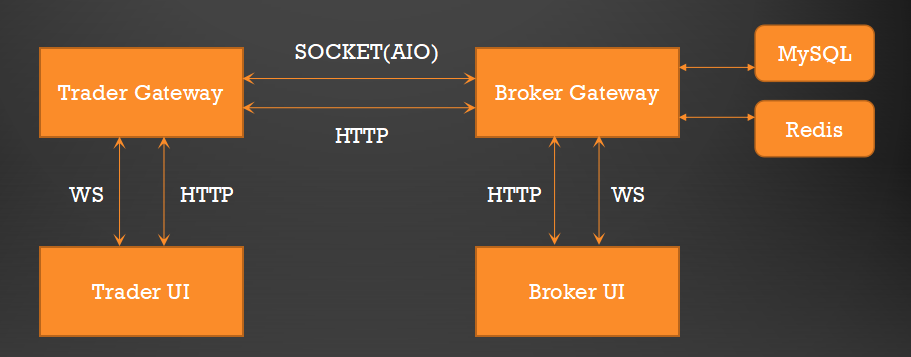
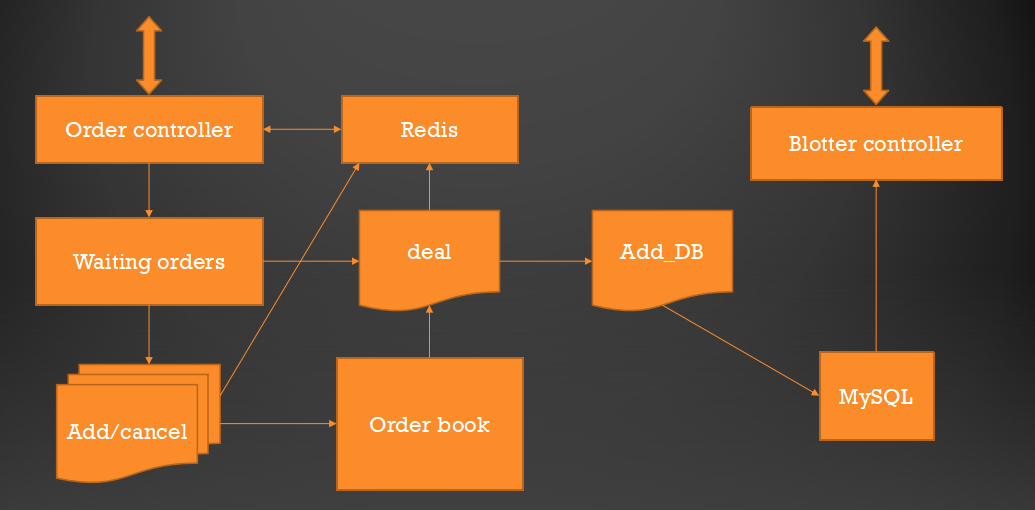
OTC-Trading System需求和设计文档

1. **需求分析**
2. 架构——分布式架构
   1. Broker端与trader端分离
   2. Broker端与trader端多对多连接，可分开部署在不同机器上独立工作，不同broker具有不同市场，各市场独立运作，充分竞争。Broker可对不同的trader提供不同权限，trader需要身份认证。
   3. 主要数据（例如订单）需要在至少一处实例中进行持久化，并保证最终一致性，例如以接收order的broker端做作为数据分布存储的master端。
3. 功能需求
   1. 核心业务：每个市场对应一个broker和一种期货，多个trader可在不同市场中提交四种类型的订单（market市价单、limit限价单、stop止损单、cancel取消单），进行实时交易，即订单一旦满足交易条件则立即交易；小时间尺度上（涉及机器运行速度），遵循“价格优先，时间其次”的准则，即当交易进行中，若市场改变，产生了更优交易情况，需要以最优价格进行后续成交。
   2. 市场信息查询：连接的trader可以实时获取到当前市场最新深度（depth），获取自己发送订单的实时状态（state），发送订单所产生的完整交易信息与市场的部分交易信息（无法查看与自己无关的交易双方信息）（blotter）。
   3. 各部分业务功能：
      1. Trader UI
         1. 查看实时市场深度
         2. 查看订单与交易记录
         3. 产生四种类型的订单（market、limit、stop、cancel）
      2. Trader gateway
         1. 连接broker gateway
         2. 接受trader UI连接
         3. 订阅不同期货的最新市场深度
         4. 发送订单到broker端
         5. 维护不同交易员所产生的所有订单
         6. 提供冰山策略
      3. Broker gateway
         1. 接受trader gateway连接
         2. 接受broker UI连接
         3. 维护接收到的不同公司的订单并撮合交易，并能正确处理并发收到的订单
         4. 维护市场运行状态，包括但不限于市场深度
         5. 推送所产生的交易和市场深度变化
      4. Broker UI
         1. 连接到broker gateway
         2. 查看当前市场
         3. 查看进行中的和交易完成的所有订单，支持筛选和分组
4. **系统设计**
5. **系统通信架构**



