# NEO源码阅读记录

## 基础结构类型

1. UInt160

用来标记Address的散列值

1. UInt256

用来标记区块数据、交易数据、资产类型的散列值

1. ECPoint

用来标记一个公钥

## IO模块

### DB

LevelDB的接口封装类，Key-Value数据库，常用操作接口：

1. Get(key)
2. Put(key, value)
3. TryGet(key, out value)
4. Write(write\_batch)

### DbCache : DataCache

1. 用Dictionary记录key-value数据，常用接口:
2. GetAndChange(key, factory)

用key查询内存中的Dictionary，如果没有缓存过，则查询LevelDB，并把结果缓存在Dictionary中，如果数据库中也没有，则用factory函数创建一个并缓存

1. TryGet(key)

先在Dictionary中找，如果没有再查询LevelDB并缓存，还没有则返回null

1. Delete(key)

从Dictionary中删除缓存

1. Commit()

把Dictionary中标记成增、删、改的数据写回到LevelDB数据库里

1. Dictionary缓存的数据标记了四种状态：None, Add, Changed, Deleted，这个标记是为了记录Dictionary中的数据对应LevelDB里的数据状态差异，方便把缓存中的数据同步写回数据库。
2. None:没有差异
3. Add:增加到缓存里的数据
4. Changed:从数据库里获取后，在缓存中已修改过的数据
5. Delete:数据库里有，但缓存中已删除的数据

### Core模块

### BlockBase

1. MerkleRoot //该区块中所有交易的Merkle树的根

该变量的生成代码：

MerkleRoot = MerkleTree.ComputeRoot(Transactions.Select(p => p.Hash).ToArray());

将Block中所有Transaction作为最底层的叶子节点，按照Merkle树（类似二叉树）的生成算法，从叶子节点开始，每两个节点生成一个父节点，并将这两个节点的Hash合并后生成这个父节点的Hash。如此逐层向上，直至构建完整个Merkle树。

Merkle树，可以理解为二叉树，其中每个节点有一个对应的Hash值，这里的MerkleRoot就是根节点的Hash。

1. ConsensusData

Nonce是或Number once的缩写，在密码学中Nonce是一个只被使用一次的任意或非重复的随机数值。创世块的ConsensusData使用比特币创世块的Nonce值2083236893。

***在Neo里，ConsensusData没有实际作用？***

1. NextConsensus //下一个区块的记账合约的散列值

生成：NextConsensus = GetConsensusAddress (GetValidators(transactions).ToArray());

***作用：验证本区块的合法性？***

### LevelDBBlockChain

1. 程序流程

内部使用一个独立线程，通过AutoResetEvent来控制线程的运行和挂起。

1. 对外主要接口：

* AddBlock(block)

在主线程中调用，先把要写入数据库的block保存在block\_cache中，等线程函数完成数据写入后再从block\_cache中删除。

* GetHeader(height) / GetHeader(hash)

获取区块头，先在header\_cache中找，没有再去数据库中查询。

因为header\_cache只是做短暂的保存，所以大部分时候是要查数据库获取的。

1. 主要成员

* List<UInt256> header\_index

记录链上所有区块头的Hash数据

* Dictionary<UInt256, Header> header\_cache

区块头的缓存，只是短暂的保存，在数据写入LevelDB后就会清除

* Dictionary<UInt256, Block> block\_cache

区块数据的缓存，只是短暂的保存，在数据写入LevelDB后就会清除