

撮合引擎：交易所是怎么把买单卖单凑成一笔交易的

从买一卖一到亲手写一个交易所——订单簿、撮合规则与下单方式全解剖

王建硕

费曼式写法 · 由多个 AI 代理撰写与互相审校

导读

从买一卖一到亲手写一个交易所——订单簿、撮合规则与下单方式全解剖

先别碰代码，先想想菜市场

这本书有一个大胆的承诺：读完它，你能亲手写出一个交易所的撮合系统。

听起来像吹牛。交易所——纳斯达克的机房、每秒几十万笔的订单、动辄上亿的成交额——怎么看都不像“读完一本书就能写”的东西。但这个承诺是认真的，因为交易所的核心远没有它的名声吓人。剥掉西装、屏幕和监管文件，剩下的那台机器只做两件事：**记账，和按规则排队**。它更像菜市场管理员立起的那面挂单墙，而不是什么金融黑魔法。

真正的难点在别处：规则有几十条，每一条都简单，但每一条都必须写对——而且在每秒几十万张订单的冲击下依然条条写对。“买一卖一为什么永远隔一条缝”“为什么改单要重新排队”“成交价为什么总取先到那张单的价”“为什么撮合引擎几乎都是单线程”——这本书不肯定义，只回答这种“为什么必须这样，不这样会出什么事”。因为写系统的人不缺API文档，缺的是对每个规则背后道理的直觉。

这本书怎么带你走

我们用十二章走完一条路：前四章凑近那面墙，把订单簿上每个数字的来历讲清楚；第五章上手术台，把一次撮合拆成逐秒的慢动作，那是全书的技术核心；第六到八章把视野拉开——五花八门的下单方式、开盘定价的集合竞价、A股美股期货币所的规则对照；第九到十一章收拢成工程：成交之后的消息网、五十行代码的内核、单线程定序与断电重放；最后一章装上护栏，把一台实验室玩具变成能托付真金白银的机器。

每一章可独立读，但连起来是一条线：从**“交易所是什么”**到**“我现在就会写一个”**。书末你会得到一张完整的架构图纸，每个零件为什么长这样，你都亲眼看着它长出来的。

适合谁

默认你是工程师或理工背景的读者：爱因果、爱类比、烦空话。不需要任何金融背景——每个名词第一次出现都会当场用大白话解释；也不灌你行业黑话——能不用的词一律不用。如

果你只是想弄懂"我下的单到底经历了什么"，前八章够了；如果你真要动手写，十章以后是你的施工图。

翻开第一章吧。我们从一面墙开始。

第一章 · 一面挂单墙

先忘掉K线图

提到交易所，大多数人脑子里冒出的是红绿闪烁的K线、满屏跳动的数字、西装革履的交易员。这些全是表象。我们要造的那台机器，跟K线一点关系都没有——K线只是它吐出来的日志的一种画法。

剥到最后，一个交易所只做两件事：**记账**，和**按规则排队**。就这两件。剩下的全部复杂性，都是这两件事在真实世界里撞出来的褶子。这本书的目标很具体：读完它，你能坐下来，从零写出一个能跑、能成交、能经得起真实订单捶打的撮合系统。所以每一章我们都不满足于"是什么"，而是追问"为什么必须这样，不这样会出什么事"。

菜市场里那面墙

想象一个卖苹果的菜市场。没有交易所的时候，交易靠双边议价——你走到一个摊位，问多少钱一斤，摊主说五块，你说四块五，成交。这个价格只有你们俩知道。隔壁摊可能四块就卖了，你不知道，你亏了也没人告诉你。

这种市场有三个毛病。第一，**找不着对手**：你想买的时候，未必有摊主刚好想卖。第二，**价格不透明**：同一时刻市场里有一百个成交价，谁是公道价？没人知道。第三，**说了不算**：摊主答应卖你，转头卖给出价更高的人，你没办法。

现在市场管理员出来说：别吵了，立一面墙。想买的，把"我愿出什么价、买多少斤"写在纸条上贴左边；想卖的，贴右边。墙对所有人公开。规则只有一条：**左边的最高价和右边的最低价一旦碰上，就必须成交**，谁也不许反悔。

这面墙，就是交易所的全部本质。它的学名是订单簿（order book）——一本公开的账本，记着"谁在什么价位想干什么"。而那个不停盯着墙、发现左右两边价格一碰上就宣布"成交"的管理员，就是撮合引擎（matching engine）——我们要写的那台机器。

大白话：交易所不是一个“地方”，也不是一堆屏幕。它就是一面公开的墙，加一个铁面无私、只认规则的管理员。你写的撮合引擎，就是那个管理员。

为什么“公开”和“不许反悔”这么重要

值得停下来想一想，这面墙凭什么比摊主吆喝强。

因为它把三个毛病一次治了。想买的和想卖的都往同一面墙上贴，**对手盘集中了**——这叫流动性聚合，说人话就是：想成交的人都来同一个地方排队，互相找得着。墙对所有人公开，**价格透明了**——此刻最公道的买价卖价就写在墙的最顶上，谁都能看。纸条一旦贴上、规则一旦启动，成交是机器执行的，**谁也反悔不了**。

注意第三个好处最容易被忽略，但它恰恰是交易所能收钱的根本原因：人们愿意把订单交给它，是因为相信它会**铁面无私地按规则办事**——不会因为你是谁就让你插队，不会因为行情剧烈就偷偷改规则。所以贯穿这本书的一条暗线是：我们写的每一行撮合逻辑，本质上都在捍卫这个信任。规则写错了、实现有歧义了、同一笔订单在不同时候得到不同待遇了——信任就没了，这比亏一笔钱严重得多。

墙的两半边，就是“买一”和“卖一”

墙上最顶上的两张纸条，是全天最重要的两个价格。

左边（想买的人）出价最高的那张，叫买一——best bid，此刻市场上有人愿意出的最高买价。右边（想卖的人）要价最低的那张，叫卖一——best ask，此刻市场上有人愿意接受的最低卖价。

正常情况下，买一一定比卖一**低**。这不难想通：如果最高买价是10块，最低卖价是9块，那管理员早就该宣布他们成交了——他俩的存在本身就是矛盾的。所以只要管理员还在干活，墙上的买一和卖一之间永远留着一条缝。这条缝叫价差（spread），它是这本书后面很多章的主角。

大白话：买一和卖一不是行情软件编出来的数字，它们就是墙上最顶上的两张纸条。价差就是买卖双方还在僵持的距离——谁也不肯再让一步。

从墙到机器，只换了三样东西

真实的交易所把这面墙搬进了机房，搬的过程中换了三样东西：

- **纸条换成了订单**。一张结构化的指令：谁、买什么、多少、出什么价、还有些什么附加条件。第六章会把这个"附加条件"翻个底朝天。
- **管理员换成了撮合引擎**。一段代码，每来一张新订单，就按固定规则检查它能不能跟墙上的旧订单成交，能成交多少、按什么价格成交，剩下的怎么办。
- **"大家围观看墙"换成了行情广播**。墙上每动一下，机器就把变化推给所有订阅者。你在行情软件里看到的买一卖一，就是这个广播的远方回声。

换完这三样，本质一点没变。这也是为什么这本书敢承诺"读完你能写一个"：要实现的从来不是华尔街的神秘黑箱，而是一台规则完全公开、行为必须可复现的排队机器。它的难点不在"想不到"，在于**规则有几十条，每条都得写对，而且在每秒几十万张订单的冲击下依然条条写对**。

