# Auto Replication 利用数据库备份自动设置多主从库(mysql v5.7 mulit slave)

1. 文件列表
   1. auto.replication.py (主文件)  
      [auto.replication.py](/download/attachments/5280585/auto.replication.py?version=2&modificationDate=1494300878000&api=v2)
   2. config.py (配置文件)  
      [config.py](/download/attachments/5280585/config.py?version=1&modificationDate=1490865058000&api=v2)
   3. recover\_report.py(报告发送模块)  
      [recover\_report.py](/download/attachments/5280585/recover_report.py?version=1&modificationDate=1490865058000&api=v2)
   4. auto.replication.template.html(HTML邮件模板文件)  
      [auto.replication.template.html](/download/attachments/5280585/auto.replication.template.html?version=1&modificationDate=1490865058000&api=v2)
2. 使用说明
   1. 利用sshfs建立共享目录
      1. [利用sshfs 挂载远程目录](/pages/viewpage.action?pageId=5931419)
   2. 拷贝备份文件进行恢复
      1. 拷贝备份文件
      2. 执行脚本

获取change\_master

mysql -uqihang.li -p'yourPassword' -BNe "select Host,User\_name,User\_password,port,Enabled\_auto\_position,Channel\_name from mysql.slave\_master\_info;"|awk --posix '/^[0-9]{3}\./{print "change master to master\_host=\""$1"\",master\_user=\""$2"\",master\_password=\""$3"\",MASTER\_PORT="$4",master\_auto\_position="$5" for channel \""$6"\";"}'

代码展示

auto.replication.py

#!/usr/bin/python

# encoding:utf-8

# author:sultan

# version:0.1

# datetime:2017-02-16 10:33:59

import commands

import datetime

import logging

import os

import re

import shutil

import statvfs

import sys

import time

import traceback

# from warnings import filterwarnings

import MySQLdb

# import pydevd

import tarfile

import multiprocessing

from recover\_report import recover\_report

from config import configure, \_\_getopts\_\_

# pydevd.settrace('localhost', port=4466, stdoutToServer=True, stderrToServer=True)

# pydevd.settrace('172.16.12.179', port=51234, stdoutToServer=True, stderrToServer=True)

# filterwarnings('error', category=MySQLdb.Warning)

class preflight:

applications = {} # 三方应用

mysql\_config = {} # mysql配置

backup\_file\_list = [] # 备份文件列表

expired\_file\_list = [] # 过期文件

def check\_applications(self, \_app\_list): # 检查所需三方应用

for \_app in \_app\_list:

\_status, \_applicationPath = commands.getstatusoutput("which %s" % (\_app))

if \_status == 0:

logging.info("找到应用 %s,位置 %s" % (\_app, \_applicationPath))

self.applications[\_app] = \_applicationPath

else:

logging.error("未找到应用 %s 还原操作即将退出" % (\_app))

raise SystemExit("未找到应用 %s 还原操作即将退出" % (\_app))

return

def check\_diskfree(self, \_disk\_path, \_disk\_free\_percent, \_disk\_free\_size): # 检查磁盘空间

free\_percent, free\_size = self.\_\_disk\_stat\_\_(\_disk\_path)

if free\_percent > \_disk\_free\_percent and free\_size > \_disk\_free\_size:

logging.info("磁盘剩余空间满足需求 剩余空间 %sG (%s%%)" % (("%.2f" % free\_size), ("%.2f" % free\_percent)))

pass

else:

logging.error("磁盘空间不足,退出程序 剩余空间 %sG (%s%%) 还原操作即将退出" % (("%.2f" % free\_size), ("%.2f" % free\_percent)))

raise SystemExit

return

def check\_DBinstans(self, \_defaults\_file, \_instance): # 检查db实例

self.\_\_get\_mysql\_config\_\_(\_defaults\_file, \_instance)

with open(self.mysql\_config["pid"], 'r') as \_pidfile:

\_pid = \_pidfile.read().strip()

\_status, \_pidcount = commands.getstatusoutput("ps -p %s|wc -l" % (\_pid))

if \_status == 0 and int(\_pidcount) > 1:

logging.info("mysql实例正常")

else:

logging.error("mysql实例未正常启动 还原操作即将退出")

raise SystemExit("mysql实例未正常启动 还原操作即将退出")

return

def check\_backup\_dir(self, \_backup\_dir, \_backup\_file\_reg, \_check\_days, \_max\_backup\_file\_num,

\_exclude\_host\_name, \_threading\_count): # 检查备份目录

\_backup\_file\_list = []

for \_dir in os.listdir(\_backup\_dir):

if os.path.isdir(\_backup\_dir + \_dir):

for \_dir\_path, \_dir\_names, \_file\_names in os.walk(\_backup\_dir + \_dir):

if \_file\_names <> []:

\_curdir\_match\_file\_list = [] # 当前目录下匹配备份文件名规则的所有文件

for \_file in sorted(\_file\_names, reverse=True):

if re.match((\_backup\_file\_reg), \_file):

\_curdir\_match\_file\_list.append(\_file)

for \_file in \_curdir\_match\_file\_list:

\_status, \_output = commands.getstatusoutput( # 检查文件是否被打开

"%s %s/%s" % (self.applications["lsof"], \_dir\_path, \_file))

if os.path.basename(\_dir\_path) in \_exclude\_host\_name: # 判断是否在排除服务器内

logging.warn("%s在还原排除列表内,将跳过还原操作" % (\_dir\_path))

break

elif \_output.\_\_len\_\_() == 0:

\_file\_stat = os.stat("%s/%s" % (\_dir\_path, \_file))

\_backup\_file\_list.append(

("%s/%s" % (\_dir\_path, \_file), \_file\_stat.st\_size,

\_file\_stat.st\_ctime)) # (文件名,文件大小,文件修改时间)

if (datetime.datetime.now() - datetime.datetime.fromtimestamp( # 过期时间判断

\_file\_stat.st\_ctime)).days > \_check\_days:

logging.warn("%s 文件%s 过期.文件时间:%s" % (\_dir\_path, \_file,

time.strftime("%Y-%m-%d %X",

time.localtime(

\_file\_stat.st\_ctime))))

self.expired\_file\_list.append(

(("dirPath", \_dir\_path), ("fileName", \_file),

("changeTime", time.strftime("%Y-%m-%d %X",

time.localtime(

\_file\_stat.st\_ctime))))) # (目录,文件名,修改时间)

