <code>true</code>：表示需要排序。</li> <li>■ <code>false</code>：表示不需要排序。</li> </ul> </li> <li>• <code>max_error</code>：表示最大容忍的错误行数。值为 <code>INT</code> 类型，超过这个数值导入任务执行会失败。</li> <li>• <code>inc inc_replace</code>：表示增量旁路导入的两种模式，取值须使用英文单引号包起来。 <ul style="list-style-type: none"> <li>■ <code>inc</code>：表示增量导入，检查主键是否重复，支持 <code>INSERT</code> 和 <code>IGNORE</code> 语义。</li> <li>■ <code>inc_replace</code>：表示增量导入，但不检查主键是否重复，相当于 <code>REPLACE</code> 语义的增量导入。</li> </ul> </li> </ul>
enable_parallel_dml	<p>加载数据的并行度。</p> <h3>46.3.5.1 说明</h3> <p>一般情况下，<code>enable_parallel_dml</code> Hint 和 <code>parallel</code> Hint 必须配合使用才能开启并行 DML。不过，当目标表的 Schema 上指定了表级别的并行度时，仅需指定 <code>enable_parallel_dml</code> Hint。</p>
parallel(N)	<p>加载数据的并行度，必填项，取值是大于 1 的整数。</p>

## 46.3.6 使用示例

`INSERT INTO SELECT` 语句中增量旁路导入的操作步骤与全量旁路导入的步骤一样，只需要将 `full` 字段的取值替换为 `inc` 或 `inc_replace` 即可。

MySQL 模式

Oracle 模式

使用旁路导入将表 `tbl2` 中的部分数据导入到 `tbl1` 中。

1. 查询表 `tbl1` 是否有数据，此时显示表为空。

```
obclient [test]> SELECT * FROM tbl1;  
Empty set
```

2. 查询表 `tbl2` 是否有数据。

```
obclient [test]> SELECT * FROM tbl2;
```

查询到表 `tbl2` 有数据。

```
+-----+-----+-----+  
| col1 | col2 | col3 |  
+-----+-----+-----+  
| 1 | a1 | 11 |  
| 2 | a2 | 22 |  
| 3 | a3 | 33 |  
+-----+-----+-----+  
3 rows in set
```

3. 使用旁路导入将表 `tbl2` 中的数据导入到表 `tbl1` 。

- 指定 `INSERT INTO SELECT` 语句的 Hint。

```
obclient [test]> INSERT /*+ DIRECT(true, 0, 'inc_replace') enable_parallel_dml  
parallel(16) */ INTO tbl1 SELECT t2.col1,t2.col3 FROM tbl2 t2;  
Query OK, 3 rows affected  
Records: 3 Duplicates: 0 Warnings: 0
```

- 不指定 `INSERT INTO SELECT` 语句的 Hint。

设置配置项 `default_load_mode` 的值为 `INC_DIRECT_WRITE` 或 `INC_REPLACE_DIRECT_WRITE`。



```
obclient [test]> ALTER SYSTEM SET default_load_mode ='INC_DIRECT_WRITE';
```

```
obclient [test]> INSERT INTO tbl1 SELECT t2.col1,t2.col3 FROM tbl1 t2;
```

4. 验证表 `tbl1` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl1;
```

查询结果如下：

```
+-----+-----+
| col1 | col2 |
+-----+-----+
| 1 | 11 |
| 2 | 22 |
| 3 | 33 |
+-----+-----+
3 rows in set
```

结果显示表 `tbl1` 中已导入数据。

5. (可选)在 `EXPLAIN EXTENDED` 语句的返回结果的 `Note` 中，查看是否通过旁路导入写入的数据。

```
obclient [test]> EXPLAIN EXTENDED INSERT /*+ direct(true, 0, 'inc_replace')
enable_parallel_dml parallel(16) */ INTO tbl1 SELECT t2.col1,t2.col3 FROM tbl2 t2;
```

返回结果如下：

