

OTC-Trading System 需求和设计文档

一、需求分析

1. 架构——分布式架构

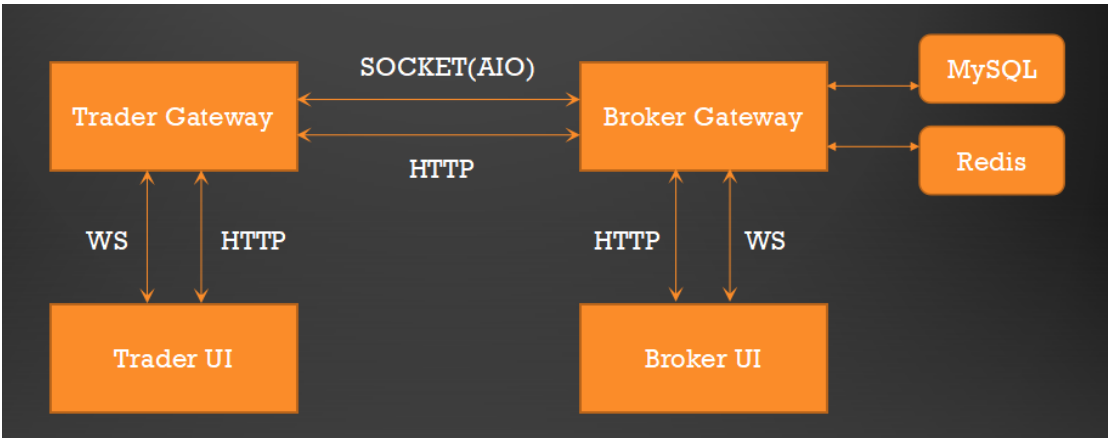
- a) Broker 端与 trader 端分离
- b) Broker 端与 trader 端多对多连接, 可分开部署在不同机器上独立工作, 不同 broker 具有不同市场, 各市场独立运作, 充分竞争。Broker 可对不同的 trader 提供不同权限, trader 需要身份认证。
- c) 主要数据 (例如订单) 需要在至少一处实例中进行持久化, 并保证最终一致性, 例如以接收 order 的 broker 端做作为数据分布存储的 master 端。

2. 功能需求

- a) 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。
- b) 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)。
- c) 各部分业务功能:
 - i. 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)
 - ii. 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. 维护不同交易所产生的所有订单
 - 6. 提供冰山策略
 - iii. 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
 - iv. 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