这本书怎么走

我们会从墙上最顶上的两张纸条出发。第二章先把订单簿上的每个数字——买一卖一、深度、最小价位——讲清楚它们从哪来；第三、四章分别解剖两种最基本的订单：愿意排队的限价单，和不惜代价立刻成交的市价单；第五章把一次撮合拆成逐秒的慢动作，那是全书的技术核心；第六章把真实交易所支持的所有下单方式过一遍；第七章讲开盘收盘时用的另一套玩法——集合竞价；第八章对比A股、美股、期货、加密货币交易所的规则差异，让你知道哪些是"物理定律"、哪些只是"地方法规"；第九章讲成交之后系统还要干的事；第十、十一章挽起袖子写代码，从数据结构到崩溃恢复；最后一章讲上线前必须装上的护栏——涨跌停、熔断、风控。

下一章，我们凑近那面墙，看清楚上面的每一行字。

第二章·订单簿：买一卖一是从哪冒出来的

把第一章那面墙拍张照

第一章结尾，管理员把墙立起来了。现在我们来读这面墙——你打开任何一个行情软件，看到的"盘口"，就是这面墙的一张快照。

一个真实的盘口长这样（以某只股票为例）：

卖五 10.05 800股

卖四 10.04 300股

卖三 10.03 1200股

卖二 10.02 500股

卖一 10.01 600股

—————

买一 10.00 1000股

买二 9.99 2400股

买三 9.98 700股

买四 9.97 900股

买五 9.96 1500股

中间那道横线，就是第一章说的"僵持的缝"。横线以下是想买的人在排队，出价从高到低；横线以上是想卖的人在排队，要价从低到高。离缝最近的两行——买一和卖一——是全市场的注意力中心，因为下一笔成交几乎必然发生在它们俩身上。

每个数字回答一个问题

盘口上的每样东西都对应一个交易者真正关心的问题，我们一个个过。

买一卖一回答"我现在立刻成交，能拿到什么价"。你想立刻买入，就得接受卖一的价10.01；想立刻卖出，就得接受买一的价10.00。"立刻"的代价，就是吃那一分钱的缝。这

个缝——卖一减买一——叫价差 (spread)。它同时是这个市场"贵不贵"的体温计：缝越小，说明买卖双方咬得越紧，进出成本越低。大盘股、主流币的缝常常只有最小一跳；冷门资产的缝可能宽到几个百分点，你买进来再立刻卖出去，光缝就亏一截。

每一行后面的数量回答"我能成交多少"。买一的1000股意思是：在10.00这个价位上，总共排着1000股的买单。你要卖500股，它们全按10.00接住；你要卖5000股，前1000股按10.00成交，剩下的就得去找9.99的买二、9.98的买三……一路往下啃。这个"每个价位上堆了多少量"的纵向画面，叫深度 (depth)。深度厚，大单砸进来价格也稳；深度薄，一笔不大的单就能把价格打穿好几格——第四章讲市价单时，这会变成真金白银的教训。

大白话

大白话：买一卖一是"缝有多宽"，深度是"墙有多厚"。宽度决定你立刻进出的成本，厚度决定你的大单会把价格砸出多大的坑。

价格的台阶：为什么没有人挂10.005

看上面的盘口，所有价格都停在一分钱的整数倍上。这不是巧合，是规则。交易所规定价格只能按固定的格子走，这个格子叫最小价位 (tick size) ——价格变动的最小单位。A股是0.01元，美股是0.01美元（低价股另有规定），期货每个品种有自己的tick，比如螺纹钢期货是1元/吨。

为什么要立这条规矩？想象一下没有它会怎样：你挂10.00买，我挂10.001，比你高千分之一就排到你前面；下一个人挂10.0001……加价可以无限小，"比别人好一点"变得毫无成本，排队就失去了意义，价格之间的缝也会被磨到无穷小，谁都没法判断真实的市场意愿。tick size相当于规定了"**插队必须付的最低票价**"：你想排到我前面，至少得多掏一分钱。这一分钱让"改善价格"成了一次真实的表态，而不是免费的挤占。

同样，数量方向也有格子，叫最小数量 (lot size) ——比如A股最少买100股（一手），科创板另有规定；加密货币交易所通常允许0.001个比特币这样的碎单。你写的撮合系统收单时第一件事，就是校验价格和数量是否落在格子上——不在格子上的订单直接拒掉，别让脏数据进门。

同一价位上，还藏着一支队伍

盘口把每个价位显示成一行总量，但"买一 10.00 1000股"这1000股背后，可能是一个挂的1000股，也可能是十个人各挂的100股。撮合引擎眼里，这一行不是一个总数，而是一支**按到达先后排好的队**。有人在这个价位卖出时，先到的买单先成交，后到的继续排——这就是第三章的主角"时间优先"。

这也是为什么有些交易者愿意花大钱看更深的数据：普通行情只告诉你每个价位多少总量，而逐笔数据（第九章会讲）能告诉你每一张单的到达顺序，让你算出自己的单排在这支队伍的第几个。

订单簿其实是一棵树加一个队列

现在把这章的画面翻译成你后面要写的数据结构。一个订单簿 = 两个按价格排序的容器：买单簿按价格从高到低，卖单簿按价格从低到高。每个价格档位里，是一个按时间先后排队的订单队列。最顶上的档位就是买一和卖一。

先不用管用什么具体的数据结构实现（第十章专门讲），你只需要记住这个形状：**横向是一格一格的价格台阶，纵向是每格里面的一支队伍**。横着比价格，竖着比时间。整个撮合世界的规则，都长在这个形状上。

下一章我们就来当一回排队的人：看看"价格优先、时间优先"这八个字，到底怎么决定你的单子什么时候轮到。

第三章·限价单与排队的艺术

两种人，两种命

面对买一10.00、卖一10.01的那道缝，每个交易者只有两种姿态：要么**跨过缝**，接受对方的价格立刻成交；要么**挂进队伍**，报一个自己的价格，等别人来跨。这一章讲第二种人用的武器——限价单（limit order）："我要买，但我最多出10.00，一分钱不多给。"

限价单是订单簿的建筑材料。墙上贴的每一张纸条都是限价单，买一卖一、深度、价差，全是限价单堆出来的。理解撮合，一半是理解限价单怎么排队。

八个字的铁律：价格优先，时间优先

全世界的连续竞价交易所，排队规则归根结底就八个字：**价格优先，时间优先**。

价格优先：买单里出价高的排在出价低的前面；卖单里要价低的排在要价高的前面。所以"买一"永远是当前最高买价——谁出得高，谁站墙顶。这条几乎没争议，因为它就是"拍卖"的直觉。

时间优先：同一个价格上，先到的先成交。10.00元上排着三张买单：甲的500股（9:30:01到）、乙的300股（9:30:02到）、丙的1000股（9:30:05到）。这时来一张卖出600股的市价单，成交价10.00，前500股给甲（甲的单全部成交、离开队列），剩下100股给乙（乙的单成交100股、还剩200股继续排头）。丙的1000股一股没轮上——哪怕他的单子最大。

这套规则合起来叫价格-时间优先（price-time priority），也叫FIFO（先进先出）。它是目前主流交易所（A股、美股、期货、绝大多数加密货币交易所）的默认规则。也有例外——比如部分期货市场用按比例分配（pro-rata），同价位的单按大小分蛋糕——但那是少数派，本书以FIFO为主线。

大白话：横向比价格，谁的条件好谁站前面；同价竖着比时间，谁先到谁先走。队列头的人永远先成交，后面的人只能等。

为什么你的单子总是排在别人后面

理解了规则，再看真实盘口，你会产生一种憋屈：你想在10.00买，发现前面已经排了2400股；等轮到你，黄花菜都凉了。你加价到10.01抢跑？那你已经跨过了缝，等于直接买卖一，排队就白排了。

