**TradingSystem说明文档**

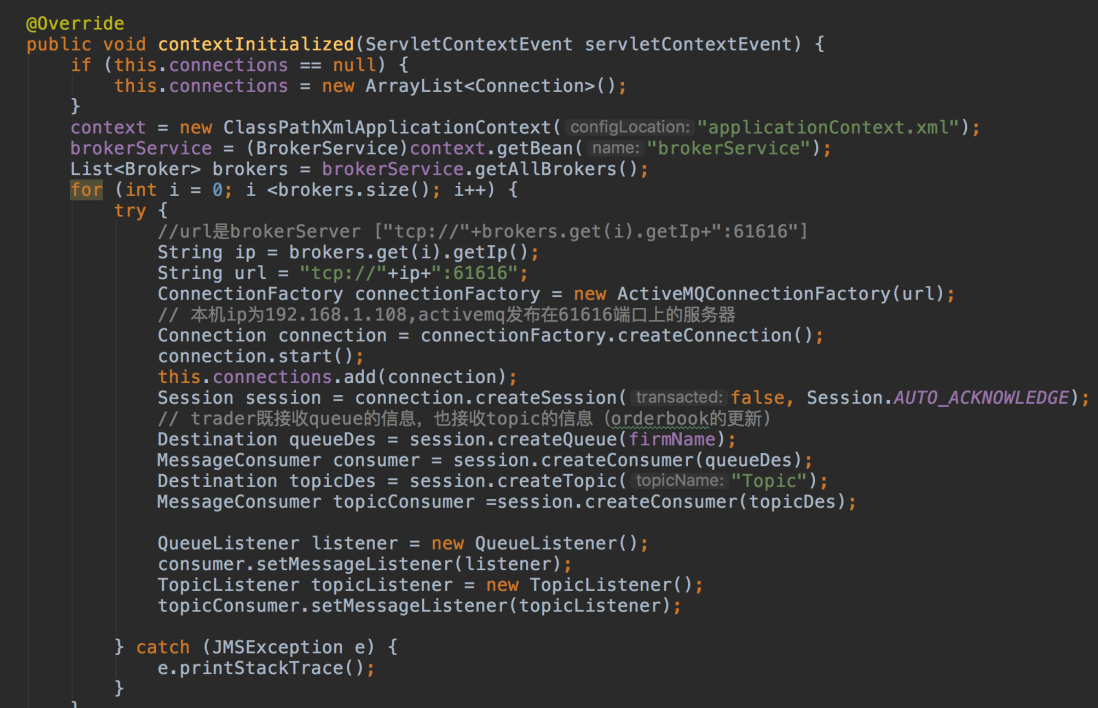
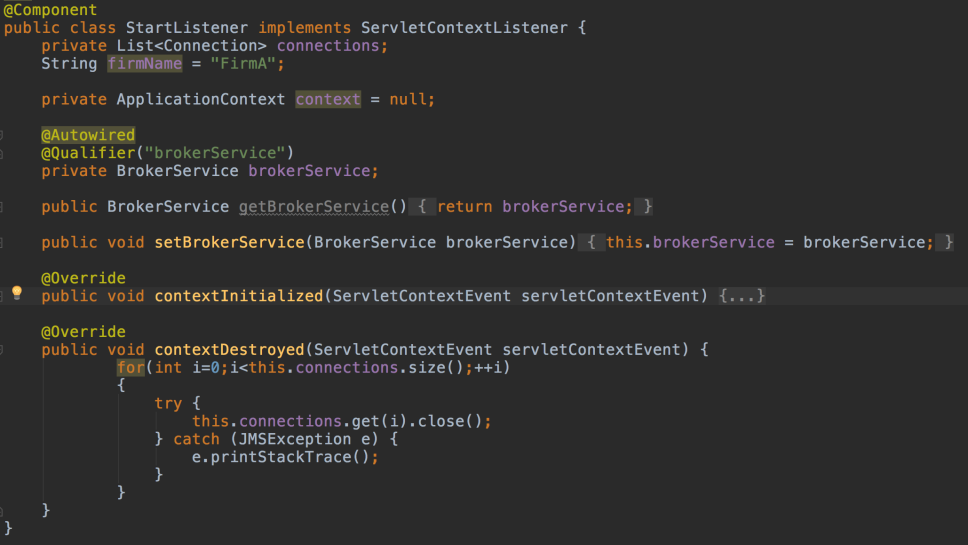
5140379052 楼思琦 5140379026 徐瑾卿 51403790 吴佳雯

**基本功能**

1. **Trader Gateway Server**

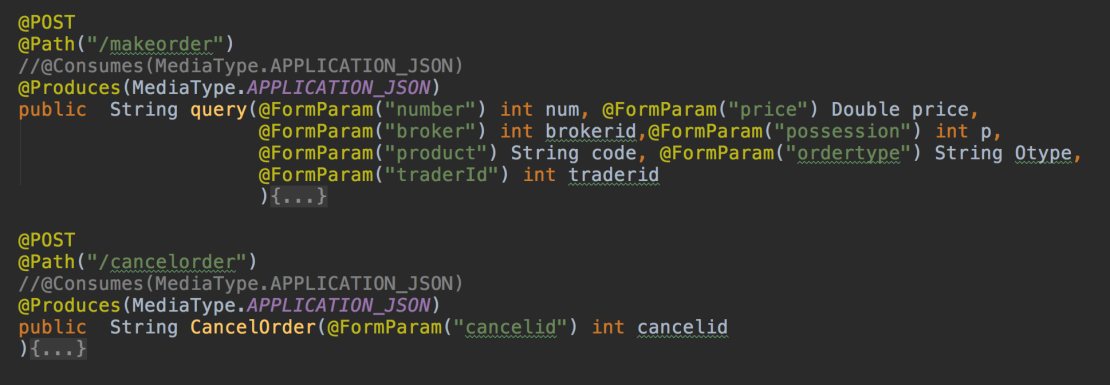
* Connect to brokerServer &Subscribe to market depth of each product

通过JMS对每一个Broker服务器生成一个Connection，注册QueueListener和TopicListener分别P2P操作（下订单、接收订单结果等）和订阅产品orderbook更新。代码中StartListener在服务器开启时最先调用，注册JMS监听器。

