交易模式

快捷模式：智能合约托管交易币和保证金，买家和卖家直接交易。

针对无币场景：

1) 允许(买家或卖家)自定义保证金，即买家保证金被设置为0%，则买家无币也可以购买。

2) 主动送币，场景：空投（随机送），挖矿，看广告，签到。

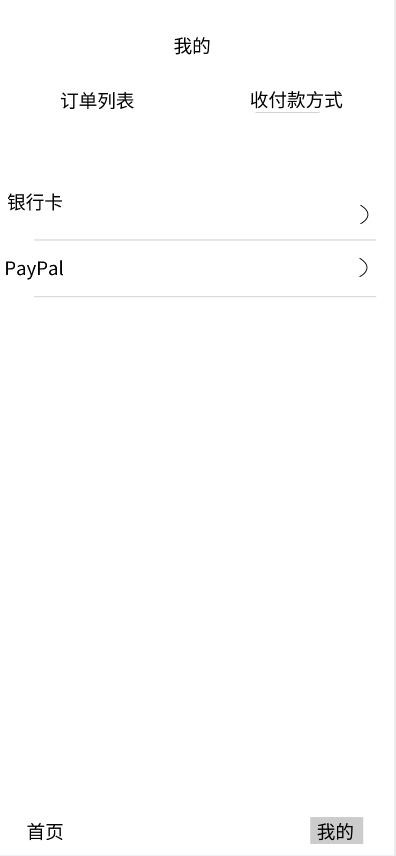
涉及角色

卖家：交保证金，确认收款，取消订单，赎回后的保证金可继续出售。

买家：交保证金，法币转账, 取消订单。

收付款方式

用户通过我的-收付款方式，主动添加收付款账号，此外在其它页面点击购买/出售按钮时，若用户未添加过收付款方式，则调转到该页面。

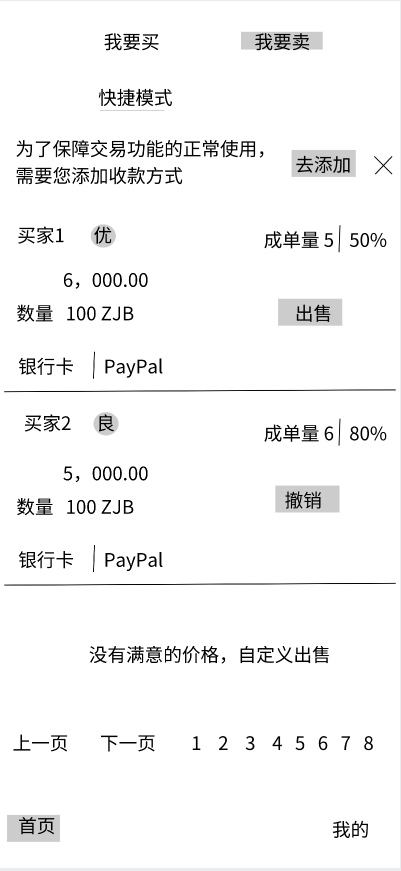


点击右侧的箭头(添加银行卡为例)，则进入“添加银行卡”页面（“已添加银行卡”页面类似），如下图所示



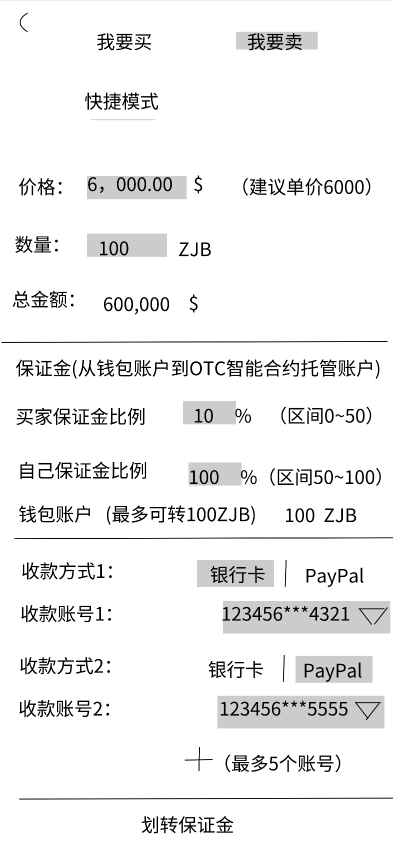
我要买：

要购买的话，首先需要有卖家出售（仅自己创建订单有撤销入口），如下图所示



卖家点击“没有满意的价格，自定义出售”

快捷模式：

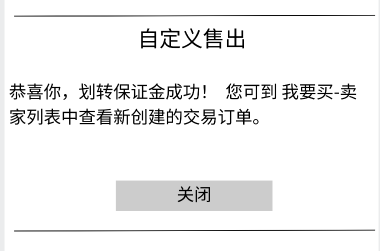


1、若点击左上角回退图标“<”，则返回前面我要卖-买家列表页面。

2、点击下面“划转保证金”

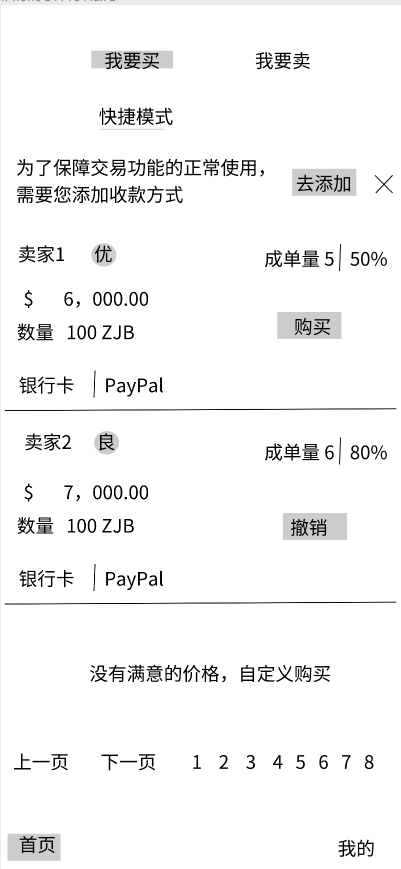
若失败，则弹框提示“划转保证金失败”以及失败原因，叉掉弹框，则返回前面我要卖-买家列表页面。

若成功，进入“自定义售出”弹框，如下图所示



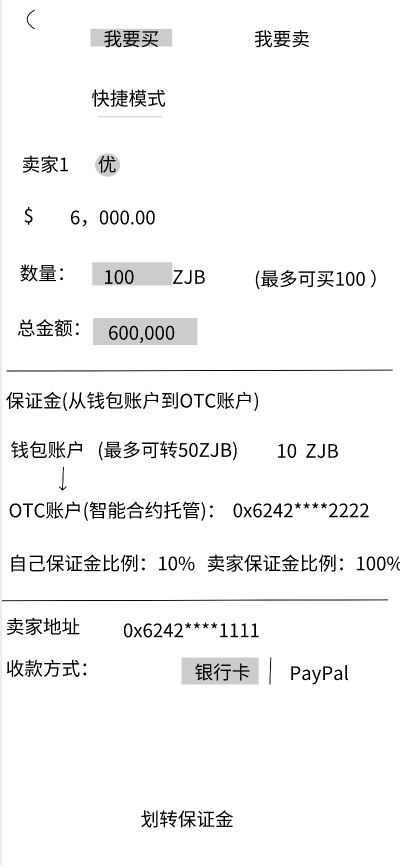
点击“关闭”，返回前面我要卖-买家列表页面。

快捷模式，卖家列表（仅自己创建订单有撤销入口）如下图所示



用户从卖家列表中选择一个进行购买

快捷模式：

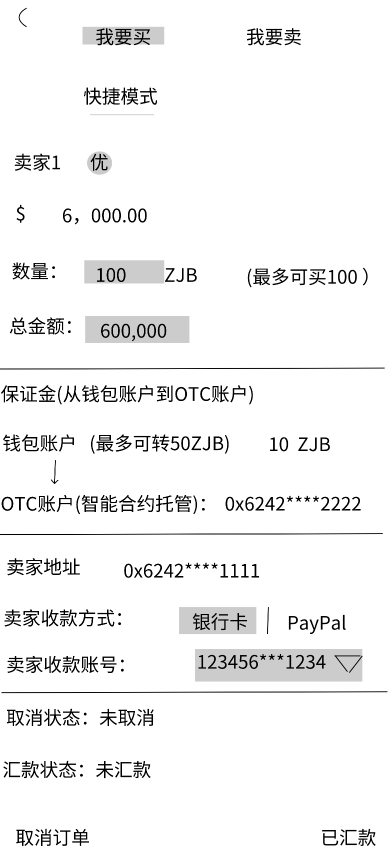


1、点击左上角回退图标“<”，则返回前面我要买-卖家列表页面。

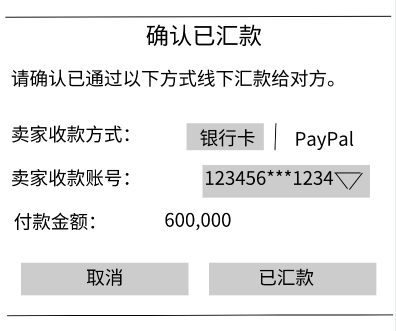
2、点击下面“划转保证金”

