

证券AI产品 “量化” 子功能界面需求文档

一、项目目标

将金融量化功能添加到大模型产品中，通过左侧量化策略库枚举与右侧大模型问答交互形式，为用户提供便捷、准确的金融量化分析服务，降低量化分析门槛，提高投资决策效率。

二、需求说明

(一) 界面布局

1. 首页布局（参考整体效果图）：屏幕左侧（参考策略库示例）设量化策略库区域，枚举多因子选股、量化择时等常见策略，每个策略项含名称、简介及“查看详情”按钮。屏幕右侧为Chat形式的交互界面。



整体效果

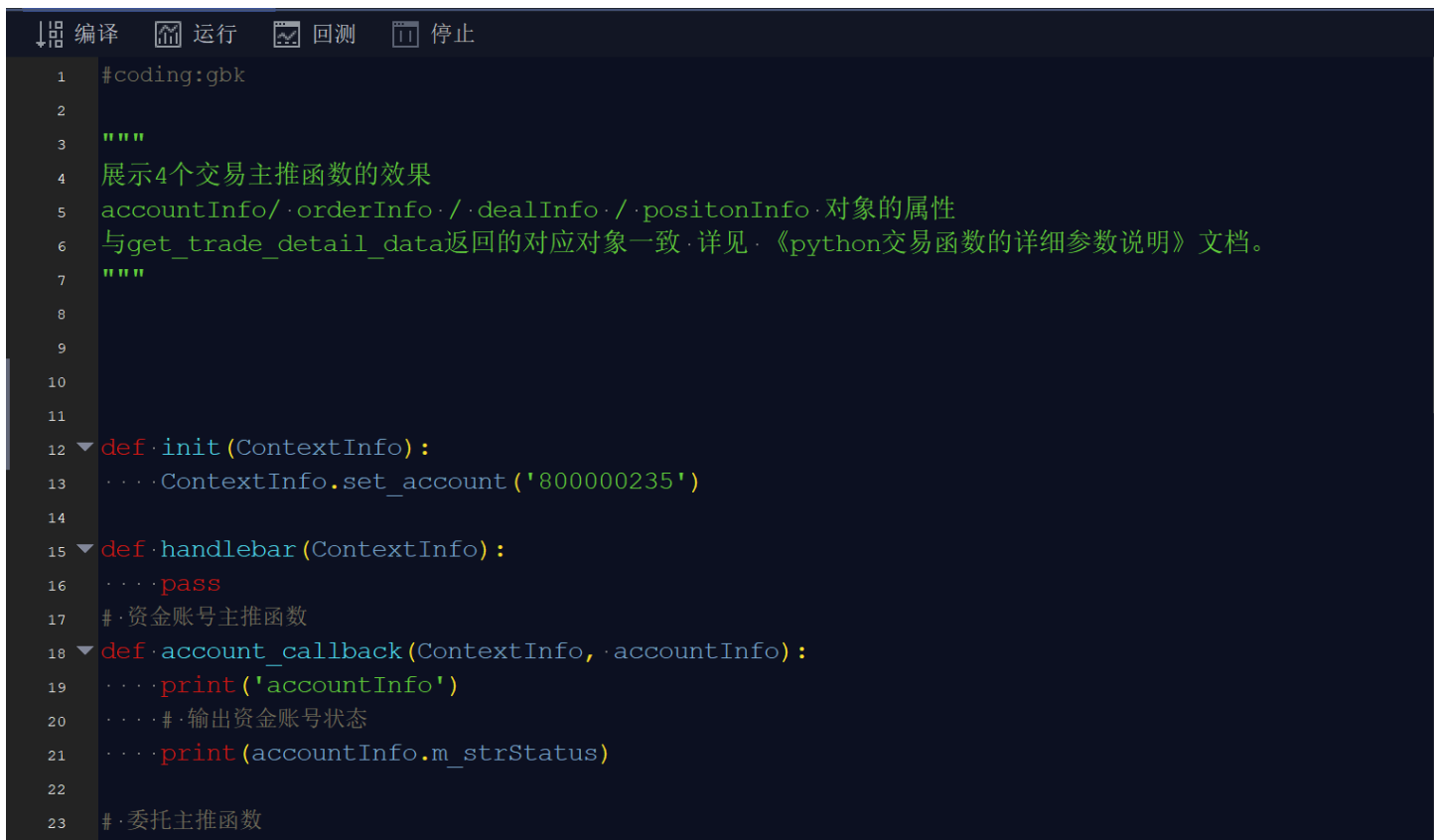


策略库示例

2. 操作界面布局（参考整体效果图）：屏幕右侧用户通过Chat进行交互（参考腾讯元宝示例），用户可通过自然语言描述对选中策略进行调整以及参数设置，最终输出代码（代码框需与代码执行器绑定，界面可参考代码执行器示例）。代码执行后屏幕左侧输出代码执行结果（可包含图例、代码执行内容及结果等）



整体效果



代码执行器示例

(二) 大模型问答功能

1. Chat问答界面：屏幕右侧保持大模型问答界面，用户可输入量化相关问题，系统快速显示回答。

2. **量化功能调用**：识别量化策略相关问题，调用策略库中策略，结合大模型分析能力，给出准确回答和建议。

（三）策略生成与回测功能

1. **策略生成**：根据用户输入条件，大模型生成个性化量化投资策略代码，含策略参数、交易规则等详细信息，提供“保存策略”、“运行”、“回测”、“停止”按钮。
2. **策略回测**：用户可对生成或选择的策略进行回测，设置回测参数，系统实时显示回测进度和关键指标，回测完成后生成详细报告。

(四) 用户交互与反馈功能

1. **用户输入提示：** 问答界面提供输入提示，如常见问题示例、格式要求等，引导用户明确提问。
2. **回答结果展示与反馈：** 以清晰、易懂方式呈现回答结果，用户可对结果评价和反馈，系统据此优化回答质量和策略推荐效果。

