读代码从创世区块源码解析开始

一、生成创世区块

1.首先回顾一下生成创世区块的命令为：

`geth --datadir data0 init genesis.json`

2.了解操作创世区块的目录为：

```
geth main 方法在 /cmd/geth/main.go。init 调用的是 cmd/geth/chaincmd.go 中 initCommand
, initCommand 调用的是 initGenesis(ctx *cli.Context)
```

生成创世区块的具体方法

```
func initGenesis(ctx *cli.Context) error {
    // Make sure we have a valid genesis JSON
    genesisPath := ctx.Args().First()
    if len(genesisPath) == 0 {
        utils.Fatalf("Must supply path to genesis JSON file")
    }

    file, err := os.Open(genesisPath)
    if err != nil {
        utils.Fatalf("Failed to read genesis file: %v", err)
    }
    defer file.Close()

    genesis := new(core.Genesis)
    if err := json.NewDecoder(file).Decode(genesis); err != nil {
        utils.Fatalf("invalid genesis file: %v", err)
    }
    // Open an initialise both full and light databases
    stack := makeFullNode(ctx)
    for _, name := range []string{"chaindata", "lightchaindata"} {
        chaindb, err := stack.OpenDatabase(name, 0, 0)
        if err != nil {
            utils.Fatalf("Failed to open database: %v", err)
        }
        _, hash, err := core.SetupGenesisBlock(chaindb, genesis)
        if err != nil {
            utils.Fatalf("Failed to write genesis block: %v", err)
        }
```

```
            log.Info("Successfully wrote genesis state", "database", name, "hash", hash
)
        }
        return nil
}
```

首先通过genesisPath := ctx.Args().First()代码确保有一个合法的创世区块JSON文件，然后打开文件获得文件内容，这里最重要的是json.NewDecoder方法对获得的JSON文件内容进行解码，然后创建创世区块并写入数据库core.SetupGenesisBlock(chaindb, genesis)，接下来我们详细分析core.SetupGenesisBlock方法

二、创世区块核心代码

的文件在 `/core/genesis.go`

结构体

```
// 下面是创世区块的数据类型，该类型都有JSON格式与之对应，所有当加载JSON文件时候，可以直接映射到该类型中。
type Genesis struct {
    Config     *params.ChainConfig `json:"config"`
    Nonce      uint64              `json:"nonce"`
    Timestamp  uint64              `json:"timestamp"`
    ExtraData  []byte              `json:"extraData"`
    GasLimit   uint64              `json:"gasLimit"   gencodec:"required"`
    Difficulty *big.Int            `json:"difficulty" gencodec:"required"`
    Mixhash    common.Hash         `json:"mixHash"`
    Coinbase   common.Address      `json:"coinbase"`
    Alloc      GenesisAlloc        `json:"alloc"      gencodec:"required"`

    // These fields are used for consensus tests. Please don't use them
    // in actual genesis blocks.
    Number     uint64      `json:"number"`
    GasUsed    uint64      `json:"gasUsed"`
    ParentHash common.Hash `json:"parentHash"`
}
```