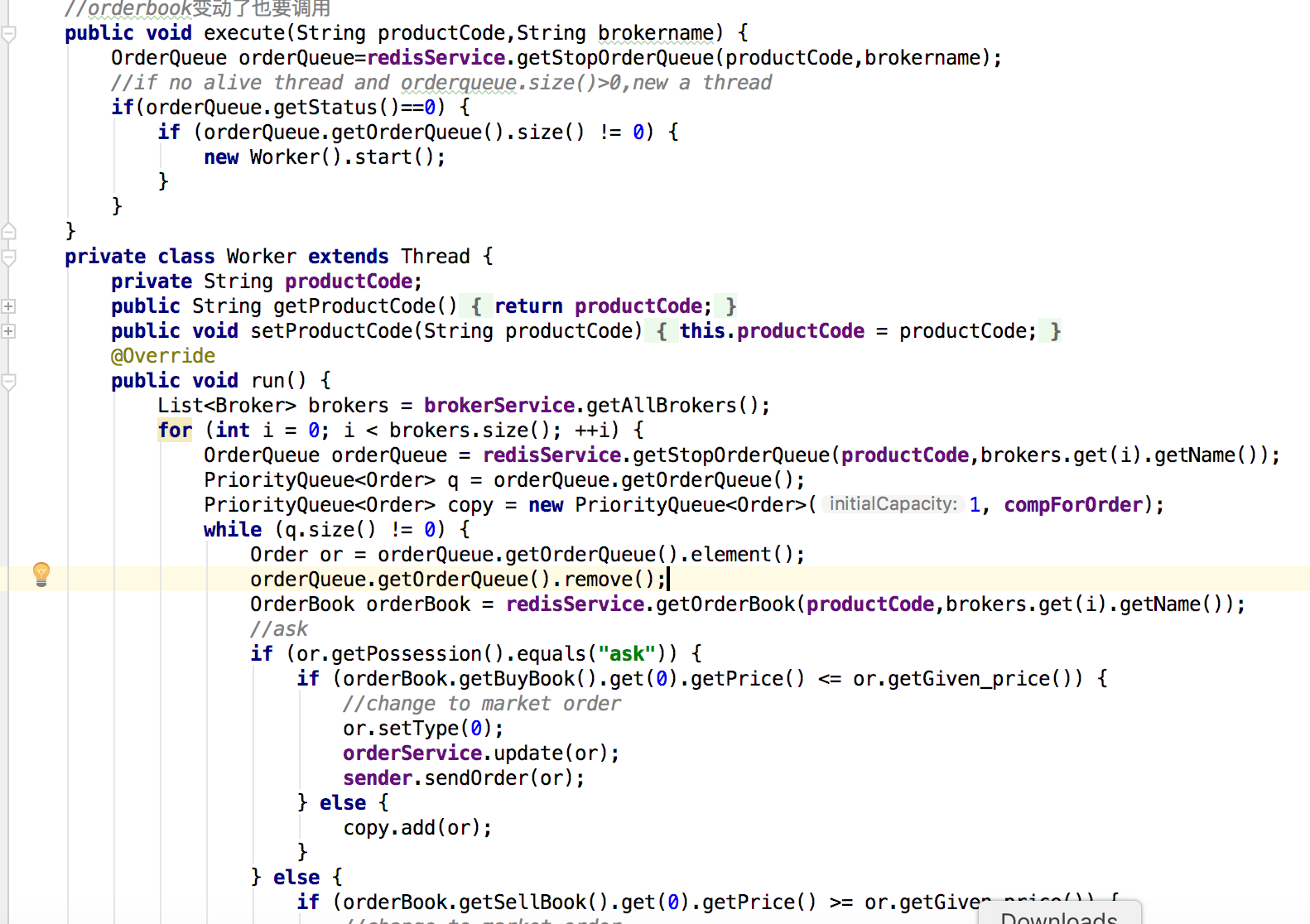
Listener中根据与Broker端指定的属性协议进行数据读取，主要根据Type属性区别不同的数据。当TopicListener收到OrderBook更新时，调用Websocket接口刷新前端UI界面。

* Manager different types of order

通过RESTful和前端UI进行交互，根据接口参数生成相应订单并保存至本地数据库，经过分单算法处理后，通过JMS发给对应的broker进行下单操作。



如果订单类型是Stop Order，那么插入redis中的Stop Order队列，当每次接收到Broker端商品价格更新时检查队列，看是否满足stopOrder设置的价格，如果达到条件则把该订单转换成Market Order发给Broker。



* ICEBerg

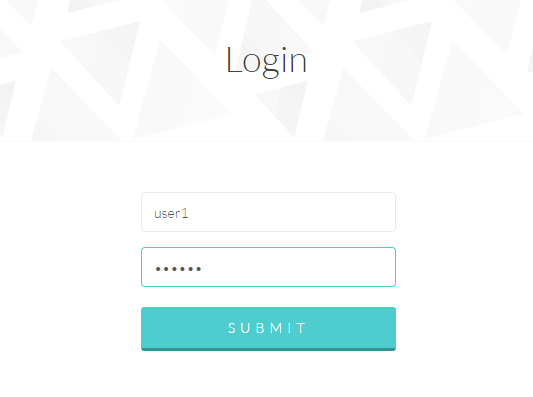
每隔2分钟起一下线程，扫描数据库中数量大于100并且尚未交易完的订单，获得待分单队列。计算过去2分钟内的成交量占总成交量的比值，乘以该订单的number，得出此时应该分单的数量，生成新的market order，并与原订单关联。