如上图，其中trader gateway和broker gateway中既包含通信模块，又包含业务处理模块。其中gateway之间通信既有socket通信做市场深度等信息的实时更新，又有rest api的调用。Gateway和UI之间通过websocket实现消息推送。

1. **Broker后端设计**

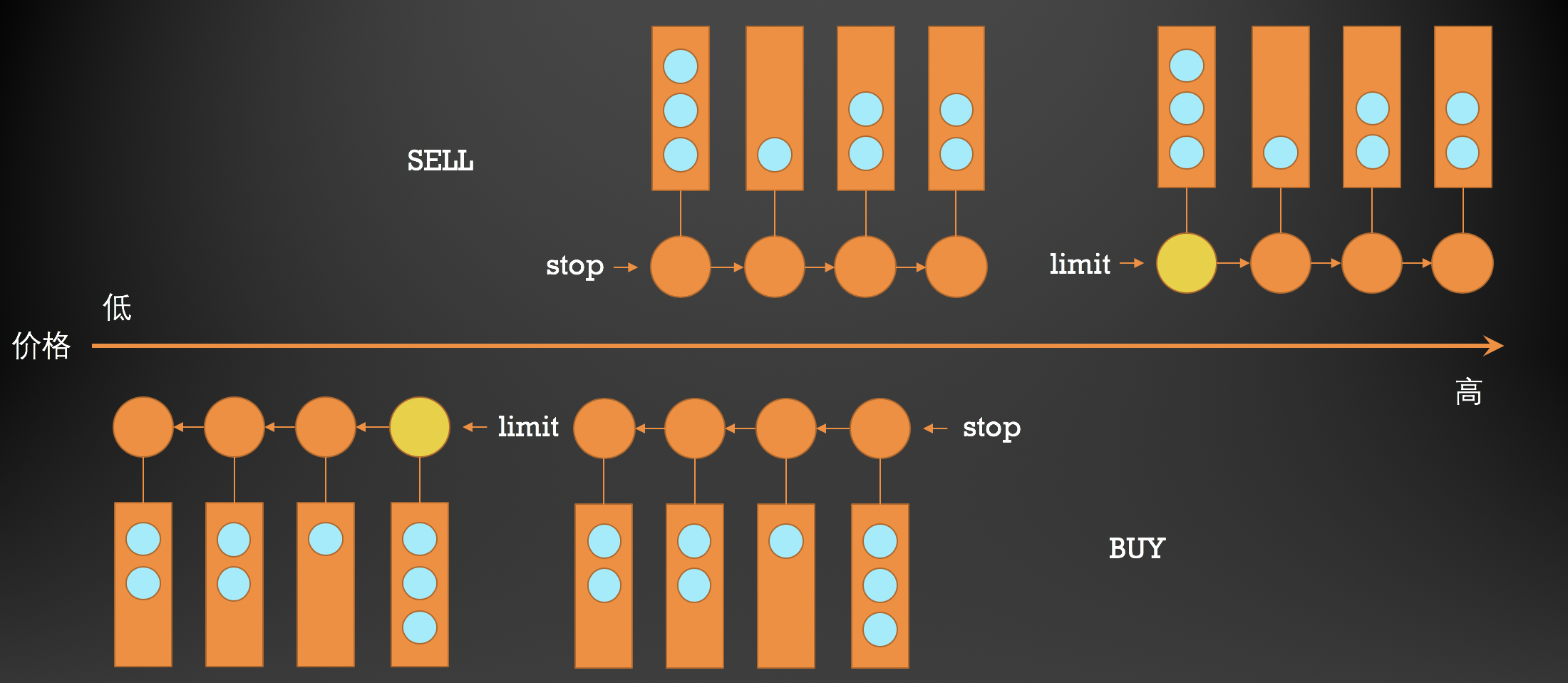


broker端的整体结构如上图所示。

1. order book模块

order book由price node list和 waiting orders两个数据结构组成，并且执行了deal，add和cancel等处理线程。

* + 1. Price Node List

****

上图表示Order Book中两条price node list的结构，其中一个橙色圆形代表一个PriceNode，一个PriceNode代表的即是OrderBook中的某一级深度的价格，维护的属性为Price、OrdernodeList和下一级price node的引用，PriceNode从前指向后的顺序为价格由优到劣（具体体现为买方订单链价格由高指向低，卖方订单链价格由低指向高）。每一条PriceNodeList 维护两个Pricenode的引用（分别表示limit order和stop order链），sellOrBuy属性（标识买卖方）和另一条PricenodeList的引用（为实现激活stop order功能）。

PricenodeList类主要实现的业务逻辑为添加订单、移除订单、取消订单、激活止损单、获取当前深度和获取当前交易候选订单。

**A)添加订单**：根据订单类型（stop或limit）选择对应的Pricenode链，从第一个非空的头结点开始遍历，若遍历过程中要插入订单的价格位于前后Pricenode的价格之间，则创建新的Pricenode，插入订单（若在头结点之前插入，则广播更新后深度）；若同价格Pricenode已存在，则直接插入对应Pricenode中OrdernodeList的末尾，返回值为Boolean类型。

