redis\_mdld.py

# 初始化

class Config

def \_\_init\_\_

config\_flist: dict

config\_optlist:dict

config\_slist: list

def init\_flist: # 期货列表

遍历115\_6380:qdb:securityex:derivatives:\*:CODE #> qdb:securityex:derivatives:IC01:CODE

return { "IH01": "IH1908", ...} 共计12个合约

def init\_optlist: # 期权列表

遍历170:OPLST:01\*

取InstrumentCode[7:]建字典 # 510050P1909M03000

return {"1909M03000":("10001746", "10001750"), ...}

def init\_slist: # 现货列表

从redis\_mdld.yaml读入

return ["S510050", "I000001"]

# run(key)

9:30-11:30 13:00-15:00 每隔一秒调用一次

ts = key # 3\*3600 + h\*3600+m\*60+s #type: int 头天21:00:00以来的秒数,

cur\_ts为当前key

rd.set("MDLD:cur\_ts", key)

# MDLD:key共计4\*3600+2个键值,循环写入,ts>cur\_ts则为昨天的数据

遍历flist:

取期货行情, KZ:F%s:LATEST, BP1,SP1 #> IH1908 mget

写入MDLD:ts:F:F%s #> IH1908 d={"BP1":

遍历slist.values():

取现货行情, KZ:S%s:LATEST, BP1, SP1

写入MDLD:ts:S:S510050

取净值, KZ:JZ0000KZE%s:NEW, B1, S1

写入MDLD:ts:JZ:S510050

if 510050:

pe = latest

pe\_510050\_SP1

pe\_510050\_BP1

JZ\_510050\_SP1,

JZ\_510050\_BP1

遍历optlist.values(): {"1909M03000": [10001677, 10001678], ...} .items()

取CP两个合约行情170:MD:01+ InstrumentID, 写入MDLD:ts:OP:10001750

MDLD:ts:OP: C1909M03000 CALL

170:MD:0110001677

MDLD:ts:OP: P1909M03000 PUT

170:MD:0110001678

根据pe计算A5和OP价

MDLD:ts:A5:1909P2950M

MDLD:ts:PO:1909P2950M

根据pe计算A5和OP价

PO = Px + Pc - Pp

A5 = PO – Pe

A5S = Px + PcBP1 - PpSP1 – Pe

A5B = Px + PcSP1 - PpBP1 - Pe

MDLD:ts:A5:1909M02950 ｛ “A5”:5.4 , "A5B", "A5S", “Pe”= ｝

MDLD:ts:PO:1909M02950

根据pe计算A13

MDLD:ts:A13:IH01 { "B" : IH\_BP1 – pe\_510050\_SP1 ,

"S" : IH\_SP1 – pe\_510050\_BP1,

"L" : IH\_LATEST – pe\_510050\_LATEST,

"C" : IH\_BP1 – JZ\_510050\_SP1,

"R" : IH\_SP1 – JZ\_510050\_BP1 }

…IH02/03/04 / IF01/02/03/04 / IC01/02/03/04

波动率

MDLD:ts:V:C1909MV0000 2922

MDLD:ts:V:C1909MVN050 2872

MDLD:ts:V:C1909MV0050 2972

Ceiling / Floor？

均线数据: [LIST]

5s/10s/15s/30s/1m/3m/5m/10m/15m/30m/1h/2h/4h/1d/

......