1. **TraderUI**

* Login

Trader可以进行登录操作，绑定user id，针对之后的下单、查看数据等进行区分。



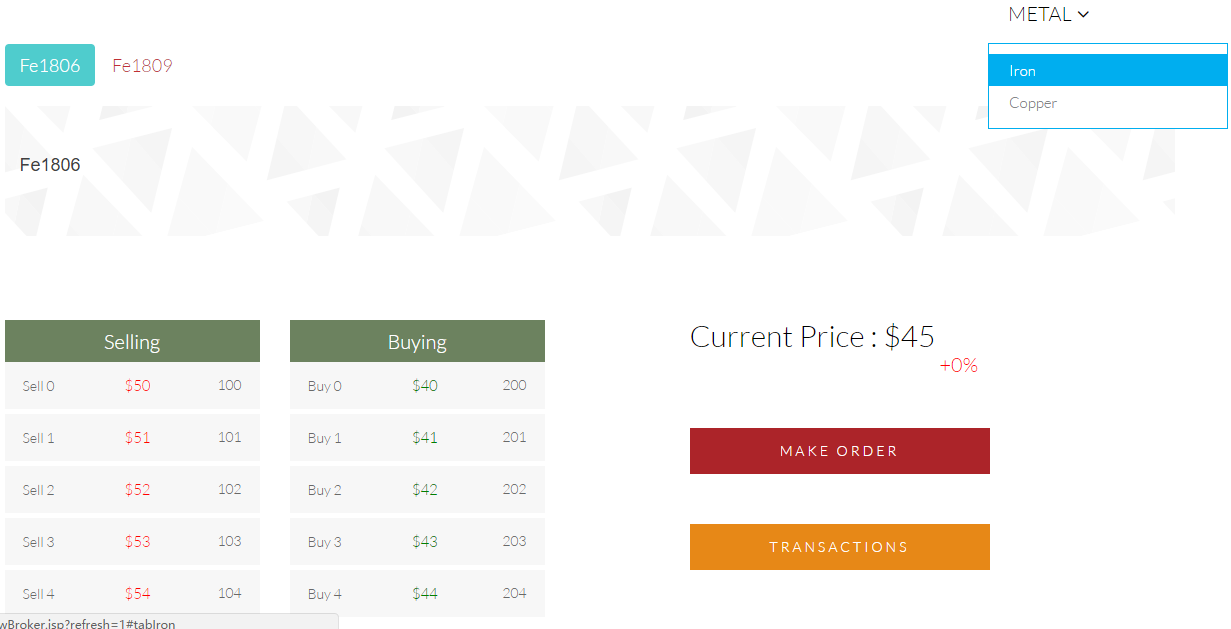
* Specity the broker

用户在登录后选择不同broker，才能查看不同broker下对应的商品信息。Broker信息列表从restful接口动态获取。



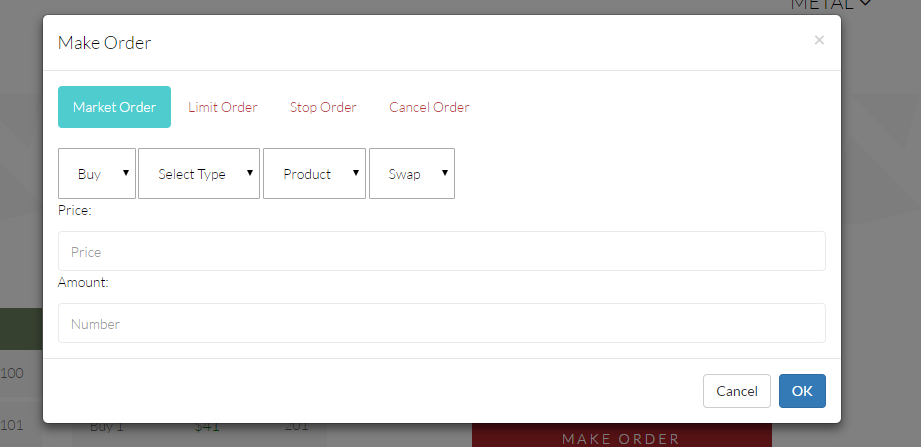
* View the lastest market depth

用户通过选择商品大类、商品属性，来确定某一种特定的交易商品，并且查看当前的market depth情况、当前价格、价格波动情况。价格高于当前价格的显示红色，低于当前价格的显示绿色。波动为正的显示红色，波动为负的显示绿色。



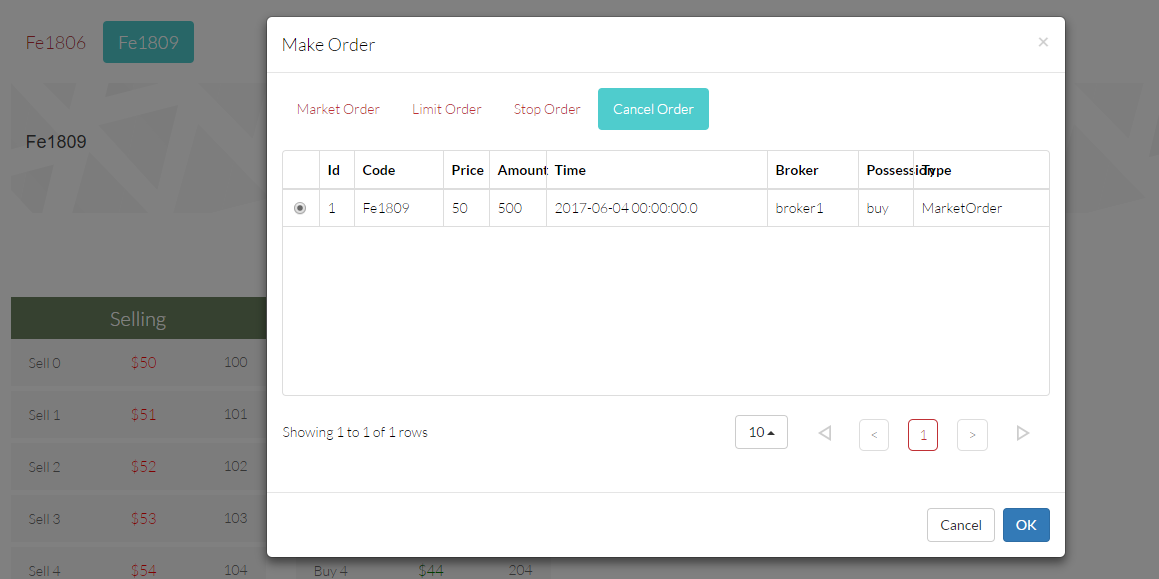
* Initiate an order(Need to specity the broker)

点击MAKE ORDER按钮会出现下单弹出框，使用的是bootstrap的modal。由于broker在查看商品信息之前就已经选好，因此这里就不用用户选择，网页后端直接保留brokerid。同时后端还会绑定用户的id。