SetupGenesisBlock 加载创世区块

```
// SetupGenesisBlock writes or updates the genesis block in db.
//
// The block that will be used is:
//
```

```
//                          genesis == nil       genesis != nil
//                       +-----------------------------------------
//     db has no genesis |  main-net default  |  genesis
//     db has genesis    |  from DB           |  genesis (if compatible)
//
// The stored chain configuration will be updated if it is compatible (i.e. does no
t
// specify a fork block below the local head block). In case of a conflict, the
// error is a *params.ConfigCompatError and the new, unwritten config is returned.
// 如果存储的区块链配置不兼容那么会被更新(). 为了避免发生冲突,会返回一个错误,并且新的配置和
原来的配置会返回.
// The returned chain configuration is never nil.

// 在这里genesis 如果是 testnet dev 或者是 rinkeby 模式， 那么需要将我们自己创建的JSON文
件传入。如果是mainnet或者是私有链接。那么为空即可
func SetupGenesisBlock(db ethdb.Database, genesis *Genesis) (*params.ChainConfig, c
ommon.Hash, error) {
    if genesis != nil && genesis.Config == nil {
        return params.AllProtocolChanges, common.Hash{}, errGenesisNoConfig
    }

    // Just commit the new block if there is no stored genesis block.
    stored := GetCanonicalHash(db, 0) //从数据库中，获取genesis对应的区块对象
    if (stored == common.Hash{}) { //如果没有区块 最开始启动geth会进入这里。
        if genesis == nil {
            //如果genesis是nil 而且stored也是nil 那么使用主网络
            // 如果是test  dev  rinkeby 那么genesis不为空 会设置为各自的genesis
            log.Info("Writing default main-net genesis block")
            genesis = DefaultGenesisBlock()
        } else { // 否则使用配置的区块
            log.Info("Writing custom genesis block")
        }
        // 将创世区块写入数据库，这个commit方法，接下来我们会详细分析
        block, err := genesis.Commit(db)
        return genesis.Config, block.Hash(), err
    }

    //检查创世区块是否已经被写入到数据库
    if genesis != nil { //如果genesis存在而且区块也存在 则对比这两个区块是否相同
        block, _ := genesis.ToBlock()
        hash := block.Hash()
        if hash != stored {
            return genesis.Config, block.Hash(), &GenesisMismatchError{stored, hash
}
        }
    }

    // Get the existing chain configuration.
```

```
    // 获取当前存在的区块链的genesis配置
    newcfg := genesis.configOrDefault(stored)
    // 获取当前的区块链的配置
    storedcfg, err := GetChainConfig(db, stored)
    if err != nil {
        if err == ErrChainConfigNotFound {
            // This case happens if a genesis write was interrupted.
            log.Warn("Found genesis block without chain config")
            err = WriteChainConfig(db, stored, newcfg)
        }
        return newcfg, stored, err
    }
    // Special case: don't change the existing config of a non-mainnet chain if no
new
    // config is supplied. These chains would get AllProtocolChanges (and a compat
error)
    // if we just continued here.
    // 特殊情况：如果没有提供新的配置，请不要更改非主网链的现有配置。
    // 如果我们继续这里，这些链会得到AllProtocolChanges（和compat错误）。
    if genesis == nil && stored != params.MainnetGenesisHash {
        return storedcfg, stored, nil   // 如果是私有链接会从这里退出。
    }

    // Check config compatibility and write the config. Compatibility errors
    // are returned to the caller unless we're already at block zero.
    // 检查配置的兼容性,除非我们在区块0,否则返回兼容性错误.
    height := GetBlockNumber(db, GetHeadHeaderHash(db))
    if height == missingNumber {
        return newcfg, stored, fmt.Errorf("missing block number for head header has
h")
    }
    compatErr := storedcfg.CheckCompatible(newcfg, height)
    // 如果区块已经写入数据了,那么就不能更改genesis配置了
    if compatErr != nil && height != 0 && compatErr.RewindTo != 0 {
        return newcfg, stored, compatErr
    }
    // 如果是主网络会从这里退出。
    return newcfg, stored, WriteChainConfig(db, stored, newcfg)
}
```

ToBlock

这个方法使用genesis的数据，加载到内存中，ethdb.NewMemDatabase，是一个内存数据库，通过statedb.IntermediateRoot计算，然后创建了一个block并返回。

```
// ToBlock creates the block and state of a genesis specification.
```

```
func (g *Genesis) ToBlock() (*types.Block, *state.StateDB) {
    db, _ := ethdb.NewMemDatabase()
    statedb, _ := state.New(common.Hash{}, state.NewDatabase(db))
    for addr, account := range g.Alloc {
        statedb.AddBalance(addr, account.Balance)
        statedb.SetCode(addr, account.Code)
        statedb.SetNonce(addr, account.Nonce)
        for key, value := range account.Storage {
            statedb.SetState(addr, key, value)
        }
    }
    root := statedb.IntermediateRoot(false)
    head := &types.Header{
        Number:     new(big.Int).SetUint64(g.Number),
        Nonce:      types.EncodeNonce(g.Nonce),
        Time:       new(big.Int).SetUint64(g.Timestamp),
        ParentHash: g.ParentHash,
        Extra:      g.ExtraData,
        GasLimit:   new(big.Int).SetUint64(g.GasLimit),
        GasUsed:    new(big.Int).SetUint64(g.GasUsed),
        Difficulty: g.Difficulty,
        MixDigest:  g.Mixhash,
        Coinbase:   g.Coinbase,
        Root:       root,
    }
    if g.GasLimit == 0 {
        head.GasLimit = params.GenesisGasLimit
    }
    if g.Difficulty == nil {
        head.Difficulty = params.GenesisDifficulty
    }
    return types.NewBlock(head, nil, nil, nil), statedb
}
```

Commit

利用Commit方法方法利用rawdb.WriteChainConfig(db, block.Hash(), config)函数把给定的genesis的block和state写入数据库， 这个block被认为是规范的区块链头。

```
// Commit writes the block and state of a genesis specification to the database.
// The block is committed as the canonical head block.
func (g *Genesis) Commit(db ethdb.Database) (*types.Block, error) {
    block := g.ToBlock(db)
    if block.Number().Sign() != 0 {
        return nil, fmt.Errorf("can't commit genesis block with number > 0")
    }
```

```
    rawdb.WriteTd(db, block.Hash(), block.NumberU64(), g.Difficulty)
    rawdb.WriteBlock(db, block)
    rawdb.WriteReceipts(db, block.Hash(), block.NumberU64(), nil)
    rawdb.WriteCanonicalHash(db, block.Hash(), block.NumberU64())
    rawdb.WriteHeadBlockHash(db, block.Hash())
    rawdb.WriteHeadHeaderHash(db, block.Hash())

    config := g.Config
    if config == nil {
        config = params.AllEthashProtocolChanges
    }
    rawdb.WriteChainConfig(db, block.Hash(), config)
    return block, nil
}
```