以太坊底层实现解析-存储

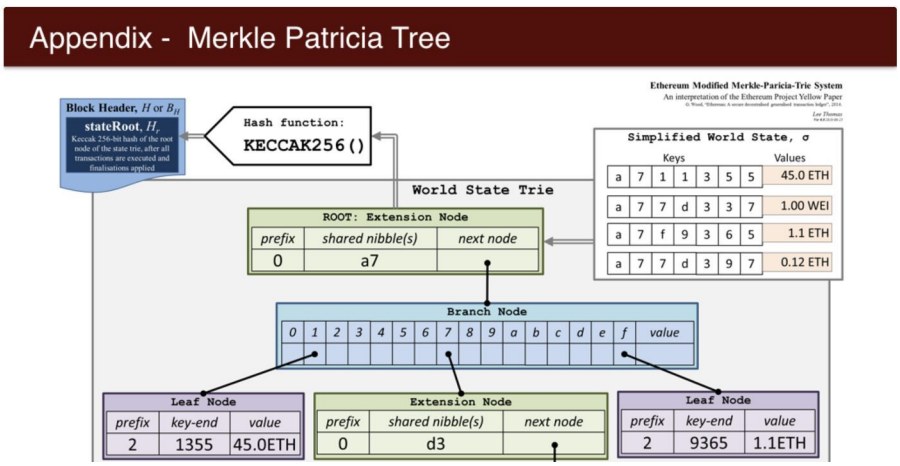
蒋天星
程序员

一、相关概念

以太坊在区块、账户信息持久化到数据库过程中，多次涉及到Merkle Patricia Tree（MPT）、Recursive Length Prefix（RLP） 编码等技术。例如：以太坊会将与交易信息相关的 MPT 树节点以 key-value 的形式持久化到数据库中，其中 key 为节点 hash，value 为节点的 RLP编码。

Merkle Patricia Tree

Merkle Patricia Tree结合了 Merkle Tree 和 前缀树 两种数据结构的优点。以太坊以此来组织管理账户、交易、收据等信息。



MPT 树有以下几个作用：

1. 存储任意长度的key-value键值对数据
2. 提供了一种快速计算所维护数据集哈希标识的机制
3. 提供了快速状态回滚的机制
4. 提供了一种称为***默克尔证明***的证明方法，进行轻节点的扩展，实现简单支付验证

MPT 树与 Merkle 树相比，有较大差异。首先，MPT树中非叶子节点的 hash 的计算方式不再直接由其两个孩子节点的 hash 计算而成，而是根据当前节点计算 hash，而孩子节点的 hash 作为当前节点的一个字段参与了计算；其次，在进行默克尔证明时，全节点只需要给出由根节点至相关叶子节点的路径；另外，在比特币中 Merkle Tree 管理的是当前区块的交易数据，而以太坊中 MPT 树可以管理全局的信息，并且能够快速进行回滚。

Recursive Length Prefix，RLP编码

二、以太坊数据库结构

以太坊采用了 levelDB 定制自己的数据库，而 levelDB 支持将文件持久化到本地文件中，因此以太坊将常用的一些元数据、区块信息、映射表、区块信息全部存储在 levelDB 中，并定期将区块信息持久化到不可更改的本地文件中。相关的模式定义在 /core/rawdb/schema.go 中。

```
/core/rawdb/schema.go
// 需要持久化的信息
const (
        freezerHeaderTable = "headers"
        freezerHashTable = "hashes"
        freezerBodiesTable = "bodies"
        freezerReceiptTable = "receipts"
        freezerDifficultyTable = "diffs"
)

// levelDB 中存储的信息
// The fields below define the low level database schema prefixing.
var (
        databaseVersionKey = []byte("DatabaseVersion")
        headHeaderKey = []byte("LastHeader")
        headBlockKey = []byte("LastBlock")
        headFastBlockKey = []byte("LastFast")
        headerPrefix       = []byte("h") // headerPrefix + num (uint64 big end
        headerTDSuffix     = []byte("t") // headerPrefix + num (uint64 big end
        headerHashSuffix   = []byte("n") // headerPrefix + num (uint64 big end
        headerNumberPrefix = []byte("H") // headerNumberPrefix + hash -> num (
        ......
)
```

为了便于理解以太坊数据库的整体架构，我整理了相关结构体之间的关系。但由于 Go 语言没有继承的语义，结构体之间没有父子的概念，只有局部与整体的概念，因此用 UML 类图表示可能存在一些问题。

