Uniswap V3 Core 白皮书(中文)

2021年3月

Hayden Adams hayden@uniswap.org	Noah Zinsmeister noah@uniswap.org	Moody Salem moody@uniswap.org
River Keefer river@uniswap.org		Dan Robinson dan@paradigm.xyz

翻译 2021年4月

杜荣政
drz@hotpot.fund

概要

Uniswap V3 是建立在以太坊区块链上，一套不受监管的自动做市协议。与协议的早期版本相比，Uniswap V3 为做市商提供了提高资金利用率、以及主动调控资金分布的能力，提高了价格预言机的准确性和方便性，并具备更灵活的手续费结构。

1 介绍

自动做市商 (AMMs) 是通过汇聚流动性，以算法为交易者完成交易提供服务。恒定函数做市商 (CFMMs)，即更广义的自动做市商 (AMMs) (Uniswap是 AMMs 中的一员)，已经在去中心化金融中得到了广泛应用，它的典型用例是在无需许可的公共区块链上，通过智能合约实现代币交易。

目前的恒定函数做市商 (CFMMs) 普遍资金利用率偏低。在 Uniswap V1 和 V2 的恒定乘积公式中，在给定价格时，流动池中只有部分资金发挥了作用。这是低效率的，尤其是当交易总是围绕着某个价格小幅波动的时候。

以前解决资金利用率问题的尝试，比如 Curve 和 YieldSpace，通过使用不同的函数来描述流动池中资产之间的关系。这就要求在给定流动池中，所有的做市商遵守单一的公式，如果做市商希望在不同的价格范围提供流动性，则可能导致流动性分散。

在本文档中，我们全面介绍了 Uniswap V3，一种新的自动做市商 (AMM)，它使得做市商能更好地控制其资金使用的价格范围，而且仅对流动性分散和Gas费用产生有限的影响。这些设计与代币价格无关，不依赖于对价格的任何假定。Uniswap V3 与其早期版本一样，基于相同的恒定乘积算法，但提供了几个重要的新特性：

- ***聚焦流动性 (Concentrated Liquidity):*** 做市商 (LPs) 有能力通过将流动性“限定”在任意价格范围内来聚焦流动性 (早期版本提供的是 $(0, +\infty)$ 的所有价格范围。)。这提高了流动池的资金效率，允许做市商接近其钟意的资产曲线，同时能与流动池中其它人提供的流动性有效集成。在第2章对本特性作了描述，第6章阐述了如何实现聚焦流动性。
- ***灵活的手续费：*** 手续费不再仅限于 0.3%。每个交易对的费率在初始化时设置 (每个交易对可以有多种费率) (3.1小节)。初期支持的费率有 0.05%，0.30%，和1%。UNI治理可以增加新的费率设置。
- ***协议费治理：*** UNI治理在协议费占手续费比例的设置方面，拥有更多的灵活性 (6.2.2小节)。

- ***增强的价格预言机：***Uniswap V3 为用户提供一种查询最近价格累计值的方法，避免在度量 TWAP 时，需要精确指定开始和结束时间来获取累计值 (5.1小节)。
- ***流动性预言机：***合约提供一种时间加权平均的流动性预言机 (5.3小节)。

Uniswap V2 核心合约设计成不能升级，所以 Uniswap V3 完全由一套新合约实现。Uniswap V3 核心合约也不能升级，除了治理可以调整部分参数 (见第4章)。

2 聚焦流动性 (Concentrated Liquidity)

Uniswap V3 的核心理念是聚焦流动性：在指定价格区间内提供流动性。

在早期版本中，流动性沿下列公式均匀分布：

$$x \cdot y = k \tag{2.1}$$

其中 x 和 y 是两种代币 X 和 Y 的数量， k 是常量。换言之，早期版本的设计是在 $(0,\infty)$ 的整个价格范围内提供流动性。这利于实现，并且允许有效地汇聚流动性，但意味着流动池中大部分资产永远不会被利用。

考虑到这一点，允许做市商在一个比 $(0,\infty)$ 小的价格区间内聚焦他们的流动性，是合情合理的。我们将聚焦到有限价格区间的流动性称作：一个头寸 (a position)。一个头寸只需要在它的价格区间内维持足够资金以支持交易，因此在保持恒定乘积算法不变的情况下，可以在该价格区间提供更大的资金量 (我们称之为：虚拟资产)。

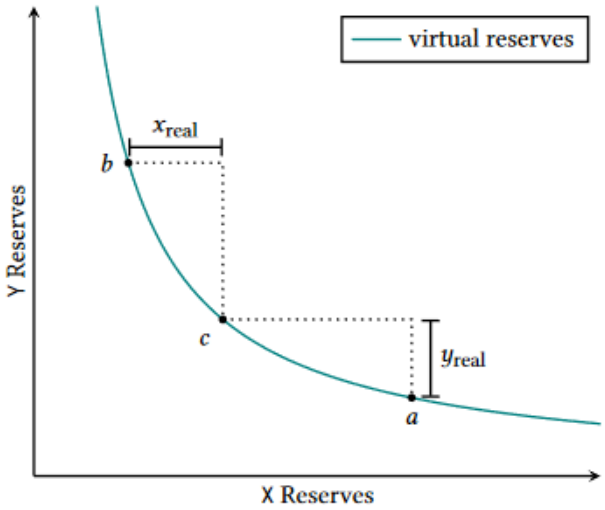

Figure 1: Simulation of Virtual Liquidity

具体而言，当价格向上变动时，一个头寸需要持有的 X 资产数量，只需要覆盖价格波动到价格区间上限，因为超出价格区间之后将耗尽 X 资产。同样，它只需要持有能将价格波动覆盖到价格区间下限的 Y 资产。图1 描绘了一个价格区间为 $\lfloor p_a, p_b \rfloor$，当前价格 $p_c \in \lfloor p_a, p_b \rfloor$ 时的相互关系。 x_{real} 和 y_{real} 表示该头寸的实际资产。

