# 使用Tick/Trade数据生成1分钟K线

## 时间要求

8小时内完成

## 函数框架

Tick 是tob行情，Trade是逐笔数据

type Tick struct {

    Symbol       string  `json:"symbol"`

    BestBidPrice float64 `json:"bbp"`

    BestBidSize  float64 `json:"bbs"`

    BestAskPrice float64 `json:"bap"`

    BestAskSize  float64 `json:"bas"`

    Timestamp    int64   `json:"timestamp"`

}

type Trade struct {

    Symbol     string  `json:"symbol"`

    Price      float64 `json:"price"`

    Size       float64 `json:"size"`

    TradeCount int64   `json:"tradecount"`

    IsBuy      bool    `json:"m"`

    Timestamp  int64   `json:"timestamp"`

}

由下面的struct接收行情然后pub k线数据

type KlinesPub struct {

    //TODO

}

func (s \*KlinesPub) OnTick(event Tick) {}

func (s \*KlinesPub) OnTrade(event Trade) {}

## K线结构

type KlineJson struct {

    Symbol        string  `json:"symbol"`

    OpenTime      int64   `json:"open\_time"`

    Open          float64 `json:"open"`

    High          float64 `json:"high"`

    Low           float64 `json:"low"`

    Close         float64 `json:"close"`

    Twap          float64 `json:"twap"`

    SellCount     int64   `json:"sell\_count"`

    SellSize      float64 `json:"sell\_size"`

    BuyCount      int64   `json:"buy\_count"`

    BuySize       float64 `json:"buy\_size"`

}

type KlinesMessage struct {

    Klines map[string]KlineJson `json:"klines"` //key: symbol

}

## 需求

1. 一次pub 需要pub所有symbol 的k线
2. k线开始的时间戳是本分钟开始的
3. 设置一个500ms的延时：下一分钟过了500ms之后再推送上分钟的k线
4. twap 3s一个点，计算本分钟的价格的twap
5. k线中的o/h/l/c 的价格是Tick中的 (BestBidPrice + BestAskPrice)/2
6. k线的open为上一根k线的close
7. SellCount/BuyCount 是来自Trade里面的TradeCount累加
8. SellSize/BuySize 是来自Trade 里面的size的累加
9. 使用zeromq 进行pub/sub的通信
10. 使用json 进行序列化
11. 程序启动后的第一根k线需要在pub端扔掉，不能pub。因为不完整
12. 添加合适的UT