**B)取消订单**：先遍历stop order链，再遍历limit order链（防止遍历过程中要取消的订单转变为limit），遇到对应取消单价格的Pricenode，则调用其中OrdernodeList的cancelOrder方法，返回值为Order类型（记录成功cancel的手数）。

**C)移除订单**：逻辑与取消订单类似，不过只需遍历limit order链，因为此方法只有交易线程会进行调用，返回值为Boolean类型。

**D)激活止损单**：当取消订单和移除订单方法造成市场深度劣化时，调用另一PricenodeList的checkStop方法，参数为变化前深度和变化后深度，将另一PricenodeList中在此深度区间内的stopOrder激活为limitOrder。

**E)获取当前深度**：返回当前PricenodeList头结点的Price值

**F)获取当前交易候选订单**：返回当前PricenodeList头结点中OrdernodeList中的第一个订单

* + 1. Order Node List

Order Node List是一个price node的组成部分，基本不涉及业务处理逻辑。其实现为一个线程安全的order node链表，每个order node中存储一个order。Order node和order可以分别获取可重入锁。Order node 提供基本的并发安全的isempty，add，remove等操作。

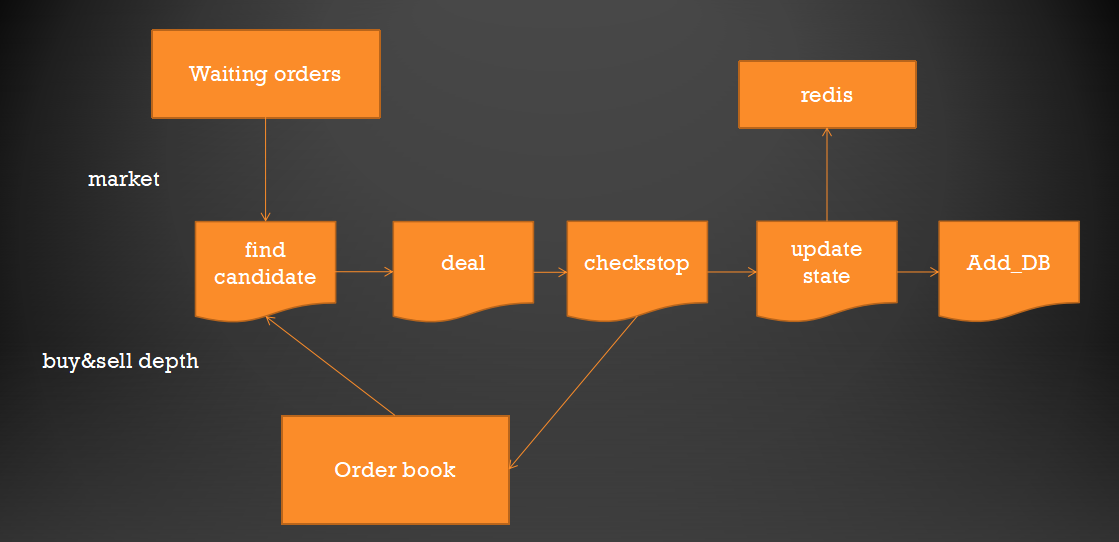
* + 1. Waiting Orders

充当了一个消息中间件的作用，controller模块接受的订单会存放到对应的队列里，orderbook的deal，add，cancel线程会不断的从这些队列里拿出订单来处理。由5条包含不同种类的order 队列组成，分别为：

优先级阻塞队列存放buylimit，优先级阻塞队列buylimit，阻塞队列stop，阻塞队列cancel，非阻塞并发队列 market。

实现了并发安全的get，add和remove方法。

* + 1. Deal处理线程



Deal线程处理交易业务，每次交易，先从buy sell两条price node list中分别取出头节点（即市场深度）的第一个order，并且从waiting orders的market队列中取第一个market order。若两个limit order不能交易，则根据买卖类型交易market和limi；如果limit之间能交易，则market和对应方的limit比较时间，时间早的一方来交易。交易后，更新订单数量， 移除数量剩余0的订单，此时市场深度可能改变，触发checkstop操作，检查是否有止损单被激活，生成成交单插入mysq中，并在redis更新订单状态。

* + 1. Add/Cancel处理线程

Add线程指add buy limit，add sell limit， add stop三个并发线程，每个线程都会从waiting orders 数据结构中，取出对应的阻塞队列里的order，并调用对应price node list 的add 方法插入order。

同理，cancel线程也会从对应的cancel队列里取出cancel order，然后调用price node list 的取消订单方法。

1. Mysql模块

Mysql模块由IDGenerator 和 AddHistory线程两部分组成

* + 1. ID generator

为每个完成交易的transaction生成id的模块。实现上， 使用了Twitter的snow flake算法。ID由3部分组成：毫秒时间戳，期货id，三位序列号（000-999）。其中，序列号是为了将一个毫秒内并发的订单id区别开来，每种期货都对应一个id generator，每次生成id时序列号都原子性的加1并mod1000，这样对于每种期货都能保证一毫秒内区分出1000个并发transaction。