当价格不在头寸的价格区间内，该头寸的流动性不再激活，也不能赚取手续费。那时，它的流动性完全由一种资产组成，因为另一种资产必然已经耗尽。当价格重新进入它的价格区间，它的流动性也重新激活。

流动性数量可以用值 L 度量，它等于 $\sqrt{k}$ 。一个头寸的真实资产，可以用下列公式和曲线表示：

$$(x + \frac{L}{\sqrt{p_b}}) \cdot (y + L\sqrt{p_a}) = L^2 \tag{2.2}$$

该曲线是公式 2.1 的平移，使其头寸在价格区间内清晰可见（图2）：

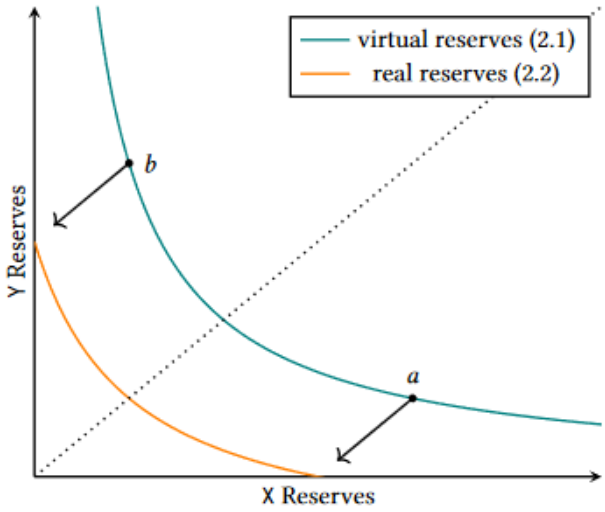

Figure 2: Real Reserves

做市商可以在他们认为合适的价格区间，创造任意数量的头寸。这样，做市商就可以在价格空间上近似地得到他们所期望的流动性分布 (参见图3，给出了一些案例)。此外，这也是一种机制：让市场决定流动性应该如何分布。理性的做市商可以通过将流动性聚焦在当前价格的狭窄区间内，在价格波动时增加或移除代币来保持流动性处于激活状态，从而提高资金效率。

2.1 限价订单 (Range Orders)

价格区间设置较小的头寸，表现得类似限价订单——如果超出价格区间，头寸从完全由一种资产组成，转变为完全由另一种资产组成 (加上累积的手续费)。这里的限价订单和传统限价订单有两点差异：

- 价格区间的狭窄度是有限的。当价格处于该价格区间内时，限价订单会被部分执行。
- 当超出头寸的价格区间时，它需要被提取出去。否则，当价格重新落回或回穿区间，头寸又会被交易回去，资产类别将再次被逆转。

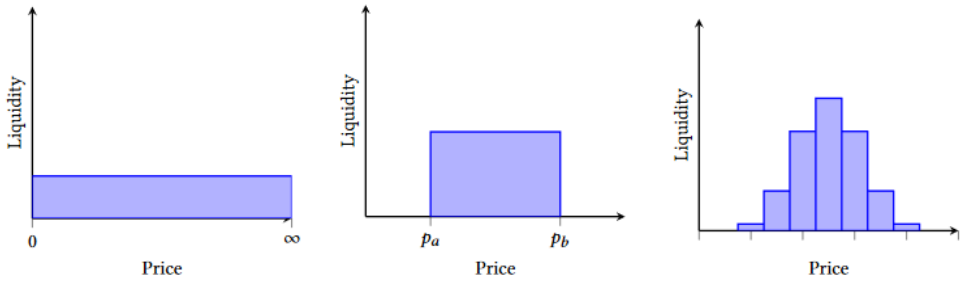

Figure 3: Example Liquidity Distributions

3 架构更改

Uniswap V3 做了很多架构更改，其中一些是为了包含聚焦流动性必需的，其中一些是独立的改进。

3.1 一个交易对多个流动池

在 Uniswap V1 和 V2 中，每个交易对对应单一一个流动池，所有的交易都统一收取 0.30% 的手续费。

尽管这一默认的手续费率过去对大多数代币都足够有效，但对于某些流动池显得过高 (比如，2种稳定币组成的流动池)，而对于某些流动池显得过低 (比如，波动剧烈或很少交易的流动池)。

Uniswap V3 为每个代币交易对引入了多个流动池，每个流动池有不同的手续费率。所有的流动池都是由同一个工厂合约创建的。工厂合约初始允许用三种手续费率创建流动池：

0.05%、0.30% 和 1%。UNI 治理可以启用其它手续费率。

3.2 无法互换的流动性 (Non-Fungible Liquidity)

3.2.1 非复利手续费 (Non-Compounding Fees)