# 程序代码

|  |
| --- |
| 参见work/src/RealTimeData/newThread.py |
| *class* PeriodThread(threading.Thread):  *def \_\_init\_\_*(self, *firstrange*:list, *secondrange*: list):  threading.Thread.\_\_init\_\_(self)  self.currentDate = str(time.localtime().tm\_year) + "-" + str(time.localtime().tm\_mon) + "-" + str(time.localtime().tm\_mday) #> "2019-07-11"   self.f\_starttime = time.mktime(time.strptime(self.currentDate + " " + *firstrange*[0] , "%Y-%m-%d %H:%M:%S")) # > 1563785375.0012558  self.f\_endtime = time.mktime(time.strptime(self.currentDate + " " + *firstrange*[1] , "%Y-%m-%d %H:%M:%S")) # > 1563785475.0012558  self.s\_starttime = time.mktime(time.strptime(self.currentDate + " " + *secondrange*[0] , "%Y-%m-%d %H:%M:%S")) # > 1563785575.0012558  self.s\_endtime = time.mktime(time.strptime(self.currentDate + " " + *secondrange*[1] , "%Y-%m-%d %H:%M:%S")) # > 1563785675.0012558   self.config = Config("redis\_mdld.yaml")  self.a5s = AverageLine(0)  self.a10s = AverageLine(1)  self.a15s = AverageLine(2)  self.a30s = AverageLine(3)  self.a1m = AverageLine(4)  self.a3m = AverageLine(5)  self.a5m = AverageLine(6)  self.a10m = AverageLine(7)  self.a15m = AverageLine(8)  self.a30m = AverageLine(9)  self.a1h = AverageLine(10)  self.a2h = AverageLine(11)  self.a4h = AverageLine(12)  self.a1d = AverageLine(13)  self.conn\_r = redis.Redis(host="168.36.1.115", port=6379, password="", charset='gb18030', errors='replace',  decode\_responses=*True*)  self.conn\_w = redis.Redis(host="192.168.40.134", port = 6379, password="", charset='gb18030',errors="replace",  decode\_responses=*True*)  *def* timeit(func):  *def* test(*self*):  start = time.time()  func(*self*)  end = time.time()  print("time used: ", end - start)  *return* test   *def* vcompute(self,*xingquan\_list*:list, *time*:str, *p\_run*:list, *cur\_ts*:int, *j*:int, *conn\_w* ):  # print(p\_run[j])  *if p\_run*[*j*] < *xingquan\_list*[0]['XingQuan'] *or p\_run*[*j*] > *xingquan\_list*[len(*xingquan\_list*) - 1]['XingQuan']:  *if j* == 0:  print("PE-500在行权价以外")  *return  if j* == 1:  print("PE在行权价以外")  *return  if j* == 2:  print("PE+500在行权价以外")  *return* cont = ""  *if j* == 1:  cont = "OOOO"  *elif j* == 0:  cont = "N050"  *elif j* == 2:  cont = "0050"  i = 0  *while* i < len(*xingquan\_list*):  *if p\_run*[*j*] < *xingquan\_list*[i]['XingQuan']:  # print(p\_run[j],xingquan\_list[i]['XingQuan'])  pre\_c\_p = 1.0 \* (float(*xingquan\_list*[i - 1]['C\_Latest']) - float(*xingquan\_list*[i]['C\_Latest'])) / (  float(*xingquan\_list*[i]['XingQuan']) - float(*xingquan\_list*[i-1]['XingQuan'])) \* (  float(*xingquan\_list*[i]['XingQuan']) - float(*p\_run*[*j*])) + float(*xingquan\_list*[i][  'C\_Latest'])  pre\_p\_p = 1.0 \* (float(*xingquan\_list*[i]['P\_Latest']) - float(*xingquan\_list*[i-1]['P\_Latest'])) / (  float(*xingquan\_list*[i]['XingQuan']) - float(*xingquan\_list*[i-1]['XingQuan'])) \* (  float(*p\_run*[*j*]) - float(*xingquan\_list*[i-1]['XingQuan'])) + float(*xingquan\_list*[i-1][  'P\_Latest'])  pre\_c\_sp1 = 1.0 \* (float(*xingquan\_list*[i - 1]['C\_SP1']) - float(*xingquan\_list*[i]['C\_SP1'])) / (  float(*xingquan\_list*[i]['XingQuan']) - float(*xingquan\_list*[i - 1]['XingQuan'])) \* (  float(*xingquan\_list*[i]['XingQuan']) - float(*p\_run*[*j*]) ) +\  float(*xingquan\_list*[i]['C\_SP1'])  pre\_p\_sp1 = 1.0 \* (float(*xingquan\_list*[i]['P\_SP1']) - float(*xingquan\_list*[i-1]['P\_SP1'])) / (  float(*xingquan\_list*[i]['XingQuan']) - float(*xingquan\_list*[i-1]['XingQuan'])) \* (  float(*p\_run*[*j*]) - float(*xingquan\_list*[i-1]['XingQuan'])) + float(*xingquan\_list*[i-1][  'P\_SP1'])  pre\_c\_bp1 = 1.0 \* (float(*xingquan\_list*[i-1]['C\_BP1']) - float(*xingquan\_list*[i]['C\_BP1'])) / (  float(*xingquan\_list*[i]['XingQuan']) - float(*xingquan\_list*[i - 1]['XingQuan'])) \* (  float(*xingquan\_list*[i]['XingQuan']) - float(*p\_run*[*j*])) + \  float(*xingquan\_list*[i]['C\_BP1'])  pre\_p\_bp1 = 1.0 \* (float(*xingquan\_list*[i]['P\_BP1']) - float(*xingquan\_list*[i-1]['P\_BP1'])) / (  float(*xingquan\_list*[i]['XingQuan']) - float(*xingquan\_list*[i-1]['XingQuan'])) \* (  float(*p\_run*[*j*]) - float(*xingquan\_list*[i-1]['XingQuan'])) + float(*xingquan\_list*[i-1][  'P\_BP1'])  # print(xingquan\_list[i]['C\_Latest'],xingquan\_list[i]['P\_Latest'])  # print({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1},{'LATEST': pre\_p\_p, 'SP1': pre\_p\_sp1, 'BP1': pre\_p\_bp1})  *conn\_w*.hmset("MDLD:" + str(*cur\_ts*) + ":V:C" + *time* + 'MV'+cont,  {'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  *if* cont == '0000':  self.a5s.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a10s.