* + 1. AddHistory

Addhistory维护一个阻塞队列， 存放交易完成后的transaction，并提供单独的一个将完成的transaction存放到mysql数据库的一个线程。这样设计原因是，deal线程中成功交易后异步调用addhistory模块，这样插入数据库的性能不会影响到交易的性能。

1. Redis 模块

使用spring的 redis template组件来存放订单的状态信息。不同订单的状态如下：

Limit order：waiting，remain：xxx， canceled，done

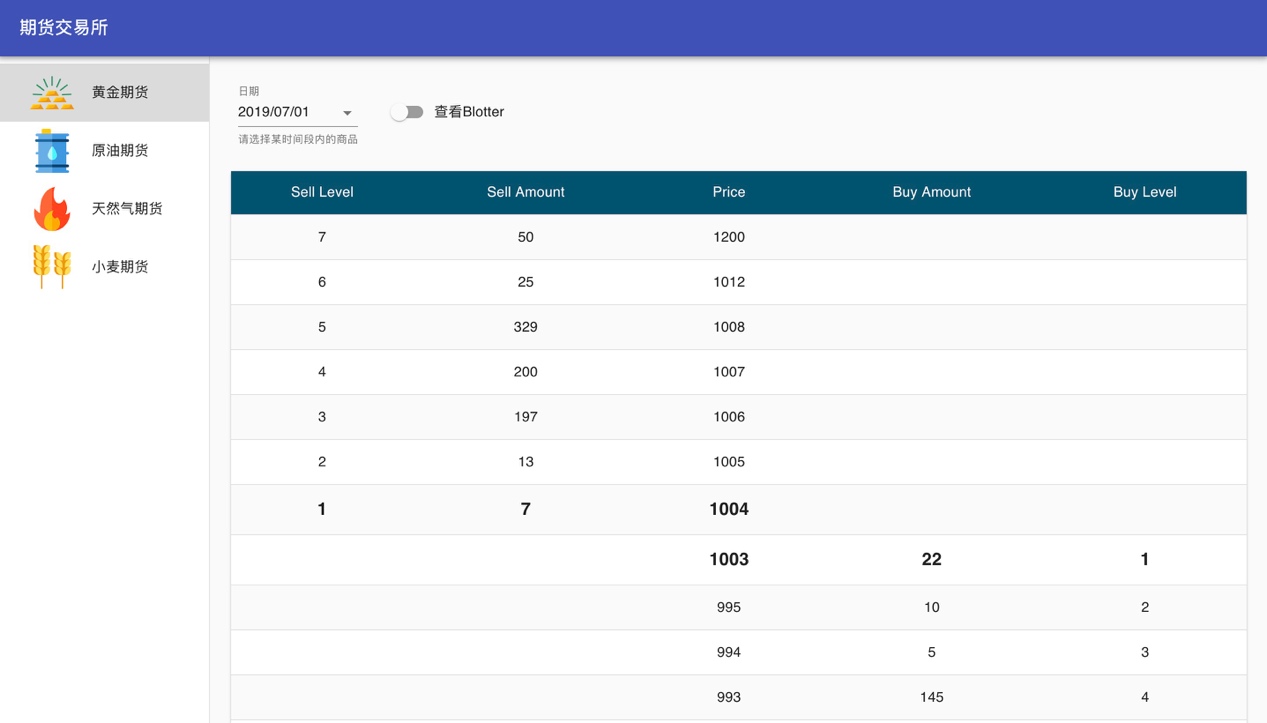
Stop：waiting，active，remain：xxx，canceled，done

