问：v3-core pool合约中的mint方法，参数是什么？

答：

* recipient，通常是NonfungiblePositionManager合约，当然，也可以不是
* tickLower, 价格下限
* tickUpper, 价格上限
* amount, 要铸造的流动性数量
* abi.encode(MintCallbackData({poolKey: poolKey, payer: msg.sender})), pool用于回调

前面四个参数最为重要，因为和pool中position息息相关

问：pool合约中，增减流动性最核心的方法是哪个？

答：

function \_modifyPosition(ModifyPositionParams memory params)

struct ModifyPositionParams {

// the address that owns the position 拥有该头寸的地址

address owner;

// the lower and upper tick of the position

int24 tickLower;

int24 tickUpper;

// any change in liquidity 流动性变化

int128 liquidityDelta;

}

function \_updatePosition(

address owner,

int24 tickLower,

int24 tickUpper,

int128 liquidityDelta, // 当新增流动性时，liquidityDelta为正数；当减少流动性时，liquidityDelta为负数

int24 tick // 池子当前的价格

) private returns (Position.Info storage position) { // 注意：返回的position是storage修饰的

liquidityDelta为正，则是增加流动性；liquidityDelta为负，则是减少流动性。

问：TICK的下限和上限分别是多少？

答：-887272和887272

问：整个池子的价格是存在什么状态变量里的？

答：存储在Slot0状态变量中，Slot0.tick

问：pool中是如何存储各LP的position的？

答：position owner, tickLower, tickUpper

以mapping的方式存储，键为

keccak256(abi.encodePacked(owner, tickLower, tickUpper))

值为Position.Info

library Position {

// info stored for each user's position

// 每个用户的头寸信息

struct Info {

// the amount of liquidity owned by this position

// 该头寸拥有的流动性数量

uint128 liquidity;

// fee growth per unit of liquidity as of the last update to liquidity or fees owed

// 截至上次更新流动性或所欠手续费时，每单位流动性的手续费增长

uint256 feeGrowthInside0LastX128;

uint256 feeGrowthInside1LastX128;

// the fees owed to the position owner in token0/token1

// token0/token1中欠头寸所有者的金额（本金+手续费）

uint128 tokensOwed0;

uint128 tokensOwed1;

}

问：当前池子中所有被position使用的tick有没有被存储下来？

答：有被存储到状态变量

mapping(int24 => Tick.Info) public override ticks;

// info stored for each initialized individual tick

// 为每个初始化的单个tick存储的信息

struct Info {

// the total position liquidity that references this tick

// 引用该tick的总头寸流动性

uint128 liquidityGross;

// amount of net liquidity added (subtracted) when tick is crossed from left to right (right to left),

// 当tick从左到右(从右到左)交叉时增加(减去)的净流动性数量

// 注意类型是有符号整数，当为正数时说明从左到右应该增加，当为负数时说明从左到右应该减去

int128 liquidityNet;

// fee growth per unit of liquidity on the \_other\_ side of this tick (relative to the current tick)

// only has relative meaning, not absolute — the value depends on when the tick is initialized

// 在tick的另一端的每单位流动性的手续费增长(相对于当前tick)

// 只有相对意义，而不是绝对意义-该值取决于tick的初始化时间

uint256 feeGrowthOutside0X128;

uint256 feeGrowthOutside1X128;

// the cumulative tick value on the other side of the tick

// tick另一端的累计tick值

int56 tickCumulativeOutside;

// the seconds per unit of liquidity on the \_other\_ side of this tick (relative to the current tick)

// only has relative meaning, not absolute — the value depends on when the tick is initialized

// 该tick的另一端每单位流动性的秒数(相对于当前tick)

// 只有相对意义，而不是绝对意义-该值取决于tick的初始化时间

uint160 secondsPerLiquidityOutsideX128;

// the seconds spent on the other side of the tick (relative to the current tick)

// only has relative meaning, not absolute — the value depends on when the tick is initialized

// 在tick的另一端花费的时间(相对于当前tick)

// 只有相对意义，而不是绝对意义-该值取决于tick的初始化时间

uint32 secondsOutside;