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a15s.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a30s.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a1m.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a3m.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a5m.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a10m.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a15m.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a30m.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a1h.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a2h.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a4h.vc00queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  *if* cont == '0050':  self.a5s.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a10s.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a15s.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a30s.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a1m.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a3m.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a5m.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a10m.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a15m.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a30m.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a1h.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a2h.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a4h.vc05queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  *if* cont == 'N050':  self.a5s.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a10s.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a15s.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a30s.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a1m.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a3m.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a5m.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a10m.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a15m.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a30m.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a1h.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a2h.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})  self.a4h.vcn5queue[*time*].append({'LATEST': pre\_c\_p, 'SP1': pre\_c\_sp1, 'BP1': pre\_c\_bp1,'pe:':*p\_run*[*j*]})   *conn\_w*.hmset("MDLD:" + str(*cur\_ts*) + ":V:P" + *time* + 'MV'+cont,  {'LATEST': pre\_p\_p, 'SP1': pre\_p\_sp1, 'BP1': pre\_p\_bp1,'pe:':*p\_run*[*j*]})  d = {'LATEST': pre\_p\_p, 'SP1': pre\_p\_sp1, 'BP1': pre\_p\_bp1,'pe:':*p\_run*[*j*]}  *if* cont == '0000':  self.a5s.vp00queue[*time*].append(d)  self.a10s.vp00queue[*time*].append(d)  self.a15s.vp00queue[*time*].append(d)  self.a30s.vp00queue[*time*].append(d)  self.a1m.vp00queue[*time*].append(d)  self.a3m.vp00queue[*time*].append(d)  self.a5m.vp00queue[*time*].append(d)  self.a10m.vp00queue[*time*].append(d)  self.a15m.vp00queue[*time*].append(d)  self.a30m.vp00queue[*time*].append(d)  self.a1h.vp00queue[*time*].append(d)  self.a2h.vp00queue[*time*].append(d)  self.a4h.vp00queue[*time*].append(d)  *if* cont == '0050':  self.a5s.vp05queue[*time*].append(d)  self.a10s.vp05queue[*time*].append(d)  self.a15s.vp05queue[*time*].append(d)  self.a30s.vp05queue[*time*].append(d)  self.a1m.vp05queue[*time*].append(d)  self.a3m.vp05queue[*time*].append(d)  self.a5m.vp05queue[*time*].append(d)  self.a10m.vp05queue[*time*].append(d)  self.a15m.vp05queue[*time*].append(d)  self.a30m.vp05queue[*time*].append(d)  self.a1h.vp05queue[*time*].append(d)  self.a2h.vp05queue[*time*].append(d)  self.a4h.vp05queue[*time*].append(d)  *if* cont == 'N050':  self.a5s.vpn5queue[*time*].append(d)  self.a10s.vpn5queue[*time*].append(d)  self.a15s.vpn5queue[*time*].append(d)  self.a30s.vpn5queue[*time*].append(d)  self.a1m.vpn5queue[*time*].append(d)  self.a3m.vpn5queue[*time*].append(d)  self.a5m.vpn5queue[*time*].append(d)  self.a10m.vpn5queue[*time*].append(d)  self.a15m.vpn5queue[*time*].append(d)  self.a30m.vpn5queue[*time*].append(d)  self.a1h.vpn5queue[*time*].append(d)  self.a2h.vpn5queue[*time*].append(d)  self.a4h.vpn5queue[*time*].append(d)  *break* i = i + 1   @timeit  *def* run(self):  key = int(time.time())-int(time.mktime(time.strptime(self.currentDate + " " + "00:00:00" , "%Y-%m-%d %H:%M:%S"))) + 3600 # > 1563785675  print("当前时间：" + str(key),time.localtime())  conn\_r = self.conn\_r  conn\_w = self.conn\_w   cur\_ts = key  conn\_w.set("MDLD:cur\_ts",cur\_ts)  pre = 'KZ:F'  f\_d\_list = dict()  # print("期货列表")  *for* key *in* self.config.config\_flist.values():  # print(pre+key+":LATEST",pre+key+":BP1",pre+key+":SP1")  