\_backup\_file = \_file

break

else:

logging.warn("%s/%s当前正在被使用,将跳过此文件 %s" % (\_dir\_path, \_file, \_output))

for \_remove\_file in [\_f for \_f in \_curdir\_match\_file\_list[\_max\_backup\_file\_num:] if

\_f <> \_backup\_file]: # 删除过多文件

logging.info("删除过多备份文件 %s/%s" % (\_dir\_path, \_remove\_file))

try:

os.remove("%s/%s" % (\_dir\_path, \_remove\_file))

except Exception as e:

logging.error("%s/%s 删除失败\n错误信息%s" % (\_dir\_path, \_remove\_file, traceback.format\_exc(e)))

# 生成还原文件列表(必还原数据库+单双日还原数据库)

\_backup\_file\_list.sort(cmp=lambda \_x, \_y: cmp(\_x[1], \_y[1])) # 按文件大小排序

self.backup\_file\_list = \_backup\_file\_list

self.expired\_file\_list.sort(cmp=lambda \_x1, \_y1: cmp(\_x1[2], \_y1[2])) # 按修改时间排序

return

def \_\_disk\_stat\_\_(self, \_path):

try:

logging.info("磁盘检查 %s" % (\_path))

\_vfs = os.statvfs(\_path)

except:

logging.warning("磁盘 %s 不存在.更改为默认磁盘 /" % (\_path))

\_vfs = os.statvfs("/")

\_available = \_vfs[statvfs.F\_BAVAIL] \* \_vfs[statvfs.F\_BSIZE] / (1024.0 \* 1024 \* 1024)

\_capacity = \_vfs[statvfs.F\_BLOCKS] \* \_vfs[statvfs.F\_BSIZE] / (1024.0 \* 1024 \* 1024)

\_available\_percent = (\_available \* 1.00) / (\_capacity \* 1.00) \* 100

return \_available\_percent, \_available

def \_\_get\_mysql\_config\_\_(self, \_file, \_instance):

if os.path.exists(\_file):

\_status, \_cfglist = commands.getstatusoutput(

"%s -c %s %s" % (self.applications["my\_print\_defaults"], \_file, \_instance))

if \_status == 0:

for \_item in \_cfglist.split("\n"):

if re.match(r"^--socket=.\*", \_item):

self.mysql\_config["socket"] = \_item.split("=")[1]

elif re.match(r"^--port=.\*", \_item):

self.mysql\_config["port"] = \_item.split("=")[1]

elif re.match(r"^--pid(\_|-)file=.\*", \_item):

self.mysql\_config["pid"] = \_item.split("=")[1]

elif re.match(r"^--datadir=.\*", \_item):

self.mysql\_config["datadir"] = \_item.split("=")[1]

else:

logging.error("mysql 配置文件不存在 %s" % (\_file))

return

def \_\_enter\_\_(self):

return self

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

return

class partition\_recover:

backup\_element\_list = {} # 解压后部分还原必须元素文件列表

backup\_root\_path = "" # 解压后备份根目录

backup\_db\_dir\_list = [] # 解压后数据库目录列表

restore\_db\_list = [] # 最终被还原的数据库列表

table\_list = {} # 数据表列表

backup\_Gtid = [] # 数据库GTID

channel\_name='' # CHANNEL NAME

def \_\_init\_\_(self, \_file\_path, \_mysql\_user, \_mysql\_pass, \_mysql\_socket, \_host\_name):

self.conn = MySQLdb.connect(host='localhost', user=\_mysql\_user, passwd=\_mysql\_pass,

unix\_socket=\_mysql\_socket) # 还原实例mysql连接

self.conn.autocommit(True)

self.cur = self.conn.cursor(cursorclass=MySQLdb.cursors.DictCursor)

self.cur.execute("SET FOREIGN\_KEY\_CHECKS = 0;")

try:

self.cur.execute("show global variables like 'version';")

\_version = self.cur.fetchall()[0]["Value"]

self.cur.execute("show global variables like 'innodb\_default\_row\_format';")

\_default\_row\_format = self.cur.fetchall()[0]["Value"]

if re.match(r'^5.7.\*', \_version) and \_default\_row\_format == 'dynamic':

self.cur.execute("set global innodb\_default\_row\_format='Compact';")

except:

logging.warning("[%s] 当前数据库版本 不需要设置innodb\_default\_row\_format='Compact'" % (\_host\_name))

self.backup\_file\_path = \_file\_path # 还原文件名

self.channel\_name=re.findall('[a-zA-Z]{2,}\d{1,}-?\d\*', os.path.basename(\_file\_path))[0].lower()

self.host\_name = \_host\_name # 线程标示| 还原服务器名 | 还原目录名

self.decompress\_target\_dir = self.\_\_check\_dir\_\_(\_file\_path, True) # 备份文件解压目录

self.abnormal\_dir = self.\_\_check\_dir\_\_(\_file\_path, False) # 还原异常处理目录

return

def decompress\_backup\_file(self, \_innobackupex\_path, \_element\_db\_list, \_element\_db\_schema, \_element\_db\_grants,

\_exclude\_db):

def check\_file\_element(\_threadname, \_file\_list, \_dir\_path, \_element\_db\_list, \_elemet\_db\_schema,

\_elemet\_db\_grants):

\_element\_file\_list = {}

# 检查文件必要内容

for \_file in \_file\_list:

if \_element\_db\_list == \_file and os.stat("%s/%s" % (\_dir\_path, \_file)).st\_size > 0:

\_element\_file\_list["backup\_db\_list"] = "%s/%s" % (\_dir\_path, \_file)

elif \_elemet\_db\_schema == \_file:

\_element\_file\_list["backup\_db\_schema"] = "%s/%s" % (\_dir\_path, \_file)

elif \_elemet\_db\_grants == \_file:

\_element\_file\_list["backup\_db\_grants"] = "%s/%s" % (\_dir\_path, \_file)

if \_element\_file\_list.\_\_len\_\_() == 3:

logging.info("[%s]-%s 元素文件完整有效" % (\_threadname, \_dir\_path))

return \_element\_file\_list

else:

logging.error("[%s]-%s 元素文件有异常" % (\_threadname, \_dir\_path))

raise SystemExit("[%s]-%s 元素文件有异常" % (\_threadname, \_dir\_path))

def apple\_log(\_threadname, \_innobackupex\_path, \_apply\_path): # apply-log

\_status, \_output = commands.getstatusoutput(

"%s --apply-log --export %s" % (\_innobackupex\_path, \_apply\_path))

if \_status == 0:

logging.info("[%s]-%s apply-log正常" % (\_threadname, \_apply\_path))

return

else:

logging.error("[%s]-%s apply-log失败 还原操作即将退出" % (\_threadname, \_apply\_path, \_output))

raise SystemExit("apply-log失败 还原操作即将退出")