// true iff the tick is initialized, i.e. the value is exactly equivalent to the expression liquidityGross != 0

// these 8 bits are set to prevent fresh sstores when crossing newly initialized ticks

// true如果tick被初始化，即该值完全等价于表达式liquidityGross != 0

// 这8位被设置为在跨越新初始化的tick时防止新sstores

bool initialized;

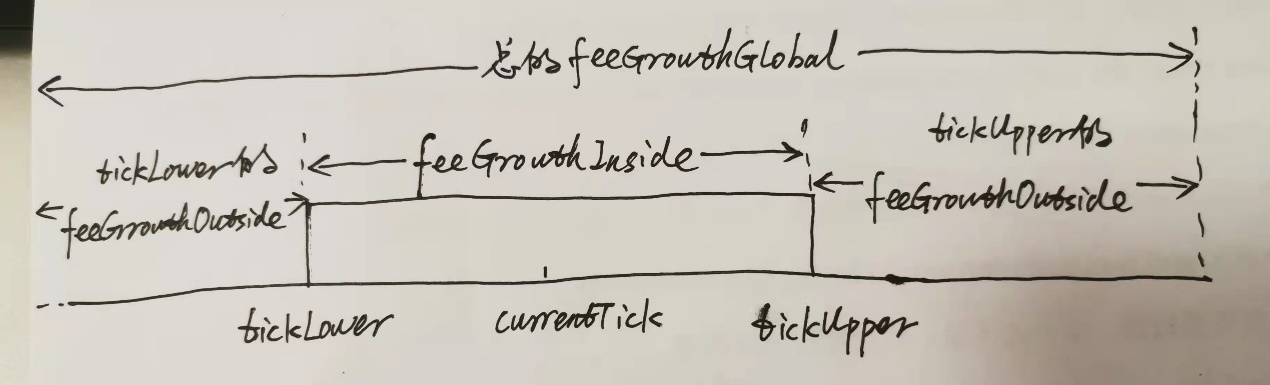
}

问：增减流动性时的大的逻辑是什么样的？

答：

* 更新该position的tickLower和tickUpper这两个tick的信息，具体更新逻辑请看“增减流动性时如何更新对应的tick信息？”
* 获取该position的tickLower和tickUpper之间每单位流动性手续费增长数据，即feeGrowthInside

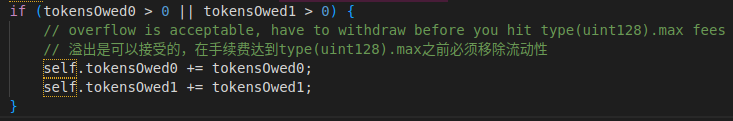
获取的方式如下，举其中一种场景，currentTick在tickLower和tickUpper之间的场景



feeGrowthInside=feeGrowthGlobal-tickLower的feeGrowthOutside- tickUpper的feeGrowthOutside

* 获取到了feeGrowthInside之后，用feeGrowthInside减去position的一个属性feeGrowthInsideLast，就得到了最近这段时间以来该position每单位流动性积累的新的手续费，再用该值乘以该position在本次更新流动性之前的liquidity，就得到了最近这段时间以来该position积累的手续费，即tokensOwed0, tokensOwed1，然后把其加到该position的tokensOwed0, tokensOwed1属性上。





* 更新该position的liquidity，或增或减
* 把该position的feeGrowthInsideLast更新为上面算出来的最新的feeGrowthInside
* 计算本次增减流动性LP需要付出或得到多少amount0,amount1