latest, bp1, sp1 = conn\_r.mget([pre+key+":LATEST",pre+key+":BP1",pre+key+":SP1"])  d = dict()  d['LATEST'] = latest  d['BP1'] = bp1  d['SP1'] = sp1  # print(d)  self.a5s.fqueue[key].append(d)  self.a10s.fqueue[key].append(d)  self.a15s.fqueue[key].append(d)  self.a30s.fqueue[key].append(d)  self.a1m.fqueue[key].append(d)  self.a3m.fqueue[key].append(d)  self.a5m.fqueue[key].append(d)  self.a10m.fqueue[key].append(d)  self.a15m.fqueue[key].append(d)  self.a30m.fqueue[key].append(d)  self.a1h.fqueue[key].append(d)  self.a2h.fqueue[key].append(d)  self.a4h.fqueue[key].append(d)  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":F:F" + key, d)  pre = 'KZ:'  pre1 = 'KZ:JZ0000KZE'  # print("现货列表")  pe = 0  pe\_510050\_SP1 = 0  pe\_510050\_BP1 = 0  jz\_510050\_SP1 = 0  jz\_510050\_BP1 =0  *for* key *in* self.config.config\_slist:  latest, bp1, sp1 = conn\_r.mget([pre + key + ":LATEST", pre+key + ":BP1", pre + key + ":SP1"])  d = dict()  d['LATEST'] = latest  d['BP1'] = bp1  d['SP1'] = sp1  # print(d)  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":S:" + key,d)  new, b1, s1 = conn\_r.mget([pre1 + key + ":NEW", pre1 + key + ":BP1", pre1 + key + ":SP1"])  *if* new == *None*:  new = 0  *if* b1 == *None*:  b1 = 0  *if* s1 == *None*:  s1 = 0  d = dict()  d['LATEST'] = new  d['BP1'] = b1  d['SP1'] = s1  self.a5s.squeue[key].append(d)  self.a10s.squeue[key].append(d)  self.a15s.squeue[key].append(d)  self.a30s.squeue[key].append(d)  self.a1m.squeue[key].append(d)  self.a3m.squeue[key].append(d)  self.a5m.squeue[key].append(d)  self.a10m.squeue[key].append(d)  self.a15m.squeue[key].append(d)  self.a30m.squeue[key].append(d)  self.a1h.squeue[key].append(d)  self.a2h.squeue[key].append(d)  self.a4h.squeue[key].append(d)  # print(d)  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":JZ:" + key,d)  *if* key == "S510050":  pe = int(conn\_w.hget("MDLD:"+str(cur\_ts) + ":S:" + key,"LATEST"))  pe\_510050\_SP1 = int(conn\_w.hget("MDLD:"+str(cur\_ts) + ":S:" + key, "SP1"))  pe\_510050\_BP1 = int(conn\_w.hget("MDLD:" + str(cur\_ts) + ":S:" + key, "BP1"))  jz\_510050\_SP1 = int(conn\_w.hget("MDLD:"+ str(cur\_ts) + ":JZ:" + key, "SP1"))  jz\_510050\_BP1 = int(conn\_w.hget("MDLD:" + str(cur\_ts) + ":JZ:" + key, "BP1"))  conn\_r = redis.Redis(host="168.36.1.170", port=6379, password="", charset='gb18030', errors='replace',  decode\_responses=*True*)  # print("期权列表")  op\_c\_p\_price = list()  date\_list = set()  *for* pxname, (icode\_c, icode\_p)*in* self.config.config\_optlist.items():  #> '1909M02750': ['10001709', '10001710']  # print(item[0][:4])  date\_list.add(pxname[:4]) #> “1909”  tem1 = dict()  tem1['Latest'], tem1['SP1'], tem1['BP1'] = conn\_r.hmget("MD:01" + icode\_c,'LATEST','SP1','BP1') #> "MD:0110001709"  tem2 = dict()  tem2['Latest'], tem2['SP1'], tem2['BP1'] = conn\_r.hmget("MD:01" + icode\_p,'LATEST','SP1','BP1')  # print(tem1, tem2)  *if* tem1['Latest'] == *None or* tem1['BP1'] == *None or* tem1['SP1'] == *None or* tem2['Latest'] == *None or* tem2['SP1'] == *None or* tem2['BP1'] == *None*:  *continue* # print(tem1)  price\_dict= dict()  price\_dict['Name'] = pxname  price\_dict['XingQuan'] = int(pxname[-5:]) \* 10 #> "02750"  price\_dict['C\_Latest'] = tem1['Latest']  price\_dict['C\_BP1'] = tem1['BP1']  price\_dict['C\_SP1'] = tem1['SP1']  price\_dict['P\_Latest'] = tem2['Latest']  price\_dict['P\_BP1'] = tem2['BP1']  price\_dict['P\_SP1'] = tem2['SP1']  op\_c\_p\_price.append(price\_dict)  # print(tem1)  # print(tem2)  *if* tem1['Latest'] != *None and* tem1['BP1'] != *None and* tem1['SP1'] != *None*:  td = {"LATEST": tem1['Latest'], 'BP1': tem1['BP1'], 'SP1': tem1['SP1']}  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":OP:" + "C" + pxname, td)   self.a5s.opcqueue[pxname].append(td)  self.a10s.opcqueue[pxname].append(td)  self.a15s.opcqueue[pxname].append(td)  self.a30s.opcqueue[pxname].append(td)  self.a1m.opcqueue[pxname].append(td)  self.a3m.opcqueue[pxname].append(td)  self.a5m.opcqueue[pxname].append(td)  self.a10m.opcqueue[pxname].append(td)  self.a15m.opcqueue[pxname].append(td)  self.a30m.opcqueue[pxname].append(td)  self.a1h.opcqueue[pxname].append(td)  self.a2h.opcqueue[pxname].append(td)  self.a4h.opcqueue[pxname].append(td)  *if* tem2['Latest'] != *None and* tem2['SP1'] != *None and* tem2['BP1'] != *None*:  td = {"LATEST":tem2['Latest'],'BP1':tem2['BP1'],'SP1':tem2['SP1']}  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":OP:" + "P" + pxname, td)  self.a5s.oppqueue[pxname].append(td)  self.a10s.oppqueue[pxname].append(td)  self.a15s.oppqueue[pxname].append(td)  