这就是限价单的核心博弈，一笔明算账：

- **挂单（排队）的好处**：你按自己的价格成交，不吃价差。买在买一、卖在卖一，一进一出比对手省一个缝。很多交易所还给挂单方手续费返佣（第八章讲maker/taker），等于交易所贴钱请你来排队。
- **挂单的代价**：你可能**成交不了**。价格掉头走了，你的单在墙上风干。更微妙的是一种叫**逆向选择**的风险：你的买单终于轮到成交的那一刻，往往是有聪明人急着卖出的那一刻——他为什么急着卖给你？也许他知道一些你还不知道的事。排队排到了，不一定是好事。

所以“挂在哪里”是一门权衡：挂得离缝越近，成交概率越大，但被聪明人“挑中”的逆向选择风险也越大；挂得越远越安全，但可能永远轮不到。做市商（专门靠双边挂单赚价差的人）整天就在算这笔账。

改单=重新排队，这是很多人不知道的坑

你挂了10.00买1000股，排在队尾。过了一会儿你想改成1005股——只加5股，队位还在吗？

不在了。绝大多数交易所的规则是：改数量（部分市场改小不改大除外）、改价格，都视为撤掉旧单、挂入新单，时间戳重新算，回到队尾。唯一普遍不改队位的操作是**减少数量**（只减不增），因为你没有占任何人便宜。

这条规则的逻辑很硬：队位是用“先到”换来的资产。如果允许改大数量还不丢队位，我可以用一张1股的小单占住队首，等行情来了再悄悄改成100万股——等于免费插队。堵死这个

漏洞的办法就是一刀切：**任何让订单变得更"占便宜"的修改，都重新排队。**

大白话

大白话：队列位置是用到达时间换来的。你可以主动放弃（撤单、改单），但系统绝不允许你拿着旧队位换新条件。写撮合系统时，cancel/replace 基本等于 cancel + new。

这章给写系统的人留下了什么

三个必须实现的机制，先记下来：

1. 每个价格档位内部是一条严格按到达顺序的**FIFO队列**，撮合永远从队首开始。
2. 撤单是把订单从队列里摘出去；**改价和增量改单 = 撤旧 + 挂新**。
3. 部分成交后，剩余部分**保留原队位**继续排（前面甲成交完离场、乙剩的200股继续排头，就是这个规则）。

下一章换边站：看看不愿意排队的人——市价单——是怎么跨过那道缝的，以及跨过去的时候会发生什么。

第四章 · 市价单：不惜代价立刻成交

不愿排队的人

限价单说："这个价，爱来不来。"市价单说："**不管什么价，现在、立刻、马上给我成交。**"

市价单（market order）是一种不填价格的订单：你只告诉交易所买还是卖、多少量，价格完全交给市场此刻的状态决定。它的撮合方式简单到粗暴——买单直接从卖一开始往上吃，卖单从买一往下吃，吃到量够为止。这个动作有个形象的名字：扫单（walking the book），像一把扫帚从墙顶往下扫。

拿第二章的盘口演示。当时卖一10.01有600股，卖二10.02有500股，卖三10.03有1200股。现在来一张"市价买入1500股"：

- 先吃掉卖一：600股 @ 10.01（卖一这一格清空）；
- 不够，继续吃卖二：500股 @ 10.02（卖二清空）；
- 还差400股，吃卖三：400股 @ 10.03，卖三剩800股。

一张单，三个价，总共1500股，均价10.0167。成交完，新的卖一变成了10.03——**这一张单把整个市场的报价抬高了两格。**

滑点：你以为的价格和你拿到的价格

下单那一刻你看到的卖一是10.01，最后均价却是10.0167。这个差距叫滑点（slippage）——预期成交成本和实际成交成本之间的差。

滑点不是bug，是物理。墙上的量就那么多，你要的多于顶格的量，就只能往深处吃，越深越贵。滑点的大小由第二章说的"深度"直接决定：深度厚，1500股可能一格都吃不完，滑点是零；深度薄，同样一张单能把价格顶出去老远。

大白话：市价单买的是一个"确定的成交"，放弃的是"确定的价格"。限价单反过来：价格确定，成交不确定。每个下单动作都是在这两样东西之间做取舍。

薄市场里的惨案

滑点在极端情况下会变成灾难。想象一个冷门币，卖一10.01只有50个，卖二直接跳到15.00，再往上是20.00。你市价买入200个，拿到的均价是16块多——比屏幕上看到的价格贵了60%。你看到的10.01，只买到了50个。

真实世界里这类事故一再发生。2010年5月6日美股"闪崩"（Flash Crash），几分钟内道指暴跌近千点又拉回，事后调查报告的核心情节之一，就是大量市价单和止损单在瞬间被抽干的订单簿上横扫，有的股票在几分钱的价格上成交——不是因为有人觉得它就值几分钱，而是那一刻墙上真的没有买单了。

所以交易所普遍给市价单上镣铐，常见三种：

- **价格保护带**：市价单最多只能扫过N个价位，超出部分撤掉或转为限价单挂在保护带上。
- **市价转限价**（market-to-limit）：扫完第一格后，剩余部分自动变成按最后成交价挂着的限价单。欧洲不少交易所默认这么干。
- **干脆不提供市价单**：A股长期以来就没有纯粹的市价单，只有"最优五档即刻成交剩余撤销"这类带边界的变体——本质就是市价单加保护带。

你自己写交易所时，至少要实现其中一种。裸奔的市价单是一颗留给你用户的炸弹，也是留给自己的。

市价单存在的理由

既然这么危险，为什么还有市价单？因为有一种情况下"确定成交"比"价格好"重要得多：**你必须离场的时候**。爆仓强平、风险急刹、止损出逃——这些场景里，晚一秒的代价远大于一个差的成交价。市价单是逃生通道，逃生通道不看票价。

这也是为什么做市商喜欢市价单：对方慌不择路跨缝而来，自己稳稳坐在买一卖一等着接，价差和滑点都进了自己口袋。市场里每一次"立刻"，都有人在对面收"立刻"的钱。

给写系统的人

市价单在撮合引擎里的实现要点，一句话说透：**它不是"价格为0或无穷大"的限价单，虽然这么实现也能跑**——买单限价设为系统最高价、卖单设为0，然后走正常撮合循环。但你要额外处理两件事：扫到墙尽头还有剩余怎么办（撤掉？转限价？报错？——这是产品决策，要显式做）；以及保护带逻辑插在哪里（每成交一格检查一次偏离）。这些分支在第五章的完整解剖里会看到它们的位置。

下一章是全书的手术台：我们把"一张新订单进门"之后的每一微秒拆开，看撮合循环到底长什么样。

第五章 · 一次撮合的完整解剖

上手术台

前四章把零件都摆出来了：订单簿是价格台阶加排队，限价单排队，市价单扫墙。这一章把它们装在一起，回答全书最关键的一个问题：**一张新订单进门之后，撮合引擎的每一行逻辑到底做什么？**

答案是一段循环，行话叫撮合循环（matching loop）。先记住它的伪代码，然后我们逐行解剖：

```
while 新订单还有剩余量:
    看对面墙顶（买单看卖一，卖单看买一）
    if 对面没单了，或价格碰不上: break
    和队首那张单成交，价格取对面那张单的价
    成交量 = min(双方剩余量)，双方各扣掉
    对面那张如果扣完，从队列移除，换下一张队首
循环结束后:if 新订单还有剩余: 按规则处理剩余
```

就这些。全部撮合引擎——从纳斯达克的到一个小币所的——核心都是这段循环的变体。魔鬼在四个细节里。

细节一：什么叫“价格碰得上”

对限价买单，“碰得上”= 卖一价 $\leq$ 你的限价。你限价10.02买，卖一10.01，碰上；卖一涨到10.03，碰不上，循环停。限价卖单对称：买一价 $\geq$ 你的限价。市价单没有限价，永远碰得上，直到对面被扫空。

