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/intodb\_pkl\_20190719/readdata.py |
| #-\*- coding: UTF-8 -\*- *import* redis *import* time *from* collections *import* deque *import* yaml  *class* Config:  *def \_\_init\_\_*(self, *fn\_config*:str):  self.config\_flist = self.init\_flist()  self.config\_optlist = self.init\_optlist()  self.config\_slist = self.init\_slist(*fn\_config*)  print(self.config\_flist)  print(self.config\_optlist)  print(self.config\_slist)   *def* init\_flist(self):  conn = redis.Redis(host="168.36.1.115", port=6380, password="", charset='gb18030', errors='replace',  decode\_responses=*True*)  pre = "qdb:securityex:derivatives:"  code = "CODE"  t = ['IC', 'IF', 'IH']  d = dict()  *for* i *in* t:  *for* j *in* range(1, 5):  d[i + "0" + str(j)] = conn.get(pre + i + "0" + str(j) + ":CODE")  *return* d   *def* init\_optlist(self):  conn = redis.Redis(host="168.36.1.170", port=6379, password="", charset='gb18030', errors='replace',  decode\_responses=*True*)  keys = conn.keys()   re = dict()  num = 0  *for* key *in* keys:  *if* key.startswith("OPLST:01"):  num = num + 1  # print(key)  code = conn.hget(key, 'InstrumentCode')  re\_key = code[7:]  *if not* re\_key *in* re.keys():  re[re\_key] = ['', '']  *if* code[6] == 'P':  re[re\_key][1] = conn.hget(key, 'InstrumentID')  *if* code[6] == 'C':  re[re\_key][0] = conn.hget(key, 'InstrumentID')  # print(num)  *return* re   *def* init\_slist(self,*filename*: str):  *with* open(*filename*, 'r') *as* file:  cont = file.read()  res = yaml.load(cont, Loader=yaml.FullLoader)  *return* res['config']['slist']  *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']  self.interval\_list = [5,10,15,30,60,180,300,600,900,1800,3600,7200,14400,60\*60\*24]  self.config = Config('read\_yaml.yaml')  self.fqueue = {}  self.squeue = {}  self.opcqueue = {}  self.oppqueue = {}  self.v00queue = {}  self.v05queue = {}  self.vn5queue = {}  *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] = deque(maxlen=self.interval\_list[self.interval])  *for* key *in* self.config.config\_optlist.items():  self.oppqueue[key] = deque(maxlen=self.interval\_list[self.interval])  *for* key *in* self.config.config\_optlist.keys():   self.appendVQueue00(cur\_ts,conn\_w,':V:C'+key[:4]+'MV0000')  self.operate(cur\_ts,':V:C'+key[:4]+'MV0000',self.interval)  self.appendVQueue00(cur\_ts,conn\_w,':V:P'+key[:4]+'MV0000')  self.operate(cur\_ts,':V:P'+key[:4]+'MV0000',self.interval)  self.appendVQueueN5(cur\_ts,conn\_w,':V:C'+key[:4]+'MVN050')  self.operate(cur\_ts,':V:C'+key[:4]+'MVN050',self.interval)  self.appendVQueueN5(cur\_ts,conn\_w,':V:P'+key[:4]+'MVN050')  self.operate(cur\_ts,':V:P'+key[:4]+'MVN050',self.interval)  self.appendVQueue05(cur\_ts,conn\_w,':V:C'+key[:4]+'MV0050')  self.operate(cur\_ts,':V:C'+key[:4]+'MV0050',self.interval)  self.appendVQueue00(cur\_ts,conn\_w,':V:P'+key[:4]+'MV0050')  self.operate(cur\_ts,':V:P'+key[:4]+'MV0050',self.interval)  *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*:  latest = latest + d['LATEST']  bp1 = bp1 + d['BP1']  sp1 = sp1 + 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):  prefix = 'MDLD:'  c\_d = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*)  self.fqueue.append(c\_d)  *def* appendSQueue(self,*cur\_ts*:int, *conn\_w*, *keyword*:str):  prefix = 'MDLD:'  c\_d = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*)  self.squeue.append(c\_d)  *def* appendOPcQueue(self,*cur\_ts*:int,*conn\_w*,*keyword*:str):  prefix = 'MDLD:'  c\_d = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*)  self.qpcqueue.append(c\_d)   *def* appendOPpQueue(self,*cur\_ts*:int, *conn\_w*, *keyword*:str):  prefix = 'MDLD:'  c\_d = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*)  