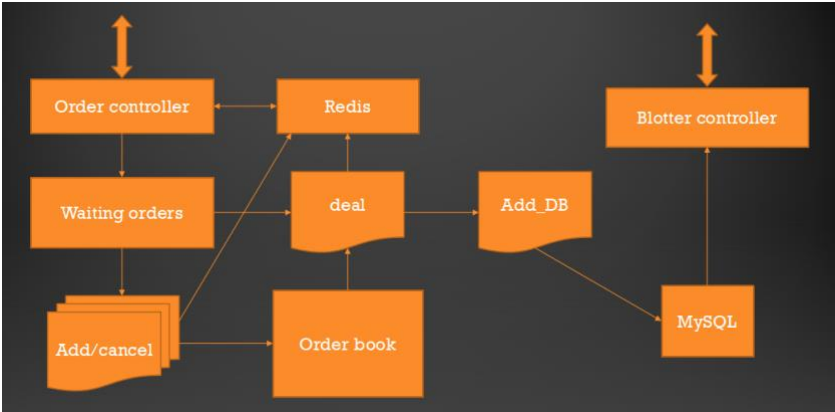
二、 系统设计

1.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。

2.Broker 后端设计

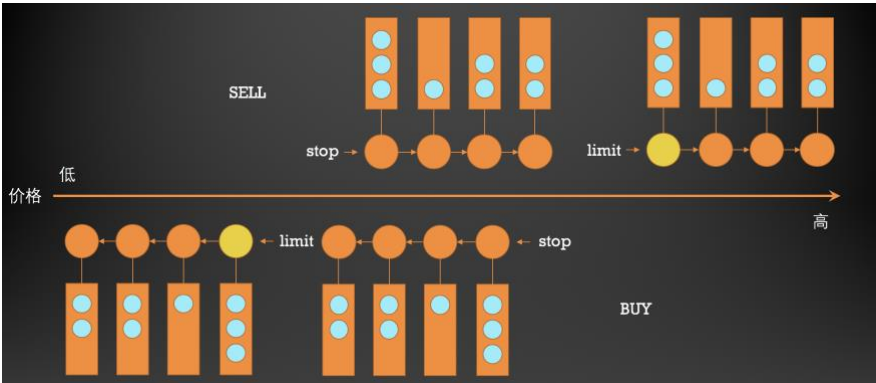


broker 端的整体结构如上图所示。

a) 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。

i. 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

ii. 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。

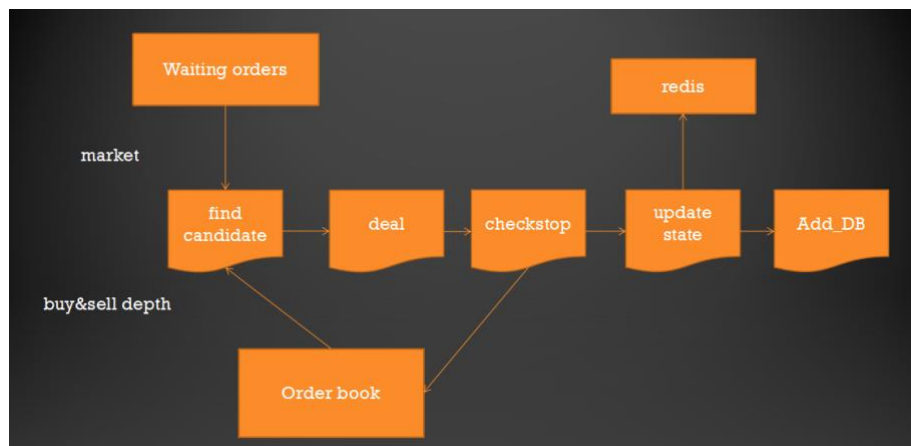
iii. Waiting Orders

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。

iv. 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 limit；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 mysql 中，并在 redis 更新订单状态。

v. 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。

b) Mysql 模块

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

i. 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 snowflake 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。

ii. 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。

c) 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。


3. Broker 前端设计

a) Order book 界面


期货交易所




黄金期货



原油期货



天然气期货



小麦期货

日期

2019/07/01

查看Blotter


请选择某时间段内的商品

Sell Level	Sell Amount	Price	Buy Amount	Buy Level
7	50	1200		
6	25	1012		
5	329	1008		
4	200	1007		
3	197	1006		
2	13	1005		
1	7	1004		
		1003	22	1
		995	10	2
		994	5	3
		993	145	4


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）

b) Blotter 界面


期货交易所




黄金期货



原油期货



天然气期货



小麦期货

日期

2019/07/01

查看Blotter

请选择某时间段内的商品

选择查询起始时间

June 11th 11:50

选择查询终止时间

June 11th 11:52

查询

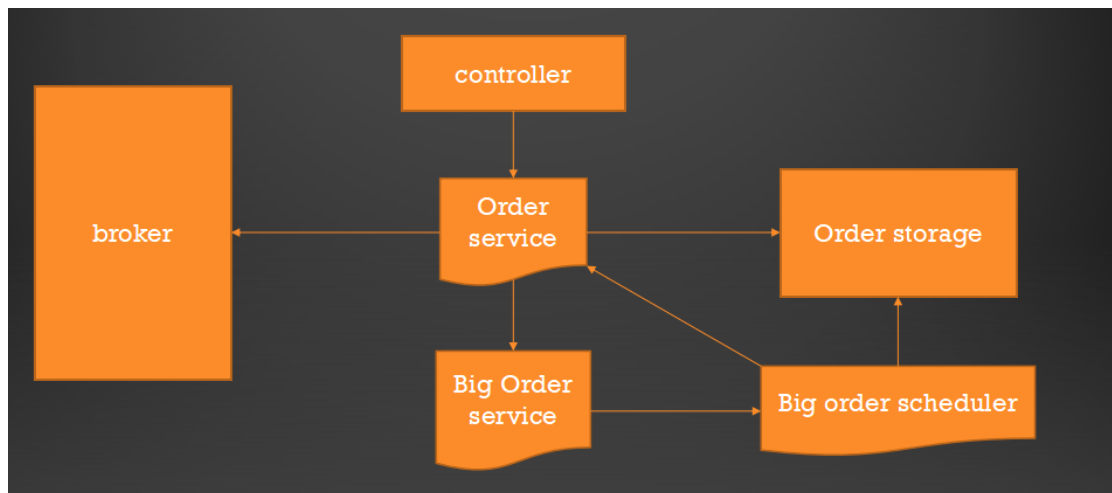
TradeID	Product	Period	Price	Qty	InitTrader	InitCompany	InitSide	CompTrader	CompCompany	CompSide
0100156022505806400	01	2019/07/01	995	4		CorpA	sell		CorpA	buy
0100156022505814600	01	2019/07/01	996	24		CorpA	sell		CorpA	buy
0100156022505854800	01	2019/07/01	1005	11		CorpA	sell		CorpA	buy
0100156022505864400	01	2019/07/01	1005	5		CorpA	sell		CorpA	buy
0100156022505886800	01	2019/07/01	998	2		CorpA	buy		CorpA	sell
0100156022505896300	01	2019/07/01	994	11		CorpA	buy		CorpA	sell
0100156022505904400	01	2019/07/01	994	7		CorpA	buy		CorpA	sell

