import abc

import gzip

import logging

import math

import os.path

import socket

import sys

import urllib2

from contextlib import contextmanager

from datetime import datetime, timedelta

from shutil import copyfile

import paramiko

from etl.bbg\_transport.config import config\_api

from etl.bbg\_transport.dto import RequestItem, RequestOptionItem

from etl.bbg\_transport.parser import BtParser

from etl.bbg\_transport.util import BbgConfig, BtRepoBase

from etl.core.file\_util import file\_stream

from etl.core.util import parse\_args, struct

from etl.enum.cor\_da import BtStatusEnum

USAGE = ['BBG Transport Agent', ['direction', {'help': 'REQUEST or RESPONSE'}]]

class BbgFtp(object):

def \_\_init\_\_(self, config=None):

logging.getLogger(\_\_name\_\_).setLevel(int(config\_api().get('log\_level')))

self.config = None

self.\_ftp\_client = None

self.\_ftp\_transport = None

self.config = config if isinstance(config, BbgConfig) else BbgConfig()

@property

def is\_local\_ftp(self):

return self.config.data[BbgConfig.KEY\_BBG\_FTP\_HOST] == BbgConfig.INTERNAL\_HOST\_FLAG

@property

def ftp\_client(self):

if self.is\_local\_ftp:

return None

if self.\_ftp\_client is None:

self.\_ftp\_connect()

return self.\_ftp\_client

def \_ftp\_connect(self):

host = self.config.data[BbgConfig.KEY\_BBG\_FTP\_HOST]

port = int(self.config.data[BbgConfig.KEY\_BBG\_FTP\_PORT])

uname = self.config.data[BbgConfig.KEY\_BBG\_FTP\_USER]

mkey = paramiko.RSAKey.from\_private\_key\_file(self.config.data[BbgConfig.KEY\_BBG\_FTP\_PASSWORD])

try:

self.\_ftp\_transport = paramiko.Transport(host, port)

self.\_ftp\_transport.set\_keepalive(2)

self.\_ftp\_transport.connect(username=uname, pkey=mkey)

self.\_ftp\_client = paramiko.SFTPClient.from\_transport(self.\_ftp\_transport)

except Exception:

host = self.config.data[BbgConfig.KEY\_BBG\_FTP\_HOST\_2]

self.\_ftp\_transport = paramiko.Transport(host, port)

self.\_ftp\_transport.set\_keepalive(2)

self.\_ftp\_transport.connect(username=uname, pkey=mkey)

self.\_ftp\_client = paramiko.SFTPClient.from\_transport(self.\_ftp\_transport)

def is\_connected(self):

if self.\_ftp\_client is None or self.\_ftp\_transport is None:

return False

return self.\_ftp\_transport.is\_active()

def \_\_del\_\_(self):

self.cleanup()

def cleanup(self):

if self.\_ftp\_client:

try:

self.\_ftp\_client.close()

except Exception:

pass

if self.\_ftp\_transport:

try:

self.\_ftp\_transport.close()

except Exception:

pass

self.\_ftp\_client = None

self.\_ftp\_transport = None

def send\_request\_file(self, source\_path):

file\_name = os.path.basename(source\_path)

target\_path = os.path.join(self.config.data[BbgConfig.KEY\_BBG\_FTP\_OUT\_LOC].strip(), file\_name)

if self.is\_local\_ftp:

copyfile(source\_path, target\_path)

else:

if not self.is\_connected():

self.\_ftp\_connect()

with open(source\_path) as f:

self.ftp\_client.putfo(f, target\_path)

def get\_response\_file(self, source\_path):

file\_name = os.path.basename(source\_path)

target\_path = os.path.join(self.config.data[BbgConfig.KEY\_DATA\_FILE\_IN\_LOC].strip(), file\_name)

if self.is\_local\_ftp is True:

copyfile(source\_path, target\_path)

return os.path.exists(target\_path)

else:

try:

if not self.is\_connected():

self.\_ftp\_connect()

with open(target\_path, 'wb') as f:

self.ftp\_client.getfo(source\_path, f)

return True

except IOError as e:

if e.errno == 2:

try:

if os.path.exists(target\_path):

os.remove(target\_path)

except IOError:

pass

return False

else:

logging.exception("Error encountered: {}".format(e.message))

raise e

class BbgAgent(BtRepoBase):

\_\_metaclass\_\_ = abc.ABCMeta

NO\_ITEMS\_PROCESSED = 100

ITEMS\_PROCESSED = 0

def \_\_init\_\_(self, processing\_status):

super(BbgAgent, self).\_\_init\_\_()

logging.getLogger(\_\_name\_\_).setLevel(int(config\_api().get('log\_level')))

self.\_processing\_status = processing\_status

self.\_run\_date = None

self.\_bbg\_ftp = None

self.\_config = None

self.\_errors = []

self.\_ins\_upd\_by = 'Python ({})'.format(os.path.basename(\_\_file\_\_))

