**代码：**

// Authored by 李凌瑶 in 2024/06/21

use rand::Rng; // 引入随机数生成器

use std::time::{Duration, Instant};

#[derive(Debug, Clone, Copy)]

enum Side { // 定义了两种交易方向：买入（BID）和卖出（ASK）

    BID,

    ASK,

}

#[derive(Debug, Clone, Copy)]

struct Trade {

    price: f32,

    quantity: f32,

    side: Side,

}

#[derive(Debug, Clone, Copy)]

struct Position {

    avg\_price: f32,

    quantity: f32,

    side: Side,

}

fn calc(pos: Position, trades: Vec<Trade>) -> (Position, f32) {

    let mut position = pos;

    let mut realized\_pnl = 0.0;

    for trade in trades {

        let (new\_avg\_price, new\_quantity, trade\_pnl) = match (position.side, trade.side) {

            (Side::BID, Side::BID) | (Side::ASK, Side::ASK) => {

                // 如果持仓和交易同向，合并持仓。计算新的平均价格new\_avg\_price、新的持仓数量new\_quantity（当前持仓数量加上交易数量）。

                let new\_avg\_price = (position.avg\_price \* position.quantity + trade.price \* trade.quantity) / (position.quantity + trade.quantity);

                (new\_avg\_price, position.quantity + trade.quantity, 0.0)

            },

            (Side::BID, Side::ASK) => {

                // 计算盈亏(pnl)：如果交易数量小于或等于持仓数量，则按交易数量计算盈亏；如果交易数量大于持仓数量，则按持仓数量计算盈亏。

                let pnl = (trade.price - position.avg\_price) \* position.quantity.min(trade.quantity);

                // 计算新的平均价格和持仓数量

                let (new\_avg\_price, new\_quantity,new\_side) = if position.quantity > trade.quantity {

                    (position.avg\_price, position.quantity - trade.quantity, position.side)

                } else {

                    (trade.price, trade.quantity-position.quantity, trade.side)

                };

                position.side = new\_side;

                (new\_avg\_price, new\_quantity, pnl)

            },

            (Side::ASK, Side::BID) => {

                // 同理

                let pnl = (position.avg\_price-trade.price) \* position.quantity.min(trade.quantity);

                let (new\_avg\_price, new\_quantity,new\_side) = if position.quantity > trade.quantity {

                    (position.avg\_price, position.quantity - trade.quantity,position.side)

                } else {

                    (trade.price, trade.quantity-position.quantity,trade.side)

                };

                position.side = new\_side;

                (new\_avg\_price, new\_quantity, pnl)

            },

        };

        position.avg\_price = new\_avg\_price;

        position.quantity = new\_quantity;

        realized\_pnl += trade\_pnl;

    }

    (position, realized\_pnl)

}

fn main() {

    let mut rng = rand::thread\_rng();

    let mut trades: Vec<Trade> = Vec::new();

    // 生成10000个随机成交数据

    for \_ in 0..10000 {

        let price: f32 = rng.gen\_range(0.0..1000.0);

        let quantity: f32 = rng.gen\_range(1.0..100.0);

        let side = if rng.gen() { Side::BID } else { Side::ASK };

        trades.push(Trade { price, quantity, side });

    }

    let init\_pos = Position {

        avg\_price: 100.0,

        quantity: 10.0,

        side: Side::BID,

    };

    let start\_time = Instant::now();

    let (updated\_pos, pnl) = calc(init\_pos, trades);

    let duration = start\_time.elapsed();

    println!("Updated Position: {:?}", updated\_pos);

    println!("Realized PnL: {}", pnl);

    println!("Total time taken: {:?}", duration);

    // 计算平均运行时间

    let average\_time\_per\_trade = duration / 10000;

    println!("Average time per trade: {:?}", average\_time\_per\_trade);

}

**Cargo.toml配置依赖：**

[package]

name = "pro2"

version = "0.1.0"

edition = "2021"

[dependencies]

serde = { version = "1.0.194", features = ["derive"] }

serde\_json = "1.0.110"

thiserror = "1.0.56"

rand = "0.8.5"

**运行结果（随机生成了10000个trade，输出平均处理用时，稳定在20ns）：**