以太坊定义了 KeyValueReader、KeyValueWriter 两个接口用于管理存储在 levelDB 中的数据。同时，为了支持批量处理、内容压缩、属性查看等功能，以太坊在此基础上组合了 io.Closer、Batcher、Iteratee、Statee、Compactor 等接口形成了 KeyValueStore 接口。

KeyValueStore 接口有两种实现，首先是封装了 levelDB 的 leveldb.Database，还有一个是 memorydb.Database 用于系统测试。AncientReader、AncientWriter 两个接口定义了管理持久

化数据的方法，在 ethdb.KeyValueStore 的基础上组合这两个接口就形成了以太坊高层使用的 ethdb.Database。

freezerTable 用于维护 headers、hashes、bodies、receipts 和 diffs 五张表，它们分别代表一组文件，freezerTable 实现了对于这组文件的基础操作。

rawdb.freezer 是 AncientStore（由 AncientReader、AncientWriter 接口组合形成）的实现，基于 freezerTable 它实现了接口的相关功能，并且 freezer 会定时从 leveldb.Database 中将区块数据持久化到 freezerTable 中，并删除 leveldb.Database 中的数据。

ethdb.Database 有三种实现，其中 nofreezedb、freezedb 是数据库级别的，而 table 是在前两者的基础上进行了封装处于 table 级别。nofreezedb 实际上就是一个 KeyValueStore ，在实现 AncientStore 接口的所有方法时，直接返回了 errNotSupport 的错误。freezedb 由 KeyValueStore 与 freezer 两者组合形成。

三、代码

1. 接口定义

具体请查看 /core/database.go

2. KeyValueStore 接口实现

```
// leveldb.Database /ethdb/memeory/database.go
type Database struct {
        db   map[string][]byte
        lock sync.RWMutex
}

func (db *Database) Has(key []byte) (bool, error) {
        db.lock.RLock()
        defer db.lock.RUnlock()

        if db.db == nil {
                return false, errMemorydbClosed
        }
        _, ok := db.db[string(key)]
        return ok, nil
}
```

memorydb.Database 的实现比较简单，直接将所有的 key-value 数据放进一个 map 中，并在读写操作前分别申请读写锁避免并发读写冲突。同时 memeorydb.Database 还支持遍历、批量读写等操作。

```
// leveldb.Database /ethdb/leveldb/database.go
type Database struct {
        fn string      // filename for reporting
        db *leveldb.DB // LevelDB instance

        compTimeMeter      metrics.Meter // Meter for measuring the total time
        compReadMeter      metrics.Meter // Meter for measuring the data read c
        compWriteMeter     metrics.Meter // Meter for measuring the data writte
        writeDelayNMeter   metrics.Meter // Meter for measuring the write delay
        writeDelayMeter    metrics.Meter // Meter for measuring the write delay
        diskSizeGauge      metrics.Gauge // Gauge for tracking the size of all
        diskReadMeter      metrics.Meter // Meter for measuring the effective a
        diskWriteMeter     metrics.Meter // Meter for measuring the effective a
        memCompGauge       metrics.Gauge // Gauge for tracking the number of me
        level0CompGauge    metrics.Gauge // Gauge for tracking the number of ta
        nonlevel0CompGauge metrics.Gauge // Gauge for tracking the number of ta
        seekCompGauge      metrics.Gauge // Gauge for tracking the number of ta
```


it channel access

```
        quitChan chan chan error // Quit channel to stop the metrics collectio

        log log.Logger // Contextual logger tracking the database path
}
```

leveldb.Database 通过 leveldb.DB 的 API 实现了 KeyValueStore 的各种接口，同时 leveldb.Database 中还设置了各种类型的 Metrics （Gauge、Counter、Meter、Histogram、Timer）以用于测量数据库的性能。

```
// leveldb.Database /ethdb/leveldb/database.go
func (db *Database) Close() error {
        db.quitLock.Lock()
        defer db.quitLock.Unlock()

        if db.quitChan != nil {
                errc := make(chan error)
                db.quitChan <- errc
                if err := <-errc; err != nil {
                        db.log.Error("Metrics collection failed", "err", err)
                }
                db.quitChan = nil
        }
        return db.db.Close()
}
```

