

🔗 c9d42d667b ▾

...

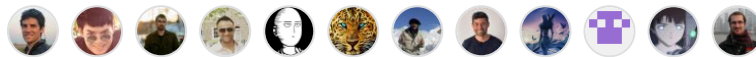
[ethermint](#) / [x](#) / [evm](#) / [keeper](#) / [statedb.go](#) / <> Jump to ▾



fedekunze all: rename go module to evmos/ethermint (#1137) ✖

🕒 History

👤 12 contributors



🛡️ 221 lines (187 sloc) | 6.15 KB ...

```

1 package keeper
2
3 import (
4     "bytes"
5     "fmt"
6     "math/big"
7
8     "github.com/cosmos/cosmos-sdk/store/prefix"
9     sdk "github.com/cosmos/cosmos-sdk/types"
10    sdkerrors "github.com/cosmos/cosmos-sdk/types/errors"
11    "github.com/ethereum/go-ethereum/common"
12    ethermint "github.com/evmos/ethermint/types"
13    "github.com/evmos/ethermint/x/evm/statedb"
14    "github.com/evmos/ethermint/x/evm/types"
15 )
16
17 var _ statedb.Keeper = &Keeper{}
18
19 // -----
20 // StateDB Keeper implementation
21 // -----
22
23 // GetAccount returns nil if account is not exist, returns error if it's not `EthAccountI`
24 func (k *Keeper) GetAccount(ctx sdk.Context, addr common.Address) *statedb.Account {
25     acct := k.GetAccountWithoutBalance(ctx, addr)
26     if acct == nil {
27         return nil
28     }

```

```

29
30     acct.Balance = k.GetBalance(ctx, addr)
31     return acct
32 }
33
34 // GetState loads contract state from database, implements `statedb.Keeper` interface.
35 func (k *Keeper) GetState(ctx sdk.Context, addr common.Address, key common.Hash) common.Hash {
36     store := prefix.NewStore(ctx.KVStore(k.storeKey), types.AddressStoragePrefix(addr))
37
38     value := store.Get(key.Bytes())
39     if len(value) == 0 {
40         return common.Hash{}
41     }
42
43     return common.BytesToHash(value)
44 }
45
46 // GetCode loads contract code from database, implements `statedb.Keeper` interface.
47 func (k *Keeper) GetCode(ctx sdk.Context, codeHash common.Hash) []byte {
48     store := prefix.NewStore(ctx.KVStore(k.storeKey), types.KeyPrefixCode)
49     return store.Get(codeHash.Bytes())
50 }
51
52 // ForEachStorage iterate contract storage, callback return false to break early
53 func (k *Keeper) ForEachStorage(ctx sdk.Context, addr common.Address, cb func(key, value common.Ha
54     store := ctx.KVStore(k.storeKey)
55     prefix := types.AddressStoragePrefix(addr)
56
57     iterator := sdk.KVStorePrefixIterator(store, prefix)
58     defer iterator.Close()
59
60     for ; iterator.Valid(); iterator.Next() {
61         key := common.BytesToHash(iterator.Key())
62         value := common.BytesToHash(iterator.Value())
63
64         // check if iteration stops
65         if !cb(key, value) {
66             return
67         }
68     }
69 }
70
71 // SetBalance update account's balance, compare with current balance first, then decide to mint or
72 func (k *Keeper) SetBalance(ctx sdk.Context, addr common.Address, amount *big.Int) error {
73     cosmosAddr := sdk.AccAddress(addr.Bytes())
74
75     params := k.GetParams(ctx)
76     coin := k.bankKeeper.GetBalance(ctx, cosmosAddr, params.EvmDenom)
77     balance := coin.Amount.BigInt()