在早期版本中，赚取的手续费是作为流动性存在流动池中的。这意味着，即便没有追加投资，池内的流动性也会随着时间推移而不断增加，手续费是复利收益。

在 Uniswap V3 中，由于头寸的不可替代特性，这已经不可能了。取而代之的，手续费收益是以支付手续费的代币方式被单独存放 (参见 6.2.2 小节)。

3.2.2 取消ERC-20代币

在 Uniswap V1 和 V2 中，流动池合约同时也是一个 ERC-20 代币合约，其代币代表在流动池中持有的流动性。虽然这很方便，但它其实与 Uniswap V2 的理念不太一致，即任何不需要在核心合约中实现的内容，都应该在外围合约中实现，并且一个规范的 ERC-20 实现，不鼓励创建改进的 ERC-20 代币封装合约。按理，ERC-20 代币应该作为核心合约中某个流动性头寸的封装，在外围合约中实现。

Uniswap V3 所做的更改，使完全可替换的流动性代币变得不可能。由于定制流动性供应的特性，现在收取手续费的方式是单独存放的代币，而不是在流动池中自动重投的流动性。

所以，在 V3 的流动池合约中没有实现 ERC-20 标准。任何人都可以在外围合约中创建一个 ERC-20 代币，使某个流动性仓位可以被替换，但它需要额外的逻辑来处理手续费的分布和重投。或者，任何人都可以创建一个外围合约，将某个流动性仓位 (包括赚取的手续费) 封装成一个 ERC-721 代币。

4 治理

工厂合约有一个业主 (owner)，初期由 UNI 代币持有者控制。业主没有能力终止任何核心合约的操作。

同 Uniswap V2 一样，Uniswap V3 中 UNI 治理也可以收取协议费用。在 Uniswap V3 中，UNI 治理有更大的灵活性，可以选择在手续费中收取 $\frac{1}{N}$（ $4 \le N \le 10$ ）或为 0 的协议费用。该参数可以在每一个流动池中单独设置。

UNI 治理也可以增加其它的手续费率。当增加新的手续费率时，同时要定义相应的价格刻度单位 (tickSpacing，见 6.1 小节)。一旦手续费率被增加到工厂合约，就不能被移除 (且刻度单位也不能更改)。初始化支持的手续费率和刻度单位有：0.05% (刻度单位为10，近似于每刻度价格偏差 0.10%)，0.30% (刻度单位为60，近似于每刻度价格偏差 0.60%)，和 1% (刻度单位为200，近似于每刻度价格偏差 2.02%)。

最后，UNI 治理可以将业主权转移到其它地址。

5 预言机升级

Uniswap V2 引入了时间加权平均价格 (TWAP) 的预言机，Uniswap V3 对其做了3处重要的升级。

最重要的是，Uniswap V3 不再需要用户从外部跟踪累加器的先前值。Uniswap V2 需要用户在一个时间段的开始和结束时，分别记录累加器的值，来计算时间加权平均价格。Uniswap V3 把累加器记录放在核心合约中，从而允许外部合约无需记录累加器的值，直接在链上计算最近一段时间的加权平均价格。

另一个改变是，Uniswap V2 是累计价格和，从而允许用户计算时间加权的算术平均价格。而 Uniswap V3 跟踪的是价格的对数 (价格的对数值) 和，从而允许用户计算时间加权价格的几何平均。

最后，在价格累加器之外，Uniswap V3 增加了一个流动性累加器。流动性累加器可以被其它合约使用，用于确定某个交易对的哪一个流动池 (参见 3.1 小节)，有最可靠的时间加权平均

价格。

5.1 预言机观察值 (Oracle Observations)

和 Uniswap V2 一样，Uniswap V3 在每个区块的开始运行一个价格累加器，乘以自上一个区块以来过去的秒数。

一个 Uniswap V2 的流动池，仅存储该价格累加器的最近值——即，最后一次发生交易的区块中的值。因此，在 Uniswap V2 中计算平均价格时，外部调用者需要提供价格累加器的前一个值。对于很多用户而言，每个用户都必须要有自己的方法记录累加器的前一个值，或者协调一个共享的办法来降低费用。而且也没办法保证累加器能忠实反映流动池被调用的每一个区块。

在 Uniswap V3 中，流动池存储一个累加器值的列表。它通过每一个区块中首次调用流动池时，自动检查累加器的值来实现。该列表类似于一个循环缓冲区，通过在一个数组中循环，最旧的值最终被最新的值覆盖。虽然该数组最初只有存放一个值的空间，但任何人都可以初始化额外的存储插槽来扩展该数组，直到存储多达 65,536 个值 (*最大 65,536个观察值，假定每13秒1个区块，每个区块都需要存值，都允许存放至少9天的价格*)。任何希望该交易对存放更多值的人，只需要承担一次为扩展该数组，初始化额外存储插槽的 gas 消耗。

流动池开放该观察值数组给用户，同时提供一个方便的函数，以供用户查询在累加器存放时间周期内，任意时间点的值。

5.2 几何平均价格

Uniswap V2 有两个价格累加器——一个以 token1 计价的 token0 价格，一个以 token0 计价的 token1 价格。用户可以用时间段结束时的累加器值，减去开始时的值，再除以时间段的秒数，来计算任意时间段价格的算数平均值。需要注意的是，token0 和 token1 的累加器是单独跟踪的，因为 token0 的时间加权算术平均价格，并不等于对应的 token1 时间加权算术平均价格。