self.oppqueue.append(c\_d)   *def* appendVQueue00(self,*cur\_ts*:int, *conn\_w*, *keyword*:str):  prefix = 'MDLD:'  c\_d = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*)  self.v00queue.append(c\_d)   *def* appendVQueue05(self,*cur\_ts*:int, *conn\_w*, *keyword*:str):  prefix = 'MDLD:'  c\_d = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*)  self.v05queue.append(c\_d)   *def* appendVQueueN5(self,*cur\_ts*:int, *conn\_w*, *keyword*:str):  prefix = 'MDLD:'  c\_d = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*)  self.vn5squeue.append(c\_d)   *def* fillQueue(self):  *return* self.deque   *def* setDeque(self,*queue*:deque):  self.queue = *queue   def* operate(self, *cur\_ts*:int, *keyword*:str, *interval*:int, *conn\_w*):  prefix = "MDLD:"  *if not conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*, self.averageNames[self.interval] + 'LATEST'):  l = self.fillQueue(*cur\_ts*, *keyword*, self.interval)  d = self.averageCompute(l,*interval*)  c\_d = *conn\_w*.hgetall(prefix + str(*cur\_ts*) + *keyword*)  *conn\_w*.hmset(prefix + str(*cur\_ts*) + *keyword*, {\*\*c\_d, \*\*d})  *else*:  last\_average\_latest = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*, self.averageNames[self.interval] + 'LATEST')  last\_average\_sp1 = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*, self.averageNames[self.interval] + 'SP1')  last\_average\_bp1 = *conn\_w*.hget(prefix + str(*cur\_ts*) + *keyword*, self.averageNames[self.interval] + 'BP1')  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 = *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 = (last\_average\_latest \* self.interval\_list[  self.interval] - last\_first\_latest + curr\_latest) / self.interval\_list[self.interval]  new\_average\_sp1 = (last\_average\_sp1 \* self.interval\_list[self.interval] - last\_first\_sp1 + curr\_sp1) / \  self.interval\_list[self.interval]  new\_average\_bp1 = (last\_average\_bp1 \* self.interval\_list[self.interval] - last\_first\_bp1 + 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*, {\*\*c\_d, \*\*d})   *def* run(self,*key*:int):  conn\_w = redis.Redis(host="localhost", port=6379, password="", charset='gb18030', errors="replace",  decode\_responses=*True*)  cur\_ts = *key* prefix = "MDLD:" + str(cur\_ts)  total\_inf = {'flist':{},'optlist':{},'slist':{},'V':{}}  *for* key *in* self.config.config\_flist.values():  *if* len(self.fqueue) < self.interval\_list[self.interval]:  self.appendFQueue(cur\_ts,conn\_w,':F:F'+*key*)  *continue* self.appendFQueue(cur\_ts,conn\_w,":F:F"+*key*)  self.operate(cur\_ts,":F:F"+*key*,self.interval)  *for* key *in* self.config.config\_slist:  *if* len(self.squeue) < self.interval\_list[self.interval]:  self.appendSQueue(cur\_ts,conn\_w,conn\_w,':JZ:'+*key*)  *continue* self.appendSQueue(cur\_ts,conn\_w,':JZ:'+*key*,self.interval)  self.operate(cur\_ts,':JZ:'+*key*,self.interval)  *for* key *in* self.config.config\_optlist.items():  self.appendOPCQueue(cur\_ts,conn\_w,':OP:C'+*key*[0],self.interval)  self.operate(cur\_ts,':OP:C'+*key*[0],self.interval,conn\_w)  *for* key *in* self.config.config\_optlist.items():  self.appendOPpQueue(cur\_ts,conn\_w,':OP:P'+ *key*[0])  self.operate(cur\_ts,':OP:P'+*key*[0],self.interval,conn\_w)  *for* key *in* self.config.config\_optlist.keys():  self.appendVQueue00(cur\_ts,conn\_w,':V:C'+*key*[:4]+'MV0000')  self.operate(cur\_ts,':V:C'+*key*[:4]+'MV0000',self.interval)  self.appendVQueue00(cur\_ts,conn\_w,':V:P'+*key*[:4]+'MV0000')  self.operate(cur\_ts,':V:P'+*key*[:4]+'MV0000',self.interval)  self.appendVQueueN5(cur\_ts,conn\_w,':V:C'+*key*[:4]+'MVN050')  self.operate(cur\_ts,':V:C'+*key*[:4]+'MVN050',self.interval)  self.appendVQueueN5(cur\_ts,conn\_w,':V:P'+*key*[:4]+'MVN050')  self.operate(cur\_ts,':V:P'+*key*[:4]+'MVN050',self.interval)  self.appendVQueue05(cur\_ts,conn\_w,':V:C'+*key*[:4]+'MV0050')  self.operate(cur\_ts,':V:C'+*key*[:4]+'MV0050',self.interval)  self.appendVQueue00(cur\_ts,conn\_w,':V:P'+*key*[:4]+'MV0050')  self.operate(cur\_ts,':V:P'+*key*[:4]+'MV0050',self.interval) |

