1. 实现功能
2. 接收行情数据，经过计算后触发策略，发送数据至下单服务器
3. 将触发时策略信息保存至Redis
4. 具体实现
5. 实现方式：
6. 使用Netty与行情服务器进行通信，接收到数据后由Disruptor队列进行转发
7. 开启多个消费者，每个消费者循环计算所拥有的策略
8. 详细实现过程
9. ConcurrentHashMap----stratagy

存放策略信息，包含在EventHandler内部

1. EventHandler

队列消费者

1. ConcurrentHashMap-----EventHandler

存放EventHandler，创建两个ConcurrentHashMap，一个存放所有的消费者AllEventConcurrentHashMap，一个存放当前有剩余空间的消费者AvailableEventConcurrentHashMap

* 系统启动后，先创建一个消费者EventHandler，将该消费者保存至AllEventConcurrentHashMap和AvailableEventConcurrentHashMap中，同时初始化该消费者内部的StratagyConcurrentHashMap；
* 当有用户创建策略时，将该策略信息保存至Redis
* 当用户开启一个策略时，将该策略从Redis中复制一份至StratagyConcurrentHashMap，同时更新该策略状态为运行中
* 当用户停止一个策略时，将该策略从StratagyConcurrentHashMap/AvailableEventConcurrentHashMap中移除，同时更新该策略的状态并保存至Redis；同时将该消费者添加至可用消费者容器（AvailableEventConcurrentHashMap）中
* 当一个消费者中的StratagyConcurrentHashMap达到预先设定的容量时，将该StratagyConcurrentHashMap所在的消费者从AvailableEventConcurrentHashMap中移除，同时开启一个新的消费者，并将该消费者加入到StratagyConcurrentHashMap和AvailableEventConcurrentHashMap中
* 当开启的消费者数量大于预先设定的值时，提示系统过载，无法运行新的策略
* 预设最大消费者数量：10；消费者内最大策略数：20
* 策略触发后，将数据添加至全局的队列ConcurrentLinkedQueue中
* 开启单独的线程，将ConcurrentLinkedQueue中的数据通过Netty推送至下单服务器

1. 具体类

(1)策略类Stratagy

private String id; //策略ID，取UUID

private String name; //策略名称

private String userId;//策略所属人ID

private Contract active; //主动腿合约

private Contract unactive1;//被动腿合约1

private Contract unactive2;//被动腿合约2

private Contract unactive3;//被动腿合约3

private String status;

private String formula;//策略公式

1. 合约类Contract

private String exchangeNo;// 交易所

private String exchangeName;// 交易所名

private String commodityNo;// 商品代码

private String commodityName;// 商品名

private String code;// 完整的期货合约代码

private String contractNo;// 合约代码（YYMM）

private String contractName;// 合约名称

private String futuresType;// 商品类型

private Double productDot;// 每点价值

private Double upperTick;// 跳点

private String regDate;// 更新日期

private String expiryDate;// 最后交易日

private Integer dotNum; // 小数点位数

private String currencyNo;// 货币

private String currencyName;// 货币名

private Integer lowerTick;// 进阶单位

private String exchangeNo2;// 备用交易所

private Double deposit;// 保证金

private Double depositPercent;// 保证金百分比

private String firstNoticeDay;// 首次通知日

private String commodityType;// 内外盘标志

private String pyName;// 拼音名字

private String delFlag;//逻辑删除标记

private String lastTradeDay; //最后交易日(过期日)

1. 下单类Order

private String accountId;//下单账户ID

1. 下单请求类OrderReq

private Order active;

private Order unactive1;

private Order unactive2;

private Order unactive3;

1. 策略与用户关系
2. 策略与Web用户绑定
3. Web用户与下单用户绑定
4. 策略中合约与下单用户绑定

* 实现方式：

用户在创建策略时，将每个合约与下单账户进行绑定，如果没有进行绑定，则随机选取可用账户进行下单

附：

1、策略运行的方式

第一种：

（1）当用户新开始一个策略时，先判断是否有可用消费者，如果有，则将该策略添加至任意的一个消费者中；

（2）如果没有可用消费者，则判断是否可以创建新的消费者；

（3）如果可以创建，则创建一个新的消费者，并将该策略添加至该新建的消费者中

第二种：

1. 当用户新开始一个策略时，先判断是否可以创建新的消费者，如果可以，则新建一个消费者并添加该策略
2. 如果不可以创建新的消费者，则判断是否有可用的消费者

注：目前采用第二种方式

1. 策略接收行情的方式

第一种：

1. 策略运行者作为消费者接收全部的行情，然后根据策略的合约选择相应的行情进 行计算。
2. 该方式的缺点是如果策略计算时间过长，后续的行情来的太快，等到下一次计算 时间间隔比较大。

第二种：

1. 策略运行者作为消费者接收全部的行情，然后根据该消费者进行合约行情的筛选， 将满足条件的行情推送给另一线程进行计算。
2. 该方式可保证策略计算的线程中没有不必要的行情，但是会增加系统的负载，开 启过多的线程。

注：目前采用第二种方式