leveldb.Database 设置了 quitChan，在关闭数据库时，会通过 quitChan 关闭常驻内存的用于测量性能的 meter goroutine。

```
// leveldb.Database /ethdb/leveldb/database.go
// meter periodically retrieves internal leveldb counters and reports them to
// the metrics subsystem.
//
// This is how a LevelDB stats table looks like (currently):
//   Compactions
//    Level |   Tables   |    Size(MB)   |    Time(sec)  |    Read(MB)   |   Wr
//   -------+------------+---------------+---------------+---------------+-----
//      0   |          0 |       0.00000 |       1.27969 |       0.00000 |
//      1   |         85 |     109.27913 |      28.09293 |     213.92493 |
//      2   |        523 |    1000.37159 |       7.26059 |      66.86342 |
//      3   |        570 |    1113.18458 |       0.00000 |       0.00000 |
//
// This is how the write delay look like (currently):
// DelayN:5 Delay:406.604657ms Paused: false
//
// This is how the iostats look like (currently):
// Read(MB):3895.04860 Write(MB):3654.64712
func (db *Database) meter(refresh time.Duration) {
        ......
}
```

leveldb.Database 的 meter 方法在 Database 初始化时，便会启动并一直常驻内存，并周期性地调用 db.db.GetProperty("leveldb.stats") 获得状态信息（如注释所示），并通过字符串截取获得数据再存储在 Metric 中。

3. AncientStore 实现

```
// rawdb.freezerTable /core/rawdb/freezer_table.go
type freezerTable struct {
        items uint64 // Number of items stored in the table (including items re

        noCompression bool   // if true, disables snappy compression. Note: doe
        maxFileSize   uint32 // Max file size for data-files
```

```
        head   *os.File            // File descriptor for the data head of the
        files  map[uint32]*os.File // open files
        headId uint32              // number of the currently active head file
        tailId uint32              // number of the earliest file
        index  *os.File            // File descriptor for the indexEntry file (

        itemOffset uint32 // Offset (number of discarded items)

        headBytes  uint32        // Number of bytes written to the head file
        readMeter  metrics.Meter // Meter for measuring the effective amount o
        writeMeter metrics.Meter // Meter for measuring the effective amount o
        sizeGauge  metrics.Gauge // Gauge for tracking the combined size of al

        logger log.Logger   // Logger with database path and table name ambedde
        lock   sync.RWMutex // Mutex protecting the data file descriptors
}

type indexEntry struct {
        filenum uint32 // stored as uint16 ( 2 bytes)
        offset  uint32 // stored as uint32 ( 4 bytes)
}

const indexEntrySize = 6
```

上面我们提到的需要持久化的每一张表都是一个 freezerTable，实际上一个 freezerTable 会根据存储内容的大小分为多个小文件，freezerTable 通过管理这组文件完成 retrive、release、append、sync、truncate 这些基本功能。注意：freezerTable 除了在数据持久化过程中出错时进行 truncate 以及完全删除所有数据的 release 外，是不提供修改历史数据的，这也符合 Ancient 的定义了。

freezerTable 结构体中的大部分字段是为了便于索引文件的，其中 index 是指索引文件的文件描述符，表内容会进行分片，在 index 中每个文件由一个 indexEntry 保存了对应的文件名（数字）和文件大小。files 是由文件名至文件描述符的映射，这些文件可以看成是一个文件链表（新在前，旧在后），headId、tailId 分别指头节点、尾节点的文件名。itemOffset 指被丢弃的文件数目，items 指当前表中已经存储的文件数目（包括从尾部删除的文件）。

```
// rawdb.freezerTable /core/rawdb/freezer_table.go
func openFreezerFileForAppend(filename string) (*os.File, error) {
        file, err := os.OpenFile(filename, os.O_RDWR|os.O_CREATE, 0644)
        if err != nil {
                return nil, err
        }
        // Seek to end for append
        if _, err = file.Seek(0, io.SeekEnd); err != nil {
                return nil, err
        }
        return file, nil
}

func openFreezerFileForReadOnly(filename string) (*os.File, error) {}

func openFreezerFileTruncated(filename string) (*os.File, error) {}
```