@property

def config(self):

if not self.\_config:

self.\_config = BbgConfig()

return self.\_config

@property

def run\_date(self):

if not self.\_run\_date:

self.\_run\_date = datetime.now() + timedelta(seconds=-3)

return self.\_run\_date

@property

def bbg\_ftp(self):

if self.\_bbg\_ftp is None:

self.\_bbg\_ftp = BbgFtp(self.config)

return self.\_bbg\_ftp

@property

def max\_items(self):

return int(self.config.data[BbgConfig.KEY\_MAX\_REQUEST\_ITEMS])

@property

def bt\_env(self):

return self.config.data[BbgConfig.KEY\_ENVIRONMENT][0:1].upper()

@property

def bbg\_location\_in(self):

return self.config.data[BbgConfig.KEY\_BBG\_FTP\_IN\_LOC].strip()

@property

def bt\_location\_in(self):

return self.config.data[BbgConfig.KEY\_DATA\_FILE\_IN\_LOC].strip()

@property

def bt\_location\_out(self):

return self.config.data[BbgConfig.KEY\_REQ\_FILE\_OUT\_LOC]

@property

def max\_retry(self):

return int(self.config.data[BbgConfig.KEY\_MAX\_RETRY])

@property

def processing\_status(self):

return self.\_processing\_status

@property

def errors(self):

return self.\_errors

@contextmanager

def run(self):

self.reset\_state()

yield self.process\_requests\_in\_status()

self.cleanup()

def reset\_state(self):

self.\_run\_date = None

self.\_errors = []

def cleanup(self):

if self.\_bbg\_ftp:

self.bbg\_ftp.cleanup()

def get\_next\_request(self):

request = self.request\_repo.get\_next\_request\_by\_status\_less\_process\_date(self.processing\_status, self.run\_date)

if request:

request.process\_date = datetime.now()

self.request\_repo.save(request)

return request

def process\_requests\_in\_status(self):

rtn = BbgAgent.NO\_ITEMS\_PROCESSED

request = self.get\_next\_request()

while request:

try:

if self.process\_request(request):

rtn = BbgAgent.ITEMS\_PROCESSED

except Exception as e:

msg = "{} [{}]: {}".format(type(e),

e.\_\_class\_\_.\_\_bases\_\_,

getattr(e, 'message', '#UNK#'))

logging.info("Error while processing RequestId {}: {}".format(request.bt\_request\_id, msg))

self.errors.append("RequestId {}: {}".format(request.bt\_request\_id, msg))

self.update\_request(request=request, status\_code=BtStatusEnum.BTERROR.value)

self.batch\_upsert(request\_id=request.bt\_request\_id, batch\_seq=0, error\_text=msg)

request = self.get\_next\_request()

if len(self.errors) > 0:

raise RuntimeError(self.errors)

return rtn

def update\_request(self, request, status\_code=None, retry\_count=None, data\_file\_path=None):

if not any([status\_code, retry\_count, data\_file\_path]):

return

request.bt\_status\_code = status\_code

if status\_code:

request.status\_date = datetime.now()

request.retry\_count = retry\_count or 0

request.data\_file\_path = data\_file\_path

request.row\_update\_by = self.\_ins\_upd\_by

request.row\_update\_date = datetime.now()

self.request\_repo.save(request)

self.\_sync\_lookback\_requests(request)

def batch\_upsert(self,

request\_id,

batch\_seq,

batch\_id=None,

file\_path=None,

error\_text=None,

bbg\_time\_started=None,

bbg\_time\_finished=None):

if not any([file\_path, error\_text, bbg\_time\_started, bbg\_time\_finished]):

return

if batch\_id:

batch = self.batch\_repo.get\_by\_batch\_id(batch\_id)

else:

batch = self.batch\_repo.get\_by\_bt\_request\_id\_file\_sequence\_no(request\_id,

batch\_seq) or self.batch\_repo.BtRequestBatch()

if not batch.bt\_request\_id:

batch.row\_insert\_by = self.\_ins\_upd\_by

batch.row\_insert\_date = datetime.now()

batch.bt\_request\_id = request\_id

batch.file\_sequence\_no = batch\_seq

batch.request\_file\_path = file\_path or batch.request\_file\_path or '#UND#'

if file\_path:

batch.response\_file\_path = os.path.join(self.bbg\_location\_in,

'{}.txt'.format(os.path.splitext(os.path.basename(file\_path))[0]))

elif batch.request\_file\_path == '#UND#':

batch.response\_file\_path = '#UND#'

elif batch.response\_file\_path:

pass

else:

batch.response\_file\_path = '#UND#'

batch.is\_error\_response = bool(error\_text)

batch.error\_text = error\_text

batch.bbg\_time\_started = bbg\_time\_started

batch.bbg\_time\_finished = bbg\_time\_finished

batch.row\_update\_by = self.\_ins\_upd\_by

batch.row\_update\_date = datetime.now()

self.batch\_repo.save(batch)