\_next\_dir = os.path.basename(self.backup\_file\_path).split('.')[0] # 解压目录下级目录

self.db\_server\_name = \_next\_dir

with tarfile.open(self.backup\_file\_path) as \_tar:

logging.info("[%s]-开始解压备份文件%s到%s" % (self.host\_name, self.backup\_file\_path, self.decompress\_target\_dir))

# 解压tar包

try:

\_tar.extractall(self.decompress\_target\_dir)

except IOError as e:

logging.error("[%s]-备份文件%s 解压失败,将继续尝试后续操作\n[%s] 错误信息:%s" % (

self.host\_name, self.backup\_file\_path, self.host\_name, traceback.format\_exc(e)))

else:

logging.info("[%s]-备份文件%s 解压完毕" % (self.host\_name, self.backup\_file\_path))

# 获取元素文件和备份还原根目录

for \_dir\_path, \_dir\_names, \_file\_names in os.walk("%s/%s" % (self.decompress\_target\_dir, \_next\_dir)):

# 获取还原元素文件

if re.match(r"^%s/%s$" % (self.decompress\_target\_dir, \_next\_dir), \_dir\_path) and [\_x for \_x in (

\_element\_db\_list, \_element\_db\_schema, \_element\_db\_grants) if \_x in \_file\_names].\_\_len\_\_() >= 3:

self.backup\_element\_list = check\_file\_element(self.host\_name, \_file\_names, \_dir\_path,

\_element\_db\_list,

\_element\_db\_schema, \_element\_db\_grants)

continue

# 确定最终路径

if re.match(r"^%s/%s/FULL/[0-9]{4}-[0-9]{2}-[0-9]{2}\_[0-9]{2}-[0-9]{2}-[0-9]{2}$" % (

self.decompress\_target\_dir, \_next\_dir), \_dir\_path):

self.backup\_root\_path = \_dir\_path

self.backup\_db\_dir\_list = [\_d for \_d in \_dir\_names if \_d not in \_exclude\_db]

break

logging.info("[%s]-%s 开始apply-log" % (self.host\_name, self.backup\_root\_path))

apple\_log(self.host\_name, \_innobackupex\_path, self.backup\_root\_path)

return

def recover\_table\_schema(self, \_mysql\_path): # 还原表结构

def check\_cleansing\_thread\_db(\_threadname, \_cur, \_db\_list):

for \_db in \_db\_list:

\_sql = "select id,user,host,db,time,state,info from information\_schema.processlist where db='%s';" % (

\_db)

try:

\_cur.execute(\_sql)

except Exception, e:

traceback.print\_exc(e)

else:

for \_date in \_cur.fetchall():

if \_date["db"] in \_db\_list:

logging.warn("[%s]-DB:%s 可能被使用,即将关闭连接\n%s" % (\_threadname, \_date["db"], \_date.\_\_str\_\_()))

\_cur.execute("kill %s;" % (\_date["id"]))

# 清理保留的数据库

try:

\_cur.execute("drop database %s;" % (\_db))

except:

try:

\_cur.execute("drop database %s;" % (\_db))

except:

logging.info("[%s]-数据库 %s 不存在" % (\_threadname, \_db))

else:

logging.info("[%s]-删除已存在数据库 %s" % (\_threadname, \_db))

else:

logging.info("[%s]-删除已存在数据库 %s" % (\_threadname, \_db))

return

def select\_table\_list(\_threadname, \_cur, \_db\_list): # 修改表行模式

\_talbe\_list = {}

for \_db in \_db\_list:

\_talbe\_list[\_db] = []

\_cur.execute(

"select TABLE\_SCHEMA,TABLE\_NAME,ROW\_FORMAT from information\_schema.tables where table\_type='BASE TABLE' AND table\_schema = '%s' and ENGINE='innodb';" % (

\_db))

for \_date in \_cur.fetchall():

\_talbe\_list[\_date["TABLE\_SCHEMA"]].append(\_date["TABLE\_NAME"])

return \_talbe\_list

#停止 slave

try:

self.cur.execute("stop slave for channel '%s';"%(self.channel\_name))

logging.info("[%s] 成功停止Slave channel：%s" % (self.host\_name,self.channel\_name))

except:

logging.warning("[%s] 停止Slave channel：%s 失败" % (self.host\_name,self.channel\_name))

# 打开还原元素文件backup\_db\_list,所列数据库必须在解压目录内

with open(self.backup\_element\_list["backup\_db\_list"], 'r') as file\_db\_list:

self.restore\_db\_list = [\_l.strip() for \_l in file\_db\_list if \_l.strip() in self.backup\_db\_dir\_list]

logging.info("[%s]-开始清理被占用mysql session 和数据库 [%s]" % (self.host\_name, self.restore\_db\_list))

check\_cleansing\_thread\_db(self.host\_name, self.cur, self.restore\_db\_list) # 清理包含数据库列表的session连接

logging.info("[%s]-开始还原数据库表结构.表结构文件%s" % (self.host\_name, self.backup\_element\_list["backup\_db\_schema"]))

\_status, \_r = commands.getstatusoutput(

"%s -uroot <%s" % (\_mysql\_path, self.backup\_element\_list["backup\_db\_schema"]))

if \_status == 0:

logging.info("[%s]-数据库表结构文件%s 还原完毕" % (self.host\_name, self.backup\_element\_list["backup\_db\_schema"]))

else:

logging.error(

"[%s]-数据库表结构文件%s 还原异常,错误信息:%s" % (self.host\_name, self.backup\_element\_list["backup\_db\_schema"], \_r))

shutil.copy(self.backup\_element\_list["backup\_db\_schema"], "%s/%s.error.%s" % (

self.abnormal\_dir,

os.path.basename(self.backup\_element\_list["backup\_db\_schema"]),

time.strftime("%Y%m%d%H%M%S", time.localtime())))

self.table\_list = select\_table\_list(self.host\_name, self.cur, self.restore\_db\_list) # 组装table\_list

return

def exec\_partly\_recover(self, \_datedir):

def discard\_table(\_threadname, \_cur, \_db, \_table):

try:

\_cur.execute("alter table %s.%s discard tablespace;" % (\_db, \_table))

logging.debug("[%s]-Table:%s.%s discard 成功" % (\_threadname, \_db, \_table))

except MySQLdb.Warning, w:

logging.warning("[%s]-Table:%s.%s discard 失败\n[%s]-%s" % (

\_threadname, \_db, \_table, \_threadname, traceback.format\_exc(w)))

except MySQLdb.Error, e:

logging.error("[%s]-Table:%s.%s discard 失败\n[%s]-%s" % (

\_threadname, \_db, \_table, \_threadname, traceback.format\_exc(e)))

return

def move\_table\_spacse(\_threadname, \_datadir, \_backup\_root, \_db, \_table):

\_status, \_r = commands.getstatusoutput(

"mv -f %s/%s/%s.{ibd,cfg} %s/%s/" % (\_backup\_root, \_db, \_table, \_datadir, \_db))

if \_status == 0:

logging.debug("[%s]-文件%s/%s/%s.{ibd,cfg} 移动到%s/%s/ 成功" % (

\_threadname, \_backup\_root, \_db, \_table, \_datadir, \_db))

else:

logging.error(

"[%s]-文件%s/%s/%s.{ibd,cfg} 移动到%s/%s/ 失败\n%s" % (

\_threadname, \_backup\_root, \_db, \_table, \_datadir, \_db, \_r))

return

def chmod\_table\_spacse(\_threadname, \_datadir, \_db, \_table):

\_status, \_r = commands.getstatusoutput(

"chmod 640 %s/%s/%s.{ibd,cfg}" % (\_datadir, \_db, \_table))

if \_status == 0:

logging.debug("[%s]-文件%s/%s/%s.{ibd,cfg}权限修改成功" % (\_threadname, \_datadir, \_db, \_table))

else:

logging.error("[%s]-文件%s/%s/%s.{ibd,cfg}权限修改失败\n%s" % (\_threadname, \_datadir, \_db, \_table, \_r))

\_status, \_r = commands.getstatusoutput(

"chown mysql:mysql %s/%s/%s.{ibd,cfg}" % (\_datadir, \_db, \_table))

if \_status == 0:

logging.debug("[%s]-文件%s/%s/%s.{ibd,cfg}所有者修改成功" % (\_threadname, \_datadir, \_db, \_table))

else:

logging.error("[%s]-文件%s/%s/%s.{ibd,cfg}所有者修改失败\n%s" % (\_threadname, \_datadir, \_db, \_table, \_r))

return

def import\_table\_spacse(\_threadname, \_cur, \_db, \_table, \_datadir, \_abnormal\_dir):

try:

\_cur.execute("alter table %s.%s import tablespace;" % (\_db, \_table))

logging.debug("[%s]-Table:%s.%s import 成功" % (\_threadname, \_db, \_table))

except MySQLdb.Warning, w:

logging.warning("[%s]-Table:%s.%s import异常\n[%s]-%s" % (

\_threadname, \_db, \_table, \_threadname, traceback.format\_exc(w)))

commands.getstatusoutput(

"mv -f %s/%s/%s.{ibd,cfg} %s/" % (\_datadir, \_db, \_table, \_abnormal\_dir))

except MySQLdb.Error, e:

logging.error("[%s]-Table:%s.%s import 失败\n[%s]-%s" % (

\_threadname, \_db, \_table, \_threadname, traceback.format\_exc(e)))

commands.getstatusoutput(

"mv -f %s/%s/%s.{ibd,cfg} %s/" % (\_datadir, \_db, \_table, \_abnormal\_dir))

return

def table\_check(\_threadname, \_cur, \_db, \_table):

try:

\_cur.execute("select 1 from %s.%s order by 1 desc limit 1;" % (\_db, \_table))

logging.debug("[%s]-Table:%s.%s 还原成功" % (\_threadname, \_db, \_table))

except MySQLdb.Warning, w:

logging.warning("[%s]-Table Check:%s.%s 异常\n[%s]-%s" % (

\_threadname, \_db, \_table, \_threadname, traceback.format\_exc(w)))

except MySQLdb.Error, e:

logging.warning("[%s]-Table Check:%s.%s 异常\n[%s]-%s" % (

\_threadname, \_db, \_table, \_threadname, traceback.format\_exc(e)))

return

for \_db, \_tables in self.table\_list.items():

logging.info("[%s]-开始 %s 数据库部分还原" % (self.host\_name, \_db))

for \_t in \_tables:

discard\_table(self.host\_name, self.cur, \_db, \_t)

move\_table\_spacse(self.host\_name, \_datedir, self.backup\_root\_path, \_db, \_t)

chmod\_table\_spacse(self.host\_name, \_datedir, \_db, \_t)

import\_table\_spacse(self.host\_name, self.cur, \_db, \_t, \_datedir, self.abnormal\_dir)

table\_check(self.host\_name, self.cur, \_db, \_t)

logging.info("[%s]-已完成 %s 数据库部分还原完成" % (self.host\_name, \_db))

return

def get\_gtid(self, \_xtra\_binlog\_info):

if os.path.isfile("%s/%s" % (self.backup\_root\_path, \_xtra\_binlog\_info)):

with open("%s/%s" % (self.backup\_root\_path, \_xtra\_binlog\_info)) as \_gtid\_file:

for \_line in \_gtid\_file:

self.backup\_Gtid += re.findall(r"\w{8}-\w{4}-\w{4}-\w{4}-\w{12}:[\d:-]\*", \_line)

logging.info("[%s]-GTID:%s" % (self.host\_name, ','.join(self.backup\_Gtid)))

else:

self.backup\_Gtid = "ERROR GTID"

logging.error("[%s]-GTID获取失败" % (self.host\_name))

return

def \_\_check\_dir\_\_(self, \_path, isdecompress=False):

if isdecompress: # 如果为True 创建异常处理目录

\_dir\_path = "%s/%s" % (

os.path.dirname(\_path), time.strftime("%Y%m%d%H%M%S", time.localtime()))

else: # 否则创建

\_dir\_path = "%s/abnormal\_%s" % (

os.path.dirname(\_path), time.strftime("%Y%m%d%H%M%S", time.localtime()))

if not os.path.exists(\_dir\_path):

os.makedirs(\_dir\_path)

logging.info("[%s]-%s目录不存在,已将其创建" % (self.host\_name, \_dir\_path))

return \_dir\_path

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

self.cur.execute("SET FOREIGN\_KEY\_CHECKS = 1;")

self.conn.commit()

self.cur.close()

self.conn.close()

if not os.listdir(self.abnormal\_dir): # 如果异常处理目录为空,即删除

logging.info("[%s]-删除异常处理目录%s" % (self.host\_name, self.abnormal\_dir))

shutil.rmtree(self.abnormal\_dir)

else:

shutil.copy(self.backup\_element\_list["backup\_db\_schema"], "%s/%s.abnormal.%s" % (

self.abnormal\_dir,

os.path.basename(self.backup\_element\_list["backup\_db\_schema"]),

time.strftime("%Y%m%d%H%M%S", time.localtime())))

logging.warning("[%s]-异常处理目录%s非空,请检查日志和目录" % (self.host\_name, self.abnormal\_dir))

shutil.rmtree(self.decompress\_target\_dir) # 删除解压目录

logging.info("[%s]-删除解压目录%s" % (self.host\_name, self.decompress\_target\_dir))

if exc\_type <> None:

logging.error("\n[%s]-exc\_type:%s\n[%s]-exc\_val:%s\n[%s]-exc\_tb:%s" % (

self.host\_name, exc\_type, self.host\_name, exc\_val, self.host\_name, traceback.format\_exc(exc\_tb)))

else:

# 备份 Xtrabackup 备份信息文件

for \_xtra\_file in configure["xtrabackup\_file\_list"]:

if os.path.isfile("%s/%s" % (self.backup\_root\_path, \_xtra\_file)):

try:

shutil.copy("%s/%s" % (self.backup\_root\_path, \_xtra\_file),

"%s/%s" % (os.path.dirname(self.backup\_file\_path), \_xtra\_file))

logging.WARNING("[%s] 将%s/%s 拷贝到%s/%s" % (

self.host\_name, self.backup\_root\_path, \_xtra\_file, os.path.dirname(self.backup\_file\_path),

\_xtra\_file))

except:

logging.WARNING("[%s] %s/%s 拷贝失败" % (self.host\_name, self.backup\_root\_path, \_xtra\_file))

return

def \_\_enter\_\_(self):

return self

class set\_replication:

current\_gtid = [] # 当前实例GTID (在停止同步后,进行更新)

restore\_object\_list = [] # restore 执行结果集

current\_change\_master\_sql = [] # 当前实例 change master sql

finally\_change\_master\_sql = [] # 最终change master sql

isSlave = True # 实例是否为从服务器 True:是slave 服务器 False:否

finally\_gtid = [] # 最终的GTID

slave\_status\_list = [] # slave status

def \_\_init\_\_(self, \_mysql\_user, \_mysql\_pass, \_mysql\_socket, ):

self.conn = MySQLdb.connect(host='localhost', user=\_mysql\_user, passwd=\_mysql\_pass,

unix\_socket=\_mysql\_socket) # 还原实例mysql连接

self.conn.autocommit(True)

self.cur = self.conn.cursor(cursorclass=MySQLdb.cursors.DictCursor)

# 确认 gtid\_mode

try:

self.cur.execute("show global variables like 'gtid\_mode';")

\_gtid\_mode = self.cur.fetchall()[0]["Value"]

except Exception, e:

logging.error("未能正确获取 gtid\_mode 状态\n%s" % (traceback.format\_exc(e)))

raise UserWarning("未能正确获取 gtid\_mode 状态")

if \_gtid\_mode <> "ON":

logging.error("实例未开启GTID功能 gtid\_mode:%s" % (\_gtid\_mode))

raise UserWarning("实例未开启GTID功能 gtid\_mode:%s" % (\_gtid\_mode))

return

def stop\_replication(self):

self.cur.execute("stop slave;")

logging.info("已关闭 master-slave replication")

try:

self.cur.execute("show master status")

self.current\_gtid = re.findall(r"\w{8}-\w{4}-\w{4}-\w{4}-\w{12}:[\d:-]\*",

self.cur.fetchall()[0]["Executed\_Gtid\_Set"])

logging.info("实例 当前GTID为 %s" % (",".join(self.current\_gtid)))

except Exception, e:

logging.error("%s" % (traceback.format\_exc(e)))

return

def get\_change\_master\_sql(self):

self.cur.execute(

"select Host,User\_name,User\_password,port,Enabled\_auto\_position,Channel\_name from mysql.slave\_master\_info;")

\_slave\_result = self.cur.fetchall()

for \_row in \_slave\_result:

self.current\_change\_master\_sql.append(

"change master to master\_host='%s',master\_user='%s',master\_password='%s',MASTER\_PORT=%d,master\_auto\_position=%d for channel '%s';" % (

\_row["Host"], \_row["User\_name"], \_row["User\_password"],

\_row["port"], \_row["Enabled\_auto\_position"],

\_row["Channel\_name"]))

if self.current\_change\_master\_sql <> []:

logging.info("当前实例 change master sql \n%s" % ("\n".join(self.current\_change\_master\_sql)))

else:

self.isSlave = False

logging.warning("当前实例非 Slave 服务器")

self.finally\_change\_master\_sql += self.current\_change\_master\_sql

for \_restore\_object in self.restore\_object\_list:

\_channel\_name = self.\_get\_result\_channel\_name\_(\_restore\_object=dict(\_restore\_object))

if \_channel\_name not in [\_c["Channel\_name"] for \_c in

\_slave\_result]:

self.finally\_change\_master\_sql.append(

"change master to master\_host='%s',master\_user='%s',master\_password='%s',MASTER\_PORT='%s',master\_auto\_position=%s for channel '%s';" % (

"{ip\_address}", "replication", "pass", "{port}", 1, \_channel\_name))

return

def start\_replication(self,\_restore\_object\_list):

def modife\_gtid(\_current\_gtid, \_restore\_object\_list):

def \_get\_result\_gtid\_(\_restore\_object):

# 还原对象GTID

\_new\_gtid = re.findall(r"\w{8}-\w{4}-\w{4}-\w{4}-\w{12}:[\d:-]\*", \_restore\_object["GTID"])

if \_new\_gtid <> []:

logging.info("新还原对象GTID 是 %s" % (','.join(\_new\_gtid)))

else:

logging.error("未能正确获取新还原对象GTID : [%s]" % (\_new\_gtid))

raise UserWarning("未能正确获取新还原对象GTID: [%s]" % (\_new\_gtid))

return \_new\_gtid

\_gtid = []

for \_restore\_object in \_restore\_object\_list:

\_new\_gtid = \_get\_result\_gtid\_(dict(\_restore\_object))

\_gtid+=\_new\_gtid

for \_c\_gtid in \_current\_gtid:

\_c\_uuid = re.findall(r"\w{8}-\w{4}-\w{4}-\w{4}-\w{12}", \_c\_gtid)[0]