```
+-----+-----+
+-----+-----+
| Query Plan |
+-----+-----+
+-----+-----+
| ===== |
===== |
```

```

|ID|OPERATOR |NAME |EST.ROWS|EST.TIME(us)| |
|-----|
|0|PX COORDINATOR | |3 |27 ||
|1| └─EXCHANGE OUT DISTR |:EX10001 |3 |27 ||
|2| └─INSERT | |3 |26 ||
|3| └─EXCHANGE IN DISTR | |3 |1 ||
|4| └─EXCHANGE OUT DISTR (RANDOM):EX10000 |3 |1 ||
|5| └─SUBPLAN SCAN |ANONYMOUS_VIEW1|3 |1 ||
|6| └─PX BLOCK ITERATOR | |3 |1 ||
|7| └─TABLE FULL SCAN |t2 |3 |1 ||
|=====|
|=====|
| Outputs & filters: |
|-----|
| 0 - output(nil), filter(nil), rowset=16 |
| 1 - output(nil), filter(nil), rowset=16 |
| dop=16 |
| 2 - output(nil), filter(nil) |
| columns([{tbl1: ({tbl1: (tbl1.__pk_increment(0x7efa518277d0), tbl1.col1
(0x7efa518119c0), tbl1.col3(0x7efa51811e00))})})], partitions(p0), |
| column_values([T_HIDDEN_PK(0x7efa51827c10)], [column_conv(INT,PS:(11,0),
NULL,ANONYMOUS_VIEW1.col1(0x7efa51826f50))(0x7efa51828030)],
[column_conv(INT, |
| PS:(11,0),NULL,ANONYMOUS_VIEW1.col3(0x7efa51827390))(0x7efa5182ff60)]) |
| 3 - output([T_HIDDEN_PK(0x7efa51827c10)], [ANONYMOUS_VIEW1.col1
(0x7efa51826f50)], [ANONYMOUS_VIEW1.col3(0x7efa51827390)]), filter(nil),
rowset=16 |
| 4 - output([T_HIDDEN_PK(0x7efa51827c10)], [ANONYMOUS_VIEW1.col1
(0x7efa51826f50)], [ANONYMOUS_VIEW1.col3(0x7efa51827390)]), filter(nil),

```

```

rowset=16 |
| dop=16 |
| 5 - output([ANONYMOUS_VIEW1.col1(0x7efa51826f50)], [ANONYMOUS_VIEW1.
col3(0x7efa51827390)]), filter(nil), rowset=16 |
| access([ANONYMOUS_VIEW1.col1(0x7efa51826f50)], [ANONYMOUS_VIEW1.col3
(0x7efa51827390)]) |
| 6 - output([t2.col1(0x7efa51825f10)], [t2.col3(0x7efa518267c0)]), filter(nil),
rowset=16 |
| 7 - output([t2.col1(0x7efa51825f10)], [t2.col3(0x7efa518267c0)]), filter(nil),
rowset=16 |
| access([t2.col1(0x7efa51825f10)], [t2.col3(0x7efa518267c0)]), partitions(p0) |
| is_index_back=false, is_global_index=false, |
| range_key([t2.__pk_increment(0x7efa51856410)]), range(MIN ; MAX)always true |
| Used Hint: |
| ----- |
| /*+ |
| |
| USE_PLAN_CACHE( NONE ) |
| PARALLEL(16) |
| ENABLE_PARALLEL_DML |
| DIRECT(TRUE, 0, 'INC_REPLACE') |
| */ |
| Qb name trace: |
| ----- |
| stmt_id:0, stmt_type:T_EXPLAIN |
| stmt_id:1, INS$1 |
| stmt_id:2, SEL$1 |
| Outline Data: |
| ----- |

```