问：增减流动性时如何更新对应的tick信息？

答：

首先，一个tick可能会被多个position用到，有可能被用作tickLower，也有可能被用作tickUpper。

更新某个position时需要更新其tick信息，tickLower和tickUpper的更新逻辑有细微的不同。

以tickLower为例

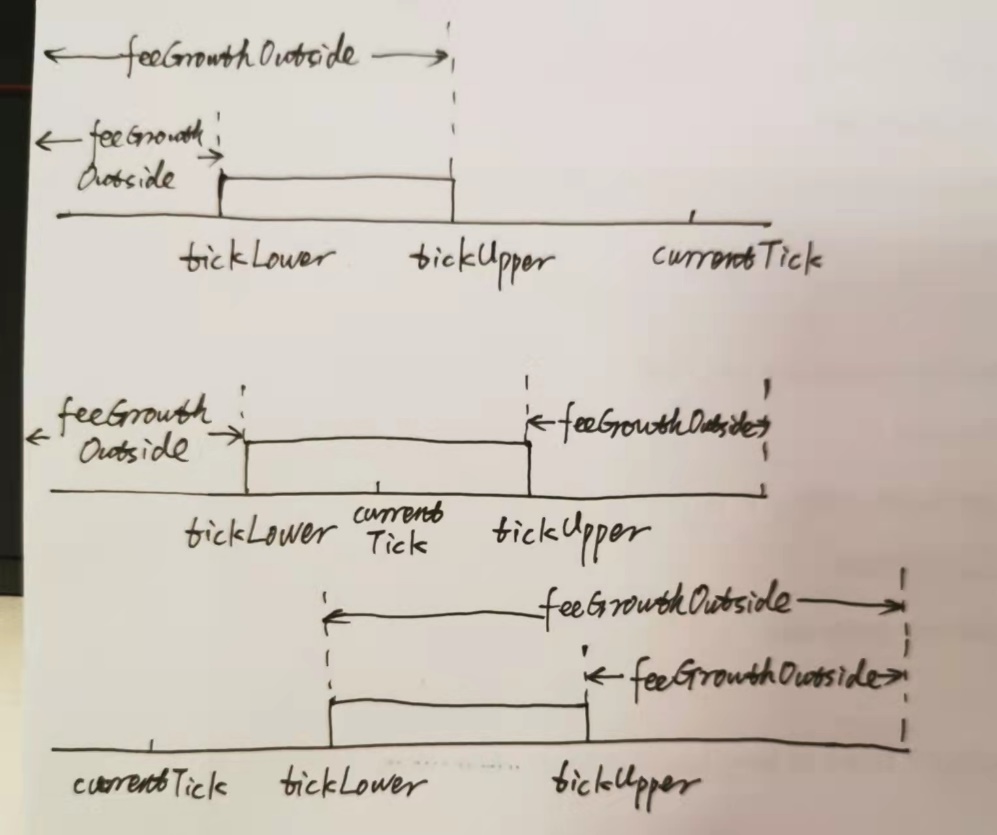
* 首先更新该tick的总流动性，如果position是新增流动性，那么tick的总流动性也相应增加，如果position是减少流动性，那么tick的总流动性也相应减少。一个tick可能被多个position使用，那么tick的总流动性就是使用这个tick的positions的流动性的总和。tick的总流动性和tick是lower tick还是upper tick无关。
* 如果一个tick的总流动性从0变非0，或从非0变0，则称该tick被flipped了。
* 更新当tick从左到右(从右到左)交叉时增加(减去)的净流动性数量。