freezerTable 在打开文件时，会设置相应的读写权限，以限定只能用于 append 或 truncate.

```
// rawdb.freezerTable /core/rawdb/freezer_table.go
func newCustomTable(path string, name string, readMeter metrics.Meter, writeMe
        if err := os.MkdirAll(path, 0755); err != nil {
                return nil, err
        }
        var idxName string
        if noCompression {
                idxName = fmt.Sprintf("%s.ridx", name)
```

```
        }
        offsets, err := openFreezerFileForAppend(filepath.Join(path, idxName))
        if err != nil {
                return nil, err
        }
        tab := &freezerTable{
                index:         offsets,
                files:         make(map[uint32]*os.File),
                readMeter:     readMeter,
                writeMeter:    writeMeter,
                sizeGauge:     sizeGauge,
                name:          name,
                path:          path,
                logger:        log.New("database", path, "table", name),
                noCompression: noCompression,
                maxFileSize:   maxFilesize,
        }
        if err := tab.repair(); err != nil {
                tab.Close()
                return nil, err
        }
        size, err := tab.sizeNolock()
        if err != nil {
                tab.Close()
                return nil, err
        }
        tab.sizeGauge.Inc(int64(size))

        return tab, nil
}
```

freezerTable 在创建时在指定文件夹下创建并打开相应表的 indexEntry 文件，其中 index 是指当前表所在的文件。

```
// rawdb.freezerTable /core/rawdb/freezer_table.go
func (t *freezerTable) repair() error {
        buffer := make([]byte, indexEntrySize)
        stat, err := t.index.Stat()
        if err != nil {
                return err
        }
        if stat.Size() == 0 {
                if _, err := t.index.Write(buffer); err != nil {
                        return err
                }
        }
        if overflow := stat.Size() % indexEntrySize; overflow != 0 {
                truncateFreezerFile(t.index, stat.Size()-overflow) // New file
        }
        if stat, err = t.index.Stat(); err != nil {
                return err
        }
        offsetsSize := stat.Size()
        var (
                firstIndex  indexEntry
                lastIndex   indexEntry
                contentSize int64
                contentExp  int64
        )
        t.index.ReadAt(buffer, 0)
        firstIndex.unmarshalBinary(buffer)

        t.tailId = firstIndex.filenum
        t.itemOffset = firstIndex.offset
```

```
        t.head, err = t.openFile(lastIndex.filenum, openFreezerFileForAppend)
        if err != nil {
                return err
        }
        if stat, err = t.head.Stat(); err != nil {
                return err
        }
        contentSize = stat.Size()

        contentExp = int64(lastIndex.offset)

        for contentExp != contentSize {
                if contentExp < contentSize {
                        t.logger.Warn("Truncating dangling head", "indexed", c
                        if err := truncateFreezerFile(t.head, contentExp); err
                                return err
                        }
                        contentSize = contentExp
                }
                if contentExp > contentSize {
                        t.logger.Warn("Truncating dangling indexes", "indexed",
                        if err := truncateFreezerFile(t.index, offsetsSize-inde
                                return err
                        }
                        offsetsSize -= indexEntrySize
                        t.index.ReadAt(buffer, offsetsSize-indexEntrySize)
                        var newLastIndex indexEntry
                        newLastIndex.unmarshalBinary(buffer)
                        // We might have slipped back into an earlier head-file
                        if newLastIndex.filenum != lastIndex.filenum {
                                // Release earlier opened file
                                t.releaseFile(lastIndex.filenum)
                                if t.head, err = t.openFile(newLastIndex.filenu
                                        return err
                                }
                                if stat, err = t.head.Stat(); err != nil {
                                        // TODO, anything more we can do here?
                                        // A data file has gone missing...
                                        return err
                                }
                                contentSize = stat.Size()
                        }
                        lastIndex = newLastIndex
                        contentExp = int64(lastIndex.offset)
                }
        }
        if err := t.index.Sync(); err != nil {
                return err
        }
        if err := t.head.Sync(); err != nil {
                return err
        }
        t.items = uint64(t.itemOffset) + uint64(offsetsSize/indexEntrySize-1) /
        t.headBytes = uint32(contentSize)
        t.headId = lastIndex.filenum

        if err := t.preopen(); err != nil {
                return err
        }
        t.logger.Debug("Chain freezer table opened", "items", t.items, "size",
        return nil
}
```

编辑于 2021-08-21 22:03

▲ 赞同 ▼ 添加评论 分享 喜欢 收藏 申请转载 ···

文章被以下专栏收录

以太坊源码阅读

推荐阅读

以太坊介绍（二）：区块链数据的存储和更新

NoBoundary

基于Java语言构建区块链（一）—— 基本原型

王维

如何在不同链部署地址完全相同的合约

廖雪峰

简述三大区块

目前市场上区块
非像我们想象的
区块链，而是基
层平台去开发自
介绍三种主流的
比特币、以太坊

BlockDAO

还没有评论

写下你的评论...