Uniswap V3 使用时间加权几何平均价格，就不需要单独跟踪两种 token 的累加器。两种 token 的价格几何平均值互为倒数。在 Uniswap V3 中，由于要实现提供定制化流动性，实现几何平均价格也变得更容易。此外，因为跟踪的价格的对数值，而不是价格，累计值可以存储在较少的字节中，而且价格的对数值，在相同精度的情况下可以表示更广泛的价格范围 (*为了在可容忍的精度范围内，支持所有可能的价格，Uniswap 用一个 224 位固定位浮点数表示每种价格。Uniswap V3 只需要用一个带符号的24位整型数表示 $\log_{1.0001}\sqrt{P}$ ，同样可以监测小到一个刻度、或一个基点［0.01%］的价格波动*)。

最后，本文给出了时间加权几何平均价格，能更真实反映平均价格的理论依据 (*虽然算术平均价格被广泛应用，但在度量几何布朗运动过程［通常采用该种建模方式描述价格波动］时，它在理论上是不太准确的。几何布朗运动过程的算术平均值，倾向于加大高价格的权重［因为同样百分比的变化，高价格的变化绝对值更大］。*)。

与跟踪价格的累计值不同，Uniswap V3 存放的是当前刻度值的累计和 ($\log_{1.0001} P$ ，以 1.0001为底数的价格对数值，精度为1个基点，即 0.01%)。 任意给定时间点的累计值等于合约历史上每秒钟的价格对数 $\log_{1.0001} P$ 求和：

$$a_t = \sum_{i=1}^{t} \log_{1.0001}(P_i) \tag{5.1}$$

下列公式用于估算任意时间段 t_1 到 t_2 的时间加权几何平均价格 (P_{t_1,t_2}):

$$P_{t_1,t_2} = (\prod_{i=t_1}^{t_2} P_i)^{\frac{1}{t_2-t_1}} \tag{5.2}$$

要计算它，您可以用时间点 t_2 的累计值减去 t_1 的累计值，除以时间段的秒数，再计算 1.0001^x 的幂值，得到时间加权几何平均价格：

$$\log_{1.0001}(P_{t_1,t_2}) = \frac{\sum_{i=t_1}^{t_2} \log_{1.0001}(P_i)}{t_2 - t_1} \tag{5.3}$$

$$\log_{1.0001}(P_{t_1,t_2}) = \frac{a_{t_2} - a_{t_1}}{t_2 - t_1} \tag{5.4}$$

$$P_{t_1,t_2} = 1.0001^{\frac{a_{t_2}-a_{t_1}}{t_2-t_1}} \tag{5.5}$$

5.3 流动性预言机

除了时间加权平均价格，Uniswap V3 还在每个区块的开头，跟踪累计了当前的 L 值(在当前价格区间的虚拟流动性)。 链上其它合约使用它，可以使价格预言机更加健壮 (例如，在判断使用哪种手续费率的流动池作为价格预言机时)。流动性累加器的值和价格累加器同时存储。

6 聚焦流动性的实现

本节描述聚焦流动性如何工作，以及在合约中实现聚焦流动性的原理。

6.1 价格刻度(*Ticks*)和价格区间(*Ranges*)

为了实现聚焦流动性，流动池可以提供的价格是由一系列离散的价格刻度(*ticks*)标记。做市商可以在任意两个价格刻度之间提供流动性(不相邻的两个刻度)。

每个区间(*Range*)可以由一对带符号整数的区间索引号界定：下届刻度 (*a lower tick*) 记作(i_l), 上届刻度 (*a upper tick*) 记作(i_u)。价格刻度代表合约中虚拟流动池的价格可变区间。我们假定价格总是由其中一种 *token* 的价格来表示，如: *token0*，相对应另一种 *token* 则为：*token1*。价格锚定是 *token0* 还是 *token1* 无关紧要，并不影响合约逻辑(除了可能出现的舍入误差)。

理论上，每一个价格 p , 都可以用 1.0001 的 n 次方计算得到。用整形数字 i 标记，价格计算公式为：

$$p(i) = 1.0001^i \tag{6.1}$$

每个刻度相对于它的相邻刻度，都有 .01% (1个基点) 的价格变动。

参见 6.2.1 小节的技术考量，流动池实际保存的是价格 p 的平方根，它可以用 $\sqrt{1.0001}$ 的 n 次方计算得到。考虑上边的公式，转化为价格的平方根计算公式：

$$\sqrt{p}(i) = \sqrt{1.0001}^i = 1.0001^{\frac{i}{2}} \tag{6.2}$$

例如， $\sqrt{p}(0)$ –价格刻度0的价格平方根–是1，刻度1的 $\sqrt{p}(1)$ 是 $\sqrt{1.0001} \approx 1.00005$ ，刻度 –1的 $\sqrt{p}(-1)$ 是 $\frac{1}{\sqrt{1.0001}} \approx 0.99995$ 。

给流动性添加某个区间时，如果一个或两个刻度尚未被用作现有头寸的边界，则该刻度将被初始化 (*initialized*)。

并非所有的刻度都可以初始化。流动池有一个参数：刻度*单位 (tickSpacing,* t_s)，只有能被刻度单位整除的刻度下标，该刻度才能初始化。例如，如果 *tickSpacing* 是2，则只有 (...–4, –2, 0, 2, 4...) 标记的刻度才能初始化。越小的刻度单位允许越紧凑和更精确的区间界定，但可能在交易 (swap) 时造成更多 gas 开销 (因为交易时越过每一个已初始化的刻度，都会造成一次 gas 消耗)。

每当价格越过一个已初始化的刻度，虚拟流动池将被移进 (*kick in*) 或移出 (*kick out*)。越过一个已初始化刻度的 gas 开销是恒定的，与该刻度被移进/移出的头寸 (*positions*) 数量无关。