（五）数据接入选择功能

1. **数据接入选择**：提供一个可勾选的数据种类库，客户可根据实际需要，选择需要接入的数据种类；
2. **数据库编辑上传**：客户可上传本地数据，新建数据库。

三、腾讯元宝（DeepSeek r1满血版）回答示例

问题：帮我生成一段量化回测代码，规则如下，HS300日线下运行，20个交易日进行一次调仓，每次买入在买入备选因子评分前10的股票，每支股票各分配当前可用资金的10%(权重可调整)

元宝

帮我生成一段量化回测代码，规则如下：
HS300日线下运行，20个交易日进行一次调仓，每次买入在买入备选池中因子评分前10的股票，每支股票分配当前可用资金的10%（权重可调整）

已深度思考 (用时81秒)

好的，用户让我帮他生成一段量化回测代码，规则是基于HS300日线下运行，每20个交易日调仓一次，买入备选池中因子评分前10的股票，每支分配当前可用资金的10%。首先，我需要确认用户的需求是否明确，有没有遗漏的细节。

首先，HS300日线下运行，可能是指当HS300指数的收盘价低于其N日均线时，才执行交易策略。这里可能需要用户确认具体是哪个均线周期，比如20日、30日还是其他。如果用户没有说明，可能需要默认一个值，比如20日均线，并在代码中作为参数，方便后续调整。

接下来，调仓周期是20个交易日，也就是每周调一次。但需要确认是否在满足HS300条件下才调仓，还是无论条件如何都每20天调仓。根据用户描述，应该是当HS300在日线下运行时才调仓，所以需要先判断HS300是否满足条件，再执行调仓。

买入备选中的因子评分前10的股票，这里需要明确备选池的来源。是HS300成分股吗？用户提到的是买入备选中的股票，可能备选池已经预先确定，比如HS300成分股。但代码中可能需要获取这些成分股的历史数据，并进行因子评分。因子评分具体是什么因子？用户没有说明，可能需要留出接口，比如自定义因子或者示例因子，比如动量、市盈率等。这里可能需要用占位符，让用户替换成自己的因子计算逻辑。