Market： waiting， remain：xxx，done

Cancel：waiting，success，fail

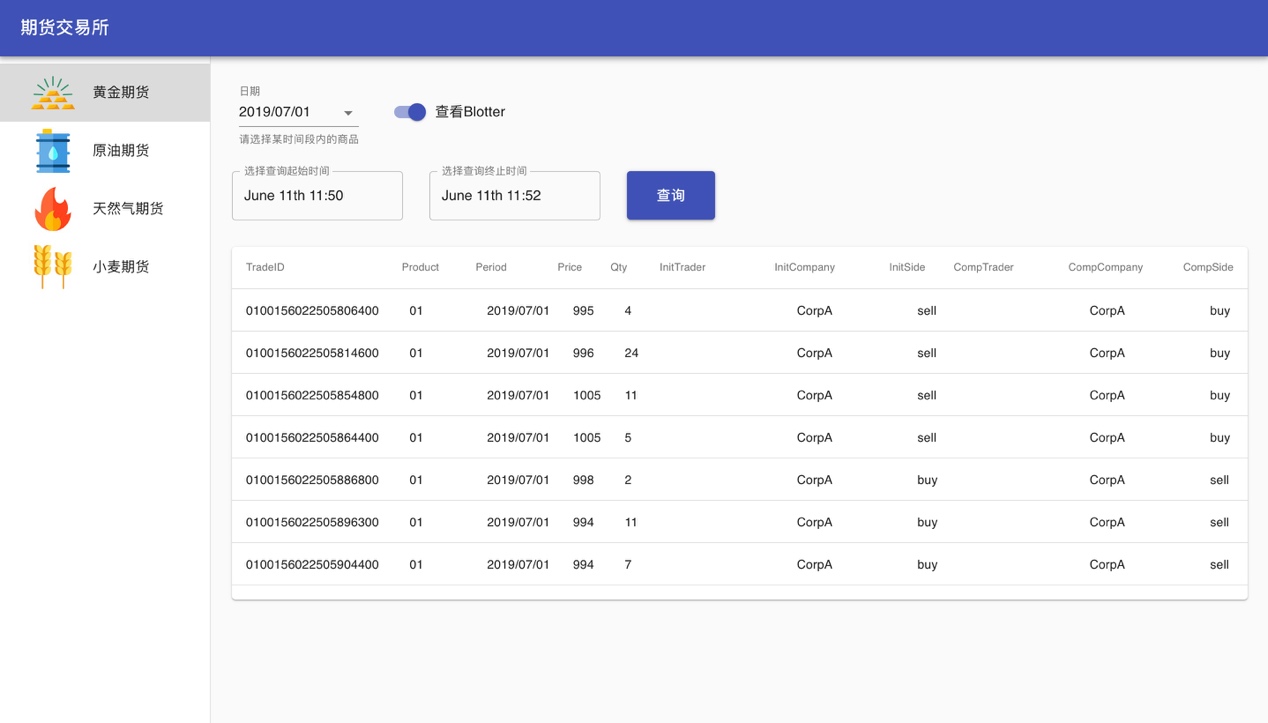
在deal，add，checkstop等线程中都要更新订单状态。

1. **Broker前端设计**
   1. Order book界面



上图为Broker UI主界面，左侧和上方选择框和Trader UI大体相似，下方表格模拟运行在后端系统中的OrderBook的实时结构，点开深度（即price）可以查看当前等待交易的具体订单信息（剩余手数，公司）

* 1. Blotter界面

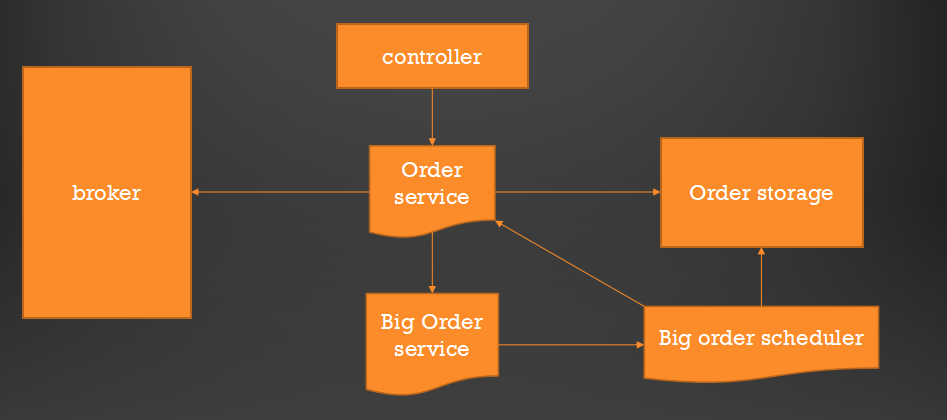


上图为Broker端的查看Blotter界面，和Trader UI的查看Blotter界面几乎一样，去除了“只看我的订单”按钮。

**4.** **Broker 前后端通信**

通过WebSocket进行长连接，每个连接session对应于一种期货，使用map结构存储。Key为product，value为查看该期货的session集合。在web前端启动时，提供查看的期货目标，从而将连接session加入此数据结构，用以对期货信息（order book）进行对应的推送。

1. **Trader gateway后端设计**



Trader gateway后台的结构如上图所示。

* 1. Order Storage

用于维护整个gateway下向broker所发的所有订单信息，并保存其状态，根据broker，交易员，期货对订单进行划分，并提供一些常用接口，包括（broker，交易员，期货）的条件筛选。

* 1. 活跃订单状态更新（Order State Service）

本项目中，活跃订单状态不通过socket连接进行主动推送，而是通过rest API进行查询，因此采用拉模式。Order state service主要启动一个线程进行周期性的订单状态更新。根据前端的连接情况，取出所有用户正在查看的期货id，主动向broker端查询这些期货中本公司所发订单的状态；得到结果后，对订单状态进行更新，并将其推给前端展示更新后状态，也包括删除已结束的订单，触发cancel信息的推送等。

* 1. Order Service

使用spring框架下的restTemplate模块，用来调用Broker的对应的order的rest api，实现向broker发送对应market、limit、stop、cancel订单以及查询blotter的功能。在发送订单后，收到broker端返回的id后，生成新的order存放到Order storage中。

