def smartSendOrder(self, tradeSeries, buyPriceMode, sellPriceMode,

buyVolMode, sellVolMode, sliceFund, sliceVol=100):

'''

自定义模拟交易函数

:param self:

:param tradeSeries: 策略交易股票Series

:param buyPriceMode: 1:开盘买入 2：收盘买入 3: 最高价买入

:param sellPriceMode: 1:开盘卖出 2：收盘卖出

:param buyVolMode: 1：sliceFund等金额买入 2：等笔数买入

:param sellVolMode: 1：报单的股票全部卖出

:param sliceFund: 每期可用资金

:param sliceVol: 每期买入笔数

:return:

'''

buyStock = tradeSeries[tradeSeries == 1].index

sellStock = tradeSeries[tradeSeries == -1].index

priceSeries = copy.deepcopy(tradeSeries)

# TODO 1. 依据规则产生价格和成交量

if not buyStock.empty:

# 1.1.1 产生买入价格

if buyPriceMode == 1:

priceSeries[buyStock] = self.open()[buyStock]

elif buyPriceMode == 2:

priceSeries[buyStock] = self.close()[buyStock]

elif buyPriceMode == 3:

loc = self.getLoc(self.TradeTime)

TEntryLength= 3 # 进场通道周期

range1= 10 # 场动量阀值

range2= 10 # 入场动量阀值

EntryIvLength= 30 # 离场通道阀值

Entrysht = (self.high\_DF().rolling(TEntryLength).max().iloc[loc - 1][buyStock])\*(1-float(range1)/float(100))-0.01

tempIdenty1 = self.high\_DF().iloc[loc - 1][buyStock] != self.low\_DF().iloc[loc - 1][buyStock]

tempIdenty2 = self.low\_DF().iloc[loc][buyStock] < Entrysht

tempIdenty3 = self.open\_DF().iloc[loc][buyStock]<1.0995\*self.close\_DF().iloc[loc-1][buyStock]

temp\_min=np.minimum(Entrysht,self.open\_DF().iloc[loc][buyStock])

tempIdenty =tempIdenty1[tempIdenty1.values == True]&tempIdenty2[tempIdenty2.values == True]&tempIdenty3[tempIdenty3.values == True]

priceSeries[tempIdenty[tempIdenty.values == True].index] = temp\_min[tempIdenty[tempIdenty.values == True].index]

priceSeries = priceSeries.drop(tempIdenty[tempIdenty.values == False].index)

# 1.1.2 产生买入交易量

if buyVolMode == 1:

number=5

if self.TradeTime=="2017-09-22":

print buyStock

if len(buyStock) < int(number):

singleAmt = float(sliceFund) / number

if len(buyStock) >= int(number):

singleAmt = float(sliceFund) / float(len(buyStock))

elif buyVolMode == 2:

sliceVol = float(sliceVol)

if sliceVol < 100.:

print(u'买入笔数不能小于100')

sys.exit()

if not sellStock.empty:

# 1.2.1 产生卖出价格

if sellPriceMode == 1:

priceSeries[sellStock] = self.open()[sellStock]

elif sellPriceMode == 2:

priceSeries[sellStock] = self.close()[sellStock]

elif sellPriceMode == 3:

loc = self.getLoc(self.TradeTime)

EntryIvLength = 30

EntryLnglv = self.high\_DF().rolling(EntryIvLength).max().iloc[loc - 1][sellStock]

temp\_max = np.maximum(EntryLnglv, self.open\_DF().iloc[loc][sellStock])

tempIdenty4=self.high\_DF().iloc[loc][sellStock] > EntryLnglv

priceSeries[tempIdenty4[tempIdenty4.values == True].index] = temp\_max[tempIdenty4[tempIdenty4.values == True].index]

priceSeries = priceSeries.drop(tempIdenty4[tempIdenty4.values == False].index)

# 1.2.2 卖出的成交量默认为买入时的成交量

# TODO 2. 报单

for i in range(len(priceSeries)):

action = tradeSeries[priceSeries.index[i]]

code = priceSeries.index[i]

price = priceSeries[code]

if action == 1: # 买入

fund = self.getCaiptal() # 资金

orderVol = 0

if buyVolMode == 1:

orderVol = np.round(singleAmt / float(price), decimals=-2) # 取100整数

elif buyVolMode == 2:

orderVol = sliceVol

amt = price \* orderVol

amtAfCost = self.amountAfCostF(action, amt, self.costF(action, amt)) # 已计算手续费

# 2.1 资金检验

if fund >= amtAfCost:

self.sendOrder(code, action, price, orderVol)

else:

lots = np.round(fund / price, decimals=-2)

if lots:

self.sendOrder(code, action, price, lots)

elif action == -1: # 卖出

if code in self.getPosition(): # 只对已持有的股票卖出

pos = self.getPosition(code)

orderVol = -0

if sellVolMode == 1:

orderVol = self.getPosition()[code] # 卖出该股票所有的头寸

# 2.2 仓位检验

if pos >= orderVol:

self.sendOrder(code, action, price, orderVol)

elif pos > 0:

self.sendOrder(code, action, price, pos)

return buyPriceMode, sellPriceMode