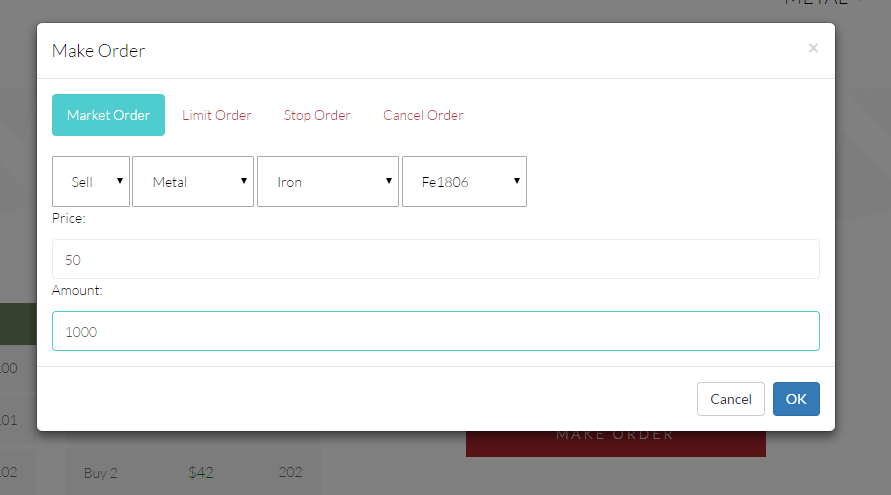
* Cancel an order

用户在弹出框点击Cancel Order后，会进行tab切换，出现可以取消的order列表。用户可以选择某一行order进行取消操作。



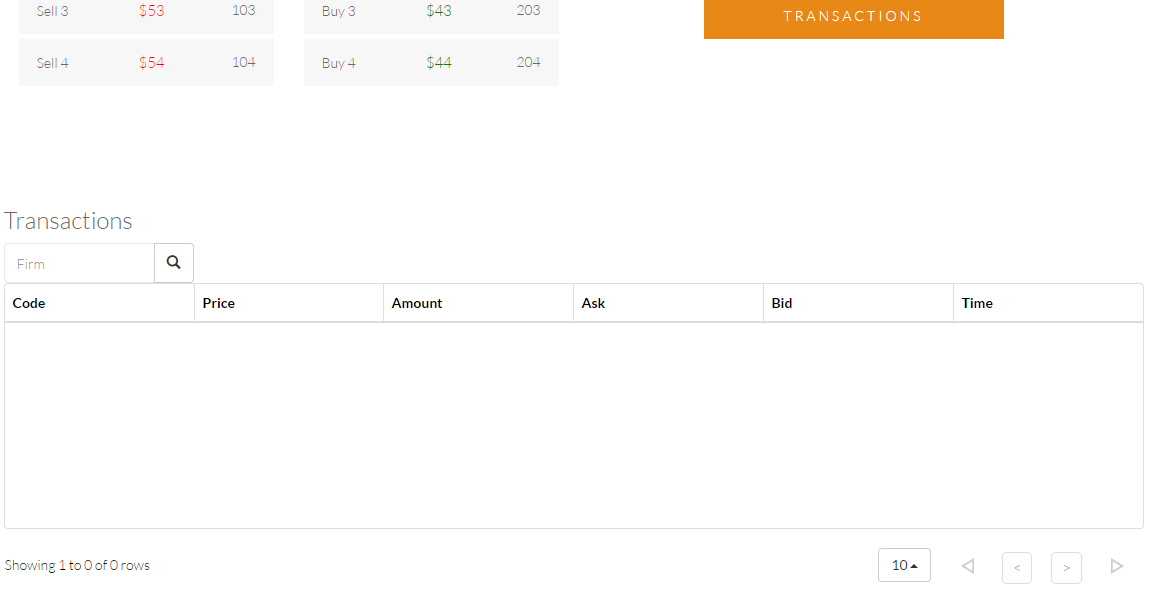
* Fill an order

用户在另外三种类型的下单中，可以选择Buy或Sell，然后通过三级联动下拉框选择要交易的商品。然后填写要交易的价格和数量。



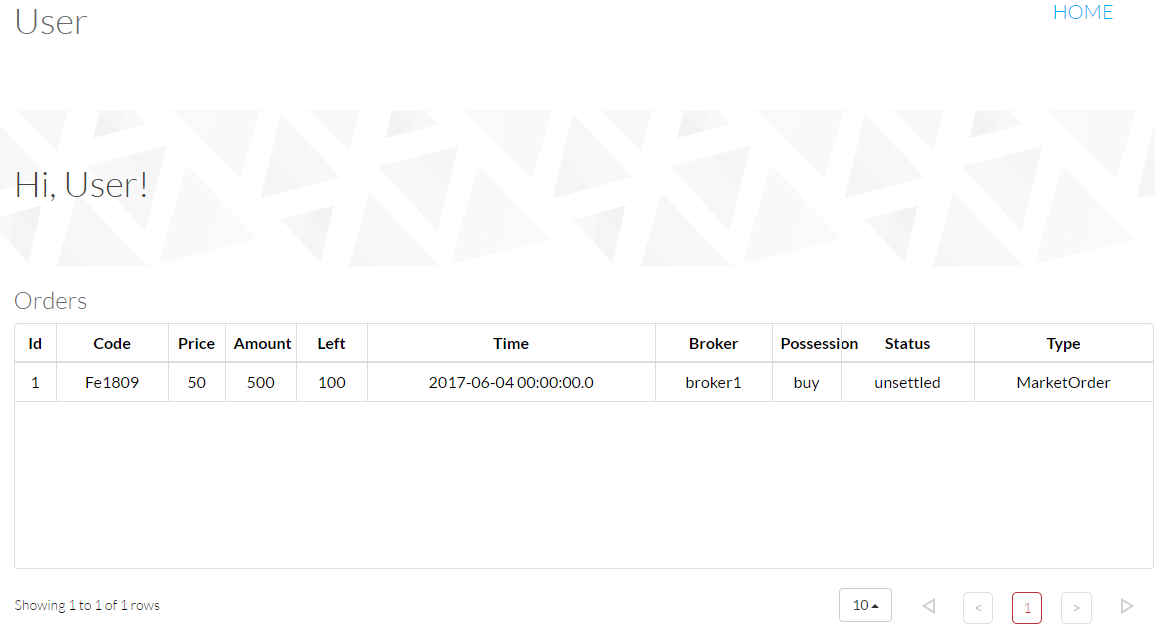
* Show product transactions

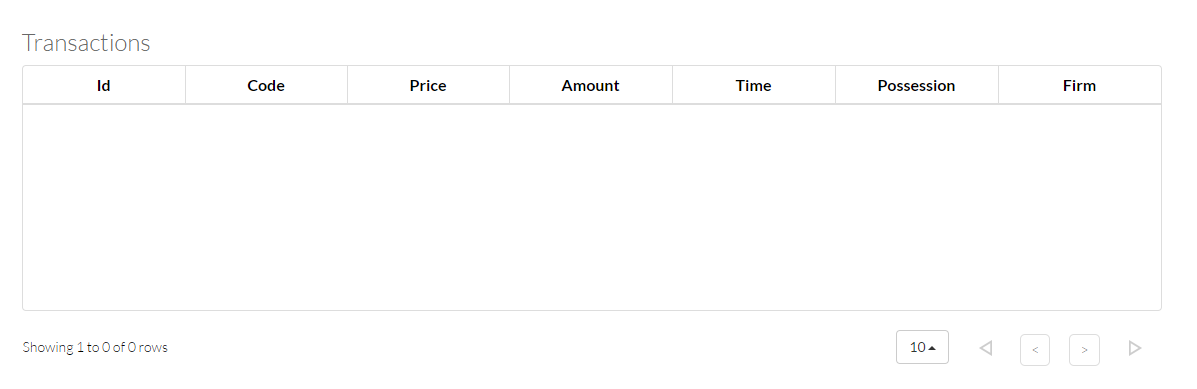
点击TRANSACTIONS按钮可以看到当前商品的交易列表。可以对公司名称进行查询。



* Show user orders and transactions

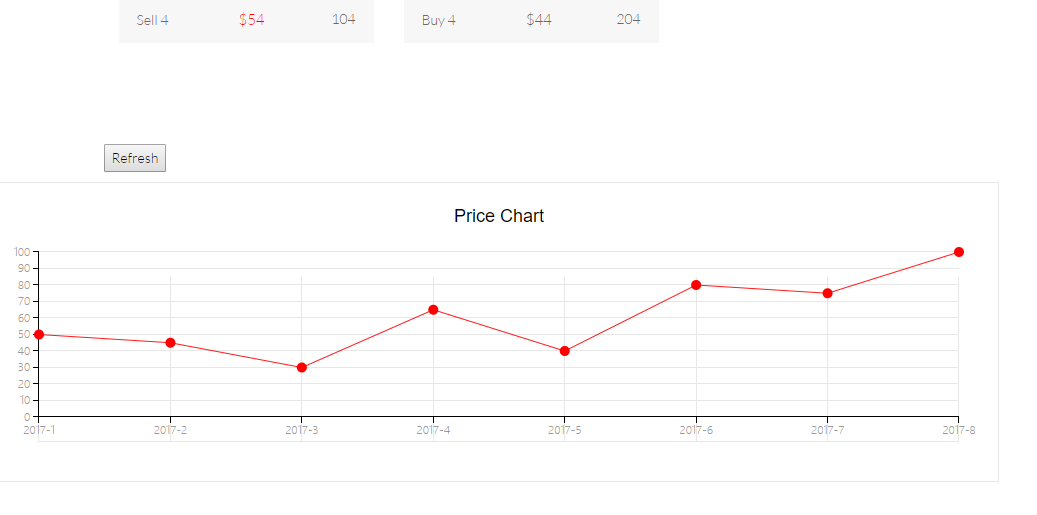
在User页面中，可以差那可能到用户关联的所有order列表和transaction列表。