```
| /*+ |  
| BEGIN_OUTLINE_DATA |  
| PARALLEL(@"SEL$1" "t2"@"SEL$1" 16) |  
| FULL(@"SEL$1" "t2"@"SEL$1") |  
| USE_PLAN_CACHE( NONE ) |  
| PARALLEL(16) |  
| ENABLE_PARALLEL_DML |  
| OPTIMIZER_FEATURES_ENABLE('4.3.3.0') |  
| DIRECT(TRUE, 0, 'INC_REPLACE') |  
| END_OUTLINE_DATA |  
| */ |  
| Optimization Info: |  
| ----- |  
| t2: |  
| table_rows:3 |  
| physical_range_rows:3 |  
| logical_range_rows:3 |  
| index_back_rows:0 |  
| output_rows:3 |  
| table_dop:16 |  
| dop_method:Global DOP |  
| available_index_name:[tbl2] |  
| stats info:[version=0, is_locked=0, is_expired=0] |  
| dynamic sampling level:0 |  
| estimation method:[DEFAULT, STORAGE] |  
| Plan Type: |  
| DISTRIBUTED |  
| Note: |  
| Degree of Parallelism is 16 because of hint |
```

```
| Direct-mode is enabled in insert into select |
```

```
+-----+
```

```
-----+
```

```
77 rows in set (0.009 sec)
```

使用旁路导入将表 `tbl4` 中的部分数据导入到 `tbl3` 中。

1. 查询表 `tbl3` 是否有数据，此时显示表为空。

```
obclient [test]> SELECT * FROM tbl3;
```

```
Empty set
```

2. 查询表 `tbl4` 是否有数据。

```
obclient [test]> SELECT * FROM tbl4;
```

查询到表 `tbl4` 有数据。

```
+-----+-----+-----+
```

```
| COL1 | COL2 | COL3 |
```

```
+-----+-----+-----+
```

```
| 1 | a1 | 11 |
```

```
| 2 | a2 | 22 |
```

```
| 3 | a3 | 33 |
```

```
+-----+-----+-----+
```

```
3 rows in set (0.000 sec)
```

3. 使用旁路导入将表 `tbl4` 中的数据导入到表 `tbl3`。

- 指定 `INSERT INTO SELECT` 语句的 Hint。

```
obclient [test]> INSERT /*+ direct(true, 0, 'inc_replace') enable_parallel_dml
parallel(16) */ INTO tbl3 SELECT t2.col1, t2.col3 FROM tbl4 t2 WHERE ROWNUM
<= 10000;
```

```
Query OK, 3 rows affected
```

```
Records: 3 Duplicates: 0 Warnings: 0
```

- 不指定 `INSERT INTO SELECT` 语句的 Hint。

设置配置项 `default_load_mode` 的值为 `INC_DIRECT_WRITE` 或 `INC_REPLACE_DIRECT_WRITE`。

```
obclient [test]> ALTER SYSTEM SET default_load_mode ='INC_DIRECT_WRITE';
```

```
obclient [test]> INSERT INTO tbl3 SELECT t2.col1, t2.col3 FROM tbl4 t2 WHERE  
ROWNUM <= 10000;
```

4. 验证表 `tbl3` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl3;
```

查询结果如下：

```
+-----+-----+  
| col1 | col3 |  
+-----+-----+  
| 1 | 11 |  
| 2 | 22 |  
| 3 | 33 |  
+-----+-----+  
3 rows in set
```

结果显示表 `tbl3` 中已导入数据。

5. (可选)在 `EXPLAIN EXTENDED` 语句的返回结果的 `Note` 中，查看是否通过旁路导入写入的数据。

```
obclient [test]> EXPLAIN EXTENDED INSERT /*+ direct(true, 0, 'inc_replace')  
enable_parallel_dml parallel(16) */ INTO tbl3 SELECT t2.col1,t2.col3 FROM tbl4 t2  
WHERE ROWNUM <= 10000;
```