资金分配方面，每支股票分配当前可用资金的10%，总共有10支，这样刚好是100%。但用户提到权重可调整，所以可能需要参数化这个比例，比如每个股票分配固定比例，或者动态计算。这里需要注意可用资金的计算，每次调仓时，可用资金是当前的现金，还是总资产减去已持仓的价值？通常，可用资金是指现金部分，但需要明确。在回测中，卖出原有持仓，再买入新的，可能需要先卖出再买入，或者调仓时先卖后买。

接下来，代码的结构需要考虑使用什么框架。用户可能使用的是聚宽（JoinQuant）、掘金（MyQuant）或者其他平台。由于用户没有指定，可能需要选择一个通用的框架，比如使用Backtrader，或者提供伪代码，但考虑到中文社区中聚宽比较常见，可能使用聚宽的API来编写，但需要确认用户的环境。

另外，数据获取部分，HS300的历史数据，以及股票池的历史数据需要处理，在聚宽中，可以使用get_index_stocks获取指数成分股，然后获取这些股票的财务或市场数据来计算因子。

关于回测的细节，比如手续费、滑点等，用户没有提到，可能需要默认添加，并在代码中作为参数，方便用户调整。

现在，整理步骤：

1. 确定HS300的均线条件，比如收盘价低于20日均线。

2. 每20个交易日检查条件，如果满足，则调仓。

3. 调仓时，获取备选池来源（如HS300成分股）。

4. 计算各股票的因素子评分，排序取前10。

5. 卖出当前持仓的不在前10的股票。

6. 买入前10的股票，每只分配可用资金的10%。

7. 处理订单，考虑手续费和滑点。

需要注意的问题：

● 设定止损目标：比如每隔20个交易日，可能需使用交易日历，避免无交易日。

● 因子评分的计算可能需要历史数据，比如过去一段时间的动量、市盈率等，需要提前准备好数据。

● 资金分配时，需考虑可用资金是否足够，以及股票价格是否可买（比如是否够买一手）。

● 确保停盘期间市价的情况，需处理，避免无法成交的股票。

● 初始资金设置，应同时考虑交易费用由自己设定。

可能的代码结构（以聚宽为例）：

● 初始化函数：设置策略参数。

● 每天检查是否触发调仓，并检索HS300条件。

● 计算因子，排序选股。

● 维持持仓。

需要测试的注意事项：比如当可用资金不足以购买10只股票时如何处理，或者选股不足10只该怎么办。

最后，代码中需要注释说明各个变量，方便用户理解和修改参数。可能还需要添加参数，如均线周期、调仓频率、选股数量、资金分配比例等，以提高灵活性。

```
# 回测基础设置
start_date = '2018-01-01'
end_date = '2023-12-31'
benchmark = '000300.XSHG' # 沪深300指数
universe = get_index_stocks('000300.XSHG') # 获取HS300成分股
capital_base = 1000000
refresh_rate = 20 # 调仓周期

# 策略参数
MA_PERIOD = 20 # 均线周期
TOP_N = 10 # 选股数量
WEIGHT = 0.1 # 单股仓位比例

def initialize(context):
    context.set_commission(Commission(buy_cost=0.0003, sell_cost=0.0013)) # 设

def handle_data(context, data):
    # 获取HS300指数数据
    hs300 = context.history(benchmark, ['close'], 1, '1d')
    current_price = hs300['close'][-1]
    ma = context.history(benchmark, ['close'], MA_PERIOD, '1d')['close'].mean()

    # 仅当指数在均线下方时运行策略
    if current_price > ma:
        return

# 到达调仓日执行
if context.current_date.strftime('%Y-%m-%d') in context.trading_days:
    # 获取备选股因子评分 (需替换实际因子计算)
    factor_scores = {}
    for stock in universe:
        # 示例因子: 20日收益率 (需替换为实际因子)
        returns = context.history(stock, ['close'], 20, '1d')['close'].pct_change()[1:]
        factor_scores[stock] = returns

    # 排序选股
    selected = sorted(factor_scores.items(), key=lambda x: x[1], reverse=True)[0:TOP_N]
    selected_stocks = [s[0] for s in selected]

    # 调整仓位
    rebalance_portfolio(context, selected_stocks)

def rebalance_portfolio(context, target_stocks):
    # 卖出非目标持仓
    for stock in context.portfolio.positions:
        if stock not in target_stocks:
            order_target(stock, 0)

    # 计算可用资金
    available_cash = context.portfolio.cash

    # 分配资金买入
    for stock in target_stocks:
        # 计算目标市值
        target_value = available_cash * WEIGHT
        current_position = context.portfolio.positions[stock].amount

        # 获取最新价格
        price = context.current(stock, 'price')
```

```
# 到达调仓日执行
if context.current_date.strftime('%Y-%m-%d') in context.trading_days:
    # 获取备选股因子评分 (需替换实际因子计算)
    factor_scores = {}
    for stock in universe:
        # 示例因子: 20日收益率 (需替换为实际因子)
        returns = context.history(stock, ['close'], 20, '1d')['close'].pct_change()[1:]
        factor_scores[stock] = returns

    # 排序选股
    selected = sorted(factor_scores.items(), key=lambda x: x[1], reverse=True)[0:TOP_N]
    selected_stocks = [s[0] for s in selected]

    # 调整仓位
    rebalance_portfolio(context, selected_stocks)

def rebalance_portfolio(context, target_stocks):
    # 卖出非目标持仓
    for stock in context.portfolio.positions:
        if stock not in target_stocks:
            order_target(stock, 0)

    # 计算可用资金
    available_cash = context.portfolio.cash

    # 分配资金买入
    for stock in target_stocks:
        # 计算目标市值
        target_value = available_cash * WEIGHT
        current_position = context.portfolio.positions[stock].amount

        # 获取最新价格
        price = context.current(stock, 'price')
```

```
# 计算目标股数 (考虑实际可买数量)
if price > 0:
    target_amount = int((target_value / price / 100) * 100) # 按手数取整
    if target_amount > current_position:
        order(stock, target_amount - current_position)
```