若失败，则弹框提示“划转保证金失败”以及失败原因。

若成功，则进入“已付款，请放币”页面，如下图所示：



点击右下角“已汇款”，进入“确认已汇款”弹框，提示：“请确认已通过以下方式线下汇款给对方”。如下图所示



通过我的-订单列表，可点击查看订单详情信息，并继续操作订单未完成的部分。

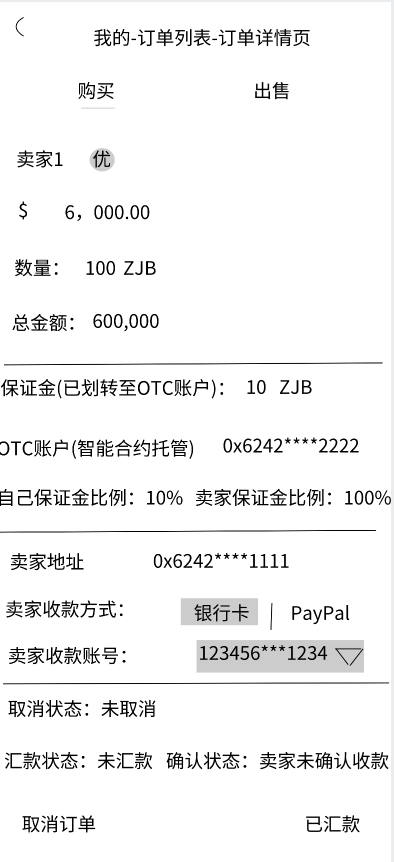
订单列表-购买列表，如下图所示



点击订单列表-购买列表中“订单详情”，进入订单详情页面

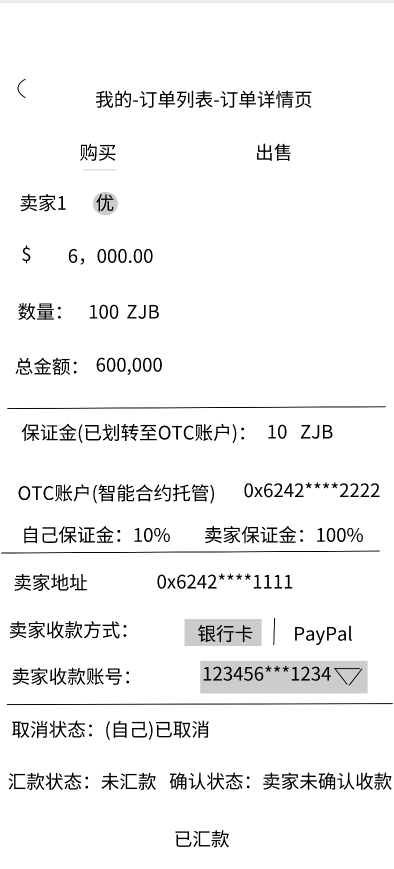
1. 买家未汇款

1）未取消，如下图所示



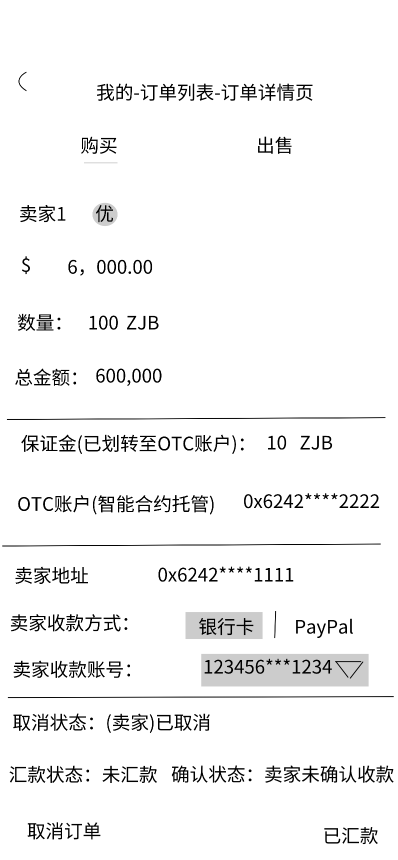
1.点击左下角“取消订单”，等卖家取消订单后，赎回保证金。  
 2.点击右下角“已汇款”，进入“确认已汇款”弹框，确认后更新链上订单状态为“已汇款”。

2）(自己)已取消，买家不能进行任何操作，如下图所示



1.点击右下角“已汇款”，进入“确认已汇款”弹框，确认后更新链上订单状态为“已汇款”。

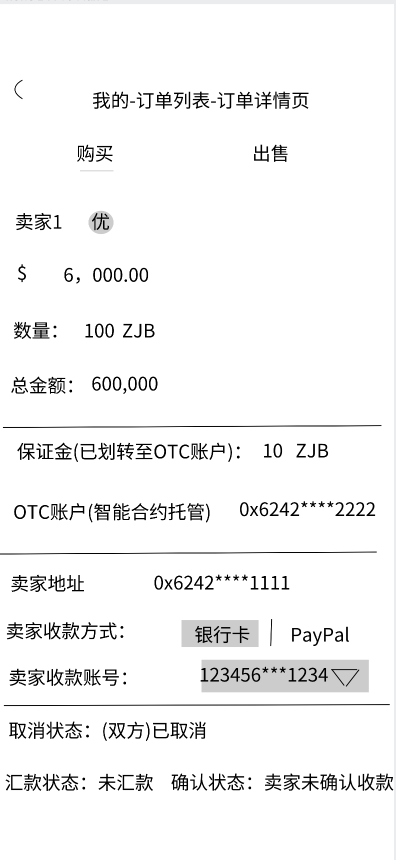
1. (卖家)已取消，如下图所示



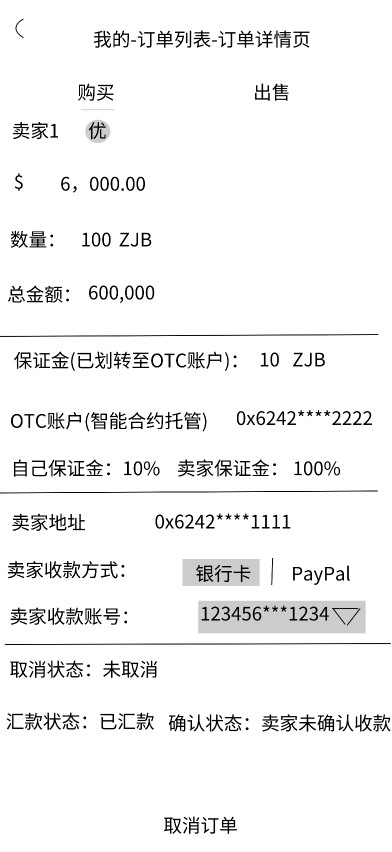
1.点击下面“取消订单”，则直接赎回保证金。

2.点击右下角“已汇款”(不知道卖家已取消，自己汇款了)，进入“确认已汇款”弹框，确认后更新链上订单状态为“已汇款”。

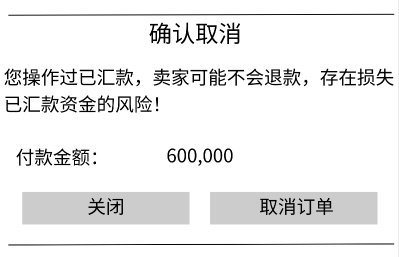
4）(双方)已取消（买家先取消，卖家再取消或卖家先取消，买家再取消），不能进行任何操作，如下图所示