当刻度被越过时，为保障移进/移出正确数量的流动性，以及每一个区间内的头寸都能获得相应份额的手续费，需要在流动池中进行某种计量。流动池合约使用存储变量保存状态，包括全局层面、每刻度层面，以及每头寸层面。

6.2 全局状态变量

合约中的全局状态有7个存储变量，与交易和流动性份额相关。（还有一些用于预言机的存储变量，如第5节所述。）

类型	变量名	符号

uint128	liquidity	L
uint160	sqrtPriceX96	$\sqrt{P}$
int24	tick	i_c
uint256	feeGrowthGlobal0X128	$f_{g,0}$
uint256	feeGrowthGlobal1X128	$f_{g,1}$
uint128	protocolFees.token0	$f_{p,0}$
uint128	protocolFees.token1	$f_{p,1}$

Table 1: Global State

6.2.1 价格和流动性

在 Uniswap V2 中，每个流动池合约保存当前池中资产，x 和 y。在 Uniswap V3 中，合约可以被想象为在保存虚拟资产——x 和 y 的值允许你用于描述合约的行为 (在两个相邻刻度之间)，同样遵循固定乘积公式。

然而，不用于保存虚拟资产，流动池合约保存两个不同的值：liquidity(L) 和 sqrtPrice($\sqrt{P}$). 这两个值与虚拟资产 x, y 之间的关系，可以用下列公式表达：

$$L = \sqrt{xy} \tag{6.3}$$

$$\sqrt{P} = \sqrt{\frac{y}{x}} \tag{6.4}$$

反过来，这两个值也可以被用于计算虚拟资产 x, y，如下：

$$x = \frac{L}{\sqrt{P}} \tag{6.5}$$

$$y = L \cdot \sqrt{P} \tag{6.6}$$

用 L 和 $\sqrt{P}$ 非常便利，是因为每次交易只有其中一个会发生变化。在一个刻度内交易时，价格 (即 $\sqrt{P}$)变化；当越过一个刻度，或者铸造或销毁流动性时，流动性 L 发生变化。这避免了如果保存虚拟资产，可能产生的一些舍入误差。

您或许已经注意到，基于虚拟资产的流动性计算公式，非常类似于 Uniswap V2 中，在获得手续费之前，基于实际资产初始化流动性 ERC20 token 数量的公式。某种角度看，流动性可以被认为是虚拟流动性 token.

或者，流动性可以被认为是每单位 $\sqrt{P}$ 变化，所对应的 token1 资产 (无论是实际还是虚拟资产) 数量变化：

$$L = \frac{\Delta Y}{\Delta \sqrt{P}} \tag{6.7}$$

我们保存 $\sqrt{P}$ ，而不是 P，使得交易计算时无需计算平方根，如 6.2.3 小节所述。

全局状态 *tick* (i_c) 保存当前价格刻度的下标，一个带符号整型数，代表当前刻度 (更具体地说，低于当前价格最近的一个刻度)。这是一种优化（也是避免对数计算精度问题的一种方法），因为在任何时候，你都可以基于当前的 sqrtPrice 计算当前刻度。具体而言，在任何时间点，下列等式都是成立的：

$$i_c = \left\lfloor \log_{\sqrt{1.0001}} \sqrt{P} \right\rfloor \tag{6.8}$$

6.2.2 手续费

每个流动池初始化了一个不变值 *fee*(γ)(immutable 变量与 constant 变量的差异，请查阅 Solidity 文档)，以一个基点 (0.0001%) 的百倍为单位，表示交易者要支付的手续费比率。

还保存了当前的协议费率： ϕ （初始化为0，但 UNI 治理可以修改）。该数字将交易者支付的手续费中按某个比例支付给协议，而不是做市商。 ϕ 只能被设置为下列有限的允许值之一：0, 1/4, 1/5, 1/6, 1/7, 1/8, 1/9 或 1/10。

全局状态还保存了两个值：feeGrowthGlobal0 （ $f_{g,0}$ ）和 feeGrowthGlobal1 （ $f_{g,1}$ ）。代表在整个合约生命周期内，每单位虚拟流动性 (L) 获得的手续费总金额。您可以将其视为在合约首次初始化时，存入的1单位流动性所获得的手续费总金额。它们以无符号 128×128 的固定小数位数字形式保存。需要注意的是，在 Uniswap V3 中，获得的手续费是 token 本身，而不是流动性，原因如 3.2.1 小节所述。

最后，全局状态还保存了每 token 总共已获得、还未被提取 (collect) 的协议费用，protocolFees0 （ $f_{p,0}$ ）和 protocolFees1 （ $f_{p,1}$ ）。它们是无符号 uint128 类型。UNI 治理可以调用 collectProtocol 函数，提取获得的协议费用。