self.a30s.oppqueue[pxname].append(td)  self.a1m.oppqueue[pxname].append(td)  self.a3m.oppqueue[pxname].append(td)  self.a5m.oppqueue[pxname].append(td)  self.a10m.oppqueue[pxname].append(td)  self.a15m.oppqueue[pxname].append(td)  self.a30m.oppqueue[pxname].append(td)  self.a1h.oppqueue[pxname].append(td)  self.a2h.oppqueue[pxname].append(td)  self.a4h.oppqueue[pxname].append(td)  px = int(pxname[-5:])\*10  pc = int(tem1['Latest'])  pp = int(tem2['Latest'])  po= px + pc - pp  a5 = po - pe  pcbp1 = int(tem1['BP1'])  ppsp1 = int(tem2['SP1'])  pcsp1 = int(tem1['SP1'])  ppbp1 = int(tem2['BP1'])  a5s = px + pcbp1 - ppsp1 - pe  a5b = px + pcsp1 - ppbp1 - pe  d = dict()  d['A5'] = a5  d['A5B'] = a5b  d['A5S'] = a5s  d['Pe'] = pe  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":A5:" + pxname,d)  conn\_w.set("MDLD:" + str(cur\_ts) + ":PO:" + pxname,po)  *for* key *in* self.config.config\_flist.keys():  d = dict()  d["B"] = int(conn\_w.hget("MDLD:" + str(cur\_ts) + ":F:F" + self.config.config\_flist[key], "BP1")) - pe\_510050\_SP1  d["S"] = int(conn\_w.hget("MDLD:" + str(cur\_ts) + ":F:F" + self.config.config\_flist[key], "SP1")) - pe\_510050\_BP1  d["L"] = int(conn\_w.hget("MDLD:" + str(cur\_ts) + ":F:F" + self.config.config\_flist[key], "LATEST")) - pe  d["C"] = int(conn\_w.hget("MDLD:" + str(cur\_ts) + ":F:F" + self.config.config\_flist[key], "BP1")) - jz\_510050\_SP1  d["R"] = int(conn\_w.hget("MDLD:" + str(cur\_ts) + ":F:F" + self.config.config\_flist[key], "SP1")) - jz\_510050\_BP1  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":A13:" + key, d)   # for i in range(len(op\_c\_p\_price)-1):  # t = 0  # while t < len(op\_c\_p\_price) - 1:  # if op\_c\_p\_price[t]['XingQuan'] > op\_c\_p\_price[t+1]['XingQuan']:  # temp\_d = op\_c\_p\_price[t]  # op\_c\_p\_price[t] = op\_c\_p\_price[t+1]  # op\_c\_p\_price[t+1] = temp\_d  # t = t+1  # print(op\_c\_p\_price)  op\_c\_p\_price = sorted(op\_c\_p\_price,key = *lambda x*: x['XingQuan'])  # print(op\_c\_p\_price)  date\_list = list(date\_list)  date\_list.sort() #> ['1908', '1909', '1912', '2003']  dangyue\_c\_p\_price = list()  xiayue\_c\_p\_price = list()  xiaji\_c\_p\_price = list()  geji\_c\_p\_price = list()  *for* i *in* range(len(op\_c\_p\_price)):  *if* op\_c\_p\_price[i]['Name'].startswith(date\_list[0]):  dangyue\_c\_p\_price.append(op\_c\_p\_price[i])  *elif* op\_c\_p\_price[i]['Name'].startswith(date\_list[1]):  xiayue\_c\_p\_price.append(op\_c\_p\_price[i])  *elif* op\_c\_p\_price[i]['Name'].startswith(date\_list[2]):  xiaji\_c\_p\_price.append(op\_c\_p\_price[i])  *elif* op\_c\_p\_price[i]['Name'].startswith(date\_list[3]):  geji\_c\_p\_price.append(op\_c\_p\_price[i])  # print(dangyue\_c\_p\_price)  # print(xiayue\_c\_p\_price)  # print(xiaji\_c\_p\_price)  # print(geji\_c\_p\_price)  pe\_p = pe + 500  pe\_n = pe - 500  p\_run = [pe\_n,pe,pe\_p]  *for* i *in* range(len(p\_run)):  *if* len(dangyue\_c\_p\_price) != 0 :  self.vcompute(dangyue\_c\_p\_price, dangyue\_c\_p\_price[0]['Name'][:4],p\_run,cur\_ts,i,conn\_w)  *if* len(xiayue\_c\_p\_price) != 0:  self.vcompute(xiayue\_c\_p\_price, xiayue\_c\_p\_price[0]['Name'][:4],p\_run,cur\_ts,i,conn\_w)  *if* len(xiaji\_c\_p\_price) != 0:  self.vcompute(xiaji\_c\_p\_price,xiaji\_c\_p\_price[0]['Name'][:4],p\_run,cur\_ts,i,conn\_w)  *if* len(geji\_c\_p\_price) != 0:  self.vcompute(geji\_c\_p\_price,geji\_c\_p\_price[0]['Name'][:4],p\_run,cur\_ts,i,conn\_w)  self.a5s.run(cur\_ts,conn\_w)  self.a10s.run(cur\_ts,conn\_w)  self.a15s.run(cur\_ts,conn\_w)  self.a30s.run(cur\_ts,conn\_w)  self.a1m.run(cur\_ts,conn\_w)  self.a3m.run(cur\_ts,conn\_w)  self.a5m.run(cur\_ts,conn\_w)  self.a10m.run(cur\_ts,conn\_w)  self.a15m.run(cur\_ts,conn\_w)  self.a30m.run(cur\_ts,conn\_w)  self.a1h.run(cur\_ts,conn\_w)  self.a2h.run(cur\_ts,conn\_w)  self.a4h.run(cur\_ts,conn\_w)  self.a1d.run(cur\_ts,conn\_w)   *def* start(self):  print("上班啦")  print("上午：")  self.operate([self.f\_starttime,self.f\_endtime])  print("下午：")  self.operate([self.s\_starttime,self.s\_endtime])  print("下班啦")   *def* operate(self, *interval*:list):   start\_time = *interval*[0]  end\_time = *interval*[1]  *if* time.time() > end\_time:  *return* ctime = time.time()  *if* ctime < start\_time:  print("等待开盘")  interval = int(start\_time) - time.time()  time.sleep(*interval*)  self.run()   ctime = time.time()  *if* ctime >= start\_time *and* ctime <= end\_time:  time.sleep(int(time.time())+1 - time.time())  *while* time.time() < end\_time:  self.run()  time1 = time.time()  time.sleep(int(time.time())+1 - time.time())  *if* int(time.time()) - int(time1) != 1:  print("miss one")  self.run() |