1. 买家已汇款
   1. 未取消，如下图所示

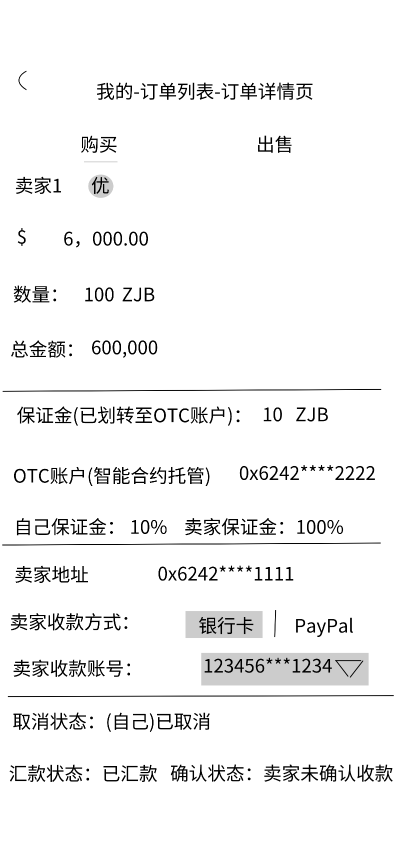


1.点击下面“取消订单”，进入“确认取消订单”弹框，提示“你操作过已汇款，取消订单存在损失已汇款资金的风险!”。如下图所示

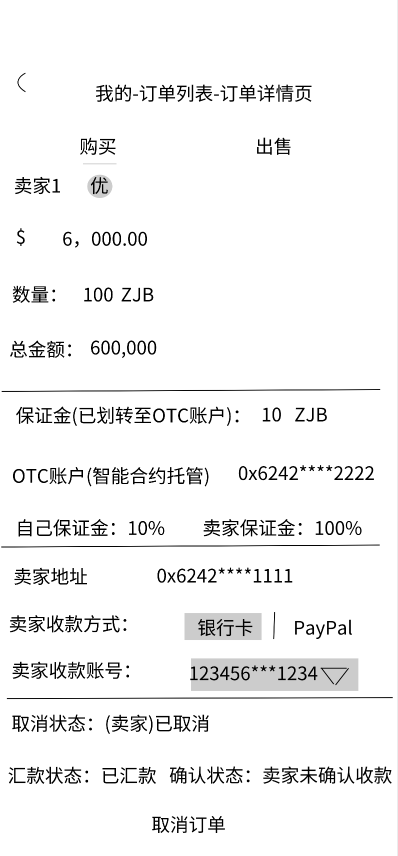


点击“关闭”则关闭弹窗。点击“取消订单”，则卖家取消订单后赎回保证金。

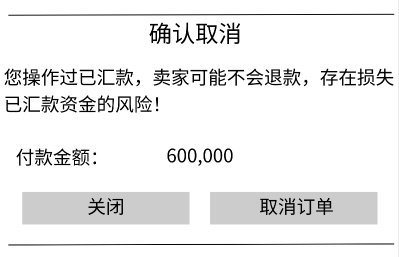
2）(自己)已取消（先汇款，再取消），不能进行任何操作，如下图所示



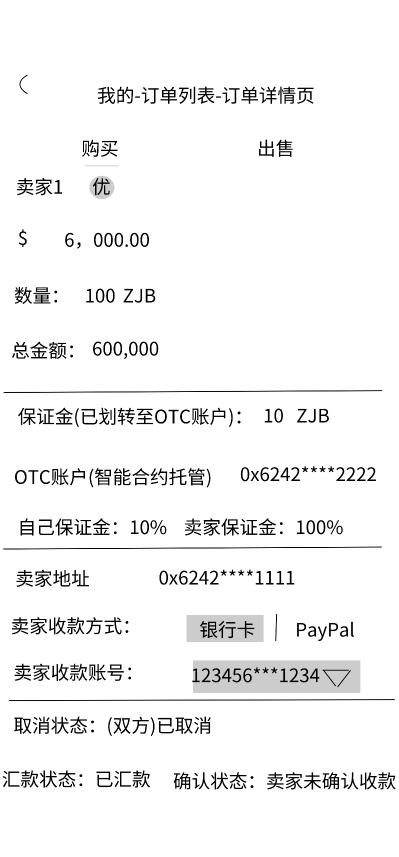
3）(卖家)已取消（买家先汇款，卖家再取消），如下图所示



1.点击下面“取消订单”，进入“确认取消订单”弹框，提示“你操作过已汇款，取消订单存在损失已汇款资金的风险!”。如下图所示

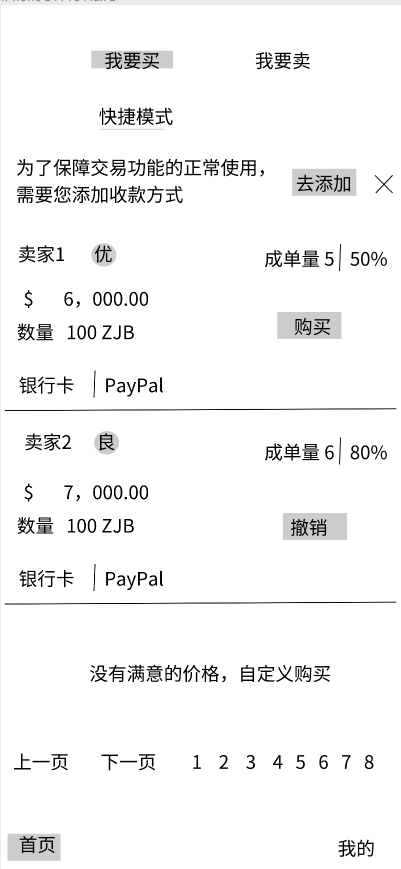


点击“关闭”则关闭弹窗。点击“取消订单”则直接赎回保证金。  
4）(双方)已取消（买家先汇款，卖家再取消，买家不得已再取消），不能进行任何操作，如下图所示



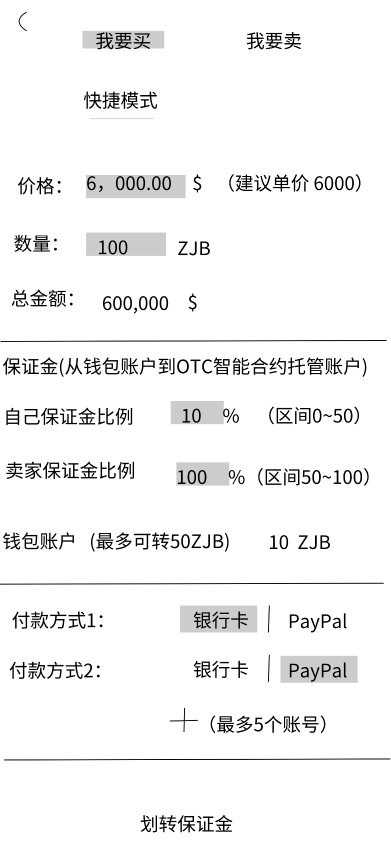
我要卖

要出售的话，首先需要有买家购买（仅自己创建订单有撤销入口），如下图所示



买家点击“没有满意的价格，自定义购买”

快捷模式：



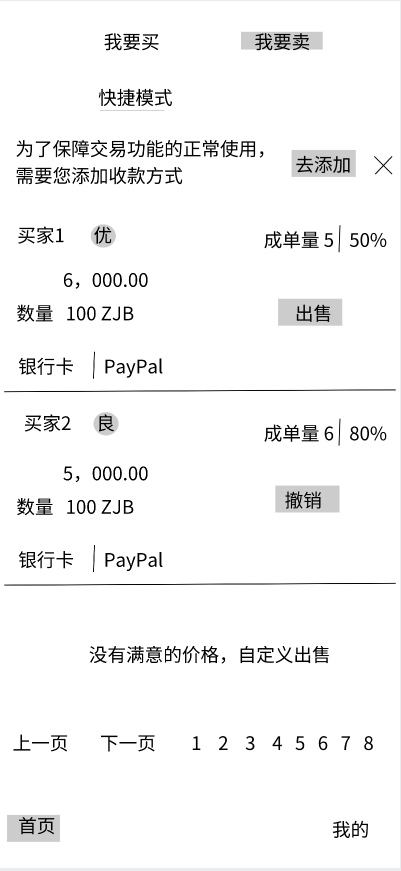
1、若点击左上角回退图标“<”，则返回前面我要买-卖家列表页面。

2、点击下面“划转保证金”