6.2.3 单一刻度内交易

对于金额较小的交易，不会越过一个价格刻度的情况下，合约表现与 $x \cdot y = k$ 恒定乘积流动池类似。

假定 γ 是手续费率，比如：0.003， y_{in} 是交易转入的 token1 数量。

首先，增加 feeGrowthGlobal1 和 protocolFees1：

$$\Delta f_{g,1} = y_{in} \cdot \gamma \cdot (1 - \phi) \tag{6.9}$$

$$\Delta f_{p,1} = y_{in} \cdot \gamma \cdot \phi \tag{6.10}$$

Δy 是 y 的增量 (扣除手续费之后)。

$$\Delta y = y_{in} \cdot (1 - \gamma) \tag{6.11}$$

如果使用虚拟资产 (x 和 y) 表示 token0 和 token1 余额，那么计算交易应转出的 token0 的计算公式为：

$$x_{end} = \frac{x \cdot y}{y + \Delta y} \tag{6.12}$$

不过，请记住 V3 合约中实际保存的是流动性 (L)和价格平方根 ($\sqrt{P}$)，而不是 x 和 y 。我们可以通过计算得到 x 和 y ，然后用它们来计算交易的执行价格。转化之后，我们得到一个简单的公式，来描述在给定 L 的情况下， $\Delta\sqrt{P}$ 和 Δy 之间的关系 (本公式可以由公式 6.7 推导得出)：

$$\Delta\sqrt{P} = \frac{\Delta y}{L} \tag{6.13}$$

$$\Delta y = \Delta\sqrt{P} \cdot L \tag{6.14}$$

同理，也有描述 $\Delta\frac{1}{\sqrt{P}}$ 和 Δx 之间关系的简单公式：

$$\Delta\frac{1}{\sqrt{P}} = \frac{\Delta x}{L} \tag{6.15}$$

$$\Delta x = \Delta\frac{1}{\sqrt{P}} \cdot L \tag{6.16}$$

用一种 token 交易另一种时，流动池合约可以先用公式 6.13 或 6.15 计算新的 $\sqrt{P}$ ，然后用公式 6.14 或 6.16 计算应该转出的 token0 或 token1。

这些公式适用于任意 $\sqrt{P}$ 没有越过下一个已初始化价格刻度的交易。如果计算所得 $\Delta\sqrt{P}$ 会使 $\sqrt{P}$ 越过下一个已初始化价格刻度，合约在完成后续交易之前，必须移动到新的刻度——先完成部分交易——然后越过该刻度，如 6.3.1 小节所述。

6.2.4 初始化刻度图 (Tick Bitmap)

如果某个刻度没有被用作任何区间的端点，也即没有投入流动性——该刻度就没有被初始化——该刻度在交易时可以被跳过。

为使寻找下一个已初始化的刻度效率更高，作为一个优化流动池保存一个已初始化刻度的位图 *tickBitmap*。如果刻度已初始化，位图中该刻度下标的对应值设为1，未初始化设为0。

如果一个刻度被用作一个新头寸 (position) 的端点，且该刻度还没有被其它流动性使用，刻度将被初始化，刻度图中相应 bit 位被设为1。1个已初始化的刻度，如果以其为端点的所有流动性都被移除，则该刻度重新回到未初始化状态，刻度图中相应 bit 位重新设为0。

6.3 每刻度的状态变量

合约需要保存每个刻度的信息，保存总的流动性数量，当越过刻度时应该被添加或移除的数量，以及在刻度之上或之下获得的手续费。

合约保存了4个以刻度下标为键值的键值对：

类型	变量名	符号
int128	liquidityNet	ΔL
uint128	liquidityGross	L_g
uint256	feeGrowthOutside0X128	$f_{o,0}$
uint256	feeGrowthOutside1X128	$f_{o,1}$

Table 2: Tick-Indexed State

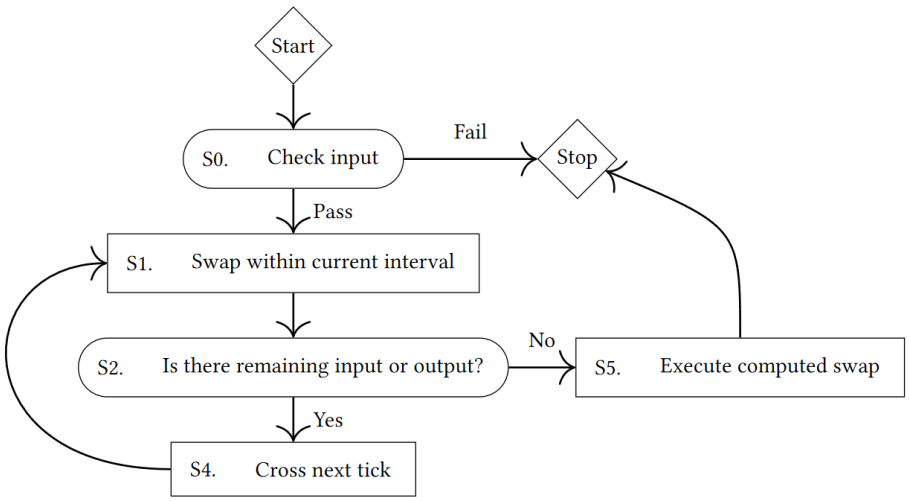

Figure 4: Swap Control Flow

每个刻度保存 ΔL ，代表该刻度被越过时应该被移进或移出的流动性数量。刻度只需要保存一个带符号整数：当刻度从左到右被越过时，增加的流动性数量 (或者，如果为负的话就是移除数量)。当刻度被越过时不需要修改该值 (只有位于该刻度的带边界头寸(position)被修改时需要修改)。

我们希望在刻度上不再有任何流动性的时候，重新将刻度置于未初始化状态。由于 ΔL 是差值，还需要保存该刻度的流动性总量：*liquidityGross*。该值可确保即便刻度的总流动性为0，我们仍然可以知道是否有至少一个头寸使用了该刻度，这告诉我们是否需要更新刻度图。

feeGrowthOutside{0,1} 用于保存给定区间内累积的手续费。由于 token0 和 token1 收取费用的公式相同，我们将省略本节后续部分的 token1 脚本。