|  |
| --- |
| 参见work/src/intodb\_pkl\_20190719/readdata.py |
| #-\*- coding: UTF-8 -\*- *import* threading *import* time *import* redis *import* yaml  *class* Config:  *def \_\_init\_\_*(self, *fn\_config*:str):  self.config\_flist = self.init\_flist()  self.config\_optlist = self.init\_optlist()  self.config\_slist = self.init\_slist(*fn\_config*)  print(self.config\_flist)  print(self.config\_optlist)  print(self.config\_slist)   *def* init\_flist(self):  conn = redis.Redis(host="168.36.1.115", port=6380, password="", charset='gb18030', errors='replace',  decode\_responses=*True*)  pre = "qdb:securityex:derivatives:"  code = "CODE"  t = ['IC', 'IF', 'IH']  d = dict()  *for* i *in* t:  *for* j *in* range(1, 5):  d[i + "0" + str(j)] = conn.get(pre + i + "0" + str(j) + ":CODE")  *return* d   *def* init\_optlist(self):  conn = redis.Redis(host="168.36.1.170", port=6379, password="", charset='gb18030', errors='replace',  decode\_responses=*True*)  keys = conn.keys()   re = dict()  num = 0  *for* key *in* keys:  *if* key.startswith("OPLST:01"):  num = num + 1  # print(key)  code = conn.hget(key, 'InstrumentCode')  re\_key = code[7:]  *if not* re\_key *in* re.keys():  re[re\_key] = ['', '']  *if* code[6] == 'P':  re[re\_key][1] = conn.hget(key, 'InstrumentID')  *if* code[6] == 'C':  re[re\_key][0] = conn.hget(key, 'InstrumentID')  # print(num)  *return* re   *def* init\_slist(self,*filename*: str):  *with* open(*filename*, 'r') *as* file:  cont = file.read()  res = yaml.load(cont, Loader=yaml.FullLoader)  *return* res['config']['slist']   *def* second(*key*:str):  print("It is " + str(*key*),time.time(),time.localtime())   *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")  *def* timeit(func):  *def* test(*self*):  start = time.time()  func(*self*)  end = time.time()  print("time used: ", end - start)  *return* test   # @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 = redis.Redis(host="168.36.1.115", port=6379, password="", charset='gb18030', errors='replace',  decode\_responses=*True*)  conn\_w = redis.Redis(host="localhost", port = 6379, password="", charset='gb18030',errors="replace",  decode\_responses=*True*)  cur\_ts = key  conn\_w.set("MDLD:cur\_ts",cur\_ts)  pre = 'KZ:F'  f\_d\_list = dict()  d = 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['LATEST'] = latest  d['BP1'] = bp1  d['SP1'] = sp1  # print(d)  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":F:F" + key, d)  d = dict()  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  # 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")  *for* item *in* self.config.config\_optlist.items():  tem1 = conn\_r.hgetall("MD:01" + item[1][0])  tem2 = conn\_r.hgetall("MD:01" + item[1][1])  # print(tem1)  # print(tem2)  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":OP:" + "C" + item[0], tem1)  conn\_w.hmset("MDLD:" + str(cur\_ts) + ":OP:" + "P" + item[0], tem2)  px = int(item[0][-5:])  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:" + item[0],d)  conn\_w.set("MDLD:" + str(cur\_ts) + ":PO:" + item[0],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:ts:A13:" + key, d)  *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()    *def* main():  # print(type(time.localtime().tm\_mon),time.localtime().tm\_year,time.localtime().tm\_mday)  t = PeriodThread(["09:30:00","11:30:00"],["16:00:30","18:00:50"])  t.start()  time.sleep(10)  # t.join()  *if* \_\_name\_\_=='\_\_main\_\_':  main() |