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

12 contributors

221 lines (187 sloc) | 6.15 KB ...

```go
package keeper

import (
	"bytes"
	"fmt"
	"math/big"

	"github.com/cosmos/cosmos-sdk/store/prefix"
	sdk "github.com/cosmos/cosmos-sdk/types"
	sdkerrors "github.com/cosmos/cosmos-sdk/types/errors"
	"github.com/ethereum/go-ethereum/common"
	ethermint "github.com/evmos/ethermint/types"
	"github.com/evmos/ethermint/x/evm/statedb"
	"github.com/evmos/ethermint/x/evm/types"
)

var _ statedb.Keeper = &Keeper{}

// -----------------------------------------------------------------------------
// StateDB Keeper implementation
// -----------------------------------------------------------------------------

// GetAccount returns nil if account is not exist, returns error if it's not `EthAccountI`
func (k *Keeper) GetAccount(ctx sdk.Context, addr common.Address) *statedb.Account {
	acct := k.GetAccountWithoutBalance(ctx, addr)
	if acct == nil {

		return nil
	}
```

```
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
```

```
	if err := k.SetBalance(ctx, addr, account.Balance); err != nil {
		return err
	}

	k.Logger(ctx).Debug(
		"account updated",
		"ethereum-address", addr.Hex(),
		"nonce", account.Nonce,
		"codeHash", codeHash.Hex(),
		"balance", account.Balance,
	)
	return nil
}

// SetState update contract storage, delete if value is empty.
func (k *Keeper) SetState(ctx sdk.Context, addr common.Address, key common.Hash, value []byte) {
	store := prefix.NewStore(ctx.KVStore(k.storeKey), types.AddressStoragePrefix(addr))
	action := "updated"
	if len(value) == 0 {
		store.Delete(key.Bytes())
		action = "deleted"
	} else {
		store.Set(key.Bytes(), value)
	}
	k.Logger(ctx).Debug(
		fmt.Sprintf("state %s", action),
		"ethereum-address", addr.Hex(),
		"key", key.Hex(),
	)
}

// SetCode set contract code, delete if code is empty.
func (k *Keeper) SetCode(ctx sdk.Context, codeHash, code []byte) {
	store := prefix.NewStore(ctx.KVStore(k.storeKey), types.KeyPrefixCode)

	// store or delete code
	action := "updated"
	if len(code) == 0 {
		store.Delete(codeHash)
		action = "deleted"
	} else {
		store.Set(codeHash, code)
	}
	k.Logger(ctx).Debug(
		fmt.Sprintf("code %s", action),
		"code-hash", common.BytesToHash(codeHash).Hex(),
	)
}

```

```go
// DeleteAccount handles contract's suicide call:
// - clear balance
// - remove code
// - remove states
// - remove auth account
func (k *Keeper) DeleteAccount(ctx sdk.Context, addr common.Address) error {
	cosmosAddr := sdk.AccAddress(addr.Bytes())
	acct := k.accountKeeper.GetAccount(ctx, cosmosAddr)
	if acct == nil {
		return nil
	}

	// NOTE: only Ethereum accounts (contracts) can be selfdestructed
	ethAcct, ok := acct.(ethermint.EthAccountI)
	if !ok {
		return sdkerrors.Wrapf(types.ErrInvalidAccount, "type %T, address %s", acct, addr)
	}

	// clear balance
	if err := k.SetBalance(ctx, addr, new(big.Int)); err != nil {
		return err
	}

	// remove code
	codeHashBz := ethAcct.GetCodeHash().Bytes()
	if !bytes.Equal(codeHashBz, types.EmptyCodeHash) {
		k.SetCode(ctx, codeHashBz, nil)
	}

	// clear storage
	k.ForEachStorage(ctx, addr, func(key, _ common.Hash) bool {
		k.SetState(ctx, addr, key, nil)
		return true
	})

	// remove auth account
	k.accountKeeper.RemoveAccount(ctx, acct)

	k.Logger(ctx).Debug(
		"account suicided",
		"ethereum-address", addr.Hex(),
		"cosmos-address", cosmosAddr.String(),
	)

	return nil
}
```