组合交易结构说明

新增手工交易持仓和策略持仓的显示界面，两者绑定同样的数据构造，区别在于通过Visibility的不同的绑定关系，形成互斥的不同显示结果。区分两个DataGrid: PositionNormalGrid：用于手工交易持仓；StrategyPositionGrid：用于策略持仓。

public readonly ObservableCollection<LogicPosition> LogicPositionView = new ObservableCollection<LogicPosition>();

组合的持仓采用逻辑持仓结构来保存逻辑持仓结构，继承自PositionDataSummary, 新增 IsCombo和 IsStrategy 两个逻辑类型的属性和 OriginalOpenCost原始开仓成本。

说明：组合持仓为某个组合的汇总持仓，区分持仓的类别（多、空）。持仓盈亏的计算根据 OriginalOpenCost 跟当前的价值比较，例如：10个组合的螺纹跨期多仓，持仓成本560\*10=5600， 当前价格为550\*10=5500， 持仓盈亏为：5500-5600 = -100。

注: 当前持仓为零的纪录也需要显示, 保留当日开仓\当日平仓\等数据

日结初始化过程：

OrderRef==1，

OrderRefPrefix=1，

读入历史组合持仓（HistoricComboPositions.xml)，将逻辑持仓中的昨仓

YdPosition = Position,

YdOpenCost=OpenCost,

日结时,过滤掉当前持仓为零的纪录

计算并保存PositionCost = PreSettlementPrice\*Position\*volumeMultiple，并读入IsStrategy, IsCombo。

维护策略及手工交易持仓：

在成交回报的回调函数中，维护策略持仓（同时也就维护了手工交易持仓，通过设置IsStrategy），通过报单引用OrderRef，在LogicOrderMappingList查找，策略及组合交易（参照OrderRef的前4位字符），然后对照LogicOrderMapping 来处理，更新LogicOrderMapping, 对于单腿的情况，则不计入逻辑持仓中。

历史持仓保存：在退出程序的时候，保存组合持仓数据，数据结构采用LogicPosition。

历史持仓核心数据解释：

OriginalOpenCost :

保存原始的开仓成本（以金额计算：OpenVolume\*VolumeMultiple\*OpenPrice）

IsStategy： 是否策略持仓

IsCombo： 是否组合持仓

YdPosition: 昨日持仓

Position：当前持仓

报单处理：

在报单时，对于组合或者策略的报单生成OrderRefPrefix（4位的字符串），生成逻辑报单，

在TQMain的public static List<LogicOrderMapping> LogicOrderMappingList中保存一份LogicOrderMapping，

注意的是：对于组合，下单板没有开仓的选项，只有自动的选项

LogicOrderMapping

/// <summary>

/// 逻辑报单与成交明细对应关系

/// 包含策略或者组合

/// 不包含普通报单

/// </summary>

[Serializable]

public class LogicOrderMapping

{

public OrderParams OrderParams; //逻辑报单

[XmlIgnore]

public List<TradeField> TradeList; //修改为成交回报明细列表

public LogicPosition Position; //逻辑持仓

/// <summary>

/// 接受报单回报

/// 计算 OpenVolume 、CloseVolume

/// 并设置 position = ydposition + OpenVolume - CloseVolume

/// 所有对应逻辑报单的账户订单数据 保存并维护在OrderDataList

/// </summary>

/// <param name="orderData"></param>

}

}

public class LogicPosition : PositionDataSummary

{

/// <summary>

/// 是否策略持仓

/// </summary>

public bool IsStategy { get; set; }

/// <summary>

/// 是否组合持仓

/// </summary>

public bool IsCombo { get; set; }

///昨仓开仓成本

public double YdOpenCost { get; set;}

}

public class OrderParams

{

/// <summary>

/// 4位的 序列号

/// 每一笔用户下单分配一个（可能会有多笔实际报单公用同一个序列号）

/// </summary>

public string OrderRefPrefix { get; set; }

/// <summary>

/// 手工：Manual

// 策略：GUID（不是 4位的 序列号）

/// </summary>

public string StrategyId { get; set; }

public bool IsCombo { get; set; }

private string \_Instrument;

/// <summary>

/// 合约

/// </summary>

public string Instrument

{

get { return \_Instrument; }

set

{

\_Instrument = value;

}

}

private OrderMode \_OrderMode = OrderMode.Auto;

/// <summary>

/// 交易方式

/// 平仓、开仓、自动

/// </summary>

public OrderMode OrderMode

{

get { return \_OrderMode; }

set

{

\_OrderMode = value;

}

}

private HedgeType \_HedgeType = HedgeType.投机;

/// <summary>

/// 套保类别

/// 投机、套利、套保

/// </summary>

public HedgeType HedgeType

{

get { return \_HedgeType; }

set

{

\_HedgeType = value;

}

}

private PricingMode \_PricingMode = PricingMode.Preset;

/// <summary>

/// 限价（设定价，对张价，己方价）

/// 市价

/// </summary>

public PricingMode PricingMode

{

get { return \_PricingMode; }

set

{

\_PricingMode = value;

}

}

/// <summary>

/// FAK / FOK

/// </summary>

private FakFok \_FakFok = FakFok.Default;

public FakFok FakFok

{

get { return \_FakFok; }

set

{

\_FakFok = value;

}

}

/// <summary>

/// 下单数量，有默认值

/// </summary>

private int \_Quant = -1;

public int Quant

{

get

{

return \_Quant;

}

set

{

\_Quant = value;

}

}

/// <summary>

/// 下单价

/// </summary>

private double \_Price;

public double Price

{

get

{

return \_Price;

}

set

{

\_Price = value;

}

}

private TradeDirection \_TradeDirection;

/// <summary>

/// 方向（多、空）

/// </summary>

public TradeDirection TradeDirection

{

get { return \_TradeDirection; }

set

{

\_TradeDirection = value;

}

}

public string InvestorID { get; set; }

/// <summary>

/// 交易者ID

/// </summary>

private string \_ExchangeID;

public string ExchangeID

{

set { \_ExchangeID = value; }

get

{

if (\_ExchangeID == null)

return Utility.GetExchangeID(Instrument);

else return \_ExchangeID;

}

}

protected double \_PriceTick = 0;

/// <summary>

/// 一跳的点数

/// </summary>

public virtual double PriceTick

{

get

{

return \_PriceTick;

}

set

{

\_PriceTick = value;

}

}

/// <summary>

/// 已成交数量

/// </summary>

public int VolumeTraded { get; set; }

}