* 1. Brokers 模块

用于Trader和broker之间的socket通信。实现为一个单例模式的broker池服务，设定待连接的broker范围，同时维持其对应的连接地址信息。每个broker中还另外绑定一个含有socket连接的broker Channel对象，用于对连接和订阅的代码进行封装，保存连接中获取的UUID身份标识。同时，由于不同broker拥有不同的市场，该对象中还应负责维持各期货的市场深度信息，即（Product，深度）的key-value对，深度为一对数字。

* 1. Big Order Service

实现冰山策略的service，支持TWAP和VWAP两种策略，并能开启scheduler线程，调用Order Service不断的将拆分出来的小份订单发送到broker上

* + 1. TWAP

根据历史数据来预测设置的total time时间内的总成交量，然后根据设置的frequency，每隔一段时间发送固定比例的固定数量的订单，价格为相应的历史时间段的平均价格。

* + 1. VWAP

根据历史数据来预测设置的total time时间内的总成交量，然后根据设置的frequency，统计每个frequency的时间段内历史交易量占总历史交易量的比例，每隔一段时间发送对应比例的变化的数量的订单，价格为相应的历史时间段的平均价格。

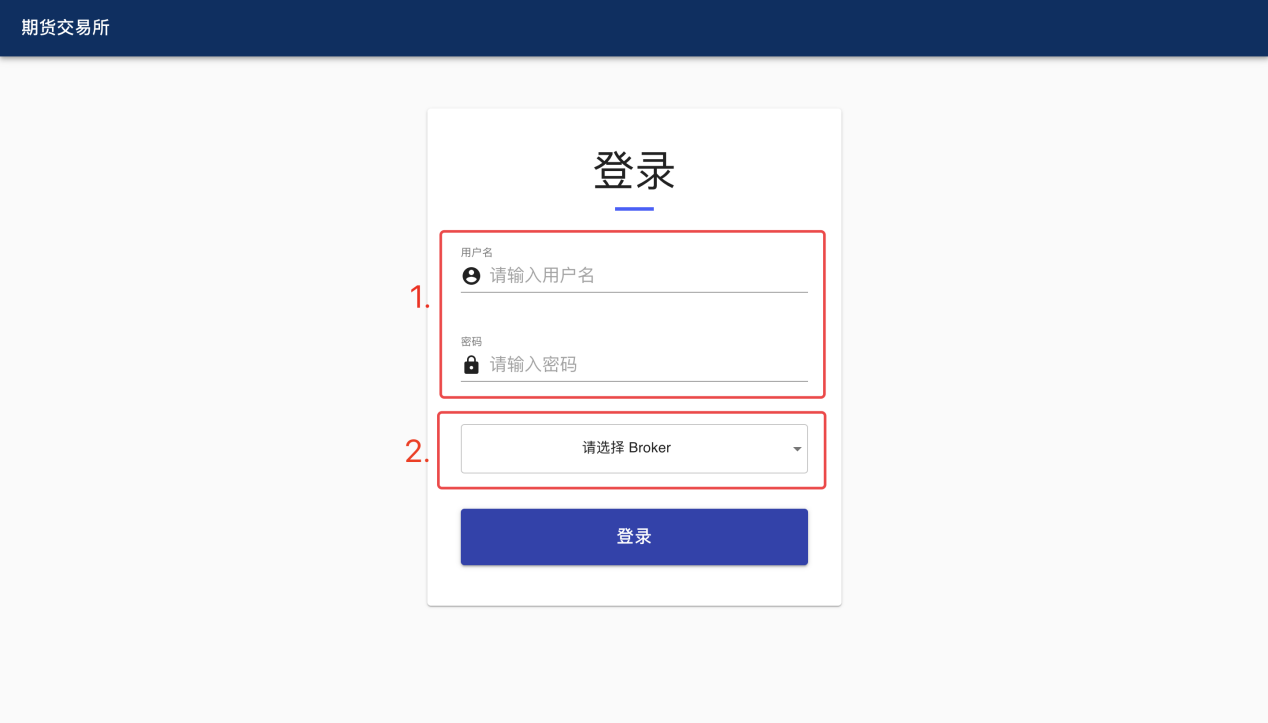
* + 1. Scheduler

根据设定的参数，新启动一个线程，每隔一段时间根据TWAP和VWAP策略的不同，调用order Service发送不同数量的拆分单，将这些拆分单存放到order storage中。并根据cancelflag来决定是否停止scheduler。