返回结果如下：

```
```shell
```

```
+-----+
+-----+
| Query Plan |
+-----+
+-----+
| =====
===== |
| ID|OPERATOR |NAME |EST.ROWS|EST.TIME(us)| |
|-----|
-- |
| 0 |PX COORDINATOR | |3 |34 ||
| 1 |  └─EXCHANGE OUT DISTR |:EX10002 |3 |33 ||
| 2 |  └─INSERT | |3 |32 ||
| 3 |  └─EXCHANGE IN DISTR | |3 |7 ||
| 4 |  └─EXCHANGE OUT DISTR (RANDOM) |:EX10001 |3 |7 ||
| 5 |  └─MATERIAL | |3 |3 ||
| 6 |  └─SUBPLAN SCAN |ANONYMOUS_VIEW1|3 |3 ||
| 7 |  └─LIMIT | |3 |3 ||
| 8 |  └─EXCHANGE IN DISTR | |3 |3 ||
| 9 |  └─EXCHANGE OUT DISTR |:EX10000 |3 |1 ||
|10 |  └─LIMIT | |3 |1 ||
|11 |  └─PX BLOCK ITERATOR | |3 |1 ||
|12 |  └─COLUMN TABLE FULL SCAN|T2 |3 |1 ||
| =====
===== |
```

```

| Outputs & filters: |
| ----- |
| 0 - output(nil), filter(nil), rowset=16 |
| 1 - output(nil), filter(nil), rowset=16 |
| dop=16 |
| 2 - output(nil), filter(nil) |
| columns([TBL3: ({TBL3: (TBL3.__pk_increment(0x7efaad22b0e0), TBL3.COL1
(0x7efaad2123f0), TBL3.COL3(0x7efaad212830))})), partitions(p0), |
| column_values([T_HIDDEN_PK(0x7efaad22b520)], [column_conv(NUMBER,PS:
(-1,0),NULL,ANONYMOUS_VIEW1.COL1(0x7efaad22a860))(0x7efaad22b930)],
[column_conv(NUMBER, |
| PS:(-1,0),NULL,ANONYMOUS_VIEW1.COL3(0x7efaad22aca0))(0x7efaad233850))] |
| 3 - output([T_HIDDEN_PK(0x7efaad22b520)], [ANONYMOUS_VIEW1.COL1
(0x7efaad22a860)], [ANONYMOUS_VIEW1.COL3(0x7efaad22aca0)]), filter(nil),
rowset=16 |
| 4 - output([T_HIDDEN_PK(0x7efaad22b520)], [ANONYMOUS_VIEW1.COL1
(0x7efaad22a860)], [ANONYMOUS_VIEW1.COL3(0x7efaad22aca0)]), filter(nil),
rowset=16 |
| is_single, dop=1 |
| 5 - output([ANONYMOUS_VIEW1.COL1(0x7efaad22a860)], [ANONYMOUS_VIEW1.
COL3(0x7efaad22aca0)]), filter(nil), rowset=16 |
| 6 - output([ANONYMOUS_VIEW1.COL1(0x7efaad22a860)], [ANONYMOUS_VIEW1.
COL3(0x7efaad22aca0)]), filter(nil), rowset=16 |
| access([ANONYMOUS_VIEW1.COL1(0x7efaad22a860)], [ANONYMOUS_VIEW1.COL3
(0x7efaad22aca0)]) |
| 7 - output([T2.COL1(0x7efaad229470)], [T2.COL3(0x7efaad229f40)]), filter(nil),
rowset=16 |
| limit(cast(FLOOR(cast(10000, NUMBER(-1, -85)))(0x7efaad227ef0))
(0x7efaad25ac30), BIGINT(-1, 0))(0x7efaad25b6d0)), offset(nil) |

```