* Show price graph

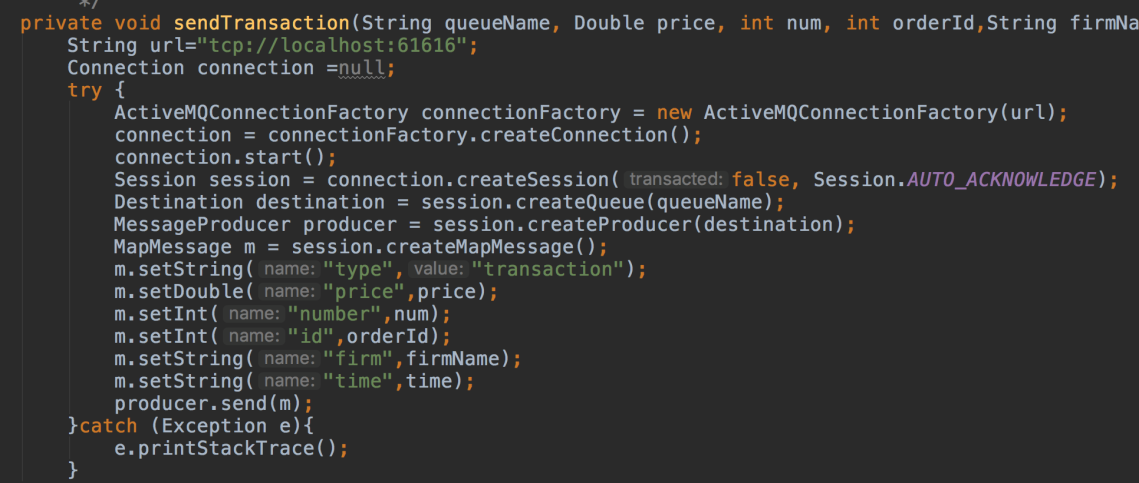
在每个商品下还能查看价格走势图，使用的是d3.js的折线图工具。



1. **BrokerServer**

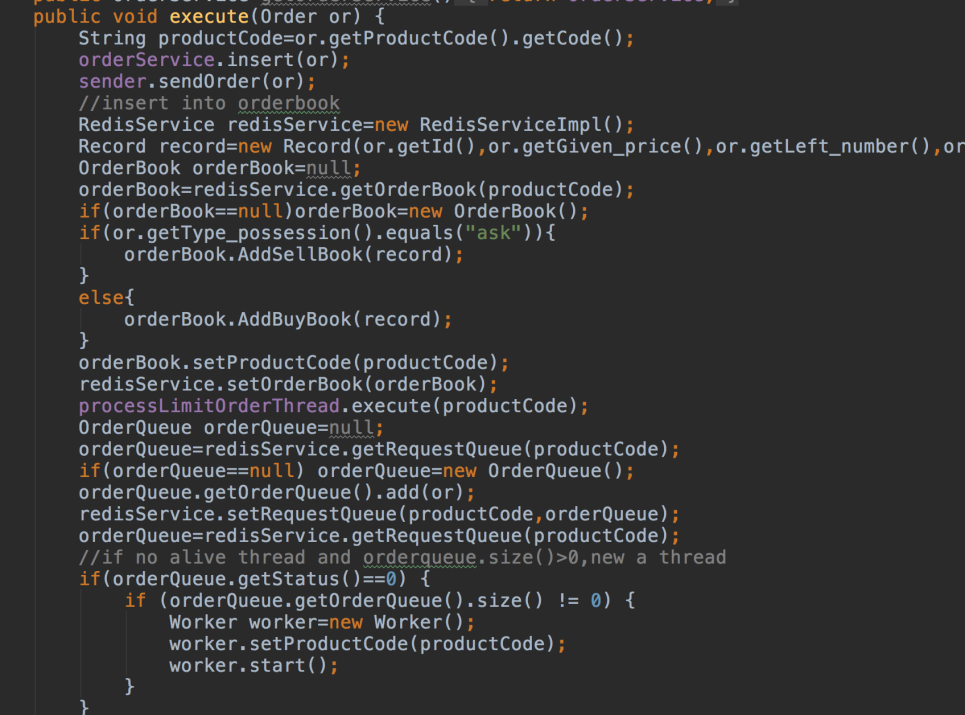
* Connect to broker Server

同样通过JMS对每一个Broker服务器生成一个Connection，注册QueueListener监听Trader端的下单操作。同时有Sender接口实现利用JMS消息发布，根据发送内容类型发布到对应Firm的队列或者是对应产品的TOPIC中，然后Trader端的监听器会取走相应的消息。下图为给成交订单的Trader发送Transaction。



* Maintain Order from various firm

JMS listener收到Trader下单的消息后，将消息读取成Order类并插入数据库，根据Order的产品Code插入对应产品的队列中，每一个产品的订单队列由一个线程进行操作。



* Maintain MarketDepth for product

为每个商品开一个处理订单的线程，保证不同商品的订单是并行处理的。如果没有可交易的订单，线程会sleep3秒，等待新订单的到来。每个product的Orderbook由两个优先队列构成（买和卖）并保存在redis当中。买的按价格降序，卖的按价格升序，相同价格按下单时间升序。

如果是market order，是立即交易的，存在orderqueue里，按照下单顺序依次交易，遇到边界情况如orderbook为空时第一笔订单是marketorder或者是买空卖空的market order，将没处理完的market order插回队列中。



如果是stop order，将其插入orderbook中，循环判断买卖队首的订单能否交易。

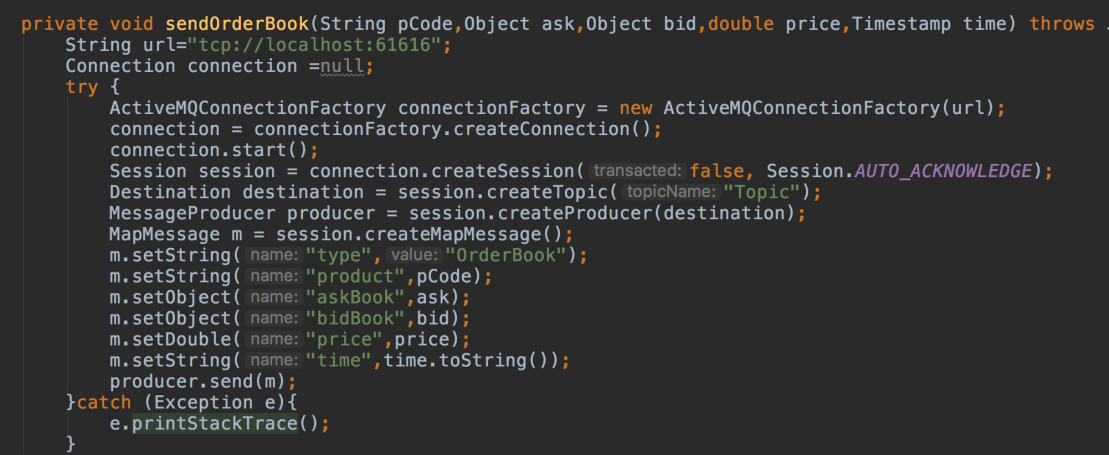


* Cancel Order

由于cancel order是一种特殊的订单，更像一种指令，因此把它存为另外一种数据类型。同样的每种商品的cancel order应该是并行处理的，将cancelorder插入队列中，如果代取消订单还没交易，则成功取消，若部分交易，则部分取消，若全部交易完了，则不能取消。取消就是改变订单的状态，并从orderbook和orderqueue中移除。有一种并发性的情况，当代取消订单正在被交易，此时交易与取消会有冲突。采用乐观锁的方式，为order加一个version字段，当订单被交易了，需要修改left\_number,当订单被取消了，需要修改状态，修改后把version加一，修改数据库前取出version的值，提交修改时检查version是否被改过，如果有，就利用spring的事务机制自动回滚，abort掉事务。