取决于当前价格在区间内还是区间外——也即，当前刻度下标 i_c 大于或等于 i，您可以用下列公式计算每单位流动性，在刻度 i 之上(f_a)以及之下(f_b)，所获得的 token0 手续费：

$$f_a(i) = \begin{cases} f_g - f_o(i) & i_c \geq i \\ f_o(i) & i_c < i \end{cases} \tag{6.17}$$

$$f_b(i) = \begin{cases} f_o(i) & i_c \geq i \\ f_g - f_o(i) & i_c < i \end{cases} \tag{6.18}$$

我们可以用上述公式计算每单位 f_{i_l,i_u} ，在两个刻度 (刻度下界 i_l 和刻度上届 i_u) 之间，累积手续费的总金额：

$$f_{i_l,i_u}(0) = f_g - f_b(i_l) - f_a(i_u) \tag{6.19}$$

每次越过刻度时需要修改 $f_{o,1}$ 。具体而言，当刻度 i 从任意方向被越过时，它的 f_o （两个token的) 应作如下修改：

$$f_o(i) := f_g - f_o(i) \tag{6.20}$$

只有被用作至少一个头寸 (position) 的下边界或上边界的刻度，才需要保存 f_o 。所以为了提升效率，除非某个头寸以该刻度为边界，否则 f_o 不会初始化 (还有，当被越过时也不需要修改)。当刻度 i 的 f_o 初始化，给其赋值的规则是假定到目前为止，产生的所有手续费都位于该刻度以下：

$$f_o = \begin{cases} f_g & i_c \geq i \\ 0 & i_c < i \end{cases} \tag{6.21}$$

由于不同刻度的 f_o 值是在不同的时间初始化，所以比较不同刻度的 f_o 值没有意义，换句话说 f_o 的值不会保持一致性。这不会对每个头寸的记账造成问题，因为如下所述，由于链上计算本身具有一致性，所有头寸(position) 需要知道的只是给定区间内 g 的增量。

最后，合约为每个刻度保存 *secondsOutside*(t_o)。这个值可以被认为是在该刻度的另一边，已经花费的时间 (相对于当前价格而言)，还可以用于在特定区间内，在刻度之上或之下，已经花费的秒数。这个值没有在合约中使用，而是为外部合约提供便利，以便它们知道给定头寸已激活了多少秒。

和 f_a 、 f_b 的计算方法类似，取决于当前价格在区间内还是区间外，在给定刻度之上花费的时间 (t_a) 和之下 (t_b)花费的时间，其计算公式如下。用 t_a 和 t_b 的值还可以计算时间范围 (t_γ)：

$$t_a(i) = \begin{cases} t - t_o(i) & i_c \geq i \\ t_o(i) & i_c < i \end{cases} \tag{6.22}$$

$$t_b(i) = \begin{cases} t_o(i) & i_c \geq i \\ t - t_o(i) & i_c < i \end{cases} \tag{6.23}$$

$$t_\gamma(i_l, i_u) = t - t_b(i_l) - t_a(i_u) \tag{6.24}$$

要计算两个时间点 t_1 和 t_2 之间花费的秒数，可以分别记录 t_1 和 t_2 的 $t_\gamma(i_l, i_u)$ 值，然后用后者减前者得到。

与 f_o 类似，如果某个刻度没有被用作任何头寸 (position) 的边界， t_o 就无需保存。所以，直到有头寸以该刻度为边界， t_o 才会被初始化。其初始化赋值规则是，以 Unix 当前时间戳为其赋值：

$$t_o(i) := \begin{cases} t & i_c \geq i \\ 0 & i_c < i \end{cases} \tag{6.25}$$

与 f_o 类似，比较不同刻度的 t_o 没有意义。只有给定开始时间 (必须在两个刻度的 t_o 都已经初始化之后) 和结束时间的范围，计算流动性在该范围内的秒数时， t_o 才有意义。

6.3.1 越过刻度

如 6.2.3 小节所述，当在已初始化的刻度之间交易时，Uniswap V3 同样遵循恒定乘积公式。当交易越过一个已初始化刻度时，合约需要添加或移除流动性， 以确保没有做市商会产生损失。这意味着从该刻度获取 ΔL ，将其合并到全局 L 变量。

合约还需要修改刻度的拥有状态，保存以该刻度为边界的区间所获得的手续费 (以及花费的描述)。 $feeGrowthOutside\{0,1\}$ 和 $secondsOutside$ 同时被更新以反应当前值，以及相对于当前刻度的正确方向。

$$f_o := f_g - f_o \tag{6.26}$$

$$t_o := t - t_o \tag{6.27}$$

一旦刻度越过后，交易如 6.2.3 小节所述继续进行，直到到达下一个已初始化刻度。

6.4 每头寸的状态变量

合约有一个键值对，其键为账户地址、下届刻度（int24 的刻度索引号）、上届刻度（同上），其值为一个 Position 结构。每个 Position 保存下面3个值：

类型	变量名	符号
uint128	liquidity	
uint256	feeGrowthInside0LastX128	$f_{\gamma,0}(t_0)$
uint256	feeGrowthInside1LastX128	$f_{\gamma,1}(t_0)$

liquidity（ ）是头寸最后一次更新后的虚拟流动性数量。具体而言，liquidity 可以被认为是 $\sqrt{x \cdot y}$ ， x, y 分别代表做市商在该区间内，向流动池中提供 token0, token1 的虚拟数量。与 Uniswap V2（在V2中，流动性份额对应的 token 数量会不断增加）不同，流动性对应的 token 数量不会随着获得手续费的增加而变化，它总是对应着 $\sqrt{x \cdot y}$ 。

liquidity 值没有反映累积获得的手续费，为了计算该头寸累积获得的未提取手续费，需要另外的状态变量，feeGrowthInside0Last（ $f_{\gamma,0}(t_0)$ ）和 feeGrowthInside1Last（ $f_{\gamma,1}(t_0)$ ），如下所述。