若失败，则弹框提示“划转保证金失败”以及失败原因，叉掉弹框，则返回前面我要买-卖家列表页面。

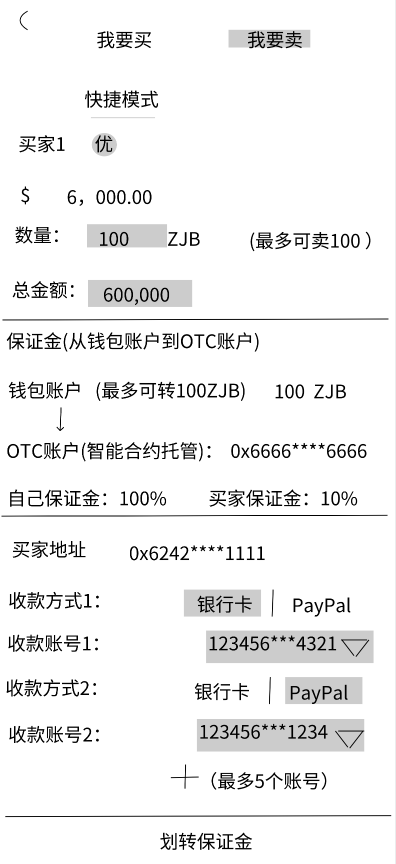
若成功，则返回前面我要买-卖家列表页面。在我要卖-买家列表中，可查看新创建的交易订单（即创建一个新的合约）。

快捷模式，买家列表（仅自己创建订单有撤销入口）如下图所示



用户从买家列表中选择一个进行出售

快捷模式：

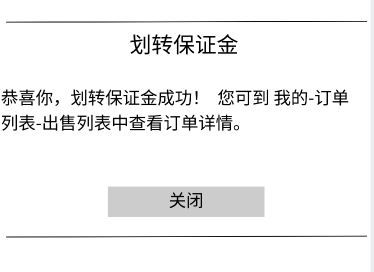


1、点击左上角回退图标“<”，则返回前面我要卖-买家列表页面。

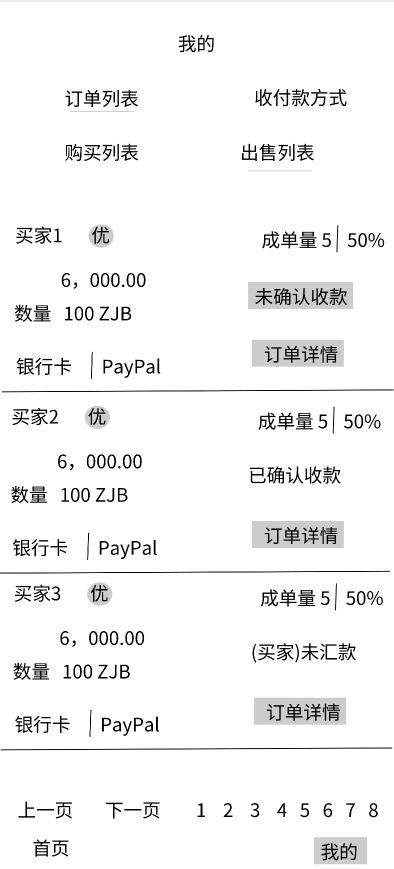
2、点击下面“划转保证金”

若失败，则弹框提示“划转保证金失败”以及失败原因，关闭后返回前面我要卖-买家列表页面。

若成功，则弹框提示“划转保证金成功，可到我的-订单列表-出售列表中查看订单详情”。

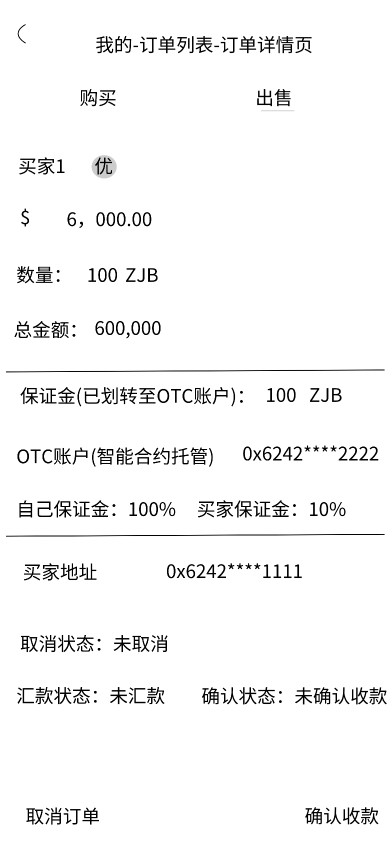


订单列表-出售列表



点击“订单详情”，进入订单详情页面，分如下几种场景

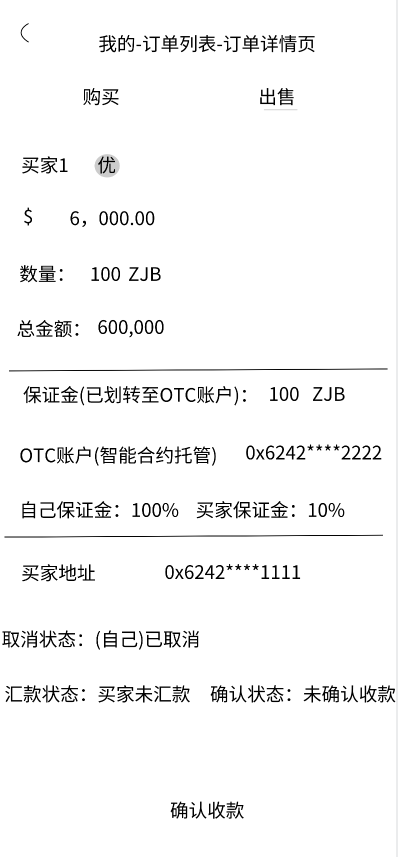
1、未确认收款  
1）未取消，如下图所示



1.点击右下角“确认收款”（前提是收到线下汇款），则交易币转给买家，赎回保证金。

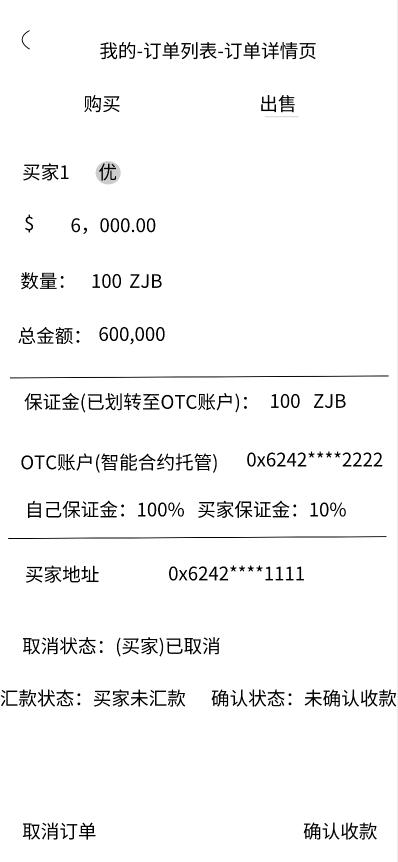
2.点击左下角“取消订单”，买家取消订单后，赎回保证金；若买家一直未操作取消订单，则可能永远无法赎回保证金。

2）(自己)已取消（卖家先取消，买家再汇款但还未来及更新链上状态为已汇款），如下图所示



1.点击下面“确认收款”（前提是收到线下汇款），则交易币转给买家，赎回保证金。

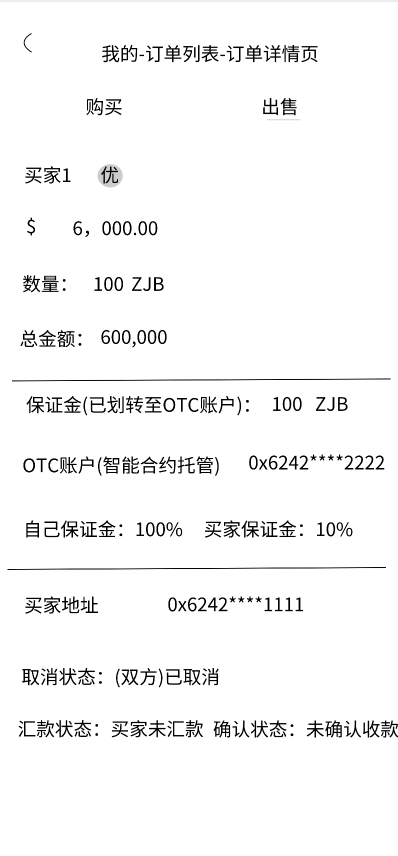
3）(买家)已取消（买家先取消，再汇款但未更新链上订单状态为已汇款或买家先汇款了但未更新链上订单状态为已汇款，再取消订单），如下图所示