* + 1. Cancel Big Order

首先将cancel flag设置为1，从而停止对应的scheduler，然后对于每一个正在进行中的拆分单，发送cancel order到broker上。

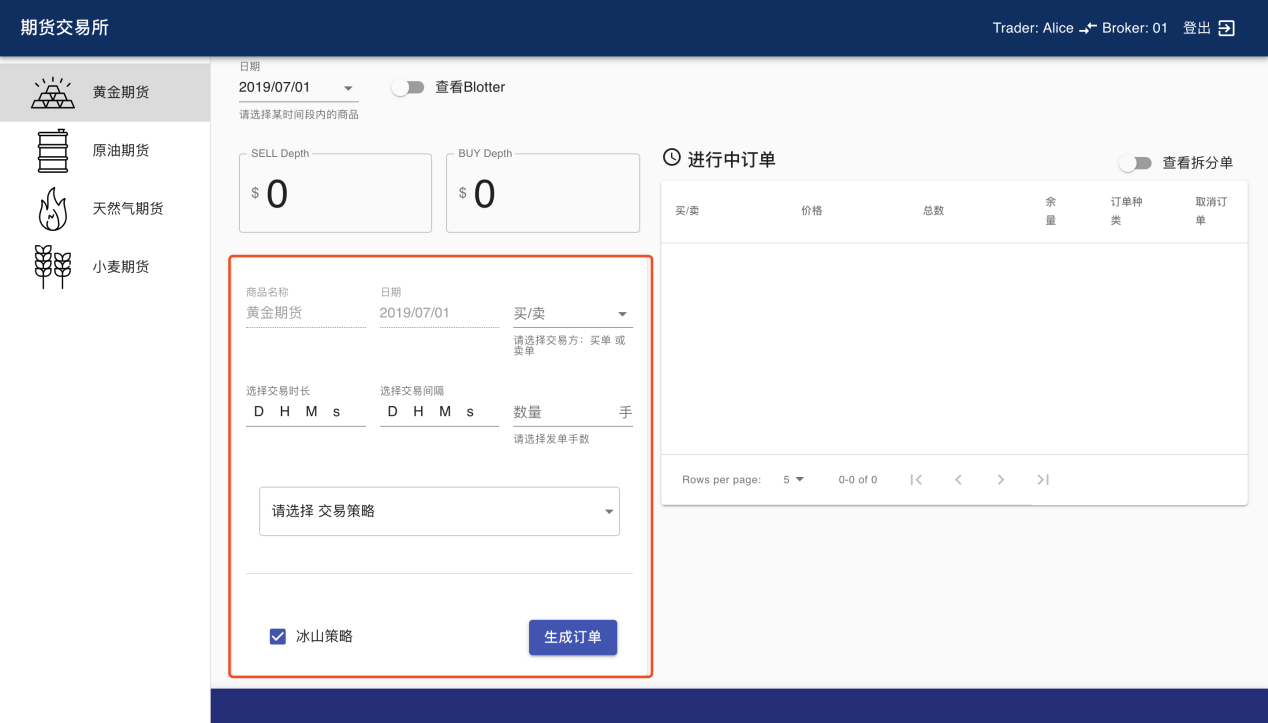
1. **Trader UI模块**



上图是trader UI 的登录界面，输入交易员用户名和密码后（区域1），在下方选择框内（区域2）选择需要连接的Broker ID，选择完毕后点击登录按钮即跳转至交易所主界面。



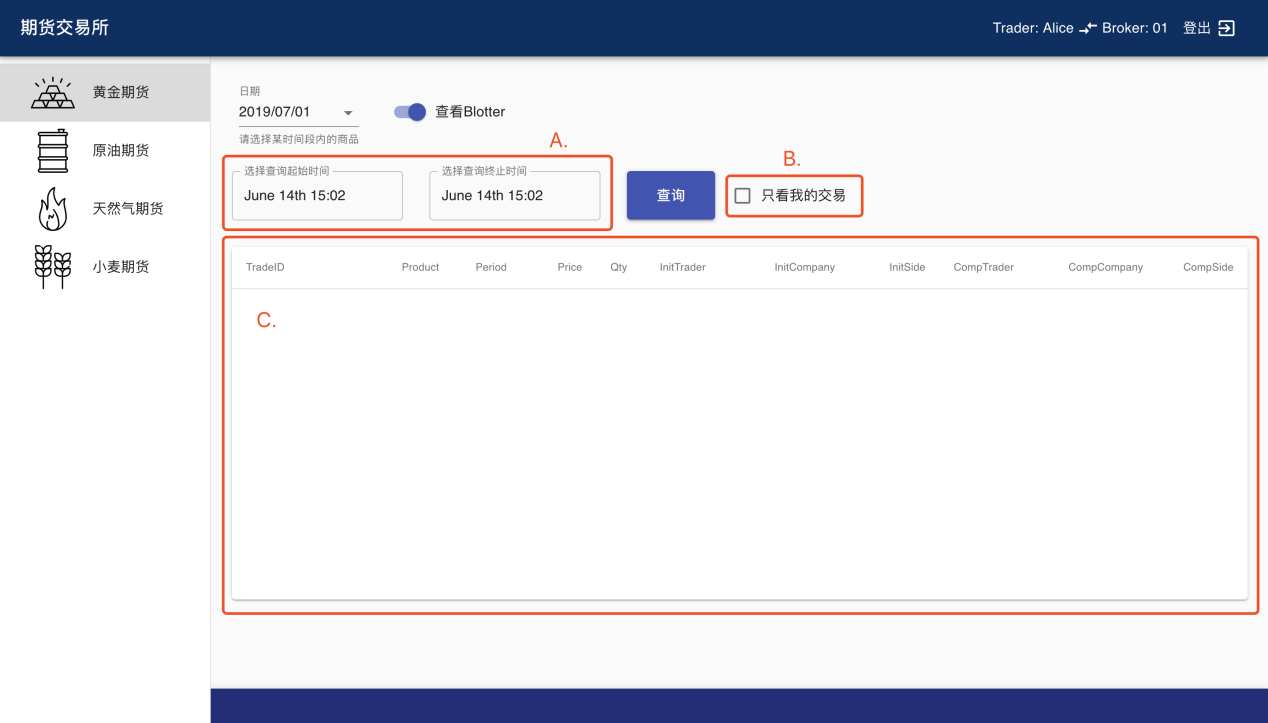
上图为交易所主界面，提供Trader端交易所的主要功能，在区域1内可进行期货种类的切换（此处商品种类指不同品类，如黄金、石油等），区域2提供同品类期货下不同日期的切换，区域1和区域2中的信息共同组成真正意义上的一种期货。区域4中的两个数字显示框代表即时市场深度，从trager gateway接收实时推送市场深度。区域5提供了发单功能，前两个输入框根据区域1和2中的信息自动填充，剩余四个输入框填入的信息分别表示：买方或卖方，出价，手数，订单种类（limit，stop，market）。区域6的选择框，勾选时区域5内变为大单信息填写页面，如下图所示：