```

| 8 - output([T2.COL1(0x7efaad229470)], [T2.COL3(0x7efaad229f40)]), filter(nil),
rowset=16 |
| 9 - output([T2.COL1(0x7efaad229470)], [T2.COL3(0x7efaad229f40)]), filter(nil),
rowset=16 |
| dop=16 |
| 10 - output([T2.COL1(0x7efaad229470)], [T2.COL3(0x7efaad229f40)]), filter(nil),
rowset=16 |
| limit(cast(FLOOR(cast(10000, NUMBER(-1, -85))(0x7efaad227ef0))
(0x7efaad25ac30), BIGINT(-1, 0))(0x7efaad25b6d0)), offset(nil) |
| 11 - output([T2.COL1(0x7efaad229470)], [T2.COL3(0x7efaad229f40)]), filter(nil),
rowset=16 |
| 12 - output([T2.COL1(0x7efaad229470)], [T2.COL3(0x7efaad229f40)]), filter(nil),
rowset=16 |
| access([T2.COL1(0x7efaad229470)], [T2.COL3(0x7efaad229f40)]), partitions(p0) |
| limit(cast(FLOOR(cast(10000, NUMBER(-1, -85))(0x7efaad227ef0))
(0x7efaad25ac30), BIGINT(-1, 0))(0x7efaad25b6d0)), offset(nil),
is_index_back=false, |
| is_global_index=false, |
| range_key([T2.__pk_increment(0x7efaad25a720)]), range(MIN ; MAX)always true
|
| Used Hint: |
| ----- |
| /*+ |
| |
| USE_PLAN_CACHE( NONE ) |
| PARALLEL(16) |
| ENABLE_PARALLEL_DML |
| DIRECT(TRUE, 0, 'INC_REPLACE') |
| */ |

```

```

| Qb name trace: |
| ----- |
| stmt_id:0, stmt_type:T_EXPLAIN |
| stmt_id:1, INS$1 |
| stmt_id:2, SEL$1 |
| stmt_id:3, parent:SEL$1 > SEL$658037CB > SEL$DCAFFB86 |
| Outline Data: |
| ----- |
| /*+ |
| BEGIN_OUTLINE_DATA |
| PARALLEL(@"SEL$DCAFFB86" "T2"@"SEL$1" 16) |
| FULL(@"SEL$DCAFFB86" "T2"@"SEL$1") |
| USE_COLUMN_TABLE(@"SEL$DCAFFB86" "T2"@"SEL$1") |
| MERGE(@"SEL$658037CB" < "SEL$1") |
| USE_PLAN_CACHE( NONE ) |
| PARALLEL(16) |
| ENABLE_PARALLEL_DML |
| OPTIMIZER_FEATURES_ENABLE('4.3.3.0') |
| DIRECT(TRUE, 0, 'INC_REPLACE') |
| END_OUTLINE_DATA |
| */ |
| Optimization Info: |
| ----- |
| T2: |
| table_rows:3 |
| physical_range_rows:3 |
| logical_range_rows:3 |
| index_back_rows:0 |
| output_rows:3 |

```

```
| table_dop:16 |
| dop_method:Global DOP |
| available_index_name:[TBL4] |
| stats info:[version=0, is_locked=0, is_expired=0] |
| dynamic sampling level:0 |
| estimation method:[DEFAULT, STORAGE] |
| Plan Type: |
| DISTRIBUTED |
| Note: |
| Degree of Parallelism is 16 because of hint |
| Direct-mode is enabled in insert into select |
| Expr Constraints: |
| cast(FLOOR(cast(10000, NUMBER(-1, -85))), BIGINT(-1, 0)) >= 1 result is TRUE |
+-----+
-----+
96 rows in set (0.007 sec)
```

## 46.4 使用 CREATE TABLE AS SELECT 语句旁路导入数据

CREATE TABLE AS SELECT 语句通过设置 DIRECT() 和 PARALLEL() Hint 来指定旁路导入的导入方式，在未指定 Hint 时能基于配置项 [default\\_load\\_mode](#) 确定导入数据的行为。

### 46.4.7 使用语法