注意这个判断是在循环**每一圈**重新做的。你限价10.02买1500股，10.01吃了600、10.02吃了500，下一格10.03——碰不上了，循环停，剩400股按10.02挂进买方队列，成为新的买二。限价既是你的成交条件，也是你的刹车。

细节二：成交价取谁的价——为什么总是"墙上那张"的价

上面伪代码里加粗的那句，是整个撮合里最容易被写错、也最体现规则哲学的地方：**成交价取被动方（先挂在墙上的那张单）的价格**，不取新来那张的价格。

例子：卖一10.01挂着，你限价10.02买。成交价是10.01，不是你出的10.02。你本来愿意多付一分，市场只收你10.01——你占了便宜。反过来，你挂10.00买在墙上，有人市价卖下来，成交价10.00，你的价。

为什么必须这样？两个理由。一是**奖励提供流动性的人**：先把订单挂上墙的人承担了风险和等待，规则把"价格改善"的好处给后到的主动方，墙才有人愿意贴。二是**确定性**：墙上那张单的价格是公开的、先来就定的，成交价取它，所有人事前都能算出自己会按什么价成交；如果取新单的价，等于成交价由后来者临时决定，墙就失去了承诺的意义。

大白话

大白话：成交价永远由"先到的、挂在墙上的那张单"说了算。后到的人要么接受，要么不成交。这让挂单者的承诺是硬承诺，也让规则简单到没有第二种解释。

细节三：部分成交是常态，不是异常

新单1500股，对面队首只有600股——成交600，对方离场，你的单剩900继续循环。反过来，对面队首有5000股，你1500股全部成交离场，对方剩3500股**保留原队位**继续排头（第三章的规则）。

所以一笔订单的"生命周期"不是"成交/没成交"两态，而是：剩余量从初始一路被削，每削一刀产生一笔成交记录（trade）。一张大单打进来，可能产生几十笔成交，每笔的对手、价格、数量都不同。你的系统里，**成交记录和订单是两张表**：订单记录"还想要多少"，成交记录记"已经发生了什么"。这个区分到第九章讲回报流时还会回来。

细节四：剩余怎么办——这里分出所有订单流派

循环停下来时，新订单可能还有剩余。怎么办？这不是一个答案，而是一组答案，每种答案是一种订单类型（第六章详述）：

- **剩余挂进簿子**：变成普通限价单排队——这是默认行为（GTC）。
- **剩余撤掉**：能成交多少算多少，剩下的不要了（IOC）。
- **要么全成要么全退**：进门前先算一遍够不够全吃，不够直接拒绝（FOK）。
- **不许成交，只许挂**：进门时若能成交，直接拒单（post-only）。

看见了吗：所谓"订单类型"，绝大多数只是撮合循环结束条件和对剩余量的处理方式不同。核心循环不用改。这是写撮合引擎时最值得领悟的一点——先写死一个干净的循环，再让订单类型成为循环的参数。

一个完整算例，逐秒走完

卖侧簿子：10.01有A的600股（先到）、B的400股；10.02有C的500股。买侧先不管。现在D下单：**限价10.02买入1200股**。

1. 循环第1圈：卖一 $10.01 \leq 10.02$ ，碰上。队首A，成交600股 @ **10.01**（A的价）。A离场。D剩600。
2. 第2圈：卖一还是10.01（B顶上成为队首）。成交400股 @ 10.01。B离场。D剩200。
3. 第3圈：卖一变成10.02， ≤ 10.02 ，碰上（限价是"最多出"，相等也算碰上）。队首C，成交200股 @ **10.02**。D剩0，循环结束。
4. D三笔成交：600@10.01 + 400@10.01 + 200@10.02，均价约10.0117。C的卖单剩300股继续排10.02队首，新卖一仍是10.02。

把这段在脑子里跑通，你就已经理解了一个撮合引擎80%的工作。

细节五：自己跟自己成交，行吗

还剩一个工程上必须回答的问题：如果一个账户的买单撞上了它自己的卖单呢？这叫自成交（self-trade）。听起来无害，但它能被用来刷成交量、制造假行情（对倒，wash trading），在受监管市场是明令禁止的。所以撮合循环里通常加一道自成交防护（STP）：发现对面队首是同一个人的单，按预设策略处理——撤新单、撤旧单、或两边都撤。这又是一颗挂在循环上的可选参数。第八章对比各交易所时会看到它们的不同选择。

循环讲完了。下一章我们把这些"参数"一个一个装上，看看真实交易所提供的下单方式全家桶。

第六章 · 下单方式全家桶

订单类型不是新引擎，是引擎的旋钮

第五章结尾埋了个伏笔：绝大多数"订单类型"只是撮合循环的参数。这一章把真实交易所的下单方式过一遍。每讲一种，我们都问同一个工程问题：**它改了循环的哪一部分？**看完你会发现，一张看起来吓人的下单表单，拆开来不过四五组旋钮。

第一组旋钮：剩余量怎么办（时效指令）

第五章说过，循环停下来时的剩余处理分出了流派。这组指令叫时效指令（time-in-force），回答"这张单活多久、剩了怎么办"：

- GTC（good-till-cancelled，撤前有效）：默认款。剩余挂进簿子一直排，直到你撤掉。有的交易所给GTC加个期限变体GTD（到某日有效）。
- IOC（immediate-or-cancel，即刻成交剩余撤销）：进撮合循环，能成交多少算多少，**剩余立刻撤掉，一秒都不排**。用法：我想要这个深度里的量，但不想要排队的不确定性。
- FOK（fill-or-kill，全成或全退）：进循环**之前**先算一遍——对面可成交量够不够我全吃？不够，整张拒绝，一股不成交；够，才进循环。用法：大额换仓，宁愿不做也不愿做一半。

工程实现：GTC什么都不用加；IOC是循环结束后把剩余丢弃而非入簿；FOK是循环前加一次"干跑"（只算不成交），量够再真跑。FOK有个坑：干跑和真跑之间不能有别的订单插进来改变簿子——这逼你接受一个原则，**撮合必须串行**，这是第十一章单线程架构的第一块基石。

第二组旋钮：只许挂，不许吃

Post-only（只做挂单方）：订单进门时检查——如果它此刻会成交，**直接拒单**（有的交易所选择把价格自动挪开一格）。为什么有人要这个？因为交易所给挂单方（maker）手续费返佣、向吃单方（taker）收费（第八章详谈），大玩家绝不允许自己"一不小心成了taker"——那等于本来收钱变成交钱。Post-only是把"我一定排队"写成了硬约束。

第三组旋钮：先藏着，等信号——条件单

前面所有单都是进门就进撮合。止损单（stop order）不一样：它进门后**先睡一觉**，不进簿子，只登记一个触发价。等市场最新成交价碰到触发价，它才醒来，变成一张普通订单冲进撮合循环。醒来的姿势又分两派：

- **止损市价单**：触发后变市价单，保证成交，不保证价。跌到止损位时往往很拥挤，实际成交价可能比触发价差不少。
- **止损限价单**：触发后变限价单，保证价格上限，不保证成交——价格一泻千里时你的限价单可能根本追不上，止损没止住。这个风险要自己想清楚。

工程实现：撮合引擎旁边多一个触发器——每笔成交产生后，拿最新价去扫一遍沉睡的止损单列表，把被触发的唤醒成普通订单重新进门。注意触发源通常是"本所最新成交价"，这就有一个微妙的顺序问题：一笔成交既改变簿子，又可能唤醒新订单，新订单再改变簿子——事件链条必须严格按序处理，又是串行的理由。

第四组旋钮：别让人看见我的底牌——冰山单

大单挂出来会吓坏市场：墙上突然出现10万股的买单，所有人都会调整行为。冰山单（iceberg order）的办法是：总量10万股，但墙上只露1000股（叫"峰值量"）；露出来的1000股被吃掉，引擎自动从水下再补1000股上来，直到总量耗尽。露出来的部分正常排队，水下部分别人看不见。

