这是一个把"每笔成交的真实盈亏轨迹"落地到系统里的小方案：用 IBKR 的成交与PnL流，实时维护每笔订单自入场以来的 **MFE/MAE**（最大有利/最大不利），并据此做风控与复盘。

为什么要做

- **行为纠偏**：把"当时其实已经到过+8%却没走"的事实量化出来，抑制处置效应与过度持仓。
- **策略评估**：按"每笔订单"的单位衡量边际改进，而不是只看组合层的平滑平均。
- **自动化风控**：当 MAE 触发阈值或 MFE 达到目标时，自动提醒/减仓/移动止损。

核心数据流（IBKR）

1. **成交流**：订阅 `reqExecutions / execDetails`（或 Client Portal 的 executions），拿到
   - `executionId`（去重主键）、`contract`（标的）、`qty`、`price`（入场价）、`time`、`side`。
2. **PnL / 持仓流**：`reqPnL / reqPnLSingle` 或 Client Portal 的 `portfolio snapshots`。
3. **行情流**：订阅标的的 **last/ bid/ ask/ high/ low**（或直接用 last 计算未实现PnL）。

> 入场后，维护一个每单的滚动高/低：
> **MFE = max(high_since_entry − entry_price)**
> **MAE = min(low_since_entry − entry_price)**（做空则方向取反）。

最小可用实现（MVP）

- **写入层**：执行事件落库（Postgres 表 `orders_rt`）：`trade_id`, `symbol`, `side`, `entry_px`, `qty`, `mfe_px`, `mae_px`, `last_px`, `updated_at`。
- **更新器**：每个行情tick：
  - `mfe_px = max(mfe_px, last_px)`（多头；空头用 `min`）
  - `mae_px = min(mae_px, last_px)`（多头；空头用 `max`）
  - 同步写回并记录一条**迷你时间序列**（RedisTimeSeries 或 Postgres `orders_rt_ts(trade_id, ts, last, mfe, mae)`）。
- **告警器**（规则引擎）：
  - **MAE 跌破阈值**：如 `MAE <= -3%` 触发Slack/短信。
  - **MFE 触达分批止盈**：如 `MFE >= +5%` → 提示减仓/调高止损。
  - **时间无进展**：入场后 `T` 分钟内 `MFE < +0.5%` → 发"无动量"提示。

- **去重&补偿**：
  - 以 `executionId` 去重；若错过回调，用周期性 `reqExecutions` 轮询补洞。
  - 大账户/顾问账户 PnL 汇总可能滞后；以**行情价×持仓**作本地兜底计算。

表结构草案

sql

```sql
-- 主表：每笔订单的滚动高低
create table orders_rt (
  trade_id text primary key,
  symbol text not null,
  side text check (side in ('BUY','SELL')) not null,
  qty numeric not null,
  entry_px numeric not null,
  mfe_px numeric not null,
  mae_px numeric not null,
  last_px numeric not null,
  opened_at timestamptz not null,
  updated_at timestamptz not null
);

-- 轻量时序：便于画“从入场到当前”的轨迹
create table orders_rt_ts (
  trade_id text,
  ts timestamptz,
  last_px numeric,
  mfe_px numeric,
  mae_px numeric,
  primary key (trade_id, ts)
);
create index on orders_rt_ts (trade_id, ts desc);
```

Copy code

事件处理伪代码

python

```python
# on_execution(exe):
upsert orders_rt with (trade_id, symbol, side, qty, entry_px=exe.price, mfe_px=ex

# on_tick(symbol, last):
for each open trade_id in symbol:
  if side == 'BUY':
    mfe_px = max(mfe_px, last); mae_px = min(mae_px, last)
  else:  # SELL
```

```
    mfe_px = min(mfe_px, last); mae_px = max(mae_px, last)
last_px = last
write orders rt & append orders rt ts
```


告警规则示例（百分比基于入场价）

- **硬止损**：`(last - entry)/entry <= -X%`
- **分层止盈**：`MFE% ≥ 5% → 提示减 1/3；≥ 8% → 再减 1/3`
- **拖尾止损**：`(last - max_run_up)/max_run_up <= -Y%`
- **时间止损**：`now - opened_at > 2h 且 MFE% < 0.5%`

健壮性与错误模式

- **缺失 `execDetailsEnd` / 回调延迟**：定时 `reqExecutions` 兜底；以 `executionId` 幂等更新。
- **PnL 聚合滞后**：大账户下用本地 `持仓×价格` 快速估算，并与官方PnL异步对账。
- **速率限制**：批量写库、指数退避；对相同tick事件做**去抖**（仅价变更才刷库）。
- **快照兜底**：周期性拉 `GET /iserver/account/orders` + `/portfolio/positions` 与内存状态对账。

可视化与复盘

- **单笔轨迹图**：入场价横线 + `last/mfe/mae` 曲线；标出告警节点。
- **分布视图**：过去N笔的 `max(MFE%)` 与 `max(drawdown%)` 直方图，指导止盈/止损参数。
- **策略对比**：同一策略的 MFE/MAE 分布 VS 另一个策略，验证改动是否提高"到达目标前被止损"的概率。

落地建议（与你现有栈对齐）

- **消息总线**：Manus/自建 Python worker 消费 IBKR 事件；
- **存储**：Postgres 为主（易做查询与报表），RedisTimeSeries 做高频缓存；
- **集成**：写 Notion/Obsidian 的日终复盘卡片，自动嵌入每单的 MFE/MAE 火花线与要点。
- **权限与回测**：生产与回测分库；回放历史 tick/exec 来回测规则的误报/漏报率。

如果你愿意，我可以直接给你：

1. 一个最小 Python 服务（IB-insync 版本）；
2. Postgres 建表脚本 + 初始告警规则；
3. 日终复盘的 Markdown 模板（自动汇总 Top MAE/MFE、错失告警、规则命中率）。

为什么要监测这个指标