增加流动性：tickLower的净流动性增加，tickUpper的净流动性减少

减少流动性：tickLower的净流动性减少，tickUpper的净流动性增加

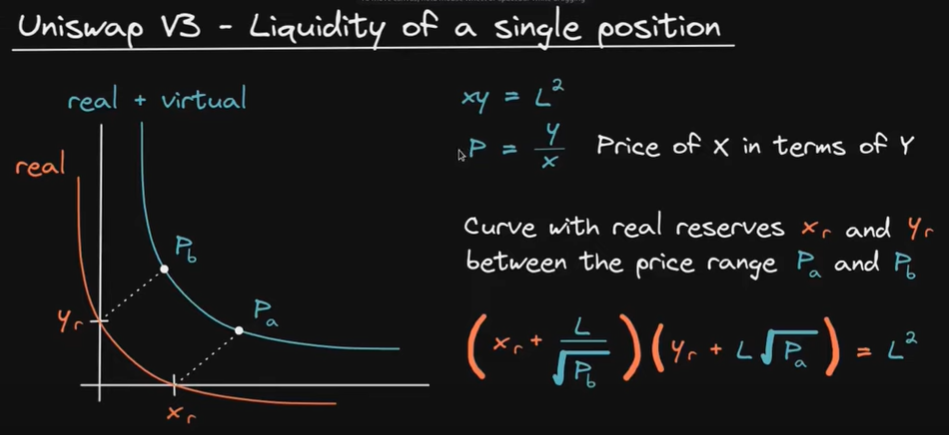
问：tick的feeGrowthOutside是什么含义？

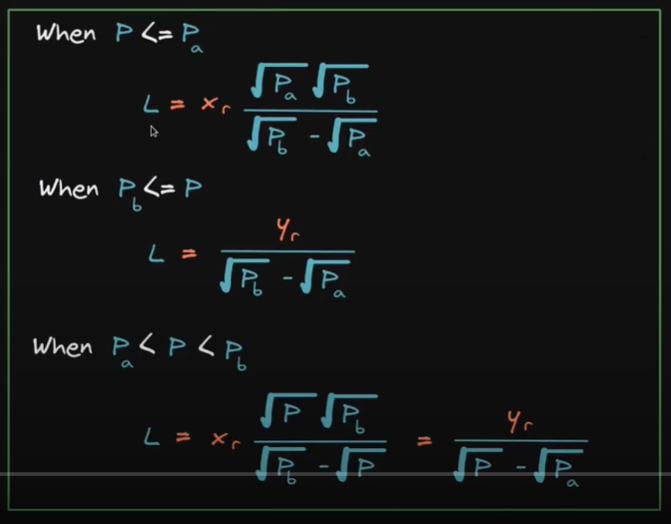
答：就是tick相对于currentTick的反方向每单位流动性积累的手续费，这对于一个position的tickLower和tickUpper没有区别。