\_c\_sequence = re.findall(r"\w{8}-\w{4}-\w{4}-\w{4}-\w{12}:([\d:-]\*)", \_c\_gtid)[0]

logging.debug("当前GTID元素 UUID:%s Sequence:%s" % (\_c\_uuid, \_c\_sequence))

if \_c\_uuid not in [re.findall(r"\w{8}-\w{4}-\w{4}-\w{4}-\w{12}", \_n\_gtid)[0] for \_n\_gtid in \_gtid]:

\_gtid.append("%s:%s" % (\_c\_uuid, \_c\_sequence))

logging.debug("当前GTID元素%s:%s不在GTID列表内,加入GTID列表" % (\_c\_uuid, \_c\_sequence))

logging.info("最终GTID:%s" % (",".join(\_gtid)))

return \_gtid

self.restore\_object\_list = \_restore\_object\_list

self.finally\_gtid = modife\_gtid(self.current\_gtid, self.restore\_object\_list)

try:

self.cur.execute("reset master;")

self.cur.execute("set global gtid\_purged='%s';" % (",".join(self.finally\_gtid)))

logging.info("gtid\_purged 已置为 \n%s:"%(",".join(self.finally\_gtid)))

if self.isSlave:

self.cur.execute("reset slave all;")

logging.info("slave status清空成功")

for \_c\_sql in self.current\_change\_master\_sql:

self.cur.execute(\_c\_sql)

logging.info("执行 %s 成功"%(\_c\_sql))

self.cur.execute("start slave;")

logging.info("启动 slave 成功")

time.sleep(2)

except Exception, e:

logging.error("%s" % (traceback.format\_exc(e)))

return

def check\_replication\_status(self):

self.cur.execute("show slave status;")

\_slave\_status\_list = self.cur.fetchall()

for \_slave\_status in \_slave\_status\_list:

self.slave\_status\_list.append((("Channel\_Name", \_slave\_status["Channel\_Name"]),

("Slave\_IO\_Running", \_slave\_status["Slave\_IO\_Running"]),

("Slave\_SQL\_Running", \_slave\_status["Slave\_SQL\_Running"]),

("Seconds\_Behind\_Master", \_slave\_status["Seconds\_Behind\_Master"]),

("Last\_IO\_Error", \_slave\_status["Last\_IO\_Error"]),

("Last\_SQL\_Error", \_slave\_status["Last\_SQL\_Error"]),)

)

return

def \_get\_result\_channel\_name\_(self, \_restore\_object):

# channel name

\_channel\_name = re.findall('[a-zA-Z]{2,}\d{1,}-?\d\*', os.path.basename(\_restore\_object["file"]))[0].lower()

if \_channel\_name <> "":

# self.channel\_name = \_channel\_name

logging.info("channel name 是 %s" % (\_channel\_name))

else:

logging.error("未能正确获取到channel name : [%s]" % (\_channel\_name))

raise UserWarning("未能正确获取到channel name : [%s]" % (\_channel\_name))

return \_channel\_name

def \_\_enter\_\_(self):

return self

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

return

def working(\_job\_name, \_mysql\_config, \_application\_list, \_configure, \_file\_path, \_file\_size, \_file\_ctime):

try:

def \_\_disk\_stat\_byte\_\_(\_path):

try:

\_vfs = os.statvfs(\_path)

except:

logging.error("检查磁盘%s不存在" % (\_path))

raise SystemExit("检查磁盘%s不存在" % (\_path))

\_available\_byte = \_vfs[statvfs.F\_BAVAIL] \* \_vfs[statvfs.F\_BSIZE]

\_capacity\_byte = \_vfs[statvfs.F\_BLOCKS] \* \_vfs[statvfs.F\_BSIZE]

\_available\_percent = (\_available\_byte \* 1.00) / (\_capacity\_byte \* 1.00) \* 100

return \_available\_percent, \_available\_byte

\_startdate = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())

\_num = 1

while \_num <= \_configure["max\_sleep\_times"]: # 备份文件解压后大小不能超出剩余空间大小

\_available\_percent, \_available\_byte = \_\_disk\_stat\_byte\_\_(\_configure["check\_disk\_path"])

if (\_file\_size \* \_configure["file\_comperss\_percent"]) < \_available\_byte:

break

else:

logging.WARNING(

"备份文件%s[%.2fG]解压后大小[%.2fG]超出剩余空间大小[%.2fG],将等待300秒(%d次)" % (

\_file\_path, \_file\_size / (1024.0 \* 1024 \* 1024),

\_file\_size \* \_configure["file\_comperss\_percent"] / (

1024.0 \* 1024 \* 1024),

\_available\_byte / (1024.0 \* 1024 \* 1024)), \_num)

\_num += 1

time.sleep(300) # 等待5分钟

else:

logging.ERROR("可用磁盘空间检查次数超出最大限制.")

raise SystemExit("可用磁盘空间检查次数超出最大限制.")

with partition\_recover(\_file\_path=\_file\_path, \_mysql\_user=\_configure["restore\_mysql\_user"],

\_mysql\_pass=\_configure["restore\_mysql\_password"],

\_mysql\_socket=\_mysql\_config["socket"], \_host\_name=\_job\_name) as decompass:

decompass.decompress\_backup\_file(\_application\_list["innobackupex"], \_configure["element\_db\_list"],

\_configure["element\_db\_schema"],

\_configure["element\_db\_grants"],

\_configure["restore\_exclude\_db"])

decompass.recover\_table\_schema(\_application\_list["mysql"])

decompass.exec\_partly\_recover(\_mysql\_config["datadir"])

decompass.get\_gtid(configure["xtrabackup\_gtid\_file"])

return (("host", \_job\_name), ("file", \_file\_path), ("startDate", \_startdate),

("finishDate", time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())),

("dbList", str(decompass.restore\_db\_list)),

("fileSize(GB)", "%.3f" % (\_file\_size / (1024.0 \* 1024 \* 1024))),

("GTID", ','.join(decompass.backup\_Gtid)),

("errorMessage", ""),)

except Exception as e:

logging.error("%s" % (traceback.format\_exc(e)))

return (("host", \_job\_name), ("file", \_file\_path), ("startDate", \_startdate),