使用说明:

- 需要替换实际因子计算逻辑 (示例使用简单收益率)
- 需完善股票池获取和过滤逻辑
- 需处理停牌、退市等异常情况
- 可调整参数: MA_PERIOD/TOP_N/WEIGHT
- 建议加入风险控制模块 (止损、最大回撤控制等)

注意事项:

- 实际使用需确保因子数据的可获得性
- 需考虑交易成本和滑点影响
- 建议在回测时加入更多风控指标分析
- 注意指数成分股动态调整问题

建议在专业量化平台 (如JoinQuant、RiceQuant) 上运行并优化该策略, 需要根据实际需求补充完整的数据处理和因子计算模块。

来自于DeepSeek R1

回答

回答

回答

问题2：加入风险控制模块以及实时数据，重新生成代码

引用 1 个网页，6 篇公众号文章

已深度思考 (用时37秒)

好的, 用户现在要求加入风险控制模块和实时数据, 重新生成之前的量化回测代码。我需要先回顾之前的对话, 看看之前的代码结构是怎样的, 然后结合用户提供的搜索结果来补充这两个模块。

首先, 之前的代码已经包括了基础的调仓逻辑、因子评分和资金分配。现在需要添加风险控制, 比如止损、最大回撤控制, 以及仓位管理。根据搜索结果中的文献6, 风险管理包括止损、止盈、波动率控制等。文献6提到, 风险管理不仅是为了规避风险, 还要提高回报的可持续性, 比如限制行业和市值敞口。此外, 文献7强调了在回测中考虑滑点和交易成本的重要性。

关于实时数据, 用户可能希望代码能够接入实时行情进行模拟或实盘交易。文献1提到的miniQMT系统支持回测、模拟和实盘, 使用事件驱动架构, 这可能对处理实时数据有帮助。文献3讨论了事件驱动回测框架, 适合实盘交易, 但代码复杂度较高。不过用户可能暂时不需要事件驱动, 但需要确保数据获取模块可以处理实时数据。

接下来, 我需要修改代码结构, 添加风险控制部分。比如在每次调仓前检查最大回撤, 如果超过阈值则清仓。同时, 在买入个股时加入止损逻辑, 比如根据ATR指标设置动态止损点。另外, 文献7提到要处理滑点和交易成本, 所以需要在订单执行时考虑这些因素。