工程实现上有两个点要想清楚。一是**补单时的队位**：新补的1000股是一张"新单"，按第三章的规则应该重新排到这个价位的队尾——所以冰山单在排队上是吃亏的，这是隐藏自己的代价。二是成交优先级：同一价位上，完全显露的单和冰山的显露部分怎么排，各交易所规则不同，写之前要定死。

一张表收拢全章

GTC → 剩余入簿；IOC → 剩余丢弃；FOK → 进门前先干跑验量；Post-only → 进门
前检查是否成交，成交则拒；止损 → 先睡，被成交价唤醒再进门；冰山 → 入簿只入峰
值，吃光自动补。

六种旋钮，没有一种动了第五章那个核心循环。它们改的是：进门前的检查（FOK、post-only）、进门的时间（止损）、入簿的量（冰山）、剩余的去向（IOC）。**这就是设计订单类型的方法论：新类型必须能表达成对现有流程的一个明确修改点，说不清楚的类型不做。**

真实交易所的表单里还有更花哨的——移动止损、OCO（二选一单）、TWAP切片——但它们要么是上面旋钮的组合，要么是交易所之外的策略层（客户端或券商系统）做的事，不属于撮合引擎本体。分清"引擎该管的"和"策略层该管的"，是交易所架构里最重要的边界之一。

下一章换个场景：当市场还没开门、没有买一卖一的时候，第一笔价格是怎么产生的？

第七章·集合竞价：开盘价是投票投出来的

没有缝的时候，第一笔价格从哪来

连续竞价有个隐含前提：墙上已经有人挂单，有买一有卖一，新订单来了就有对手。但每天早上9:25（A股开盘前），墙是**空的**——或者更准确地说，隔夜攒了一堆互相碰不上的订单，谁也不知道"此刻的公道价"是多少。这时候第一章那条"碰上就成交"的规则没法直接开跑：先放谁进来？按什么价？

交易所的答案不是"赶紧开始连续竞价"，而是换一个玩法：集合竞价（call auction）——**先把所有人的订单收上来攒着，谁都别成交，然后在一个瞬间，按一个统一的价格，一次性全部撮完**。A股早上9:15到9:25就是这样：你下的单进了交易所但不成交，只在9:25:00那一刻，全部按同一个开盘价成交。

核心原则：让最多人成交的那个价赢

集合竞价的定价规则叫最大成交量原则：找出一个价格，使得在这个价位上能成交的总量最大，它就是开盘价。

算一遍就懂了。假设9:25前攒下的订单是这样：

买单：10.00买1000股 / 9.99买800股 / 9.98买2000股

卖单：9.98卖500股 / 9.99卖1200股 / 10.00卖1500股

（注意：这里的买单限价是"最多出多少"，卖单是"最少收多少"，所以10.00的买单愿意按任何 ≤ 10.00 的价格成交。）现在逐个候选价格试算"如果开盘价定在这里，能成交多少"：

- 定9.98：愿意接受的买单=全部三档（都愿出 ≥ 9.98 ），共3800股；愿意接受的卖单=9.98的500股。成交= $\min(3800, 500)=500$ 。
- 定9.99：买单=10.00档+9.99档=1800股；卖单=9.98档+9.99档=1700股。成交= $\min(1800, 1700)=1700$ 。

- 定10.00：买单=10.00档=1000股；卖单=全部3200股。成交=**1000**。

9.99能成交1700股，最多——**开盘价定为9.99**。所有被撮中的订单，不管限价多少，一律按9.99成交：出10.00的买方少付了一分，要9.98的卖方多收了一分。统一价格对所有人都是"不比你要求的差"，这就是它能服众的原因。

大白话

大白话：连续竞价是"随到随配"，集合竞价是"攒一波、民主投票"——哪个价格能让最多订单成交，哪个价格当选，当选后所有人按同一价格成交，没人吃亏。

平票怎么办

如果两个价格的最大成交量一样呢？规则会一层层往下加裁判（各交易所措辞略不同，思路一致）：

1. **未匹配量最小**：选成交完之后，墙上剩的未成交差额最小的价；
2. 还平，**参考价最近**：选离昨收盘价（或某个基准价）最近的；
3. 还平，取中间或按规则取其一。

写实现时这就是一道清晰的算法题：枚举候选价（所有订单限价去重）→ 每个价算买侧累计量和卖侧累计量 → 取min得可成交量 → 按"最大成交量→最小剩余→最近参考价"排序选优。几十行代码。

为什么不一直用集合竞价

既然集合竞价这么"民主"，为什么开盘后就换回连续竞价？因为两者的强项正好错开：

- **集合竞价强在定价**：信息嘈杂、订单稀疏的时候（隔夜消息发酵后的开盘、大事发生后的复牌），把所有人的意愿攒起来一次性定价，比让前两张碰巧到达的订单定出价格要稳得多，也抗操纵——想操纵一个瞬间的统一价，你得扛住全市场的量。
- **连续竞价强在连续**：市场正常运转时，交易者要的是"现在就能成交"，等不起攒单。而且连续成交产生的连续价格本身就是信息，指引新订单进来。

所以成熟市场是两者接力：**开盘集合竞价定开盘价 → 白天连续竞价 → 收盘再来一次集合竞价定收盘价**。收盘价用集合竞价尤其重要——基金净值、指数、衍生品结算都拿收盘价算账，这个价必须是全天最抗操纵的一口价。

工程上的三个坑

1. **集合竞价期间要广播"虚拟开盘价"**：一边攒单一边实时算出"如果现在截止，开盘价是多少、能成交多少"，推给用户，引导大家调整订单。A股盘口的9:15-9:25你看到的价格变动就是它。这个虚拟价是只读计算，不改簿子。
2. **撤单规则分时段**：A股9:15-9:20可撤，9:20-9:25不可撤——最后五分钟锁死，防止有人用大单虚晃一枪再撤掉来操纵开盘价。你的状态机要按时间段切换"能不能撤"。
3. **没撮中的订单去哪**：集合竞价结束后，剩余未成交的限价单按价格-时间优先进入连续竞价的簿子——两套玩法共用同一本簿，交接要干净。

到这里，"一个交易所怎么定价"的两套发动机都齐了。下一章我们走出单个交易所，看看A股、美股、期货、加密货币这些真实玩家的规则差在哪——以及哪些是铁律、哪些只是地方习俗。

第八章·各家交易所的规则差在哪

物理定律与地方法规

前七章讲的都是"物理定律": 订单簿的形状、价格时间优先、撮合循环、最大成交量原则——这些是全世界交易所的公共语言, 换了哪家都一样。这一章讲"地方法规": 同样是撮合, A股、美股、期货、加密货币交易所在参数和规则上的真实差异。你要写自己的交易所, 就得知道哪些旋钮是别人已经替你调好的、哪些得自己定。

最小价位: 一分钱的讲究

第二章说tick size是"插队的最低票价"。各家定法很不一样:

- **A股**: 统一0.01元, 不管股价是2块还是2000块(茅台一股2000元, tick还是一分钱, 相对值小到可以忽略)。
- **美股**: 1美元以上的股票tick是0.01美元; 1美元以下的是0.0001美元。另外美股有个著名的"tick试点"(Tick Size Pilot, 2016年起的小范围实验), 强制部分小盘股用0.05美元的tick, 想看看把缝撑大能不能吸引做市商——监管用真实市场做实验, 这本身就说明tick是个有真实后果的政策杠杆。
- **期货**: 每个品种自己定, 而且tick对应固定的金额价值。比如螺纹钢期货tick是1元/吨、一手10吨, 所以每跳一格一手盈亏10元——交易员直接按"跳"算钱。
- **加密货币**: 每个交易对自己定, 而且通常很细。BTC/USDT的tick常见0.01美元甚至更小, 配合0.00001个的最小数量, 缝可以窄到几乎不存在。