1.点击右下角“确认收款”（前提是收到线下汇款），则交易币转给买家，赎回保证金。

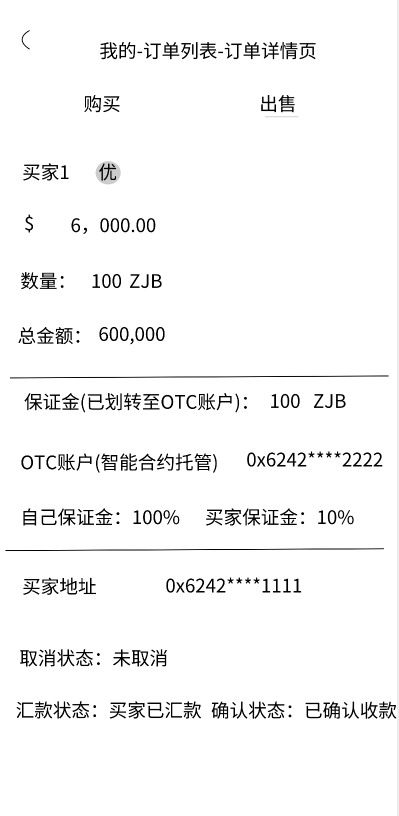
2.点击左下角“取消订单”，直接赎回交易金和保证金。

4）(双方)已取消（买家先取消，卖家再取消或卖家先取消，买家再取消），不能进行任何操作，如下图所示

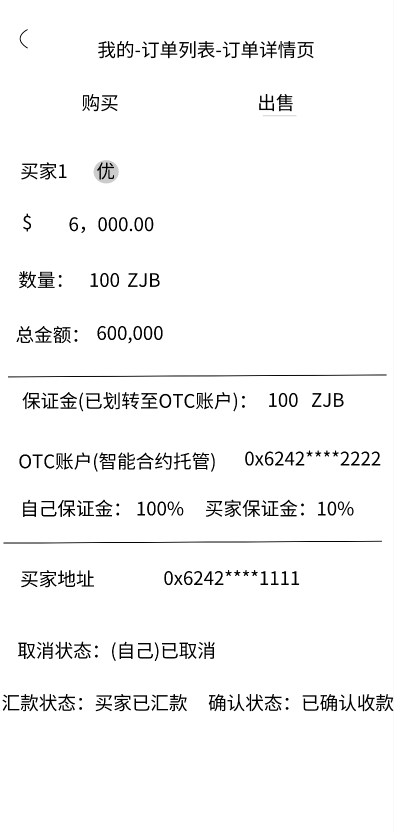


2、已确认收款

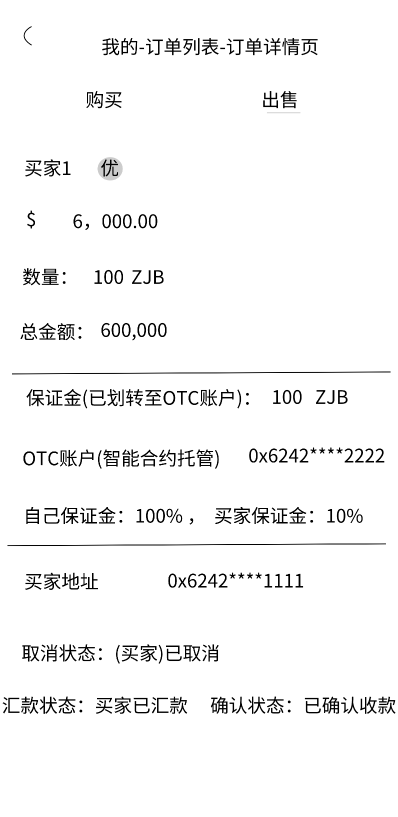
1）未取消，不能进行任何操作，如下图所示



2）(自己)已取消(买家先汇款，卖家再取消订单，卖家再已确认收款或卖家先取消订单，买家再汇款，卖家再已确认收款)，不能进行任何操作，如下图所示



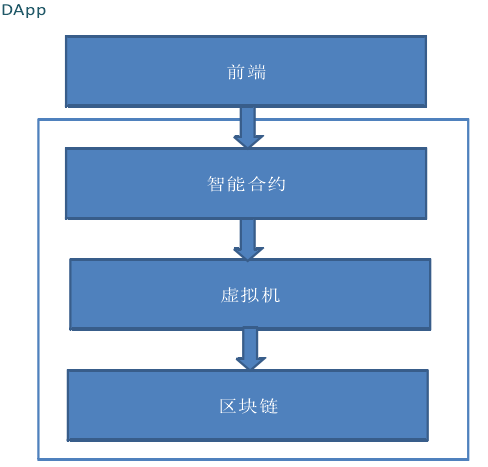
3）(买家)已取消，不能进行任何操作，如下图所示



技术方案

系统分层架构：

前端http调用智能合约，方便测试



快捷智能合约：托管交易币和保证金。

**所有的卖家挂单共用一个合约（**防止合约数量爆炸**）1个合约：N个挂单（最大1000），1个挂单： M个吃单 (订单)。**

**所有的买家挂单共用一个合约。**

**公共账户一个合约地址（用于锁死币空投释放转账）。**

**需要提前建好这3个合约地址。**

创建挂单时，若合约中挂单数量已达1000，提示：“目前列表达最大值，不可以创建，请稍后创建”

1.合约地址contractAddress，提前建好

挂单字段：

1.挂单者地址 makerAddress

2.昵称nickname

3.信用分creditScore

4单价(价格) price

5总售(购)数量 totalAmount

6当前可售(购)数量 availableAmount （后台计算出来的）

7已售(购)数量 filledAmount （后台计算出来的）

8付款方式以及账号：paymentTypeBankCardNoMap（买家仅有支付类型，卖家有账号）

9买家保证金比例(前端默认100%) buyerMortgagePercentage

10卖家保证金比例(前端默认100%) sellerMortgagePercentage

11.未锁死币数量 unlockedAmount （可操作赎回按钮直接赎回）

12块的高度(越小说明时间越早) blockHeight（小于一定高度，不允许吃单）

吃单字段：

1. 吃单者地址takerAddress

2.昵称nickname

3.单价(价格) price （挂单中已经存在，这里不需要保存？）

4.数量 amount

5.总金额 totalValue （不完全等于单价乘以数量， 会加一个[-50,50]范围内的数值）

6.（双方）取消状态：cancelStatus  
1、未取消（双方均未取消）

2、买家已取消

3、卖家已取消

4、双方已取消

7.（买家）汇款状态 paymentStatus   
 未汇款

已汇款

8.（卖家）确认状态：paymentConfirmStatus

未确认收款

已确认收款

汇款状态与确认状态可组合如下：

1、买家未汇款，卖家未确认收款

2、买家未汇款，卖家已确认收款 （特殊情况：买家汇款了但没有更新链上订单汇款状态或卖家点错了导致更新了确认状态）

3、买家已汇款，卖家未确认收款

4、买家已汇款，卖家已确认收款

9. 吃单开始时块的高度(越小说明时间越早) startBlockHeight

10.吃单结束时块的高度（双方顺利交易的时间或双方取消时间或到期锁死币空投释放时间），endBlockHeight (若锁死时间超过3个月，币释放空投转到公共账户)