("finishDate", time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())),

("dbList", str(decompass.restore\_db\_list)),

("fileSize(GB)", "%.3f" % (\_file\_size / (1024.0 \* 1024 \* 1024))),

("GTID", ','.join(decompass.backup\_Gtid)),

("errorMessage", traceback.format\_exc(e)),)

if \_\_name\_\_ == '\_\_main\_\_':

opts = \_\_getopts\_\_(sys.argv[1:])

with preflight() as pre:

pre.check\_applications(configure["check\_application\_list"])

pre.check\_DBinstans(opts["defaults-mysql-file"], opts["mysql-instance"])

if opts["backup\_file\_list"] == []:

pre.check\_backup\_dir(configure["backup\_file\_dir"], configure["backup\_file\_regexp"],

configure["check\_file\_days"],

configure["max\_backup\_file\_num"], configure["exclude\_host\_name"],

configure["threading\_count"])

else:

pre.backup\_file\_list = opts["backup\_file\_list"]

pre.check\_diskfree(configure["check\_disk\_path"], configure["check\_disk\_free\_percent"],

configure["check\_disk\_free\_size"])

#开始还原

arg\_list = []

pool = multiprocessing.Pool(processes=configure["threading\_count"])

result = []

for \_file\_path, \_file\_size, \_file\_ctime in pre.backup\_file\_list:

# for \_file\_path, \_file\_size, \_file\_ctime in pre.backup\_file\_list[:5]:

result.append(pool.apply\_async(working, (

os.path.basename(os.path.dirname(\_file\_path))

, pre.mysql\_config, pre.applications, configure, \_file\_path,

\_file\_size, \_file\_ctime)))

pool.close()

pool.join()

restore\_object\_list = [] # 执行结果

for \_res in result:

restore\_object\_list.append(\_res.get())

logging.debug("备份还原结果集%s" % (restore\_object\_list))

logging.debug("备份还原过期备份结果集%s" % (pre.expired\_file\_list))

try:

with set\_replication(\_mysql\_user=configure["restore\_mysql\_user"],

\_mysql\_pass=configure["restore\_mysql\_password"],

\_mysql\_socket=pre.mysql\_config["socket"]) as replication:

replication.stop\_replication()

replication.get\_change\_master\_sql()

replication.start\_replication(\_restore\_object\_list=restore\_object\_list)

replication.check\_replication\_status()

except Exception, e:

logging.error("%s" % (traceback.format\_exc(e)))

# 生成执行结果报告

with recover\_report() as report:

report.create\_email\_body(restore\_object\_list, replication.slave\_status\_list, replication.finally\_gtid,

replication.finally\_change\_master\_sql)

report.send\_mail(to=configure["emailto"], subjet=configure["emailsubjet"])

logging.info("备份还原报告发送成功")

config.py

#!/usr/bin/python

# encoding:utf-8

# version:0.1

# datetime:2017-02-16 10:33:59

import getopt, sys, time, os, datetime, re

import logging

\_log\_filename = "%s/logging.%s.log" % (os.path.dirname(sys.argv[0]), time.strftime("%Y%m%d%H%M%S", time.localtime()))

logging.basicConfig(level=logging.DEBUG,

format='%(asctime)s-[line:%(lineno)d]-%(levelname)s-%(message)s',

datefmt='%Y-%m-%d %H:%M:%S',

filename=\_log\_filename,

filemode='a')

console = logging.StreamHandler()

console.setLevel(logging.INFO)

formatter = logging.Formatter('%(asctime)s-%(lineno)d-%(levelname)s-%(message)s')

console.setFormatter(formatter)

logging.getLogger('').addHandler(console)

def \_\_usage\_\_():

print(

'''Usage:%s [options]

OPTIONS:

[-h] [--help] 帮助信息

[-c] [--defaults-mysql-file=] mysql实例配置文件

[-i] [--mysql-instance=] 准备执行还原的mysql实例

[-l] [--backup-file-list=] 备份还原列表,逗号分开

''' % (sys.argv[0])).decode('utf-8')

sys.exit()

return

def \_\_getopts\_\_(\_argv):

def \_\_get\_file\_status\_\_(\_file\_list):

\_file\_object\_list = []

\_list = \_file\_list.split(",")

for \_file in \_list:

\_abspath\_file = os.path.abspath(\_file)

if os.path.isfile(\_abspath\_file) and re.findall(r'backup\_[\w-]\*\_\d{8}.tar.gz$', \_abspath\_file):

\_file\_stat = os.stat(\_abspath\_file)

\_file\_object\_list.append((\_abspath\_file, \_file\_stat.st\_size,

\_file\_stat.st\_ctime))

else:

raise UserWarning("%s 不是一个有效文件" % (\_abspath\_file))

return \_file\_object\_list

try:

options, args = getopt.getopt(\_argv, "hc:i:l:",

["help", "defaults-mysql-file=", "mysql-instance=", "backup-file-list=", ])

opts = {

"defaults-mysql-file": configure["default\_mysql\_config"],

"mysql-instance": configure["default\_mysql\_instance"],

"backup\_file\_list": [],

}

for name, value in options:

if name in ("-h", "--help"):

\_\_usage\_\_()

elif name in ("-c", "--defaults-mysql-file"):

opts["defaults-mysql-file"] = value

logging.info("defaults-mysql-file Is %s" % (opts["defaults-mysql-file"]))

elif name in ("-i", "--mysql-instance"):

opts["mysql-instance"] = value

logging.info("mysql-instance Is %s" % (opts["mysql-instance"]))

elif name in ("-l", "--backup\_file\_list"):

opts["backup\_file\_list"] = \_\_get\_file\_status\_\_(value)

logging.info("backup\_file\_list is %s" % (str(opts["backup\_file\_list"])))

return opts

except Exception, ex:

print ex

\_\_usage\_\_()

configure = {

"threading\_count": 4, # 多线程数量

"check\_application\_list": ("mysqldump", "innobackupex", "my\_print\_defaults", "lsof", "mysql"), # 需要检查的应用程序

"file\_comperss\_percent": 8, # 文件压缩比

"max\_sleep\_times": 30, # 最大sleep的次数

"check\_disk\_path": "/data", # 需要检查剩余空间的磁盘路径

"check\_disk\_free\_percent": 20, # 剩余空间阈值(%)

"check\_disk\_free\_size": 200, # 剩余空间大小阈值(G)

"check\_file\_days": 2, # 检查备份文件过期天数>=

"max\_backup\_file\_num": 3, # 每个目录保留的最大备份文件数

"default\_mysql\_config": "/etc/my.cnf", # 默认的mysql配置文件

"default\_mysql\_instance": "mysqld", # 准备还原的实例(如果是多实例 可选择mysqld1|mysqld2)

"backup\_file\_dir": "/data/backup/", # 备份文件存放目录

"backup\_file\_regexp": r"^backup\_[\w-]\*\_\d{8}.tar.gz$", # 备份文件名(正则)