给你的工程启示: tick和lot**必须是每个交易品种的配置项**, 绝不能写死在代码里。

T+0与T+1: 成交和"货到手"是两回事

A股有个让新手迷惑的规则: 今天买的股票, 今天不能卖(T+1); 但期货、可转债、美股、加密货币都是T+0——随买随卖。注意这跟撮合引擎**没关系**: 撮合瞬间就完成了, "T+1"限制的是成交之后的**交收和再卖出权限**, 是结算层和风控层的规则。

这是本章最重要的架构启示：**撮合引擎只负责"谁和谁按什么价成交了"，不负责"货什么时候到账、到账前能不能再用"**。T+0还是T+1、涨跌停多少，都是撮合外围的约束层。把撮合内核和约束层分开，你的系统才能既去服务A股式的市场，也去服务币所式的市场。

自成交防护：三种斩法

第五章说过自成交要被拦。具体怎么拦，各家给用户的选项不同，常见三档（以CME、各大币所的STP为典型）：

- **撤新单** (cancel aggressor)：保留墙上旧单，把新冲进来的撤掉。
- **撤旧单** (cancel resting)：撤掉墙上自己的旧单，让新单继续跟别人成交。
- **两边都撤** (cancel both)：都不留。

STP通常按账户或机构ID判断。实现上就是撮合循环里每次准备成交前查一下对手单的归属，撞到就按模式处理。一个小但有用的细节：STP是**订单自带的参数**，同一张单上写着"如果我撞到自己怎么办"，而不是交易所全局一个开关。

手续费：maker赚钱，taker付费

加密货币交易所把一件事做到了极致：maker/taker费率——挂单被吃的一方（maker，提供流动性）手续费极低甚至是**负数**（交易所倒贴返佣），主动吃单的一方（taker，消耗流动性）付正常费率。比如maker -0.01%、taker 0.05%：你挂的单被别人吃掉，交易所给你钱。

为什么倒贴？因为订单簿的厚度是交易所最核心的资产，深度厚才能吸引大交易者，而大交易者来了做市商才有生意——所以交易所花钱买"墙厚"。美股也早就这样：交易所给挂单方每股返佣零点几美分，"挂单返佣"甚至是一些交易策略的全部利润来源。A股是例外：没有maker/taker之分，买卖都收佣金和印花税，方向还不一样（印花税只在卖出时收）。

给你的启示：费率和返佣是按"**这次成交里你是主动方还是被动方**"逐笔算的。撮合循环的每一笔成交输出里，就得标清楚谁是taker谁是maker——这是计费系统的数据源，事后补标很痛苦。

交易时段与涨跌幅：A股的特色护栏

A股主板有±10%涨跌停（创业板、科创板±20%，ST股±5%），一天就四个小时的连续竞价加两头集合竞价；美股没有个股涨跌停，但有全市场熔断和个股的"涨跌停式"临时停牌机制（LULD）；加密货币交易所7×24小时不眠，大多数没有涨跌停——波动靠市场自己消化，这也是为什么币圈单日-30%不稀奇。

这些护栏怎么实现，是第十二章的整章内容。这里只需要建立观念：**它们的差异巨大，但都不碰撮合循环本身**——涨跌停的本质就是"拒绝/不撮合超出边界的订单"，是收单时的一道校验和簿子的一道闸。

一张对照表

	A股	美股	期货	加密货币
时段	4小时+集合竞价 6.5小时+盘前盘后 日盘+夜盘 7×24			
T+?	股票T+1 T+0 T+0 T+0			
涨跌停	±10%/±20% 无（有LULD） 各品种有 多数无			
市价单	变体有保护 有 有（带保护） 有			
费率	固定佣金+印花税 maker/taker返佣 固定/按额 maker/taker明显			

看懂这张表，你就明白了为什么成熟的撮合引擎都把规则做成配置：内核不变，换一套参数就是另一个市场。下一章我们跟一笔成交走完全程：从撮合成的那一刻起，交易所还要往外发哪些消息、怎么保证谁都不丢。

第九章 · 成交之后：回报、行情与对账

成交只是开始

撮合循环跑完一圈，簿子变了，双方的钱货要划转——但先别急着划。交易所此刻还欠全世界一个交代：**把"刚刚发生了什么"准确、有序、不丢不重地通知出去**。这一章讲的就是成交之后的消息体系。别小看它——真实交易所的撮合内核可能只占代码量的两成，剩下八成都

在伺候"让别人知道"。

第一张网：成交回报，只给你看的私信

你的订单进门之后，它的一生会产生一串消息，这叫**成交回报**（execution report）——交易所发给你的私信，告诉你"你的单现在什么状态"。一张订单的状态机长这样：

已接收 → 已确认（进了簿子）→ （部分成交 → 部分成交 → ...）→ 全部成交
 ↘ 已撤销 / 已拒绝 / 已过期

每个状态变化都是一条回报，带上订单号、本次成交量、本次成交价、剩余量。第五章那张 1500 股的市价单，你会收到三条"部分成交"回报加一条"全部成交"。

设计回报流时有一个铁律：**客户端必须能从回报流重建出订单的完整状态**，不需要额外去问。因为"问"是有竞态的——你查询的瞬间状态又变了。流是唯一的真相来源，查询只能做兜底。这个原则在第十一章的日志重放里会放大成整个系统的架构哲学。

第二张网：行情广播，给所有人看的公告

回报是私信，行情是公告。交易所把簿子的变化广播给所有订阅者，这就是你在行情软件里看到的東西。行情分几个档次，贵的不一样：

- L1（一档行情）：只有买一卖一的价格和量，加最新成交价。免费行情软件给你看的就是这个。

- L2（多档行情）：买卖各5档、10档的总量——第二章那张盘口快照。这是"按价位聚合"的视图，同一价位十个人的单合并成一个总数。
- L3（逐笔/全量行情）：每一笔订单的增、删、改、成交都单独广播，不聚合。你能看到队列里每张单的生与死，能自己重建出完整的簿子和排队顺序。高频交易者花大价钱买的就是它。

注意一个因果方向，很多人会搞反：**行情不是数据库查出来的，是撮合过程的副产品**。撮合引擎每改一次簿子（新单入簿、成交、撤单），就顺带吐出一条"簿子增量"。L2行情的每一档总量，就是这些增量累加出来的。这直接决定了实现方式：不要定时截图订单簿去广播（那样要么丢变化要么发重复），要让簿子的每次修改天然产生一条增量消息。

序列号：让"丢没丢"变成可检测的

行情走网络广播，网络会丢包。怎么让接收方知道自己丢没丢？答案简单到优雅：**给每条消息编一个严格递增的序列号**（sequence number）。

你收到1001、1002、1004——立刻知道丢了1003，本地簿子不能信，必须修复。修复方式两种：**重放**（找交易所要1003这条，或从某个序号开始重发）和**快照**（重新拉一份当前完整簿子，再从快照对应的序列号继续接增量）。币所的WebSocket深度流就是这套机制：先拉快照，缓存增量，把序列号对上了再开始应用，对不上就重来。

你自己实现时有两个决策点：序列号是**全所一个**还是**每个品种一个**（后者扩展性好，主流做法）；快照里必须带"这份快照对应到哪个序列号"，否则增量接不回去。

大白话

大白话：行情流就像一本连载小说，每页有页码。页码断了，别猜，回头补那一页或者从最近的全家福重新接——绝不允许"大概齐"。金融数据的对，是"可证明的对"。

第三张网：对账，深夜里的慢功夫

白天的回报流和行情流再严密，交易所每天收盘后还要干一件笨事：对账——把每个账户"理论上应该有的钱和货"跟实际余额核一遍。理论上应该有 = 开盘余额 + 全天所有成交、费用、充提的累加。