涉及接口

1. 查询挂单列表（比如：首页中卖家/买家挂单列表页面） 从clickhouse链上查询
2. 查询订单列表（比如：我的-订单列表中购买和出售列表）从clickhouse链上查询，类似sql语句查询，可自己设置筛选条件，仅查询当前用户的
3. 查询订单详情接口（比如：我的-订单列表中的订单详情） 不需要再查数据，订单列表中数据展开而已
4. 查询代币接口，即用户可用币数量（比如：我要买/卖中划转保证金页面） 链上查询
5. 查询代币单价接口，每10分钟更新一次（比如：自定义购买/出售中划转保证金页面） 从tag（K，V）键值对查询或从中心数据库服务器查询
6. 新增挂单接口，即挂单加入合约 (比如：首页中“没有满意的价格，自定义购买/出售”中划转保证金页面)
7. 划转保证金接口，创建新订单，即更新智能合约（比如：从挂单列表中进入，我要买/卖中划转保证金页面）
8. 买家已汇款接口，更新订单，即更新链上合约状态（比如：我要买中订单详情页面，订单列表中的购买列表以及订单详情页面）
9. 卖家确认收款接口，更新订单，即更新链上合约状态（比如：订单列表中的出售列表以及订单详情页面）
10. 取消订单接口，即更新链上合约状态（比如：我要买中订单详情页面，订单列表以及订单详情页面）
11. 赎回接口 （比如：订单列表页面）

问题1：

卖家A挂单，多个买家(B, C)吃单，B和C同时从A购买相同数量的币，卖家如何判断收到的汇款来自哪一个买家，从而操作对应买家的已确认收款？

问题2：

买家E和F挂单，卖家G吃单，卖家G同时出售给买家E和F相同数量的币，卖家如何判断收到的汇款来自哪一个买家，从而操作对应买家的已确认收款？  
  
问题3：

更复杂的场景，比如：同一个卖家有多个挂单。

假设币单价0.5美分，即1000币计算总金额5美元。  
方案：

1. 出售的币必须大于等于1000且是500的整数倍。
2. 实际支付总金额在计算总金额（币数量 \* 单价）的基础上加一个数值变量（[-50，50]，即买家最大多支付50美分，最大少支付50美分）。
3. 每次执行合约时，首先生成一个区间[-50，50]内的整数随机值，比如：10，多个挂单依次执行，同一个挂单中多笔订单也依次执行，即第一个挂单中第一笔订单加10美分，第二笔订单加11美分（即在前一个订单基础上多增加1美分，若加到50，则下一个从-50开始），依次循环。
4. 提醒买家必须严格按照订单支付总金额转账，方便卖家根据收款总金额对应到买家。
5. 极端情况下问题，比如：一个卖家挂单，有101个买家都购买了相同数量的币，则至少存在2人支付总金额是相同的（[-50, 50]轮询了一遍以上），如果仅一人完成转账汇款，两个买家都操作了已汇款或都未操作已汇款，卖家可能无法确认收款是哪一个买家的。此时，若卖家察觉到两笔订单总金额相同  
    1、一方面可继续等待，看另一个买家是否汇款，若两个买家都汇款，则卖家两笔订单都操作已确认收款。
   1. 另一方面卖家可选择主动退款（汇款原账号返回），并同时取消2个订单，待买家取消订单后，订单交易完结。

优点：

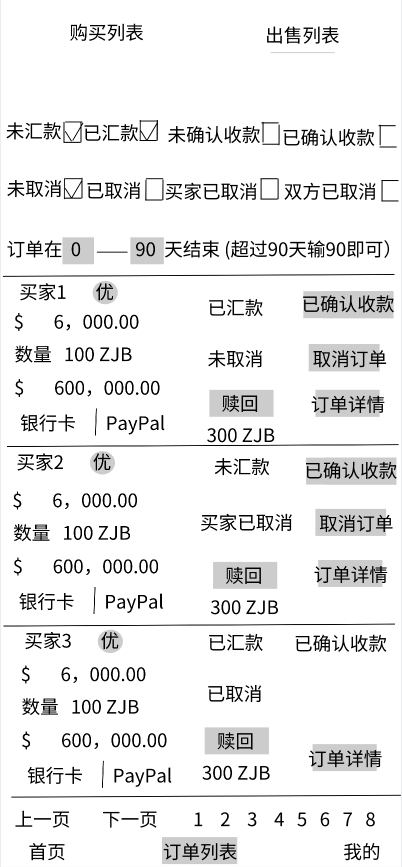
1、卖家只需要根据订单总金额便可对应到买家。

缺点：

1、买家厌恶损失心理，多付钱时不爽，少付钱时开心。对于需要多付钱的订单，可能采取取消订单的策略，等待卖家也取消订单，然后重新下单，直至遇到自己满意的少付款的订单，增加交易成本。（换一种卖家吃亏策略：实际支付总金额比计算总金额始终小， [-100, 0]，相当于给卖家给买家随机优惠0~100美分）

买入、卖出订单管理（币）





1. 从列表选择购买或出售与“自定义出售”或“自定义购买”的交易打平，同一层级展示。
2. 支持订单取消，买家操作已汇款，卖家操作已确认收款等
3. 支持查看订单详情
4. 支持按照买家是否已汇款以及卖家是否已确认收款进行筛选。
5. 支持按照取消状态进行筛选（共计4种）
6. 支持按照结束天数进行筛选（超过90天锁死币空投释放）
7. 挂单者可主动操作赎回按钮赎回未锁定的币，即赎回当前挂单所有可赎回的币（一个挂单多个吃单方时，每个订单中展示的可赎回金额相同，一个订单赎回后，其它订单不可再赎回）。
8. 吃单者不可主动操作赎回按钮赎回（币一旦从锁定状态变为未锁定状态，系统就帮助自动赎回了）。只可被动赎回，比如：双方都取消订单后或币锁死超过3个月后，交易金自动赎回。

如果是“自定义出售”或“自定义购买”，

新挂单加入合约

旧挂单移出合约—> 保存至tag（K，V）键值对，key为挂单者， value为挂单列表  
  
假设有账户A, B 每个账户都可以挂单，也可以吃单。

A作为卖家，挂单OrderA， 即A挂单出售， B直接购买

B作为买家，挂单OrderB， 即B挂单购买， A直接出售

1、可售  
用户A自己在首页-我要买中看不到OrderA (自己在首页看不到自己挂单)  
用户B在首页-我要买中可以看到OrderA

用户A在订单列表-出售列表中可以看到OrderA

用户B在订单列表-购买列表中可以看到已吃单的OrderA

用户B在首页-我要卖中可以看不到OrderB (自己在首页看不到自己挂单)

用户A自己在首页-我要卖中看到OrderB

用户A在订单列表-出售列表中可以看到OrderB

用户B在订单列表-购买列表中可以看到已吃单的OrderB

1. 不可售（比如：挂单超过3天了，不在展示在我要买或我要卖列表）

用户B在首页-我要买列表中看不到OrderA

用户A在首页-我要卖列表中看不到OrderB

吃单者创建订单

币锁死/锁定状态：不可赎回

币未锁死/未锁定状态：可赎回

卖家挂单总售数量（2000币）与买家看到的可售数量不一样（卖家100%保证金时：1000币）。

挂单时总售数量 大于 最终实际售卖数量 （需要交保证金，不可能完全售完）

挂单时总购数量 大于等于 最终实际购买数量

币质押管理（卖家挂单合约，保证金均100%，卖家出售2000币，买家购买500币）

本币账户间转账时机（Transfer操作）：

1、创建挂单，（交易金+卖家保证金） 从卖家个人钱包往合约地址

2、吃单时，（买家保证金） 从买家个人钱包往合约地址

3、双方交易达成，（买家保证金+购买币） 从合约地址往买家个人钱包

4、双方订单取消，（买家保证金） 从合约地址往买家个人钱包

5、币锁死且超过3个月，锁死保证金空投释放 从合约地址往公共钱包

1. 卖家创建挂单，即往合约地址转入2000币(transfer操作，1000交易金，1000保证金) 未锁定状态，买家看到可售数量1000，如果一直未有吃单者，卖家可随时赎回。
2. 吃单者买家往合约地址转入500币保证金锁死状态（transfer操作）  
   卖家同时需要1000币锁死状态（500交易金，500保证金），其它1000币(2000-1000)未锁定状态。  
   1）双方顺利达成交易  
   卖家500保证金不赎回但从锁定状态变为未锁定状态（继续售卖），加上原先未锁死的1000币，即卖家合约地址共1500币未锁定状态(已售500，剩余可售750)。

买家1000币（500保证金+ 500购买币）赎回个人钱包（transfer操作）。

2）双方订单取消  
卖家500交易金和500保证金全部从锁定状态变为未锁定状态，加上原先未锁死的1000币，即卖家合约地址共2000币未锁定状态(已售0，剩余可售1000)。