* Broadcast the latest transaction

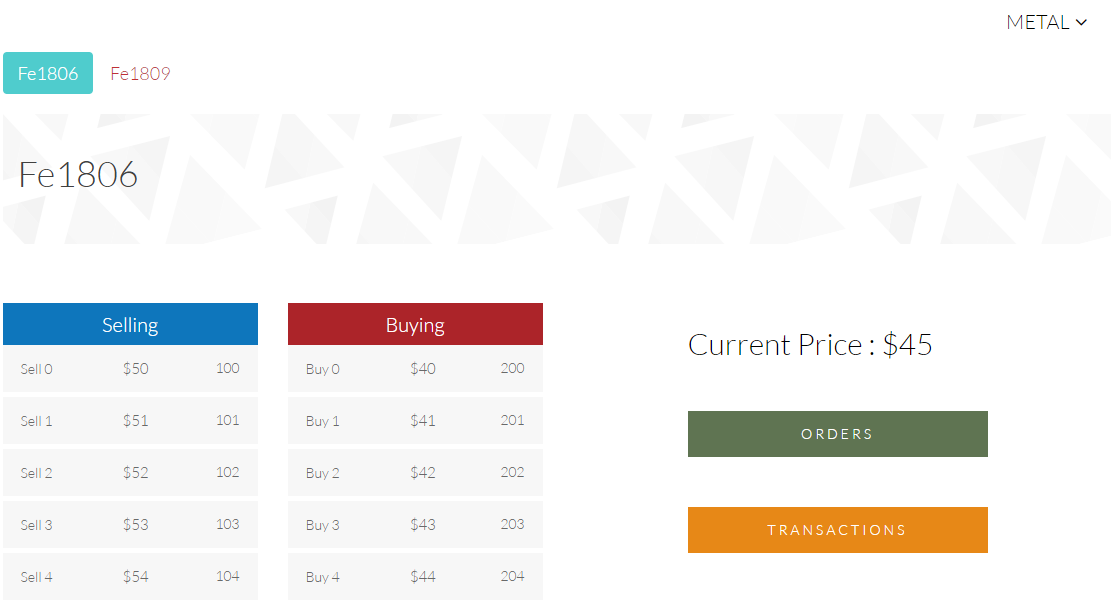
每次订单成交导致产品OrderBook变动时，线程调用JMS中sendOrderBook接口，把最新的交易信息发布到Topic中，所有订阅的Trader都可以取到该消息。



1. Broker UIClient

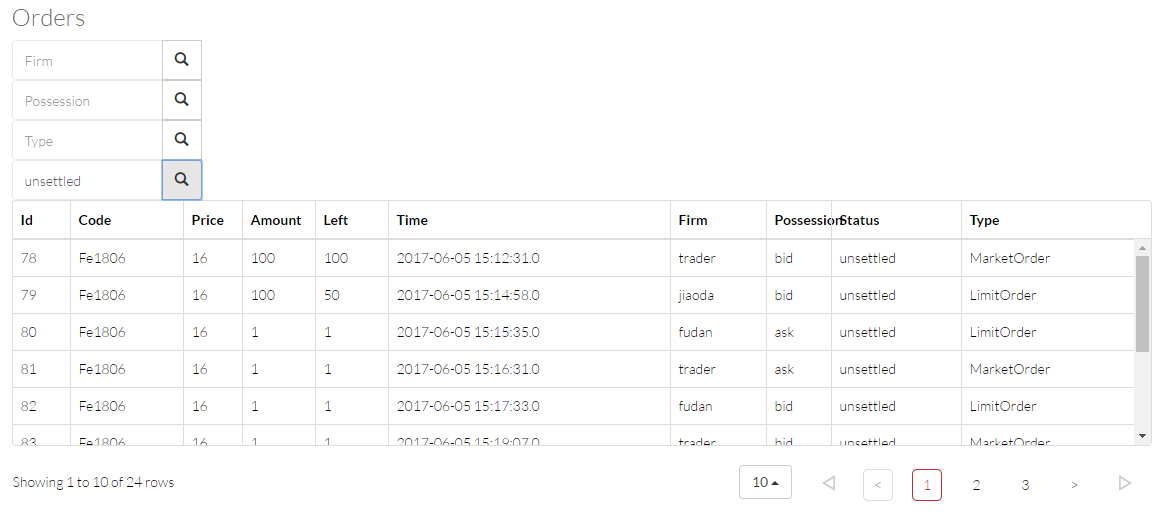
* Visualize the market depth

首次通过restful的get接口获得初始的market depth数据，之后通过web socket动态获取数据并且更新。

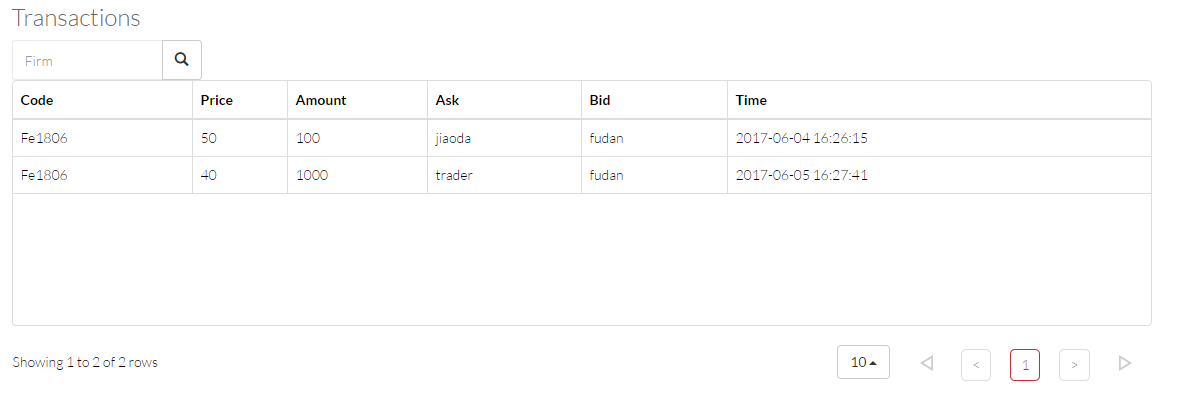


* Show the pending orders as well as completed transactions

点击ORDERS按钮可以在每个商品页面下方动态显示order列表，并且可以对公司名称、买卖属性、下单类型以及order当前情况（traded、canceled、unsettled）进行查询。