对账的意义不是"平时会错"，而是**必须能证明没错**。它是整个系统正确性的最终裁判：撮合引擎、清算、费用、任何一环出bug，对账都会不平。不平就是事故，停下来查。所以好的交易所系统，从第一天就把每笔成交设计成"可重算"的——这又为第十一章的事件溯源埋了线。

收拢一下

成交之后，交易所同时维持三张网：给当事人的回报流（私信、状态机）、给所有人的行情流（公告、L1/L2/L3、序列号可修复）、给自己的对账流（慢、全量、最终裁判）。三张网共用同一个数据源：**撮合循环吐出来的事件**。把这句话记住，下一章我们终于开始写代码——你会发现，数据结构的设计早已被前八章的规则悄悄决定了。

第十章 · 动手写一个撮合引擎

前九章都是设计文档

回到第二章那个画面：横向一格一格的价格台阶，纵向每格里一支FIFO队伍。现在把它变成代码。你会惊讶于这一步有多顺理成章——不是因为我们聪明，而是因为规则讲了九章，**数据结构已经被规则决定好了**。

三张表撑起一本簿

一个交易品种的订单簿，内存里就三样东西：

- **买侧价格表**：一个按价格从大到小排序的容器，key是价格，value是该价位的订单队列。要能快速拿到"最高价"（队首）和"给定价格的队列"。
- **卖侧价格表**：同构，按价格从小到大排，"最低价"是队首。
- **订单索引**：一个哈希表，key是订单ID，value指向订单在队列里的位置。撤单时靠它O(1)找到订单。

价格表用什么实现？教科书答案是红黑树（一种自平衡的二叉搜索树）：插入、删除、取最大/最小值都是O(log n)。工程上还有两个变体值得知道：如果价格档位稀疏，用跳表也行；如果tick很小、价格区间有限（比如国债期货价格总在95-105之间跳动），可以直接开一个**数组**，下标就是价格格数——取买一卖一变成数组扫描，快到极致。很多顶级撮合引擎就是这么干的：把"价格"从连续的量变成离散的格，是tick size规则送我们的免费性能。

每个价位的订单队列，用**双向链表**。为什么不用数组？因为高频操作是"从中间摘掉一个"（撤单）——链表O(1)，数组O(n)。订单对象里带prev/next指针，订单索引的哈希表直接存节点地址，撤单就是两步：哈希表查到节点，链表摘除。

大白话：三张表各管一件事——价格表管"最好的价在哪"（有序），链表管"同价谁先来的"（FIFO），哈希表管"我这单在哪"（快查）。撮合引擎的全部性能秘密就是：每个高频操作都落在其中一张表的 $O(1)$ 或 $O(\log n)$ 上。

核心循环：五十行伪代码

把第五章的循环翻译成更接近代码的样子（以买单为例，卖单对称）：

```
function match(order):
  while order.remaining > 0:
    level = 卖侧价格表.最低价位 // 卖一
    if level不存在: break // 对面没人了
    if order是限价单 and level.price > order.price: break // 碰不上
    maker = level.队首订单
    if maker.账户 == order.账户: 按STP处理; continue
    qty = min(order.remaining, maker.remaining)
    产生成交(price=level.price, qty, taker=order, maker=maker)
    order.remaining -= qty; maker.remaining -= qty
    if maker.remaining == 0:
      level.出队(); 订单索引.删除(maker.id)
      if level空了: 卖侧价格表.删除(level.price)
  处理剩余(order) // GTC入簿 / IOC丢弃 / FOK已前置校验
  返回成交列表
```

对照第五章的算例走一遍，每一行都对得上。注意几个被故意写对的地方：成交价取 `level.price`（被动方的价）；成交输出里标了谁是taker谁是maker（第八章的计费靠它）；STP检查在成交之前；价位空了才从价格表删——价格表只增删"档位"，不碰档位内的顺序。

复杂度：为什么这样就是快

数一下每个操作的开销：新限价单撮合，每圈循环是 $O(1)$ （取最优价、队首、min都是常数），总共跑K圈， $K=\text{成交笔数}$ ——**撮合的成本和成交量成正比，不和簿子大小成正比**。这是好性质：墙再厚，只成交一格就只花一格的时间。剩余入簿 $O(\log n)$ ，撤单 $O(1)$ ，查买一卖一 $O(1)$ （缓存队首价位）。

这就是主流撮合引擎单品种每秒几十万单、延迟个位数微秒的底气：**没有任何一步需要遍历整个簿子**。慢的系统多半是在某个地方偷偷做了全表扫描——比如每次成交后重新排序，或者用数组存队列导致撤单是 $O(n)$ 。

整数，整数，整数

代码里有一个没画出来但写错就出人命的决定：**价格和数量必须用整数，禁止浮点**。10.01元存成1001（分），0.001个币存成1000（最小单位）。浮点数的 $0.1+0.2 \neq 0.3$ ，在别的系统是误差，在交易所是事故——两边对不上账，对账（第九章那张网）立刻报警。所有交易所内部都用定点整数，显示给用户时才除以缩放因子。这条没有例外。

先单线程，别急着并发

你可能已经手痒想加锁、加线程了。忍住。下一章整章讲为什么：撮合引擎的正确性完全建立在“事件严格按序处理”上，而多线程的第一作用就是破坏顺序。业界的主流答案——包括LMAX这样做到每秒六百万笔的系统——是单线程。我们现在先把单线程版本写对，下一章再把它做快、做皮实。

到这里，你已经有了一个能撮合的撮合引擎：三张表加一个循环，五十行伪代码。但它还只是个“实验室玩具”——断电就没了，重启就失忆。下一章把它变成一台可以托付真金白银的机器。

第十一章 · 做快与做对：定序、重放与崩溃恢复

快和对，是同一个答案

上一章的五十行伪代码能成交，但离"能上线"还差三件事：它得扛住断电，它得快，它得能被证明是对的。这三件事看起来是三座山，其实山脚连着同一个决定——而这一章就是要让你相信这个决定：**单线程定序 + 一切皆可重放**。

为什么撮合引擎几乎都是单线程

直觉说多线程快。但撮合引擎的命门不是算力，是**顺序**：同一张簿子上，"先撤单后成交"和"先成交后撤单"是两个世界。第六章FOK的干跑、止损单的触发链，全都建立在"事件一件一件来，前一件彻底处理完才看下一件"之上。多线程的第一件事就是破坏这个保证，然后你得用锁把顺序再抢回来——抢回来的过程比省下的还贵，还引入死锁和竞态这些新的死法。

所以业界的主流架构是把系统切成三段流水线：

- **网关**（多线程，随便扩）：收单、校验、鉴权，把合法的订单**排成一个全局唯一的序列**——盖章编号，扔给下一站。这一步叫**定序**（sequencing），是整个系统唯一不能并行的咽喉。
- **撮合内核**（**单线程**）：从序列里一条一条取事件，跑第十章的循环，改簿子，吐成交。单线程意味着零锁、零竞态、行为完全确定。著名的高性能交易框架LMAX Disruptor用一根线程做到每秒数百万事件——因为没了锁，CPU缓存全在为同一件事工作。
- **外围**（多线程，随便扩）：把内核吐出的事件分发给行情、回报、计费、风控、清算。它们慢一点都不影响撮合，各干各的。

大白话：撮合引擎像一个只开一扇门的银行柜台——门口保安（网关）给所有人发号，柜台里一个人（内核）按号办事，后面一群人（外围）负责复印单据寄出去。多开几个柜台不会更快，只会让账对不上。

事件溯源：让断电变得不可怕

单线程内核是纯内存的，快，但断电就失忆。怎么办？答案叫事件溯源（event sourcing）：**在进入内核之前，把每个输入事件先追加写到磁盘日志里**——"10:00:01 收到D的限价买单1200股"、"10:00:02 收到撤单X".....