问：给定L, P, Pa, Pb, 其中P是当前价格，Pa<Pb，Pa是价格下限，Pb是价格上限，计算xr和yr

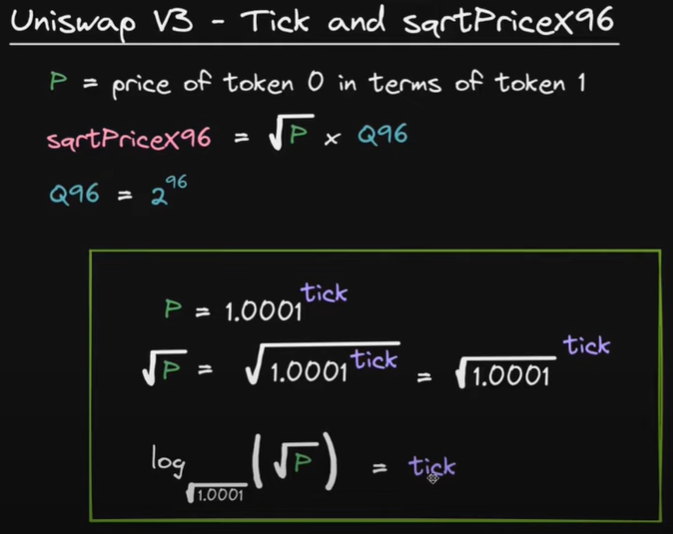
答：





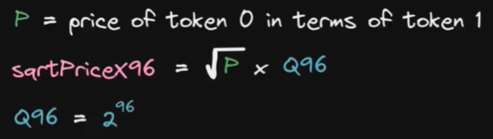
问：tick和价格P的关系是什么？

答：首先，P是指token0的价格

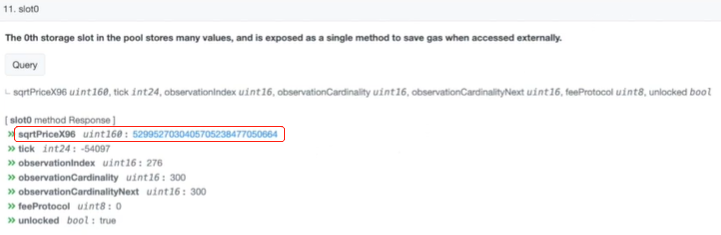


问：sqrtPriceX96是Q64.96类型，具体是怎么回事？

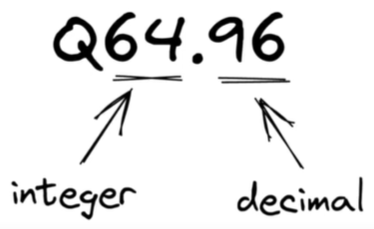
答：

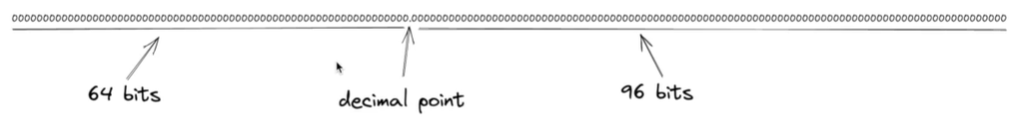


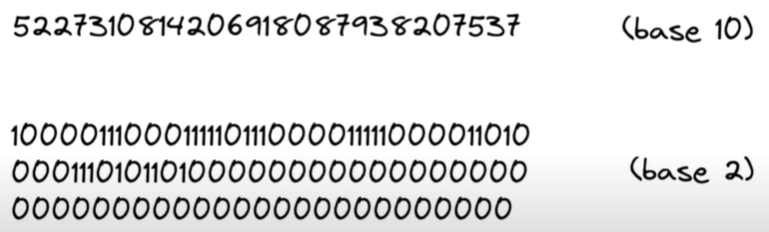


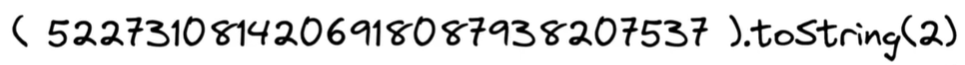




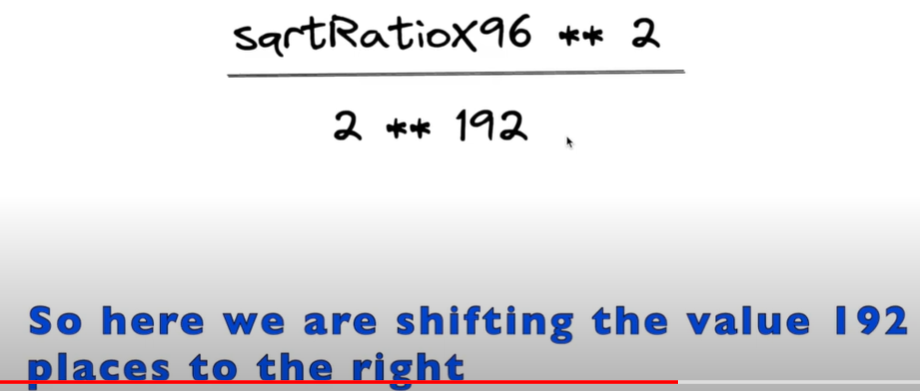


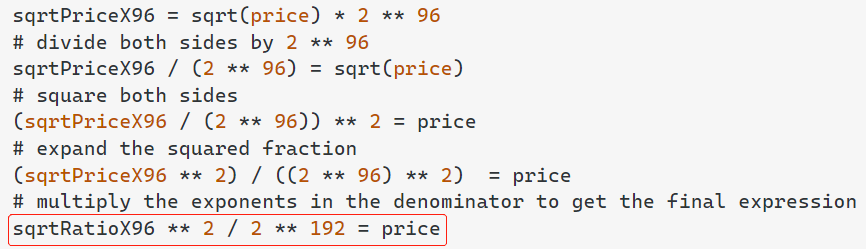


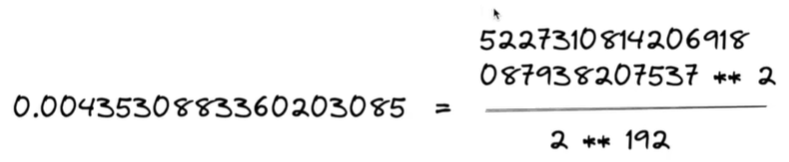












参考：

<https://ethereum.stackexchange.com/questions/98685/computing-the-uniswap-v3-pair-price-from-q64-96-number>

<https://www.youtube.com/watch?v=EV23xTgWsnY&list=PLXiAtWDhSlIvYpJQFv_OTgoOIygdbaKNP&index=20>