买家500保证金赎回个人钱包（transfer操作）。

3）有人毁约（仅一方取消或买家不汇款或卖家不确认收款）且超过3个月

双方保证金共1000(各损失500币)被空投释放了（transfer操作到一个公共账户）

卖家500交易金不赎回但从锁定状态变为未锁定状态（继续售卖），加上原先未锁死的1000币，即卖家合约地址共1500币未锁定状态(已售0，剩余可售750)。

买家无币。

币质押管理（买家挂单合约，保证金均100%，买家购买2000币，卖家出售500币）

本币账户间转账时机（Transfer操作）：

1、创建挂单，（买家保证金） 从买家个人钱包往合约地址

2、吃单时，（交易金+卖家保证金） 从卖家个人钱包往合约地址

3、双方交易达成，（买家保证金+购买币） 从合约地址往买家个人钱包

以及（卖家保证金） 从合约地址往卖家个人钱包

4、双方订单取消，（交易金+卖家保证金） 从合约地址往卖家个人钱包

5、币锁死且超过3个月，锁死保证金空投释放 从合约地址往公共钱包

（交易金） 从合约地址往卖家个人钱包

1. 买家创建挂单，则往合约地址转入2000币保证金（transfer操作），未锁定状态，如果一直未有吃单者，则买家可随时赎回2000币。
2. 吃单者卖家往合约地址转入1000币锁死状态(transfer操作，500交易金，500保证金)买家同时需要500币保证金锁死状态，其它1500币保证金未锁死状态。  
   1）双方顺利达成交易

卖家500保证金赎回个人钱包（transfer操作）。

买家1000币（500保证金+ 500购买币）赎回个人钱包（transfer操作）。原先未锁定1500币保证金继续在合约地址（已购500，剩余可购1500）。

2）双方订单取消  
卖家1000币（500保证金+ 500交易金）赎回个人钱包（transfer操作）。

买家500保证金从锁定状态变为未锁定状态，加上原先未锁定1500币保证金，即买家合约地址共2000币未锁定状态（已购0， 剩余可购2000）。

3) 有人毁约（仅一方取消或买家不汇款或卖家不确认收款）且超过3个月

双方保证金共1000(各损失500币)被空投释放了（transfer操作到一个公共账户）

卖家500交易金赎回个人钱包（transfer操作）。

买家合约地址1500币未锁死状态（已购0，剩余可购1500）。

锁死订单管理：（卖家挂单合约）

**合约中不断有新的挂单加入，为了避免合约中挂单数量过大（比如：限制最大1000个挂单），同时要移出旧的已完结的挂单，即挂单需要能够结束，现设置如下规则：**

1. **订单超过“锁死币过期时间”（比如：3个月），锁死币需要空投释放出来，否则订单一直不结束，挂单就无法结束。**
2. **挂单中未锁死的币数量少于阈值N（比如：500）时，仅可赎回，不可继续出售，原因如下：  
   1）一方面出售币的数量越来越少，继续交易的意义不大  
   2）另一方面挂单结束不了（新出售币的同时总有新赎回的保证金用于继续出售）。**
3. **挂单超过“吃单截止时间”（比如：3天，即从首次挂单，3天内允许吃单），即使未锁死的币大于阈值N，依然只能赎回，不可继续出售。因为挂单中未锁死的币数量一直大于N，原因如下：**

**1）币单价高，市场没竞争力，一直售卖不出去。（币流动性太差的挂单应该尽早剔除）  
2）不断有保证金赎回使未锁死的币数量增加。**

**总结： 需要2个过期时间 （订单中锁死币过期时间，挂单中吃单截止时间）和 1个阈值（币自动赎回）。**

**针对规则1：**

**订单超过“锁死币过期时间”**

**方案：**

**1）后台定时任务，每5分钟检测一次。**

**针对规则2和规则3：**

**挂单中未锁死的币数量少于阈值N或挂单超过“吃单截止时间”，自动赎回**

**方案：**

**1）挂单者主动赎回，用户点击进入Dapp后自动进行，则消耗自己的gas费。**

**2）后台定时任务，每5分钟检测一次，消耗是公共账户的gas费。**

**移出合约中已完结的挂单**

**方案：**

**1）后台定时任务， 判断挂单中所有订单无质押的币以及挂单中未锁死的币数量小于阈值N或挂单超过“吃单截止时间”， 若满足，则将挂单移出合约。**

锁死订单管理：（买家挂单合约）

**合约中不断有新的挂单加入，为了避免合约中挂单数量过大（比如：限制最大1000个挂单），同时要移出旧的已完结的挂单，即挂单需要能够结束，现设置如下规则：**

1. **订单超过“锁死币过期时间”（比如：3个月），锁死币需要空投释放出来，否则订单一直不结束，挂单就无法结束。**
2. **挂单中待购买币数量（总购买币数量-已购买币数量）小于阈值N（比如：500）时，未锁死的币仅可赎回，不可继续购买，原因如下：  
   1）一方面待购买币的数量越来越少，继续交易的意义不大  
   2）可能没有足够的币作为保证金抵押购买了，比如：订单未正常完结，质押币都被空投释放出去了。**
3. **挂单超过“吃单截止时间”（比如：3天，即从首次挂单，3天内允许吃单），即使待购买币的数量大于阈值N，依然只能赎回，不可继续出售。原因如下：**

**1）币单价低，市场没竞争力，一直购买不到。（币流动性太差的挂单应该尽早剔除）  
2）没有足够的币作为保证金抵押购买了，比如：订单未正常完结，质押币都被空投释放出去了。**

**总结： 需要2个过期时间 （订单中锁死币过期时间，挂单中吃单截止时间）和 1个阈值（币自动赎回）。**

**针对规则1：**

**订单超过“锁死币过期时间”**

**方案：**

**1）后台定时任务，每5分钟检测一次。**

**针对规则2和规则3：**

**挂单中待购买币数量（总购买币数量- 已购买币数量）小于阈值N或挂单超过“吃单截止时间”，自动赎回**

**方案：**

**1）挂单者主动赎回，用户点击进入Dapp后自动进行，则消耗自己的gas费。**

**2）后台定时任务，每5分钟检测一次，消耗是公共账户的gas费。**

**移出合约中已完结的挂单**

**方案：**

**1）后台定时任务， 判断挂单中所有订单无质押的币以及挂单中待购买币数量（总购买币数量- 已购买币数量）小于阈值N或挂单超过“吃单截止时间”， 若满足，则将挂单移出合约。**

enum CancelState{ uncancel, buyerCanceled, sellerCanceled, bothCanceled}

enum PaymentState{ unpayment, payment}

enum PaymentConfirmState{ paymentUnConfirm, paymentConfirmed}

**卖家挂单合约**

1. **挂单列表(map实现)保存所有的挂单信息。**
2. **优先队列（最大堆实现)中仅保存价格和订单号（订单号小的挂单时间更早），订单号用于映射到挂单列表中挂单。其中score优先级由价格和订单号共同决定。  
   1）优先按照价格排序，价格越高，优先级越高（靠近堆顶，越早移除）。  
   2）价格相同时，按照订单号(订单号越小/块的高度越低，挂单时间越早)排序，订单号越小，优先级越高。**

**3)优先队列可根据订单号进行出队操作，即满足移除订单列表中挂单时，同时移除优先队列中数据。**

1. **价格排序依赖Dapp客户端自己实现，从而简化合约实现。**

1、在创建挂单时

1）获取挂单列表makerOrders的长度size

2）若size < 1000, 则挂单直接加入挂单列表，同时将挂单也加入优先队列(价格，订单号)。

3）若size >= 1000, 则挂单列表已满，需先从两个队列中移除价格最高的挂单  
1）从优先队列获取队头元素但不删除（时间复杂度O(1)），记最高价格为maxPrice，订单号为orderId。

2）若新挂单的价格>=maxPrice，则说明新挂单价格最高，直接不允许创建。  
3）若新挂单的价格<maxPrice  
 1、则从优先队列删除队头元素（时间复杂度O(logn)，以2为底）。

2、从挂单列表中移除对应的挂单(时间复杂度O(1))，若移除的挂单有未锁定的币，则先赎回个人账户。  
 3、执行新的挂单的创建工作，即加入挂单列表(O(1))以及优先队列(时间复杂度O(logn), 以2为底)。

