交易数据结构

区块链转账的基本概念和流程

- 用户输入转账的地址和转入的地址和转出的金额
- 系统通过转出的地址的私钥对转账信息进行签名（用于证明这笔交易确实由本人进行）
- 系统对交易信息进行验证
- 把这笔交易入到本地的txpool中（就是缓存交易池）
- 把交易信息广播给其它节点

每条交易对应存储两条数据，一条是交易本身，一条是交易的元信息（meta）。交易以交易的hash为key、交易的RLP编码为value存储；元信息以txHash+txMetaSuffix为key、元信息的RLP编码为value存储。元信息中包含交易所在区块的区块hash、区块号、交易在区块中的索引。具体可以看WriteTransactions函数

```
// hash size from是缓存字段
/cores/types/transaction.go
type Transaction struct {
    data txdata
    // caches
    hash atomic.Value
    size atomic.Value
    from atomic.Value
}

type txdata struct {
    AccountNonce uint64          `json:"nonce"    gencodec:"required"`
    Price        *big.Int        `json:"gasPrice" gencodec:"required"`
    GasLimit     uint64          `json:"gas"      gencodec:"required"`
    Recipient    *common.Address `json:"to"       rlp:"nil"` // nil means contract
creation
    Amount       *big.Int        `json:"value"    gencodec:"required"`
    Payload      []byte          `json:"input"    gencodec:"required"`

    // Signature values
    V *big.Int `json:"v" gencodec:"required"`
    R *big.Int `json:"r" gencodec:"required"`
    S *big.Int `json:"s" gencodec:"required"`

    // This is only used when marshaling to JSON.
    Hash *common.Hash `json:"hash" rlp:"-"`
```

```
}
```

AccountNonce: 发送者发起的交易总数量

Price: 此次交易的gas价格

GasLimit: 本次交易允许消耗的最大Gas数

Recipient: 交易接收者的地址

Amount: 此次交易转移的以太币数量

Payload: 其它数据

V: 交易的签名数据

R: 交易的签名数据

S: 交易的签名数据

> txdata结构中没有指明交易的发送者，因为交易的发送者地址可以从签名中得到