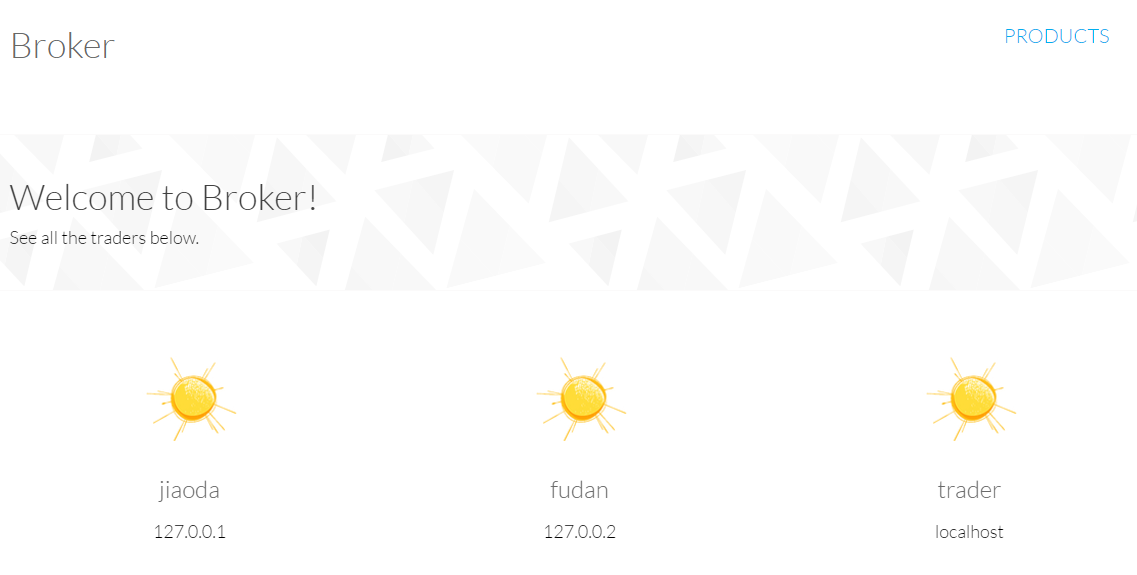


点击TRANSACTIONS按钮可以每个商品页面下方动态显示transaction列表，并且可以对公司名称进行查询。



* Show traders’ information

有一个单独页面可以查看与当前broker有关的trader的信息。



**System Architecture：**

1. **Backup database**

数据库采用MySQL+MongoDB的存储结构，产品、用户、订单信息等都存于MySQL，较为频繁的成交纪录、价格变化和交易日志存在MongoDB中。对MySQL配置主从备份，进行读写分离来提高效率。具体配置如下：

1. **Secure communication**

* RSETful CORS跨域访问

RSETful访问资源（GET方法）安全性较低，因此配置CORS过滤器，只允许指定域的资源读取。只需要在Header中对“Access-Control-Allow-Origin“属性进行设定即可。



* ActiveMQ SSL通信

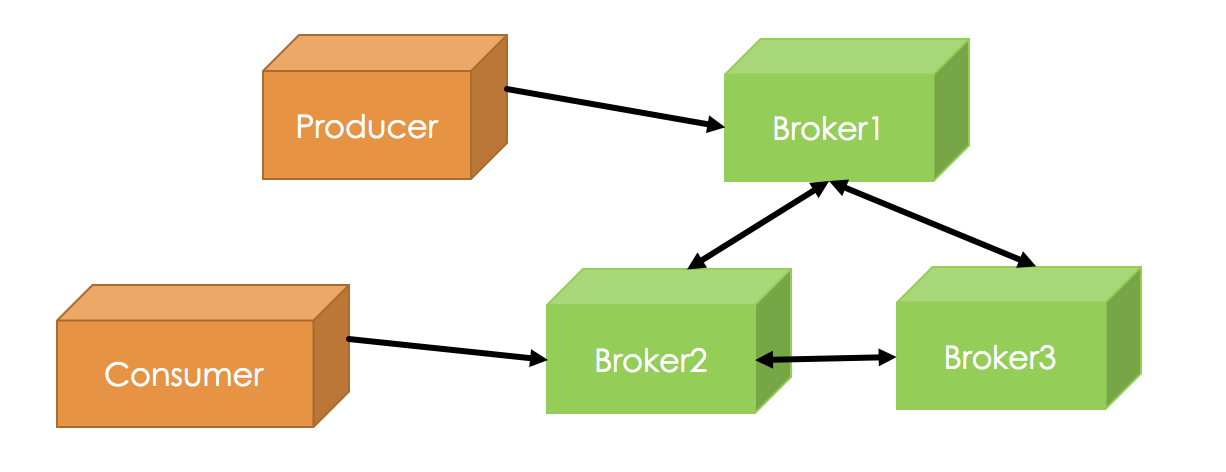
用普通tcp发送JMS消息没有任何加密，容易被拦截，非常不安全。所以可以选择用SSL通信。在ActiveMQ的配置中开启SSL Connector连接器，并引入刚刚生成的broker.ks和broker.ts文件，并设置相应的公钥，私钥密码即可。



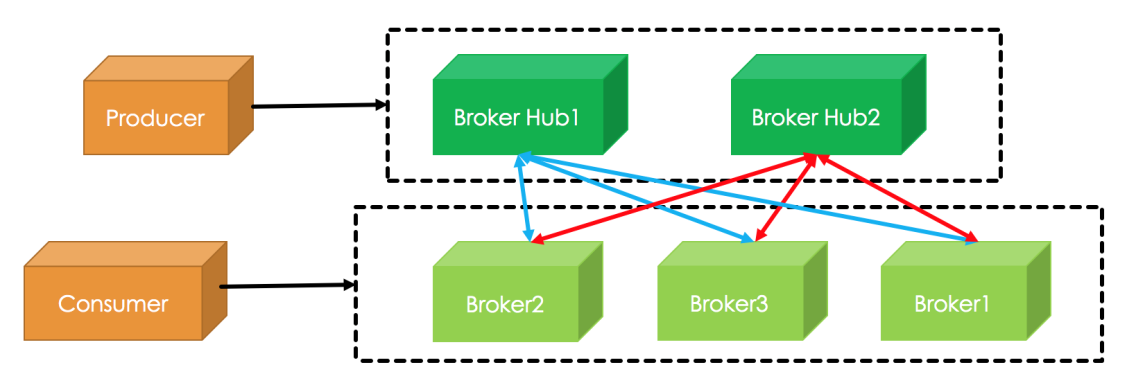
1. **Load Balance**

* JMS采用Broker-Cluster部署方式

Broker-Cluster部署方式中各个broker通过网络互相连接，并共享queue。当broker-A上面指定的queue-A中接收到一个message处于pending状态，而此时没有consumer连接broker-A时。如果cluster中的broker-B上面由一个consumer在消费queue-A的消息，那么broker-B会先通过内部网络获取到broker-A上面的message，并通知自己的consumer来消费。如下图所示。



但是如果broker数量再增多，从消息的路由角度来看，每多经过一个节点，消息处理的延时将会增加一些，如果Broker越多，情况越复杂，最终系统对外表现为消息处理有时很快，有时很慢，整体性能很不稳定，所以实际生产中，不要采用所有Broker之间两两互连的方案。采用一个hub来管理那么不管consumer连到哪个broker上，消息处理时间都相似。但是一个hub会变成单一故障节点所以配置两个Hub即可。最终部署如下图。