```

```

78     delta := new(big.Int).Sub(amount, balance)
79     switch delta.Sign() {
80     case 1:
81         // mint
82         coins := sdk.NewCoins(sdk.NewCoin(params.EvmDenom, sdk.NewIntFromBigInt(delta)))
83         if err := k.bankKeeper.MintCoins(ctx, types.ModuleName, coins); err != nil {
84             return err
85         }
86         if err := k.bankKeeper.SendCoinsFromModuleToAccount(ctx, types.ModuleName, cosmosA
87             return err
88         }
89     case -1:
90         // burn
91         coins := sdk.NewCoins(sdk.NewCoin(params.EvmDenom, sdk.NewIntFromBigInt(new(big.In
92         if err := k.bankKeeper.SendCoinsFromAccountToModule(ctx, cosmosAddr, types.ModuleN
93             return err
94         }
95         if err := k.bankKeeper.BurnCoins(ctx, types.ModuleName, coins); err != nil {
96             return err
97         }
98     default:
99         // not changed
100    }
101    return nil
102 }
103
104 // SetAccount updates nonce/balance/codeHash together.
105 func (k *Keeper) SetAccount(ctx sdk.Context, addr common.Address, account statedb.Account) error {
106     // update account
107     cosmosAddr := sdk.AccAddress(addr.Bytes())
108     acct := k.accountKeeper.GetAccount(ctx, cosmosAddr)
109     if acct == nil {
110         acct = k.accountKeeper.NewAccountWithAddress(ctx, cosmosAddr)
111     }
112
113     if err := acct.SetSequence(account.Nonce); err != nil {
114         return err
115     }
116
117     codeHash := common.BytesToHash(account.CodeHash)
118
119     if ethAcct, ok := acct.(ethermint.EthAccountI); ok {
120         if err := ethAcct.SetCodeHash(codeHash); err != nil {
121             return err
122         }
123     }
124
125     k.accountKeeper.SetAccount(ctx, acct)
126

```

```

127         if err := k.SetBalance(ctx, addr, account.Balance); err != nil {
128             return err
129         }
130
131         k.Logger(ctx).Debug(
132             "account updated",
133             "ethereum-address", addr.Hex(),
134             "nonce", account.Nonce,
135             "codeHash", codeHash.Hex(),
136             "balance", account.Balance,
137         )
138         return nil
139     }
140
141     // SetState update contract storage, delete if value is empty.
142     func (k *Keeper) SetState(ctx sdk.Context, addr common.Address, key common.Hash, value []byte) {
143         store := prefix.NewStore(ctx.KVStore(k.storeKey), types.AddressStoragePrefix(addr))
144         action := "updated"
145         if len(value) == 0 {
146             store.Delete(key.Bytes())
147             action = "deleted"
148         } else {
149             store.Set(key.Bytes(), value)
150         }
151         k.Logger(ctx).Debug(
152             fmt.Sprintf("state %s", action),
153             "ethereum-address", addr.Hex(),
154             "key", key.Hex(),
155         )
156     }
157
158     // SetCode set contract code, delete if code is empty.
159     func (k *Keeper) SetCode(ctx sdk.Context, codeHash, code []byte) {
160         store := prefix.NewStore(ctx.KVStore(k.storeKey), types.KeyPrefixCode)
161
162         // store or delete code
163         action := "updated"
164         if len(code) == 0 {
165             store.Delete(codeHash)
166             action = "deleted"
167         } else {
168             store.Set(codeHash, code)
169         }
170         k.Logger(ctx).Debug(
171             fmt.Sprintf("code %s", action),
172             "code-hash", common.BytesToHash(codeHash).Hex(),
173         )
174     }
175

```

```

176 // DeleteAccount handles contract's suicide call:
177 // - clear balance
178 // - remove code
179 // - remove states
180 // - remove auth account
181 func (k *Keeper) DeleteAccount(ctx sdk.Context, addr common.Address) error {
182     cosmosAddr := sdk.AccAddress(addr.Bytes())
183     acct := k.accountKeeper.GetAccount(ctx, cosmosAddr)
184     if acct == nil {
185         return nil
186     }
187
188     // NOTE: only Ethereum accounts (contracts) can be selfdestructed
189     ethAcct, ok := acct.(ethermint.EthAccountI)
190     if !ok {
191         return sdkerrors.Wrapf(types.ErrInvalidAccount, "type %T, address %s", acct, addr)
192     }
193
194     // clear balance
195     if err := k.SetBalance(ctx, addr, new(big.Int)); err != nil {
196         return err
197     }
198
199     // remove code
200     codeHashBz := ethAcct.GetCodeHash().Bytes()
201     if !bytes.Equal(codeHashBz, types.EmptyCodeHash) {
202         k.SetCode(ctx, codeHashBz, nil)
203     }
204
205     // clear storage
206     k.ForEachStorage(ctx, addr, func(key, _ common.Hash) bool {
207         k.SetState(ctx, addr, key, nil)
208         return true
209     })
210
211     // remove auth account
212     k.accountKeeper.RemoveAccount(ctx, acct)
213
214     k.Logger(ctx).Debug(
215         "account suicided",
216         "ethereum-address", addr.Hex(),
217         "cosmos-address", cosmosAddr.String(),
218     )
219
220     return nil
221 }

```