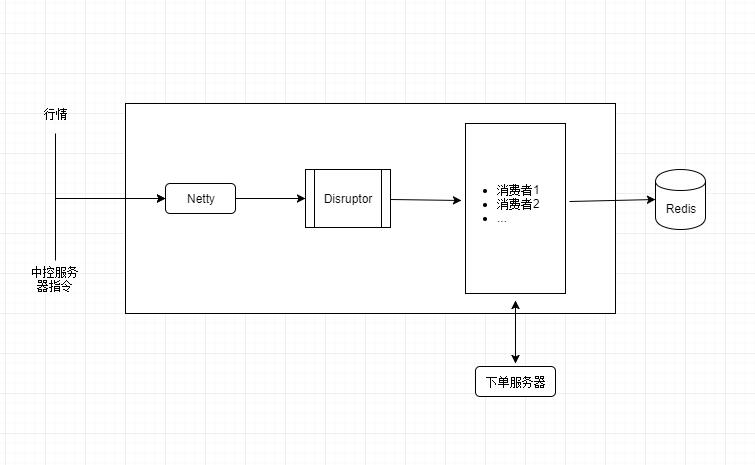
1. 实现功能

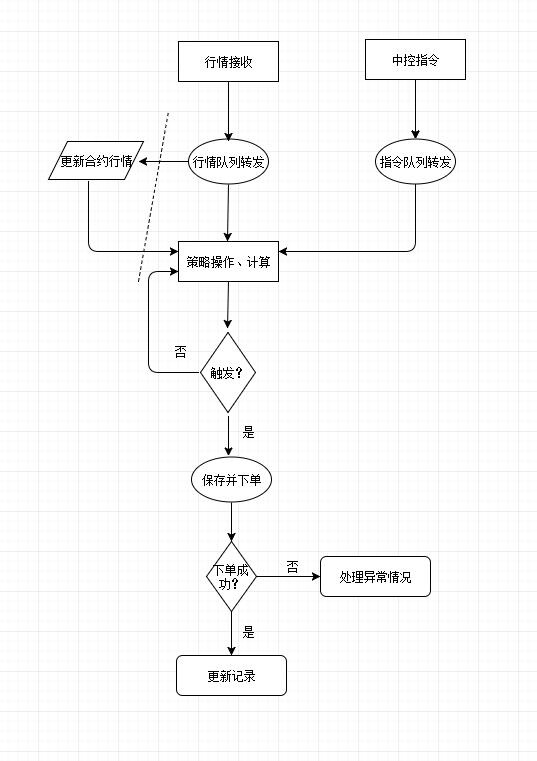
1、接收行情数据，经过计算后触发策略，发送数据至下单服务器

2、将触发时策略信息保存至Redis

1. 具体实现
2. 实现方式：
3. 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
4. 开启多个消费者，每个消费者循环计算所拥有的策略
5. 结构流程图



（图2.1）结构图



（图2.2）流程图

1. 详细实现过程
2. 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. 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~~。

第三种：

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，

更新内存中该合约的行情，同时发送一条消息给另一个处理策略的单独线程；

1. 处理策略的线程只取内存中需要的合约行情；