交易内存池

系统定义了交易内存池，存放交易信息，存放的交易是依据当前最优区块中的有效交易（valid-according-to-the-current-best-chain transactions），且可能出现在下一个区块中。

节点创建的交易、网络接收的交易会添加到内存池中，但并不是所有的交易都会添加到内存池中，如果新的交易的输入已存在内存池中的某个交易中，则丢弃。

比特币：

比特币节点需要为内存池每笔交易都分配有优先级，并选择优先级较高的交易记录来构造候选区块。而比特币的优先级是根据交易输入的块龄，交易的输入值来定的：

Priority = Sum (Value of input \* Input Age) / Transaction Size

UTXO的“块龄”是自该UTXO被记录到区块链为止所经历过的区块数，即这个UTXO在区块链中的深度;

交易记录的大小由字节来表示（一般标准的交易一个输入，两个输出的大小大概是250字节）;

一个交易想要成为“较高优先级”，需满足的条件：优先值大于57,600,000，相当于一个比特币（即100万聪），年龄为一天（144个区块），交易的大小为250个字节：

High Priority > 100,000,000 satoshis \* 144 blocks / 250 bytes = 57,600,000

// Can we complete this as a free transaction?

//判断交易单大小是否不大于MAX\_FREE\_TRANSACTION\_CREATE\_SIZE（1000字节）

// fSendFreeTransactions是否支持免费

if (fSendFreeTransactions && nBytes <= MAX\_FREE\_TRANSACTION\_CREATE\_SIZE)

{

// Not enough fee: enough priority?

// 根据nTxConfirmTarget求出优先权需要满足的条件

double dPriorityNeeded = mempool.estimateSmartPriority(nTxConfirmTarget);

// Require at least hard-coded AllowFree.

if (dPriority >= dPriorityNeeded && AllowFree(dPriority))

break;

}

Dacrs：

没有提供免费的交易,根据小费来计算其优先权：

double GetPriority() const {

return llFees / GetSerializeSize(SER\_NETWORK, PROTOCOL\_VERSION);

}

区块中用来存储交易的前50K字节是保留给较高优先级交易的。比特币节点在填充这50K字节的时候，会优先考虑这些最高优先级的交易，不管它们是否包含了矿工费。这种机制使得高优先级交易即便是零矿工费，也可以优先被处理。然后比特币的挖矿节点会选出那些包含最小矿工费的交易，并按照“每千字节矿工费”进行排序，优先选择矿工费高的交易来填充剩下的区块，区块大小上限为MAX\_BLOCK\_SIZE。如区块中仍有剩余空间，比特币的挖矿节点可以选择那些不含矿工费的交易。有些矿工会竭尽全力将那些不含矿工费的交易整合到区块中，而其他矿工也许会选择忽略这些交易。由于交易的优先值取决于它交易输入的“块龄”，所以这个交易的优先值也就随之增长了。最后，一个零矿工费 交易的优先值就有可能会满足高优先级的门槛，被免费地打包进区块。比特币交易中没有过期、超时的概念，一笔交易现在有效，那么它就永远有效。然而，如果一笔交易只在全网广播了一次，那么它只会保存在一个挖矿节点的 内存中。因为内存池是以未持久化的方式保存在挖矿节点存储器中的，所以一旦这个节点重新启动，内存池中的数据就会被完全擦除。而且，即便一笔有效交易被传 播到了全网，如果它长时间未处理，它将从挖矿节点的内存池中消失。如果交易本应该在一段时间内被处理而实际没有，那么钱包软件应该重新发送交易或重新支付 更高的矿工费。

交易内存池：

系统定义了交易内存池（CTxMemPool）存放交易信息!

class CTxMemPool;

交易内存池中由交易内存池项构成：

系统定义了交易内存池项（CTxMemPoolEntry）描述每个交易信息！

class CTxMemPoolEntry;

以下是Dacrs的内存池添加交易流程：

网络接收交易信息：

节点接收到"tx”命令时，把交易添加到内存池中（AcceptToMemoryPool）。先对交易进行检查，然后构造交易内存池项，再添加到交易内存池中，最后同步钱包。

满足以下几点才能添加到内存池中：

1.判断交易是否存在内存池中!

2.交易是否已经被确认。

3.判断交易是否是来自CoinBase，

4.交易是否在有效高度!

5.交易的输入不在交易内存池中!

6.交易不在币视图内存池中!

7.交易的输入的输出须在币视图内存池中!

8.交易的输入是有效的!

9.如果网络是MAIN类型，则交易的输入须是标准的!

10.如果限制免费交易，则交易费不能低于最小交易费（GetMinFee函数获取最小交

易费）!

11.当限制免费交易，交易费低于最小传播交易费时，必须限制交易的流量，即免费

交易的大小不能超过免费传播最大值，默认是15\*10\*1000字节，可以通过参数”-limitfreerelay"修改!

12.如果限制交易费最大值，则交易费必须小于最小传播交易费的10000倍

（CTransaction::nMinRelayTxFee \* 10000）!

13.交易的输入不能有二次花费（CheckInputs）!

添加到内存池中不在进行交易的正常检查，直接添加!

初始化交易项时，交易费的计算公式是： int64\_t nFees = view.GetValueIn(tx)-tx.GetValueOut();

优先级的计算公式是：double dPriority = view.GetPriority(tx, chainActive.Height());

高度是chainActive.Height()。

获取节点交易内存池数据：

节点发送”mempool”命令获取交易内存池数据，接收方发送”inv"命令返回数据!返回的数据是类CInv的数组，类CInv包含了类型、交易项的哈希值，类型默认是MSG\_TX。每次最多发送50000个!

节点接收到”mempool”命令后，先获取交易内存池中的所有的交易项的哈希值（queryHashes函数），然后构造类CInv对象，最后发送出去。发送之前要在内存池中查找此哈希值是否存在，因为其他线程可能移除此哈希项，依然存在的项才发送!