|  |
| --- |
| 参见work/src/RealTimeData/AverageLine.py |
| *class* AverageLine:  *def \_\_init\_\_*(self,*interval*:int):  self.interval = *interval* self.averageNames = ['A5s','A10s','A15s','A30s','A1m','A3m','A5m','A10m','A15m','A30m','A1h','A2h','A4h','A1d']  # 1d 和 4h 是相同的  self.interval\_list = [5,10,15,30,60,180,300,600,900,1800,3600,7200,14400,14400]  self.config = Config('redis\_mdld.yaml')  self.fqueue = {} # type: dict[str, deque]  self.squeue = {}  self.opcqueue = {}  self.oppqueue = {}  self.vc00queue = {}  self.vc05queue = {}  self.vcn5queue = {}  self.vp00queue = {}  self.vp05queue = {}  self.vpn5queue = {}   *for* key *in* self.config.config\_flist.values():  self.fqueue[key] = deque(maxlen=self.interval\_list[self.interval])  *for* key *in* self.config.config\_slist:  self.squeue[key] = deque(maxlen=self.interval\_list[self.interval])  *for* key *in* self.config.config\_optlist.items():  self.opcqueue[key[0]] = deque(maxlen=self.interval\_list[self.interval])  *for* key *in* self.config.config\_optlist.items():  self.oppqueue[key[0]] = deque(maxlen=self.interval\_list[self.interval])  *for* key *in* self.config.config\_optlist.keys():  self.vc00queue[key[:4]] = deque(maxlen=self.interval\_list[self.interval])  self.vp00queue[key[:4]] = deque(maxlen=self.interval\_list[self.interval])  self.vcn5queue[key[:4]] = deque(maxlen=self.interval\_list[self.interval])  self.vpn5queue[key[:4]] = deque(maxlen=self.interval\_list[self.interval])  self.vc05queue[key[:4]] = deque(maxlen=self.interval\_list[self.interval])  self.vp05queue[key[:4]] = deque(maxlen=self.interval\_list[self.interval])  self.directCompute = *False   def* timeit(func):  *def* test(*self*,*num\_list*,*interval*):  start = time.time()  func(*self*,*num\_list*,*interval*)  end = time.time()  print("compute average time used: ", end - start)  *return* test   *def* averageCompute(self,*num\_list*:list, *interval*: int):  res = {self.averageNames[self.interval]+'LATEST':*None*, self.averageNames[self.interval]+'BP1':*None*, self.averageNames[self.interval]+'SP1':*None*}  latest = 0  bp1 = 0  sp1 = 0   *for* d *in num\_list*:  # print(d)  latest = latest + float(d['LATEST'])  bp1 = bp1 + float(d['BP1'])  sp1 = sp1 + float(d['SP1'])  latest = latest / self.interval\_list[*interval*]  bp1 = bp1 / self.interval\_list[*interval*]  sp1 = sp1 / self.interval\_list[*interval*]  res[self.averageNames[self.interval]+'LATEST'] = latest  res[self.averageNames[self.interval]+'SP1'] = sp1  res[self.averageNames[self.interval]+'BP1'] = bp1  *return* res   *def* appendFQueue(self,*cur\_ts*:int, *conn\_w*, *keyword*:str, *keyname*:str):  prefix = 'MDLD:'  # print(conn\_w.hmget(prefix + str(cur\_ts) + keyword, 'LATEST','SP1','BP1'))  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST','SP1','BP1')  print(c\_d['LATEST'])  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.fqueue[*keyname*].append(c\_d)  *def* appendSQueue(self,*cur\_ts*:int, *conn\_w*, *keyword*:str, *keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST','SP1','BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.squeue[*keyname*].append(c\_d)  *def* appendOPcQueue(self,*cur\_ts*:int,*conn\_w*,*keyword*:str, *keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST','SP1','BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.opcqueue[*keyname*].append(c\_d)   *def* appendOPpQueue(self,*cur\_ts*:int, *conn\_w*, *keyword*:str, *keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST', 'SP1', 'BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.oppqueue[*keyname*].append(c\_d)   *def* appendVQueueC00(self,*cur\_ts*:int, *conn\_w*, *keyword*:str, *keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST', 'SP1', 'BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.vc00queue[*keyname*].append(c\_d)   *def* appendVQueueC05(self,*cur\_ts*:int, *conn\_w*, *keyword*:str, *keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST', 'SP1', 'BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.vc05queue[*keyname*].append(c\_d)   *def* appendVQueueCN5(self,*cur\_ts*:int, *conn\_w*, *keyword*:str,*keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST', 'SP1', 'BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.vcn5queue[*keyname*].append(c\_d)   *def* appendVQueueP00(self,*cur\_ts*:int, *conn\_w*, *keyword*:str, *keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST', 'SP1', 'BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.vp00queue[*keyname*].append(c\_d)   *def* appendVQueueP05(self,*cur\_ts*:int, *conn\_w*, *keyword*:str, *keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST', 'SP1', 'BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.vp05queue[*keyname*].append(c\_d)   *def* appendVQueuePN5(self,*cur\_ts*:int, *conn\_w*, *keyword*:str, *keyname*:str):  prefix = 'MDLD:'  c\_d = {}  c\_d['LATEST'], c\_d['SP1'], c\_d['BP1'] = *conn\_w*.hmget(prefix + str(*cur\_ts*) + *keyword*, 'LATEST', 'SP1', 'BP1')  *if* c\_d['LATEST'] == *None or* c\_d['SP1'] == *None or* c\_d['BP1'] == *None*:  *return* self.vpn5queue[*keyname*].append(c\_d)   *def* fillQueue(self,cur\_ts,*keyword*:str,interval:int, *keyname*:str):  # print(keyword,keyword[-6:])  *if keyword*.startswith(":F:F"):  *return* self.fqueue[*keyname*]  *elif keyword*.startswith(':JZ:'):  *return* self.squeue[*keyname*]  *elif keyword*.startswith(':OP:C'):  *return* self.opcqueue[*keyname*]  *elif keyword*.startswith(':OP:P'):  *return* self.oppqueue[*keyname*]  *elif keyword*.startswith(':V:C'):  *if keyword*[-6:] == 'MV0000':  # print("yes")  *return* self.vc00queue[*keyname*]  *elif keyword*[-6:] == 'MVN050':  *return* self.vcn5queue[*keyname*]  *elif keyword*[-6:] == 'MV0050':  *return* self.vc05queue[*keyname*]  *elif keyword*.startswith(':V:P'):  *if keyword*[-6:] == 'MV0000':  *return* self.vp00queue[*keyname*]  *elif keyword*[-6:] == 'MVN050':  *return* self.vpn5queue[*keyname*]  *elif keyword*[-6:] == 'MV0050':  *return* self.vp05queue[*keyname*]   *def* setDeque(self,*queue*:deque):  self.queue = *queue   def* operate(self, *cur\_ts*:int, *keyword*:str, *interval*:int, *conn\_w*, *keyname*:str):  prefix = "MDLD:"  l = self.fillQueue(*cur\_ts*, *keyword*, self.interval, *keyname*)  # print(prefix+str(cur\_ts-1)+keyword,self.averageNames[self.interval] + 'LATEST',prefix+ str(cur\_ts-self.interval\_list[self.interval])+keyword)  *if not* (str(self.interval\_list[self.interval])+'LATEST' *in* l[len(l) - 1].keys()) *or not* (str(self.interval\_list[self.interval])+'BP1' *in* l[len(l) - 1].keys()) *or not* (str(self.interval\_list[self.interval])+'SP1' *in* l[len(l) - 1].keys()) :  # print("AerageLine", prefix+str(cur\_ts-1)+keyword, conn\_w.hmget(prefix+str(cur\_ts-1)+keyword, self.averageNames[self.interval] + 'LATEST'), prefix+ str(cur\_ts-self.interval\_list[self.interval])+keyword, conn\_w.hmget(prefix+ str(cur\_ts-self.interval\_list[self.interval])+keyword, 'LATEST'))  d = self.averageCompute(l,*interval*)  *conn\_w*.hmset(prefix + str(*cur\_ts*) + *keyword*, d)  *else*:  # print("compute")  last\_average\_latest = l[len(l) - 2][self.interval\_list[self.interval]+'LATEST']  last\_average\_bp1 = l[len(l) - 2][self.interval\_list[self.interval] + 'BP1']  last\_average\_sp1 = l[len(l) - 2][self.interval\_list[self.interval] + 'SP1']  # last\_average\_sp1 = conn\_w.hget(prefix + str(cur\_ts-1) + keyword, self.averageNames[self.interval] + 'SP1')  # last\_average\_bp1 = conn\_w.hget(prefix + str(cur\_ts-1) + keyword, self.averageNames[self.interval] + 'BP1')  last\_first\_latest = l[0]['LATEST']  last\_first\_bp1 = l[0]['BP1']  last\_first\_sp1 = l[0]['SP1']  # last\_first\_latest = conn\_w.hget("MDLD:" + str(cur\_ts - self.interval\_list[self.interval]) + keyword,  # 'LATEST')  # last\_first\_sp1 = conn\_w.hget("MDLD:" + str(cur\_ts - self.interval\_list[self.interval]) + keyword,  # 'SP1')  # last\_first\_bp1 = conn\_w.hget("MDLD:" + str(cur\_ts - self.interval\_list[self.interval]) + keyword,  # 'BP1')  curr\_latest = l[len(l) - 1]['LATEST']  curr\_bp1 = l[len(l) - 1]['BP1']  curr\_sp1 = l[len(l) - 1]['SP1']  # curr\_latest = conn\_w.hget("MDLD:" + str(cur\_ts) + keyword, 'BP1')  # curr\_sp1 = conn\_w.hget("MDLD:" + str(cur\_ts) + keyword, 'SP1')  # curr\_bp1 = conn\_w.hget("MDLD:" + str(cur\_ts) + keyword, 'BP1')  new\_average\_latest = (float(last\_average\_latest) \* self.interval\_list[self.interval] - float(last\_first\_latest) + float(curr\_latest)) / self.interval\_list[self.interval]  new\_average\_sp1 = (float(last\_average\_sp1) \* self.interval\_list[self.interval] - float(last\_first\_sp1) + float(curr\_sp1)) / \  self.interval\_list[self.interval]  new\_average\_bp1 = (float(last\_average\_bp1) \* self.interval\_list[self.interval] - float(last\_first\_bp1) + float(curr\_bp1)) / \  self.interval\_list[self.interval]  d = dict()  d[self.averageNames[self.interval] + "LATEST"] = new\_average\_latest  d[self.averageNames[self.interval] + "SP1"] = new\_average\_sp1  d[self.averageNames[self.interval] + "BP1"] = new\_average\_bp1  # c\_d = conn\_w.hgetall(prefix + str(cur\_ts) + keyword)  *conn\_w*.hmset(prefix + str(*cur\_ts*) + *keyword*, d)  l[len(l) - 1][self.averageNames[self.interval] + 'LATEST'] = new\_average\_latest  l[len(l) - 1][self.averageNames[self.interval] + 'SP1'] = new\_average\_sp1  l[len(1) - 1][self.averageNames[self.interval] + 'BP1'] = new\_average\_bp1   *def* run(self,*key*:int,*conn\_w*):  cur\_ts = *key* # 秒编号  prefix = "MDLD:" + str(cur\_ts)  *for* key *in* self.config.config\_flist.values():  # print(len(self.fqueue[key]))  *if* len(self.fqueue[*key*]) < self.interval\_list[self.interval]:  # print(len(self.fqueue[key]), self.interval\_list[self.interval],  # (len(self.fqueue[key]) < self.interval\_list[self.interval]))  # self.appendFQueue(cur\_ts,conn\_w,':F:F'+key,key)  *continue* # print('hhhh')  # self.appendFQueue(cur\_ts,conn\_w,":F:F"+key,key)  self.operate(cur\_ts,":F:F"+*key*,self.interval, *conn\_w*, keyname=*key*)  *for* key *in* self.config.config\_slist:  # print(len(self.squeue[key]))  *if* len(self.squeue[*key*]) < self.interval\_list[self.interval]:  # self.appendSQueue(cur\_ts,conn\_w,':JZ:'+key, key)  *continue* # self.appendSQueue(cur\_ts,conn\_w,':JZ:'+key,key)  self.operate(cur\_ts,':JZ:'+*key*,self.interval, *conn\_w*, keyname=*key*)  *for* key *in* self.config.config\_optlist.items():  # print(len(self.opcqueue[key[0]]))  *if* len(self.opcqueue[*key*[0]]) < self.interval\_list[self.interval]:  # self.appendOPcQueue(cur\_ts, conn\_w, ':OP:C' + key[0], key[0])  *continue* # self.appendOPcQueue(cur\_ts,conn\_w,':OP:C'+key[0],key[0])  self.operate(cur\_ts,':OP:C'+*key*[0],self.interval,*conn\_w*, keyname=*key*[0])  *for* key *in* self.config.config\_optlist.items():  # print(len(self.oppqueue[key[0]]))  *if* len(self.opcqueue[*key*[0]]) < self.interval\_list[self.interval]:  # self.appendOPpQueue(cur\_ts,conn\_w,':OP:P'+ key[0],key[0])  *continue* # self.appendOPpQueue(cur\_ts, conn\_w, ':OP:P' + key[0], key[0])  self.operate(cur\_ts,':OP:P'+*key*[0],self.interval,*conn\_w*,keyname=*key*[0])  *for* key *in* self.config.config\_optlist.keys():  # print(len(self.vc00queue[key[:4]]))  *if* len(self.vc00queue[*key*[:4]]) < self.interval\_list[self.interval]:  # self.appendVQueueC00(cur\_ts,conn\_w,':V:C'+key[:4]+'MV0000', key[:4])  *continue* # self.appendVQueueC00(cur\_ts,conn\_w,':V:C'+key[:4]+'MV0000',key[:4])  self.operate(cur\_ts,':V:C'+*key*[:4]+'MV0000',self.interval, *conn\_w*, keyname=*key*[:4])  *for* key *in* self.config.config\_optlist.keys():  # print(len(self.vp00queue[key[:4]]))  *if* len(self.vp00queue[*key*[:4]]) < self.interval\_list[self.interval]:  # self.appendVQueueP00(cur\_ts, conn\_w, ':V:P' + key[:4] + 'MV0000', key[:4])  *continue* # self.appendVQueueP00(cur\_ts,conn\_w,':V:P'+key[:4]+'MV0000',key[:4])  self.operate(cur\_ts,':V:P'+*key*[:4]+'MV0000',self.interval, *conn\_w*, keyname=*key*[:4])  *for* key *in* self.config.config\_optlist.keys():  # print(len(self.vcn5queue[key[:4]]))  *if* len(self.vcn5queue[*key*[:4]]) < self.interval\_list[self.interval]:  # self.appendVQueueCN5(cur\_ts, conn\_w, ':V:C' + key[:4] + 'MVN050', key[:4])  *continue* # self.appendVQueueCN5(cur\_ts,conn\_w,':V:C'+key[:4]+'MVN050',key[:4])  self.operate(cur\_ts,':V:C'+*key*[:4]+'MVN050',self.interval, *conn\_w*, keyname=*key*[:4])   *for* key *in* self.config.config\_optlist.keys():  # print(len(self.vpn5queue[key[:4]]))  *if* len(self.vpn5queue[*key*[:4]]) < self.interval\_list[self.interval]:  # self.appendVQueuePN5(cur\_ts, conn\_w, ':V:P' + key[:4] + 'MVN050', key[:4])  *continue* # self.appendVQueuePN5(cur\_ts,conn\_w,':V:P'+key[:4]+'MVN050',key[:4])  self.operate(cur\_ts,':V:P'+*key*[:4]+'MVN050',self.interval, *conn\_w*, keyname=*key*[:4])   *for* key *in* self.config.config\_optlist.keys():  # print(len(self.vc05queue[key[:4]]))  *if* len(self.vc05queue[*key*[:4]]) < self.interval\_list[self.interval]:  # self.appendVQueueC05(cur\_ts, conn\_w, ':V:C' + key[:4] + 'MV0050', key[:4])  *continue* # self.appendVQueueC05(cur\_ts,conn\_w,':V:C'+key[:4]+'MV0050',key[:4])  self.operate(cur\_ts,':V:C'+*key*[:4]+'MV0050',self.interval, *conn\_w*, keyname=*key*[:4])   *for* key *in* self.config.config\_optlist.keys():  *if* len(self.vp05queue[*key*[:4]]) < self.interval\_list[self.interval]:  # self.appendVQueueP05(cur\_ts, conn\_w, ':V:P' + key[:4] + 'MV0050', key[:4])  *continue* # self.appendVQueueP05(cur\_ts,conn\_w,':V:P'+key[:4]+'MV0050',key[:4])  self.operate(cur\_ts,':V:P'+*key*[:4]+'MV0050',self.interval, *conn\_w*, keyname=*key*[:4]) |