其它操作优先队列的场景(删除)：

在交易(确认收款/取消订单)过程中，若未锁定量/可购量小于等于500，不再出售，立即赎回未锁定的币，或超过一定时间锁死币空投释放，此时从挂单列表中移除对应的挂单(时间复杂度O(1))，同时根据订单号在优先队列中删除对应数据。即优先队列的长度始终等于挂单列表的长度。

**contract** **OrderContract**{

address contractAddress;

uint256 randomValue; //[-50, 50]，计算实际支付金额，卖家根据收款金额对应到订单

uint256 orderId; //订单号，初始值为1，然后依次递增

mapping(uint256 => MakerOrder) public makerOrders;//挂单订单号至挂单订单映射

PriorityQueue priorityQueue ; //score为单价，同时保存订单号映射到挂单列表中挂单。

struct MakerOrder {

address makerAddress; //ZJB 个人钱包地址

string nickname; //昵称

uint256 creditScore; //信用分

uint256 price; //单价 （挂单时传入，始终不变）

uint256 totalAmount; //总售量 （挂单时传入，始终不变）

uint256 availableAmount; //可售数量（根据未锁定数量，重新计算可售数量）

uint256 filledAmount; //已售数量 （卖家确认收款后，累加售出数量）

mapping(string => string[5]) paymentTypeBankCardNoMap;//支付方式和账号

uint8 buyerMortgagePercentage; //买家保证金，默认100%

uint8 sellerMortgagePercentage; //卖家保证金，默认100%

uint256 unlockedAmount; //未锁定数量（操作赎回按钮赎回，买家或卖家的操作都可能改变该值，比如：卖家确认收款后，赎回部分保证金，未锁定数量增加）

uint256 makerBlockHeight; //挂单时块的高度

uint256 makerOrderId; //挂单订单号

mapping(uint256 => TakerOrder) takerOrders; //吃单订单号至吃单订单映射

}

struct TakerOrder {

address takerAddress; ///ZJB 个人钱包地址

string nickname; //昵称

uint256 price; //单价

uint256 totalValue; //实际支付金额（不完全等于单价乘以购买数量）

uint256 amount; //购买数量

CancelState cancelState; //取消状态, 共4种

PaymentState paymentState; //汇款状态，共2种

PaymentConfirmState paymentConfirmState; //(卖家)确认收款状态，共2种

uint256 startBlockHeight; //吃单开始时块的高度

uint256 takerOrderId; //吃单订单号

uint256 endBlockHeight; //吃单结束时块的高度

}

//创建一个合约

constructor(address \_contractAddress) {

contractAddress = \_contractAddress;

randomValue = 0;

orderId = 1;

}

//创建一个新的挂单

function addMakerOrder(address makerAddress, uint256 price, uint256 totalAmount, mapping(string => string[5]) paymentTypeBankCardNoMap) public returns (MakerOrder){ //加锁或加可重入锁？

if（可售数量小于1000或不是500的整数倍）{

return 0;

}

if（挂单列表大于1000）{

//优先队列价格最高移除，同时移除挂单列表中挂单

}

//1、获取当前块的高度makerBlockHeight

//2、generateNewOrderId生成挂单订单号makerOrderId（挂单唯一标识）

//3、根据挂单者地址makerAddress查询昵称nickname和信用分creditScore

//4、计算未锁死币数量unlockedAmount

//5、计算当前可售数量availableAmount

//6、从个人钱包makerAddress往合约地址contractAddress转币，即transfer操作，数量为totalAmount

If（卖家交易金和保证金往合约地址transfer转币成功）{

MakerOrder makerOrder = MakerOrder(makerAddress, nickname, creditScore，price，totalAmount，availableAmount, 0，paymentTypeBankCardNoMap，100,100，totalAmount，makerBlockHeight, makerOrderId，0)；

makerOrders[makerOrderId] = makerOrder;

return makerOrder;

}else{

return 0;

}

}

//创建一个吃单订单

function addTakerOrder(uint256 makerOrderId, address takerAddress, uint256 price, uint256 amount) returns (TakerOrder){ //加锁或加可重入锁？

if（购买数量小于1000或不是500的整数倍）{ //购买数量太少或不是500的倍数，不允许购买

return 0;

}

if（挂单中吃单者数量大于1000）{//同一个挂单中吃单列表太长了

return 0；

}

//1、获取当前块的高度startBlockHeight

//2、generateNewOrderId生成吃单订单号takerOrderId（吃单唯一标识）

//3、根据吃单者地址takerAddress查询昵称nickname和信用分creditScore

//4、calculateTotalValue计算实际支付总金额totalValue （误差控制在[-50, 50]美分）

//5、从个人钱包takerAddress往合约地址contractAddress转币，即transfer操作，数量为amount

If（买家保证金往合约地址transfer转币成功）{

MakerOrder makerOrder = makerOrders[makerOrderId];

mapping(bytes32=> TakerOrder) takerOrders = makerOrder.takerOrders;

TakerOrder takerOrder = TakerOrder(takerAddress, nickname，price，amount， totalValue，0, 0, 0, startBlockHeight, takerOrderId，0)；

takerOrders [takerOrderId] = takerOrder;

return takerOrder;

}else{

return 0;

}

}

//根据保证金比例计算当前可售数量（默认100%）

function refreshAvailableAmount() returns (uint256){

availableAmount = unLockedAmount /2;

return availableAmount;

}

//计算未锁死币数量, 初始转入时所有的币均未锁定

function refreshUnlockedAmount (uint256 totalAmount) returns (uint256){

unlockedAmount= unlockedAmount + totalAmount;

return unlockedAmount;

}

//获取循环增加值，每次randomValue都在原有基础上+1，超过50，赋值为-50

function getLoopAndIncValue () returns (uint256){

randomValue = randomValue + 1;

if (randomValue > 50){

randomValue = -50;

}

return randomValue;

}

//生成订单号，每次都在原有基础上+1

function generateNewOrderId() returns (uint256){

orderId = orderId + 1;

return orderId;

}

//计算实际支付总金额

function calculateTotalValue(uint256 price, uint256 amount) returns (uint256) {

uint256 result = price \* amount + getLoopAndIncValue(); //注意美元/美分单位变换

return result;

}

///买家已汇款

function buyerPayment (uint256 makerOrderId, uint256 takerOrderId) returns (uint){

}

///卖家确认收款

function sellerPaymentConfirm (uint256 makerOrderId, uint256 takerOrderId) returns (uint){

if（合约地址往买家地址transfer转币成功）{

//1、计算未锁定币的数量unlockedAmount，新增保证金释放

//2、重新计算可售数量calculateAvailableAmount

}

}

///取消订单

function cancellOrder (uint256 makerOrderId, uint256 takerOrderId, CancelState cancelState) returns (uint) {

if（双方取消）{

if（合约地址往买家地址transfer转币成功）{

//1、计算未锁定币的数量unlockedAmount，新增交易金和保证金释放

//2、重新计算可售数量calculateAvailableAmount

}

}

}

///挂单赎回指定金额

function revokeOrder (bytes32 makerOrderId, uint256 amount) returns (uint){

if（可赎回币数量足够）{

if（合约地址往卖家地址transfer转币成功）{

//1、重新计算未锁定币的数量unlockedAmount，卖家提取币

//2、重新计算可售数量calculateAvailableAmount

}

}

}

}

**contract** **ZJB** { ///不需要遵循ERC20标准

address tokenAddress;

}

用户基本信息存储方案

1. 买家或卖家昵称 nickname
2. 信用分 creditScore（链上查询的）
3. 收付款方式（存在多种收付款方式和账户，map，key为收付款方式，value为账号列表）

方案1：保存至中心数据库服务器----好处是不需要消耗gas

方案2：保存至tag（K，V）键值对，我们自己链实现的，速度相对于个人合约速度更快

方案3：保存至个人账户合约

涉及接口

1. 查询用户基本信息(比如：自定义购买/出售页面，我要买/卖中划转保证金页面，我的-收付款方式-已添加银行卡页面，确认已汇款弹框)
2. 新增收付款方式（比如：添加银行卡页面）

struct UserInfo{

address userAddress;

string nickname;

uint256 creditScore;

mapping(string => BankCard [5]) public paymentTypeBankCardMap;

}

struct BankCard{

string paymentType;

string userName;

string bankCardNo;

string bankName;

string bankNameAddress;

}