```
CREATE /*+ [DIRECT(need_sort,max_error,{inc|inc_replace})] parallel(N) */ TABLE
table_name [AS] select_sentence
```

更多 CREATE TABLE AS SELECT 语法的信息，请参见 [CREATE TABLE（MySQL 模式）](#) 和 [CREATE TABLE（Oracle 模式）](#)。

## 参数解释：

参数	描述
DIRECT()	<p>使用 Hint 启用旁路导入功能。DIRECT() 参数解释如下：</p> <ul style="list-style-type: none"> <li>• <code>need_sort</code>：表示写入的数据是否需要排序，值为 <code>bool</code> 类型： <ul style="list-style-type: none"> <li>■ <code>true</code>：表示需要排序。</li> <li>■ <code>false</code>：表示不需要排序。</li> </ul> </li> <li>• <code>max_error</code>：表示最大容忍的错误行数。值为 <code>INT</code> 类型，超过这个数值导入任务执行会失败。</li> <li>• <code>inc inc_replace</code>：表示增量旁路导入的两种模式，取值须使用英文单引号包起来。 <ul style="list-style-type: none"> <li>■ <code>inc</code>：表示增量导入，检查主键是否重复，支持 <code>INSERT</code> 和 <code>IGNORE</code> 语义。</li> <li>■ <code>inc_replace</code>：表示增量导入，但不检查主键是否重复，相当于 <code>REPLACE</code> 语义的增量导入。</li> </ul> </li> </ul> <h3>46.4.7.1 注意</h3> <p>当 <code>load_mode</code> 取值为 <code>inc_replace</code> 时，<code>LOAD DATA</code> 语句中不允许有 <code>REPLACE</code> 或 <code>IGNORE</code> 关键字。</p>
parallel(N)	加载数据的并行度，必填项，取值是大于 1 的整数。

## 46.4.8 使用示例

MySQL 模式

Oracle 模式

使用旁路导入将表 `tbl1` 中的数据导入到其他表中。

1. 创建表 `tbl1`。

```
obclient [test]> CREATE TABLE tbl1(c1 int);
```

2. 插入数据到表 `tbl1`。

```
obclient [test]> INSERT INTO tbl1 VALUES (1),(2),(3);
```

3. 查询表 `tbl1` 中的数据。

```
obclient [test]> SELECT * FROM tbl1;
```

查询结果如下：

```
+-----+
| c1 |
+-----+
| 1 |
| 2 |
| 3 |
+-----+
3 rows in set (0.027 sec)
```

4. 使用旁路导入将表 `tbl1` 中数据导入到表 `tbl2`。

- 指定 `CREATE TABLE AS SELECT` 语句的 Hint。

- 使用 `inc` Hint 旁路导入数据。

```
obclient [test]> CREATE /*+ direct(true, 0, 'inc') parallel(4) */ TABLE tbl2 AS
SELECT * FROM tbl1;
```

- 使用 `inc_replace` Hint 旁路导入数据。

```
obclient [test]> CREATE /*+ direct(true, 0, 'inc_replace') parallel(4) */ TABLE
tbl2 AS SELECT * FROM tbl1;
```

- 不指定 `CREATE TABLE AS SELECT` 语句的 Hint。

设置配置项 `default_load_mode` 的值为 `INC_DIRECT_WRITE` 或 `INC_REPLACE_DIRECT_WRITE`。

```
obclient [test]> ALTER SYSTEM SET default_load_mode ='INC_DIRECT_WRITE';
```

```
obclient [test]> CREATE TABLE tbl2 AS SELECT * FROM tbl1;
```

5. 验证表 `tbl2` 中是否已导入数据。

```
obclient [test]> SELECT * FROM tbl2;
```

查询结果如下：

```
+-----+
```

```
| c1 |
```

```
+-----+
```

```
| 1 |
```

```
| 3 |
```

```
| 2 |
```

```
+-----+
```

```
3 rows in set (0.050 sec)
```

结果显示表 `tbl2` 中已导入数据。

使用旁路导入将表 `tbl1` 中的数据导入到其他表中。

1. 创建表 `tbl1` 。

```
obclient [SYS]> CREATE TABLE tbl1(c1 int);
```

2. 插入数据到表 `tbl1` 。

```
obclient [SYS]> INSERT INTO tbl1 VALUES (1),(2),(3);
```

3. 查询表 `tbl1` 中的数据。