区域7表格显示进行中订单，即已发出但尚未被交易的订单，点击表格中一条订单右方的取消订单按钮即可发送对应订单的cancel order。区域8中的按钮勾选后显示拆分单信息，进行中订单只显示普通单和拆分后的小单，未拆分大单的信息（如为拆分手数、未交易手数等）都在左侧“进行中拆分单”表中显示，如下图所示：



区域3的按钮点选后页面切换至查看Blotter界面，如下图所示:



查看Blotter界面中区域A选择查看的时间区间，区域C表格显示查询所得的交易记录，区域B按钮勾选后过滤掉非当前登录交易员参与的交易记录。

1. **Trader 前后端通信**

通过WebSocket进行长连接，每个连接session对应于一个（broker，交易员，期货）对，同样使用map结构存储。Key为product，value为查看该期货的session集合。在web前端启动时，根据用户登录与选择选项，提供broker，该user和查看的期货目标，此后根据这些信息对不同的用户仅推送所需信息。主要有两部分数据：市场深度和订单状态，分别从broker连接对象中存储的深度数据和order storage模块中获得推送内容。

1. **Broker Gateway 和 Trader Gateway之间通信**

为保证深度实时性，使用**长连接**进行推送。采用基于socket的java **AIO**框架，实现异步非阻塞的推送。对于多个trader连入，使用Executors管理线程池，接收连接请求。

**Broker端**从trader端接收连接请求，将维持连接的channel对象存入对应的trader实例中，同时返回UUID作为trader端的标识，用以controller的身份检验。连接之后接收订阅请求，根据不同trader的预设权限，可以对其订阅的期货进行检查，通过后返回成功信息，并将连接加入order book实例的连接集中，开始进行推送。

**Trader端**在启动时，依次连接多个broker，在获取UUID标识后，依次订阅该broker上的期货。将连接、订阅单个期货实现为方法，并通过异步的socket通信，可以实现broker的动态连接和期货的动态订阅。

由于该项目设计中socket部分承担业务较少，**socket消息协议**可以简化设计。一条消息中，使用“:”进行分隔，第一部分为消息类型，标识消息内容，例如company、subscribe、depth等。其余部分为消息内容，根据类型不同进行不同的处理：company后接具体公司名；subscribe后接订阅期货id；depth后接期货id、买卖方、具体市场深度价格，同样用冒号分隔不同部分。

1. **身份验证**
   * 1. **Trader端**

通过帐号密码验证不同的trader

* + 1. **Rest api**

Broker端从trader端接收连接请求，将维持连接的channel对象存入对应的trader实例中，同时返回UUID作为trader端的标识，用以rest api的身份检验。一个trader和一个socket链接通过uuid绑定，通过这个uuid 在rest api中验证trader的身份。

1. **开发总结**

基本实现功能性需求，实现扩展性需求，在安全性、容灾性等方面有所不足

在开发时，我们最先进行Orderbook核心业务逻辑的编码，在完成订单处理逻辑并测试通过后开始着手通信及订单状态相关功能的开发，这样做保证了核心业务逻辑的健壮性和完整性，但同时也使得开发过程中职责分工存在较多的耦合，导致开发效率低下。

。

1. **项目分工**

茅悦田（516030910215）：broker UI，trader UI，order book核心数据处理逻辑，broker前后端通信

顾一辉（516030910206）：broker gateway和trader gateway的socket通信，trader市场深度的管理与前后端通信，trader端订单维护管理模块，order book部分数据处理逻辑

原帅（516030910222）： order book业务处理逻辑，mysql ，redis 模块编写，broker，trader controller层编写，trader 端big order（冰山策略）模块，order Service 模块，gateway间api调用逻辑。