@abc.abstractmethod

def process\_request(self, request):

return False

class LoadertAgent(BbgAgent):

def \_\_init\_\_(self):

super(BbgRequestAgent, self).\_\_init\_\_(BtStatusEnum.INITIAL.value)

logging.getLogger(\_\_name\_\_).setLevel(int(config\_api().get('log\_level')))

self.\_request\_template = None

@property

def request\_template(self):

if not self.\_request\_template:

path = os.path.join(os.path.abspath(os.path.dirname(\_\_file\_\_)), 'bt\_request\_template.txt')

with open(path) as f:

self.\_request\_template = f.read()

return self.\_request\_template

def process\_request(self, request):

if not request.request\_object\_data:

raise RuntimeError('request\_object\_data cannot be null (RequestID: {})'.format(request.bt\_request\_id))

request\_data = RequestItem.from\_json(request.request\_object\_data)

request\_data.request\_options = self.process\_request\_options(request\_data.program\_code,

request\_data.request\_options)

template = self.request\_template.replace(BbgConfig.TEMPLATE\_SECTIONS.FIELDS,

'\n'.join(request\_data.request\_fields))

file\_count = int(math.ceil(float(len(request\_data.request\_data\_items)) / self.max\_items))

kwargs\_request = struct(

request=request,

status\_code=BtStatusEnum.PENDING.value

)

for i in range(file\_count):

kwargs\_batch = struct(

request\_id=request.bt\_request\_id,

batch\_seq=i

)

start = i \* self.max\_items

end = start + self.max\_items

file\_name = self.create\_req\_file\_name(request.bt\_request\_id, i)

request\_data.request\_options.append(RequestOptionItem(BbgConfig.OPTION\_KEY\_REPLY\_FILE,

'{}.txt'.format(os.path.splitext(file\_name)[0])))

file\_data = template.replace(BbgConfig.TEMPLATE\_SECTIONS.OPTIONS,

'\n'.join([o.option\_string for o in request\_data.request\_options]))

data\_items = request\_data.request\_data\_items[start:end]

file\_data = file\_data.replace(BbgConfig.TEMPLATE\_SECTIONS.DATA,

'\n'.join([d.data\_item\_string for d in data\_items]))

file\_path = self.create\_req\_file(file\_name, file\_data)

kwargs\_batch['file\_path'] = file\_path

try:

self.bbg\_ftp.send\_request\_file(file\_path)

except (paramiko.SSHException,

paramiko.BadHostKeyException,

paramiko.AuthenticationException,

socket.error,

IOError) as e:

rc = request.retry\_count + 1

kwargs\_request['retry\_count'] = rc

kwargs\_request['status\_code'] = request.bt\_status\_code

msg = "{} [{}] - {}({}): {}".format(type(e),

e.\_\_class\_\_.\_\_bases\_\_,

'Retry Count Exceeded ' if rc >= self.max\_retry else '',

request.retry\_count,

getattr(e, 'message', '#UNK#'))

kwargs\_batch['error\_text'] = msg

if rc >= self.max\_retry:

kwargs\_request['status\_code'] = BtStatusEnum.BTERROR.value

self.batch\_upsert(\*\*kwargs\_batch)

self.update\_request(\*\*kwargs\_request)

return True

def process\_request\_options(self, program\_code, request\_options):

if not isinstance(request\_options, list) or not all(isinstance(i, RequestOptionItem) for i in request\_options):

request\_options = []

for o in self.config.get\_file\_option\_config(program\_code):

if bool(o.is\_default\_only) and not any([True for i in request\_options if i.option\_name == o.option\_name]):

request\_options.append(RequestOptionItem(o.option\_name, o.option\_value))

else:

[request\_options.remove(i) for i in request\_options if i.option\_name == o.option\_name]

if not bool(o.is\_deleted\_option):

request\_options.append(RequestOptionItem(o.option\_name, o.option\_value))

return request\_options

def create\_req\_file(self, file\_name, data):

file\_path = '{}{}'.format(self.bt\_location\_out, file\_name)

with open(file\_path, 'w') as output:

output.write(data)

return file\_path

def create\_req\_file\_name(self, request\_id, seq):

name\_template = BbgConfig.REQUEST\_FILE\_NAME\_TEMPLATE

file\_name = name\_template.replace('##YYYYMMDD##', datetime.today().strftime('%Y%m%d'))

file\_name = file\_name.replace('##REQUEST\_ID##', str(request\_id))

file\_name = file\_name.replace('##SEQ\_NO##', str(seq))

return file\_name.replace('##ENV##', self.bt\_env)

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

agent = LoadertAgent()

with agent.run() as x:

sys.exit(x)