```
obclient [SYS]> SELECT * FROM tbl1;
```

查询结果如下：

```
+-----+
| C1 |
+-----+
| 1 |
| 2 |
| 3 |
+-----+
3 rows in set (0.045 sec)
```

#### 4. 使用旁路导入将表 `tbl1` 中数据导入到表 `tbl2`。

- 指定 `CREATE TABLE AS SELECT` 语句的 Hint。
  - 使用 `inc` Hint 旁路导入数据。

```
obclient [SYS]> CREATE /*+ direct(true, 0, 'inc') parallel(4) */ TABLE tbl2 AS
SELECT * FROM tbl1;
```

- 使用 `inc_replace` Hint 旁路导入数据。

```
obclient [SYS]> CREATE /*+ direct(true, 0, 'inc_replace') parallel(4) */ TABLE
tbl2 AS SELECT * FROM tbl1;
```

- 不指定 `CREATE TABLE AS SELECT` 语句的 Hint。

设置配置项 `default_load_mode` 的值为 `INC_DIRECT_WRITE` 或 `INC_REPLACE_DIRECT_WRITE`。

```
obclient [SYS]> ALTER SYSTEM SET default_load_mode = 'INC_DIRECT_WRITE';
```

```
obclient [SYS]> CREATE TABLE tbl2 AS SELECT * FROM tbl1;
```

#### 5. 验证表 `tbl2` 中是否已导入数据。

```
obclient [SYS]> SELECT * FROM tbl2;
```

查询结果如下：

```
+-----+
| C1 |
+-----+
| 3 |
| 2 |
| 1 |
+-----+
3 rows in set (0.013 sec)
```

结果显示表 `tbl2` 中已导入数据。

## 46.5 相关文档

- [表与表之间的数据迁移](#)
- [SQL 执行计划简介](#)
- [连接方式概述](#)
- [删除表](#)



## 47 使用 OBLOADER 旁路导入数据

本文介绍如何使用 OBLOADER 旁路导入数据。OBLOADER 旁路导入模式的相关命令行选项如下：

### 注意

OBLOADER 旁路导入模式支持连接 OBCServer 和 ODP。对应的版本要求：

- 连接 OBCServer 时：要求 OBCServer 必须为 V4.2.0 及之后版本。
- 连接 ODP 时：要求 ODP V4.3.0 及之后版本，且 OBCServer 必须为 V4.2.1 及之后版本。

命令行选项	说明	云数据库 OceanBase & ODP	OceanBase 数 据库 & ODP	OceanBase 数 据库 & OBCServer
--direct	标识使用旁路导入。	必选	必选	必选
--parallel	服务端并发度。默认值 1，建议与租户 CPU 规格一致。 <b>建议指定该选项以保证性能稳定。</b>	非必选	非必选	非必选

--rpc-port	<p>服务端 inner rpc port。获取方式：</p> <ul style="list-style-type: none"> <li>● 连接 ODP 服务端时： <ul style="list-style-type: none"> <li>● 云数据库 OceanBase 环境下，ODP RPC Port 默认 3307。</li> </ul> </li> <li>● OceanBase 数据库环境下，默认端口 2885；如果需要自定义，可以在启动 ODP 时通过 <code>-s</code> 选项进行指定。</li> <li>● 连接 ODBServer 服务端时，sys 租户下查询系统视图 DBA_OB_SERVERS 即可获取 ODBServer 的 RPC 端口，默认端口 2882。</li> </ul>	必选	必选	必选
-u(--user)	数据库用户名。	必选	必选	必选
-P(--port)	SQL 端口号。	必选	必选	必选
-t(--tenant)	集群的租户名。	非必选 若不填可能会导致跳过分区计算	必选	必选
-c(--cluster)	数据库的集群名。	非必选	必选	-
--public-cloud	指定云数据库 OceanBase 运行环境。	必选	-	-
--no-sys	用于标识不依赖 sys 租户。仅用于 OceanBase 数据库 V4.0.0 之前的版本。	非必选	非必选	非必选

--sys-user	用于标识依赖 sys 租户的 user。若不填，默认为 root。仅用于 OceanBase 数据库 V4.0.0 之前的版本。	非必选 与 --no-sys 互斥	非必选 与 --no-sys 互斥	非必选 与 --no-sys 互斥
--sys-password	用于标识依赖 sys 租户的密码。仅用于 OceanBase 数据库 V4.0.0 之前的版本。	非必选 与 --no-sys 互斥	非必选 与 --no-sys 互斥	非必选 与 --no-sys 互斥

## 47.1 导入示例

使用以下命令，将数据导入至 OceanBase 数据库：