实时数据方面, 可能需要使用如yfinance或pandas-datareader来获取实时行情, 但具体实现可能依赖量化平台的数据接口。例如, 在聚宽平台, 可以使用get_price函数获取实时数据, 需要确保数据获取部分能处理实时和历史数据, 以便回测和实盘一致。

还需要考虑代码的模块化, 将风险控制和数据获取分离, 便于维护。例如, 将止损逻辑放在单独的函数中, 在每次下单前调用。同时, 仓位管理部分需要根据可用资金和风险参数动态调整头寸大小。

最后, 测试代码是否涵盖了所有新增功能, 确保风险控制模块在回测中正确触发, 并且实时数据能正确集成。可能还需要添加日志记录, 方便调试和监控策略执行情况。

我将基于您的要求, 结合风险控制模块和实时数据要素, 优化之前的回测代码框架。以下是增强版代码 (基于事件驱动架构设计):

我将基于您的要求, 结合风险控制模块和实时数据要素, 优化之前的回测代码框架。以下是增强版代码 (基于事件驱动架构设计):

```
# ===== 系统配置模块 =====
import pandas as pd
from datetime import datetime, timedelta

# 风险控制参数
RISK_PARAMS = {
    'max_drawdown': 0.20, # 最大回撤阈值
    'volatility_window': 60, # 波动率计算窗口
    'stop_loss_pct': 0.08, # 个股止损比例
    'position_limit': 0.8 # 总仓位限制
}

# 实时数据接口配置
DATA_SOURCE = 'Yahoo' # 可选: JoinQuant, RQData, Yahoo
LIVE_MODE = False # 实时交易模式开关

# ===== 核心交易模块 =====
class RiskManager:
    """风险控制模块 (文献6核心思想实现)"""
    def __init__(self, portfolio):
        self.portfolio = portfolio
        self.max_portfolio_value = portfolio.initial_capital
        self.drawdown = 0

    def check_system_risk(self):
        # 最大回撤控制 (文献6[6]@ref1)
        current_value = self.portfolio.value
        self.max_portfolio_value = max(self.max_portfolio_value, current_value)
        self.drawdown = (self.max_portfolio_value - current_value) / self.max_portfolio_value
        if self.drawdown > RISK_PARAMS['max_drawdown']:
            self.portfolio.liquidate_positions() # 清仓止损
            return False
        return True

    def check_position_risk(self, stock, price):
        # 个股止损逻辑 (文献7[7]@ref1)
        position = self.portfolio.positions.get(stock)
        if position:
            cost = position.cost_basis
            if (price - cost) / cost < -RISK_PARAMS['stop_loss_pct']:
```

```
def _generate_signals(data):
    """信号生成 (需补充因子计算)"""
    # 示例: 动量因子 (文献[2]@ref1)
    returns = data.pct_change(20)
    signals = returns.iloc[1:].sort_values(ascending=False)
    return signals.head(10)

def _execute_trades(context, signals):
    """订单执行模块 (含风险控制)"""
    available_cash = context.portfolio.cash * RISK_PARAMS['position_limit']

    for stock in signals.index:
        # 个股风险检查 (文献[6]@ref1)
        price = data.current(stock, 'price')
        if not context.risk_manager.check_position_risk(stock, price):
            continue

        # 计算目标仓位
        target_value = available_cash * 0.1
        current_value = context.portfolio.positions[stock].value

        # 执行调仓
        if target_value > current_value:
            order_value(stock, target_value - current_value)
```

主要增强功能说明 (综合文献1-7):

- 风险控制体系** (文献6 ➡ 核心实现)
 - 组合层面: 动态最大回撤监控 (超过20%自动清仓)
 - 个股层面: 8%止损机制 (基于持仓成本计算)
 - 仓位控制: 总仓位不超过80% (文献6风险敞口控制)
- 实时数据接口** (文献1 ➡ 架构参考)
 - 统一数据接口: 支持历史回测/实时交易模式切换
 - 数据缓存机制: 保留60天历史数据窗口
 - 模块化设计: 便于对接不同数据源 (Yahoo/聚宽等)