6.4.1 setPosition 函数

setPosition 函数允许做市商修改他们的头寸。

setPosition 函数的两个参数——lowerTick 和 upperTick——与 msg.sender 组合在一起，唯一标识一个头寸。

函数的另一个参数——liquidityDelta，给定用户希望增加或移除（如果其值为负数）的虚拟流动性数量。

首先，函数计算该头寸每种 token 的未提取手续费，然后在存入的虚拟流动性所需 token 数量中予以扣除。

为了计算某种 token 的未提取手续费，您需要知道该头寸从上一次提取以来，其价格区间（如 6.3 小节所述，根据区间的 i_l 和 i_γ 计算)的 f_γ 增量。给定区间每单位流动性，在时间点 t_0 和 t_1 之间，手续费增量的计算公式是 $f_\gamma(t_1) - f_\gamma(t_0)$（其中，头寸的 feeGrowthInside{0,1}Last 变量保存了 $f_\gamma(t_0)$ ， $f_\gamma(t_1)$ 可以从当前刻度状态计算得到)。所得值乘以该头寸的 liquidity，即可得到该头寸 token0 的未提取手续费：

$$f_u = l \cdot (f_\gamma(t_1) - f_\gamma(t_0)) \tag{6.28}$$

然后，合约把 liquidityDelta 增加进该头寸的 liquidity 变量。它同时把 liquidityDelta 增加进该区间下届刻度的 liquidityNet 变量，还要从上届刻度的 liquidityNet 变量中减去 liquidityDelta（意思是当价格增长，越过下届刻度时，新增加的流动性需要被加进来；当价格增长，越过上届刻度时，新增加的流动性需要被减出去)。如果流动池的当前价格位于该头寸的价格区间内，合约还会将 liquidityDelta 添加进全局 liquidity 变量。

最后，根据销毁或铸造的流动性数量，流动池从用户转出（或者转入，如果 liquidityDelta 为负）相应数量的 token。

需要存入的 token0（ ΔX)或 token1（ ΔY ）数量，可以想象为如果价格从当前价格（ P)移动到上届刻度或下届刻度（对应 token0 或 token1)，头寸所卖出的 token 数量。下列公式可以由公式 6.14 和 6.16 推导得到，使用哪一个取决于当前价格是低于、在其间、还是高于本头寸的价格区间：

$$\Delta Y = \begin{cases} 0 & i_c < i_l \\ \Delta L \cdot (\sqrt{P} - \sqrt{p(i_l)}) & i_l \le i_c < i_u \\ \Delta L \cdot (\sqrt{p(i_u} - \sqrt{p(i_l)}) & i_c \ge i_u \end{cases} \tag{6.29}$$

$$\Delta X = \begin{cases} \Delta L \cdot (\frac{1}{\sqrt{p(i_l}} - \frac{1}{\sqrt{p(i_u)}}) & i_c < i_l \\ \Delta L \cdot (\frac{1}{\sqrt{P}} - \frac{1}{\sqrt{p(i_u)}}) & i_l \le i_c < i_u \\ 0 & i_c \ge i_u \end{cases} \tag{6.30}$$

词汇表

英文	中文	说明
liquidity provider	做市商	uniswap 的早期版本中，翻译成流动性提供者更好，因为他们没有主动调控能力，而在 v3 中，流动性提供者已经具备主动做市能力，虽然与传统做市商有很大差异，但这个词还是翻译成做市商，感觉更容易被中国人理解。
concentrated liquidity	聚焦流动性	流动性提供者通过限定价格范围，在指定价格区间内提供流动性。
range orders	限价订单	v3 的限价订单，也可以翻译成**区间订单**，与传统的限价订单还是存在两点差异。
oracle	预言机	对外部合约提供时间加权平均价格的报价。
TWAP	时间加权平均价格	time-weighted average price，时间加权平均价格
non-fungible liquidity	无法互换的流动性	uniswap 的早期版本中，流动性是由 ERC-20 代币表示，所以LP持有的流动性份额与持有的代币一样，可以分拆，可以相互发送。而 v3 的头寸 (流动性) 与账户地址、价格区间绑定，不能分拆，不支持完全可互换。
tick	价格刻度，刻度	ticks 可以想象为一把尺子，tick 就是尺子上的刻度，每一个tick 就是尺子上的一个价格刻度。两个刻度确定一个价格区间。
tickSpacing	刻度单位	价格尺子的最小刻度单位 (基点) 为 0.01%，但实际使用的刻度单位，不是最小刻度单位，而是以最小刻度单位的倍数来衡量。如：0.3%手续费，对应的刻度单位是60个最小刻度单位，即每刻度大约0.6%的价格偏差。
the lower bound	下届刻度	一个价格区间的下边界刻度
the upper bound	上届刻度	一个价格区间的上边界刻度
range	价格区间，区间	两个价格刻度之间的范围，称之为 range (价格区间)
position	头寸	与某一个账户地址关联之后的区间，称之为一个 position (头寸)
fees	手续费	