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。

5.Trader gateway 后端设计



Trader gateway 后台的结构如上图所示。

a) 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。

b) 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。

c) 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。

d) 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。

e) 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

i. 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。

ii. 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数量的订单，价格为相应的历史时间段的平均价格。

iii. 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。

iv. 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。

6. Trader UI 模块



期货交易所

登录

1. 用户名
请输入用户名

密码
请输入密码

2. 请选择 Broker

登录

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。

期货交易所

Trader: Alice Broker: 01 登出

黄金期货

原油期货

天然气期货

小麦期货

日期

2019/07/01

查看Blotter

SELL Depth

\$ 0

BUY Depth

\$ 0

商品名称

黄金期货

日期

2019/07/01

买/卖

请选择交易方: 买单 或 卖单

价格

\$

数量

手

订单种类

请选择 Limit / Stop / Market

冰山策略

生成订单

进行中订单

查看拆分单

买/卖	价格	总数	余量	订单种类	取消订单

Rows per page: 5 0-0 of 0 |< < > >|

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：

期货交易所

Trader: Alice Broker: 01 登出

黄金期货

原油期货

天然气期货

小麦期货

日期

2019/07/01

查看Blotter

SELL Depth

\$ 0

BUY Depth

\$ 0

商品名称

黄金期货

日期

2019/07/01

买/卖

请选择交易方: 买单 或 卖单

选择交易时长

D H M s

选择交易间隔

D H M s

数量

手

请选择发单手数

请选择 交易策略

☒ 冰山策略

生成订单

进行中订单

查看拆分单

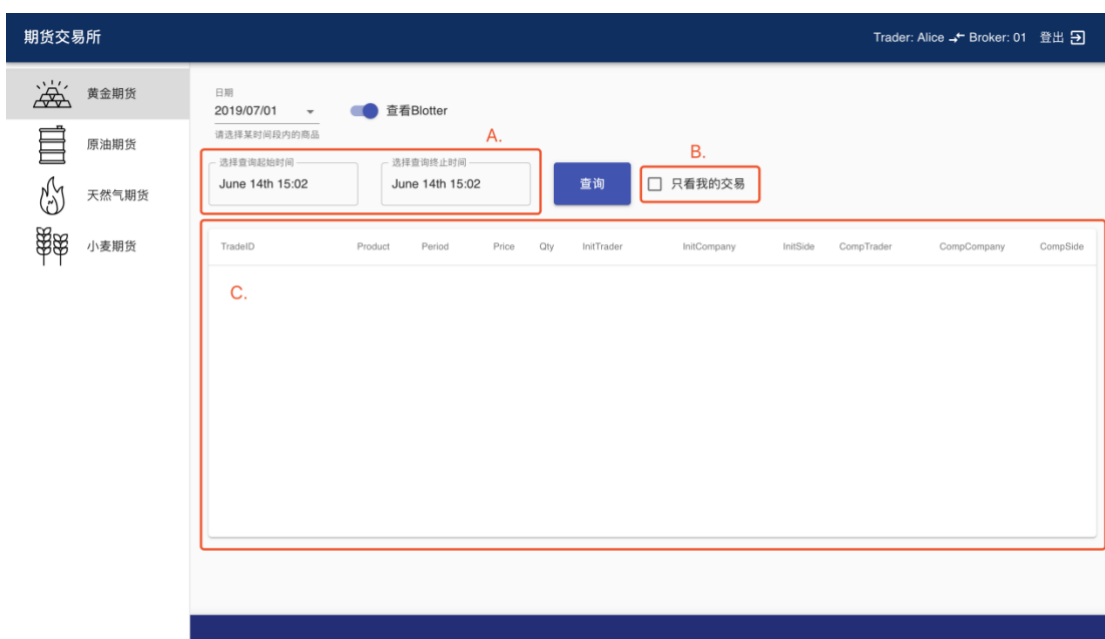
买/卖	价格	总数	余量	订单种类	取消订单

Rows per page: 5 0-0 of 0 |< < > >|

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。

7. 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。

8. 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。

9. 身份验证

i. Trader 端

通过帐号密码验证不同的 trader

ii. 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。

三、 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。

。

四、 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, orderService 模块, gateway 间 api 调用逻辑。