"element\_db\_list": "dbs.list", # 还原数据库dblist文件

"element\_db\_schema": "database\_schema.sql", # 还原数据库schema文件

"element\_db\_grants": "grants.sql", # 还原数据库grant 文件

"iscomplex\_restore\_list\_regulation": True, # 还原列表是否采用复杂规则 True:还原顺序按文件大小交叉排放,并隔天还原。False:按从小到大每天还原

"restore\_exclude\_db": "performance\_schema,test,mysql,sys", # 还原排除数据库

"xtrabackup\_file\_list": ( # Xtrabackup 备份信息文件

"xtrabackup\_binlog\_info", "xtrabackup\_binlog\_pos\_innodb", "xtrabackup\_checkpoints", "xtrabackup\_galera\_info",),

"xtrabackup\_gtid\_file": "xtrabackup\_binlog\_info", # GTID 文件

"restore\_mysql\_user": "root", # 数据库账号

"restore\_mysql\_password": "", # 数据库密码

"exclude\_host\_name": ("DB4-41.yooli-idc.net",), # 排除不恢复的数据库列表

"emailto": ("dbagroup@yooli.com",),

"emailsubjet": "auto replication 报告 %s" % (datetime.date.today().strftime("%Y-%m-%d")),

}

recover\_report.py

#!/usr/bin/python

# encoding:utf-8

# version:0.1

# datetime:2017-02-16 10:33:59

import smtplib

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

import sys, time

import logging, traceback

class recover\_report:

def \_\_init\_\_(self, \_email\_sender="dba@yooli.com", \_smtp\_server="smtp.exmail.qq.com", \_smtp\_user="dba@yooli.com",

\_smtp\_pass="urpass"):

self.email\_sender = \_email\_sender

self.smtp\_server = \_smtp\_server

self.smtp\_user = \_smtp\_user

self.smpt\_pass = \_smtp\_pass

self.email\_body = ""

return

def send\_mail(self, to, subjet='subjet'):

msg\_root = MIMEMultipart('alternative')

msg\_root['Subject'] = subjet

msg\_root["From"] = self.email\_sender

msg\_root["to"] = ','.join(to)

msg\_text = MIMEText(self.email\_body, "html", 'utf-8')

msg\_root.attach(msg\_text)

smtp = smtplib.SMTP()

smtp.connect(self.smtp\_server)

smtp.login(self.smtp\_user, self.smpt\_pass)

smtp.sendmail(msg\_root["From"], to, msg\_root.as\_string())

return

def create\_email\_body(self, \_restore\_result\_list, \_slave\_status\_list, \_finally\_gtid, \_finally\_change\_master\_sql):

def \_create\_table\_body\_(\_inData):

def \_\_html\_install\_\_(\_\_sign, \_\_list):

\_\_outText = ""

if (\_\_list.\_\_len\_\_() == 0):

return

else:

if isinstance(\_\_list, list):

for cell in \_\_list:

\_\_outText += "<" + \_\_sign + ">" + str(cell) + "</" + \_\_sign + ">"

else:

\_\_outText += "<" + \_\_sign + ">" + \_\_list + "</" + \_\_sign + ">"

return \_\_outText

\_table\_body = ""

if \_inData <> []:

for \_row in \_inData:

\_table\_body += \_\_html\_install\_\_("tr", \_\_html\_install\_\_("td", [\_v for \_k, \_v in \_row]))

\_table\_title = \_\_html\_install\_\_("tr", \_\_html\_install\_\_("th", [\_k for \_k, \_v in \_row]))

return \_table\_title + \_table\_body.replace("\n", "<br>")

else:

return ""

with open("%s/auto.replication.template.html" % (sys.path[0]), 'r') as \_html\_template:

\_HTML = \_html\_template.read()

\_html\_key = {"restore\_result\_html": \_create\_table\_body\_(\_restore\_result\_list),

"slave\_status\_list":\_create\_table\_body\_(\_slave\_status\_list),

"finally\_gtid": ",<br>".join(\_finally\_gtid),

"finally\_change\_master\_sql": "<br>".join(\_finally\_change\_master\_sql),

"curentdate": time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())}

for \_key in \_html\_key:

\_HTML = \_HTML.replace("{%%%s%%}" % (\_key), \_html\_key[\_key])

self.email\_body = \_HTML

return

def \_\_enter\_\_(self):

return self

def \_\_exit\_\_(self, exc\_type, exc\_val, exc\_tb):

if exc\_type <> None:

logging.error("exc\_type:%s\nexc\_val:%s\nexc\_tb:%s" % (

exc\_type, exc\_val, traceback.format\_exc(exc\_tb)))

return

auto.replication.template.html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en">

<head>

<meta charset="UTF-8">

<style type="text/css">

\* {

padding: 0;

margin: 0;

}

body {

font-family: verdana;

font-size: 12px;

}

.content {

width: 550px;

margin: 20px auto;

}

.content h1 {

font-family: '' 微软雅黑 '';

font-size: 18px;

padding-bottom: 5px;

}

table {

width: 100%;

}

th, td {

padding: 6px 0;

text-align: center;

}

th {

background-color: #007D28;

color: #ffffff;

}

tr {

background-color: #E8FFE8;

<!-- background-color: expression((this . sectionRowIndex % 2 = = 0) ? "#E1F1F1": "#F0F0F0") -->

}

.odd {

background-color: #FFF3EE;

}

.highlight {

background-color: #E0E0E0;

}

</style>

<Title>Auto replication 报告 {%curentdate%}</Title>

<br>

</head>

<body>

<H1>AUTO REPLICATION REPORT</H1>

<table>

{%restore\_result\_html%}

</table>

<H1>Slave status</H1>

<table>

{%slave\_status\_list%}

</table>

<H1>最终GTID</H1>

<div style="background-color: #B0C4DE">

<H3>

{%finally\_gtid%}

</H3>

</div>

<H1>CHANGE MASTER SQL</H1>

<div style="background-color: #E8FFE8">

<H3>

{%finally\_change\_master\_sql%}

</H3>

</div>

</body>

</html>

1. 附件
   1. [auto.replication.py](/download/attachments/5280585/auto.replication.py?version=2&modificationDate=1494300878000&api=v2)
   2. [auto.replication.template.html](/download/attachments/5280585/auto.replication.template.html?version=1&modificationDate=1490865058000&api=v2)
   3. [config.py](/download/attachments/5280585/config.py?version=1&modificationDate=1490865058000&api=v2)
   4. [recover\_report.py](/download/attachments/5280585/recover_report.py?version=1&modificationDate=1490865058000&api=v2)