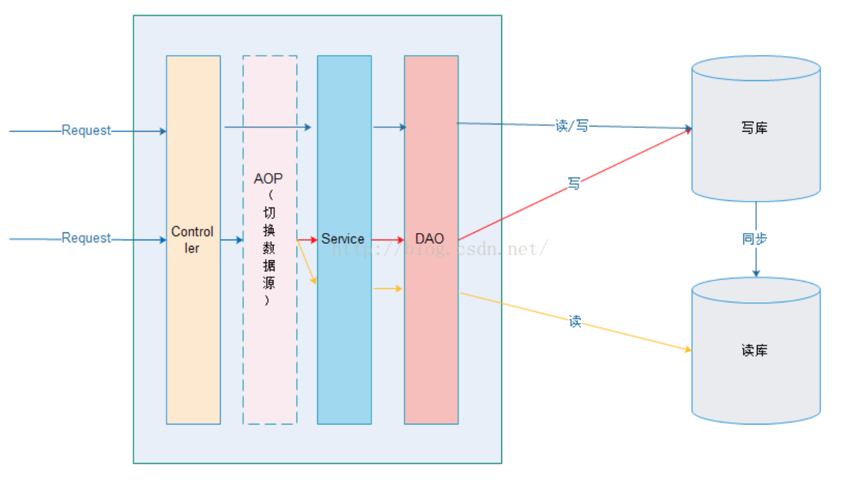
**4.Scalability：**

* **nginx**

利用nginx做反向代理，负载均衡。启动三个server，端口为81，82，83. 采用加权轮旬的方式，选择负载最小的节点。

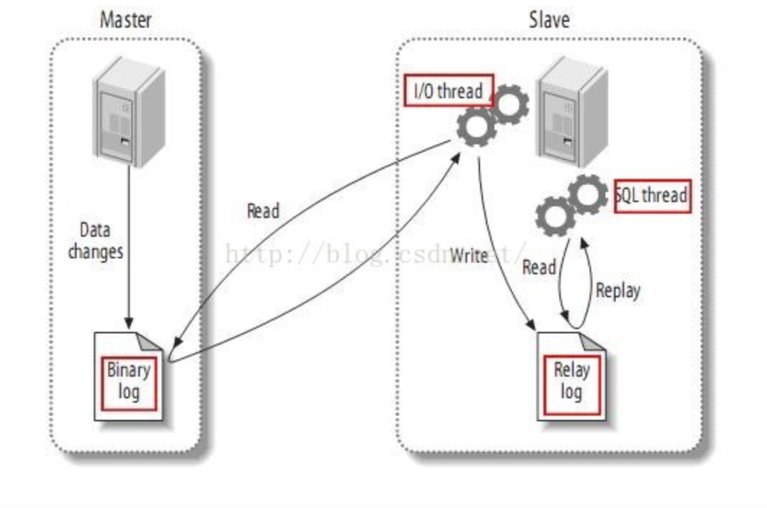


* **mysql读写分离**



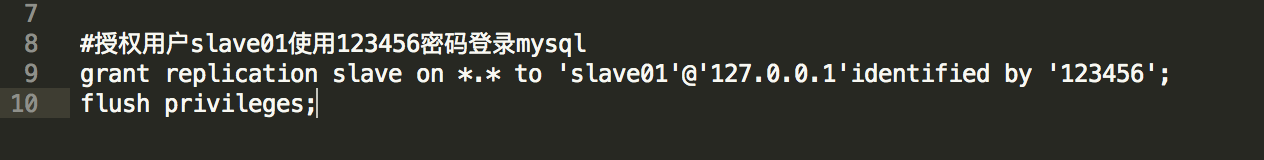
对数据库的操作读多写少，读操作对数据库的压力很大，因此对mysql做读写分离，其中一个主库负责写，其余的从库负责读。要求主库与从库数据一致，读数据到读库，写数据到写库。使用AOP对service做区分，根据service的命名来区分，带有query，get的走读库，带有update，insert，delete的走写库。

mysql主从备份

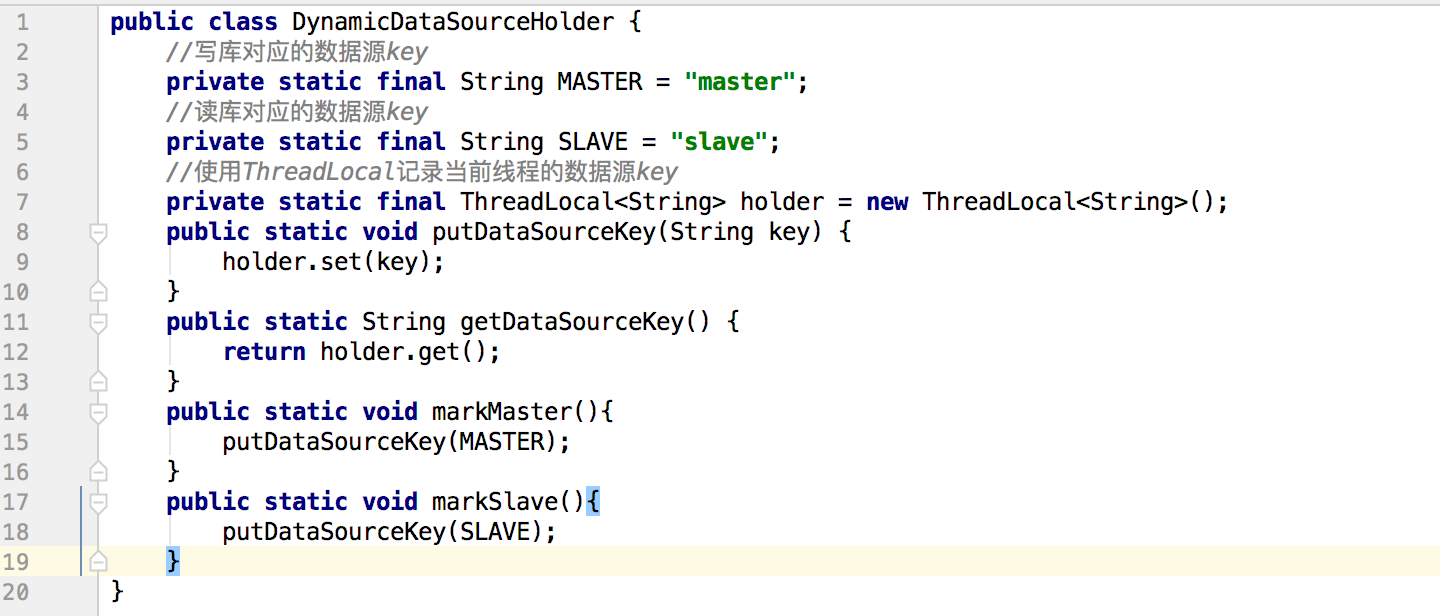


mysql配置 远程登录 自动同步





AOP配置 根据函数名做切片



5.小组分工

楼思琦 brokerserver traderserver jms通信 redis缓存 restful接口 33%

徐瑾卿 brokerserver traderserver订单处理，orderbook广播，负载均衡，分单 33%

吴佳雯 traderUI brokerUI restful接口 33%