```
bin/obloader -h xx.x.x.xx -P 2883 -u TPCHE -t oboracle -c OB4216 -p -D TPCHE --table
LINEITEM --external-data --csv -f /data/1/tpch/s4/bak/ --truncate-table --column-
separator='|' --thread 16 --rpc-port=2885 --direct --parallel=16
```

旁路导入不是连接 OBDServer 的 SQL 端口（默认 2881）而是连接 RPC 端口（默认 2882）。OBLADER 如果绕过 ODP 而直连 OBDServer，通过额外指定 RPC 端口即可实现旁路导入。ODP V4.3.0 的 RPC 端口是 2885。但是生产中不推荐客户端绕过 ODP。因为原本业务数据的主副本位置对应用客户端是透明的（客户端不需要知道数据在哪个 OBDServer 节点上，ODP 会负责 SQL 路由）。直连 OBDServer 旁路写入数据，如果主副本不在这个节点，将产生跨机事务。

以上命令将 `/data/1/tpch/s4/bak/` 目录下所有已支持的 CSV 数据文件导入到表 `LINEITEM` 中。如果您的数据文件大小为 TB 以上，请使用旁路导入提高效率。配合更大的租户资源规格，效率将更高。

## 47.2 旁路导入模式相关参数

您可以在 `{ob-loader-dumper}/conf` 目录下的 `session.config.json` 文件中配置旁路导入参数。

示例：

```
"direct_path_load": {
"rpc_connect_timeout": "15000",
"rpc_execute_timeout": "20000",
```

```
"runtime_retry_times": "5",
"runtime_retry_intervals": "50",
"task_timeout": "2592000000000",
"heartbeat_timeout": "60000000"
}
```

- `rpc_connect_timeout`: RPC 连接超时时间, 单位为毫秒。
- `rpc_execute_timeout`: RPC 执行超时时间, 单位为毫秒。
- `runtime_retry_times`: 运行时最大重试次数。如果因为某些原因操作失败, 将会根据该参数进行重试。
- `runtime_retry_intervals`: 重试间隔时间。在重试操作前等待的时间长度, 单位为毫秒。
- `task_timeout`: 配置操作的超时时间。如果在配置的时限内未完成操作, 则被视为超时。单位为微秒, 默认值为 0, 表示无超时限制。
- `heartbeat_timeout`: 设置心跳超时时间, 用于检测导入操作的活跃状态。单位为微秒, 默认值为 0, 表示不启用心跳检测。

## 47.3 注意事项

- 旁路写入需要使用 RPC 端口传输数据, 并非 SQL 协议端口。
- 基于表粒度进行整体提交, 并非会话级/事务级的提交操作。
- 暂不支持重试或者断点续传。
- 暂不支持 bit 类型数据。
- 暂不支持虚拟生成列。
- 对于数据量较小的导入任务, 不建议使用旁路导入。
- 目前旁路导入的写入请求并非幂等。极少数情况下, 数据包重发可能会导致最后排序出现数据重复, 导入无索引堆表可能出现重复数据。
- 指定 `--replace-data` 命令行选项时, 仍无法处理唯一索引冲突。
- 参数 `--thread` 命令行选项与 `--parallel` 命令行选项的区别:
  - `--thread` 表示客户端到服务端的连接池, 由客户端维护。
  - `--parallel` 为 OBCServer 可以调用的工作线程数, 用于写入数据与排序。
  - 使用时, 建议 `--thread` 与 `--parallel` 保持一致。