现在关键的魔法出现了：撮合内核是**确定性**的——同样的输入序列，必然产生同样的输出和同样的簿子状态（这就是为什么不能用随机数、不能依赖系统时钟做撮合决策）。所以恢复的时候根本不需要恢复"簿子"，只需要**重放日志**：从空簿子开始，把今天的事件按序号重新喂一遍，内核一单一单重新撮合，最终状态分毫不差。日志就是系统，内存里的簿子只是日志的缓存。

重放一天的日志太慢？定期拍快照：把某时刻的簿子状态整个存下来，记下它对应日志的哪个序号。恢复 = 加载最近快照 + 重放之后的日志。这套"日志+快照"正是第九章行情修复机制的同款——你看，同一个思想贯穿了整个系统。

高可用：主备两台机器，同一份日志

单机再稳也会坏。标准做法是主备：两台一模一样的内核，消费同一份定序日志。主机活着，备机跟着重放保持热同步；主机死了，备机接管——它的簿子和主机死前一字不差，因为它算过同样的题。切换时对外几乎无感。更讲究的交易所上三台做多数派，那已经进入分布式共识的领域，原理相同。

测试：怎么证明一台金融机器是对的

确定性给了我们一件别的系统梦寐以求的武器：**可复现的测试**。四层，一层都不能少：

1. **规则单测**：本书前五章的每个算例都该变成一个测试——限价单排队、市价单扫墙、部分成交保留队位、FOK干跑、STP三档、集合竞价最大成交量。这些是"法律条文"的判例集。
2. **不变量检查**：不管喂什么输入，某些事永远不许发生——簿子里买价不得 $\geq$ 卖价（交叉的簿子说明撮合漏了）；任何订单剩余量不得为负；所有成交的成交量必须守恒（买方得的=卖方出的）。让引擎每处理完一个事件就自检一遍，违反就停机报警——宁可停，不可错。
3. **随机对拍** (fuzzing)：写一个"笨但显然正确"的参考实现（用数组、暴力遍历，不管性能），再写一个随机订单生成器，两边喂同样的十万条随机事件，状态必须一模一样。你的快实现和笨实现任何一次分歧，都是一个真bug。
4. **生产重放**：上线后，把每天的生产日志拿到测试环境重放，结果和当天真实状态对账。这套设施反正为了崩溃恢复也得建，顺手就成了最强的回归测试。

走到这里，你的交易所内核已经能跑、能快、能恢复、能被证明正确。只剩最后一关：真实世界不总是按规则出牌——行情会疯，人会犯错，程序会有bug。最后一章，我们给这台机器装上护栏。

第十二章 · 真实世界的护栏

最后一道坎：市场会疯，机器不能疯

前十章造出来的引擎，在风平浪静的日子里完美无缺。但交易所的职业生涯是用极端日子来定义的：利好落地全市场抢筹的日子、谣言炸开夺路而逃的日子、某个交易员手滑多敲三个零的日子。护栏的意义就是：**在这些日子里，规则依然替你站岗，而且站得比人冷静**。这一章把真实交易所的四道护栏讲透——每一道都能落成一个具体的代码位置。

护栏一：涨跌停——把价格关进笼子

A股的 $\pm 10\%$ 涨跌停，实现上出乎意料地简单：**一个随时间移动的合法价格区间**。每天以昨收盘价为基准算出今日笼子的上下沿，收单时校验：限价超出笼子的订单直接拒掉；撮合时，笼子外的价位根本不存在（挂着涨停价的大量买单可以存在，但没人能以更高价卖，所以天然不成交——笼子就是这么"软"地生效的：不是不让你挂，是不让笼子外发生成交）。

注意一个细节：基准价不一定是昨收盘。除权除息日、复牌日，基准要调整。写成配置，别写死。美股的同类机制叫LULD（Limit Up-Limit Down）：不是全天一个笼子，而是以最近成交价为轴心的滚动价格带，出带的订单不进撮合，价格带被顶死15秒就暂停交易5分钟，用一次小集合竞价重新开盘——把第七章的定价机器当成"重启按钮"用，这是很漂亮的规则设计。

护栏二：熔断——全市场的深呼吸

涨跌停管单个品种，熔断（circuit breaker）管整个市场。美股的规则是：标普500指数较昨收跌7%、13%时全市场暂停15分钟，跌20%直接收市。逻辑上一句话：当所有人都在向同一个方向逃时，让市场停下来，给信息追上情绪的时间。

工程实现：熔断是一个**全局状态机**——正常交易 / 暂停（拒新单、允许撤单或全拒，按规则定） / 集合竞价重启 / 恢复。它凌驾于所有品种之上，每条指令进门前先问状态机。2016年初A股曾短暂引入指数熔断，四天后叫停——因为磁吸效应（价格快跌到熔断线时，大家抢

着在熔断前卖，反而加速了下跌)。这个教训值得记住：**护栏本身会改变人的行为，设计护栏要算进参与者会对它做出反应。**

护栏三：防手滑——胖手指检查

2005年日本瑞穗证券的操作员把"61万日元卖1股"敲成"1日元卖61万股"，系统居然接了，几分钟亏掉数亿美元。这类事故有个名字：胖手指 (fat finger)。防御措施是一组廉价的合理性校验，在收单的最前端：

- **价格偏离检查**：限价偏离最新价/昨收超过x%（比如20%）直接拒单；
- **数量上限**：单张订单不得超过N股或N元名义价值；
- **频率限制**：单账户每秒下单/撤单次数上限（顺手也挡了恶意刷流量）。

这些检查会误伤极小比例的合法订单，但拦住的事故价值远超此。原则：**越靠前端的校验越便宜，越致命的事故越值得便宜地拦。**

护栏四：风控前置与总开关

最后一道，也是最不起眼的一道：**进撮合之前先问"这个人配不配下这张单"**。买单进来，先冻结相应的钱（限价×数量）；卖单进来，先冻结相应的货。钱不够、货不足，直接拒——绝不允许"先成交、后赖账"。这就是为什么撮合引擎能放心地只关心价格和时间：信用问题已经在它上游被解决了。期货还要查保证金、持仓限额，同一套"前置校验"框架，只是检查项更多。

以及，每台交易所都需要一个kill switch（总开关）：一个按钮，按下后某账户、某网关、或全所的订单流立刻切断、在途订单批量撤掉。平时它一次都用不上；出事的那一分钟里，它是唯一重要的按钮。它必须是系统里**最可靠、最少依赖**的那条代码路径——不能因为它自己也卡在同一条出问题的链路上而按不下去。

收工：你现在拥有什么

回头看这本书攒下的零件：一面墙（订单簿）和它的形状（第二章）；两个排队的人（限价单、市价单，第三、四章）；一台手术台上解剖过的撮合循环（第五章）；四组订单旋钮（第六章）；一台开盘定价机器（第七章）；一张各地法规的对照表（第八章）；成交后的三张消

息网（第九章）；三张表五十行的内核（第十章）；单线程定序加日志重放的躯干（第十一章）；以及这一章的四道护栏。

它们拼起来就是一台完整交易所的图纸：**网关定序 → 前置校验（格式、胖手指、风控冻结） → 单线程撮合内核（循环+旋钮+护栏状态机） → 事件日志 → 行情/回报/计费/对账外围。**

每个零件的"为什么"你都知道了一一而知道为什么，才是能动手写的前提。

剩下的，就是打开编辑器了。祝你的引擎永远冷静。

撮合引擎：交易所是怎么把买单卖单凑成一笔交易的 · 王建硕