智能合约开发SDK接口说明

V2.0.5

智能合约开发SDK接口说明

1 文档概述

BCBChain SDK是专门为程序员们开发BCBChain上运行的智能合约设计的编程接口。本文档详细介绍BCBChain SDK提供的各接口及类型说明，SDK包路径为`blockchain/smcsdk/sdk`。

2 sdk

代码包`blockchain/smcsdk/sdk`封装了一些开发智能合约需要的辅助函数，以及智能合约上下文环境所定义的各类辅助接口。

2.1 functions

2.1 func Require()

函数原型：

```
func Require(expr bool, errCode uint32, errInfo string)
```

断言表达式`expr`为`true`，如果`expr`为`false`，将触发智能合约`panic`一个类型为`types.Error`的对象，其中错误码为指定的`errCode`，错误信息为指定的`errInfo`。如果`expr`为`true`，智能合约将被允许继续往下执行。

2.2 func RequireNotError()

函数原型：

```
func RequireNotError(err error, errCode uint32)
```

断言`err`不是错误，要求`err`对象必须为空，如果`err`对象不为空，将触发智能合约`panic`一个类型为`types.Error`的对象，其中错误码为指定的`errCode`，错误信息为`err`对象中的错误描述信息。如果`err`为空，智能合约将被允许继续往下执行。

2.3 func RequireOwner()

函数原型：

```
func RequireOwner()
```

断言本次智能合约调用的调用者必须是合约的拥有者。如果不满足要求，将触发智能合约`panic`一个类型为`types.Error`的对象，其中错误码为`types.ErrNoAuthorization`，错误信息为具体错误原因。如果满足要求，智能合约将被允许继续往下执行。

2.4 func RequireAddress()

函数原型：

```
func RequireAddress(addr types.Address)
```

断言 `addr` 是一个格式正确的地址，如果地址格式不正确，将触发智能合约 `panic` 一个类型为 `types.Error` 的对象，其中错误码为 `types.ErrInvalidAddress`，错误信息为具体错误原因。如果地址格式正确，智能合约将被允许继续往下执行。

2.5 func Array()

函数原型：

```
func Array(items ...interface{}) []interface{}
```

将多个对象转换成一个切片进行表示。

2.2 interfaces

2.2.1 interface ISmartContract

参见本文档章节`sdk/ISmartContract`。

2.2.1 interface IBlock

参见本文档章节`sdk/IBlock`。

2.2.3 interface ITx

参见本文档章节`sdk/ITx`。

2.2.4 interface IMessage

参见本文档章节`sdk/IMessage`。

2.2.5 interface IAccount

参见本文档章节`sdk/IAccount`。

2.2.2 interface IContract

参见本文档章节`sdk/IContract`。

2.2.6 interface IToken

参见本文档章节`sdk/IToken`。

2.2.7 interface IHelper

参见本文档章节`sdk/IHelper`。

2.2.8 interface IAccountHelper

参见本文档章节`sdk/IAccountHelper`。

2.2.9 interface IBlockChainHelper

参见本文档章节`sdk/IBlockChainHelper`。

2.2.10 interface IBuildHelper

参见本文档章节`sdk/IBuildHelper`。

2.2.11 interface IContractHelper

参见本文档章节`sdk/IContractHelper`。

2.2.12 interface IReceiptHelper

参见本文档章节`sdk/IReceiptHelper`。

2.2.13 interface IGenesisHelper

参见本文档章节`sdk/IGenesisHelper`。

2.2.14 interface IStateHelper

参见本文档章节`sdk/IStateHelper`。

2.2.15 interface ITokenHelper

参见本文档章节`sdk/ITokenHelper`。

3 sdk/types

代码包`blockchain/smcsdk/sdk/types`封装了一些开发智能合约所需要的基础数据类型。

3.1 struct Error

SDK提供的描述错误的标准类型。

3.1.1 definition

```
//Error define type for error of sdk
type Error struct {
    ErrorCode uint32 // Error code
    ErrorDesc string // Error description
}
```

3.1.2 func Error()

函数原型：

```
func (err *Error) Error() string
```

返回错误信息文本。

3.1.3 code

```
CodeOK = 200                      ErrorDesc = ""

//for stub
ErrStubDefined = 51000            ErrorDesc = "Error stub defined"
ErrGasNotEnough = 51001           ErrorDesc = "Gas limit is not enough"
ErrFeeNotEnough = 51002           ErrorDesc = "Insufficient balance to pay fee"

//for sdk
ErrAddSupplyNotEnabled = 52000    ErrorDesc = "Add supply is not enabled"
ErrBurnNotEnabled = 52001         ErrorDesc = "Burn supply is not enabled"
ErrInvalidAddress = 52002         ErrorDesc = "Invalid address"

//for contract
ErrNoAuthorization = 53000        ErrorDesc = "No authorization to execute contract"
ErrInvalidParameter = 53001       ErrorDesc = "Invalid parameter"
ErrInsufficientBalance = 53002    ErrorDesc = "Insufficient balance"
```

```
//for user defined
ErrUserDefined = 55000            ErrorDesc = "Error user defined"
```

3.2 type HexBytes

SDK提供的字节数组/切片类型，封装了一些常用的序列化接口。

3.2.1 definition

```
type HexBytes []byte
```

3.2.2 func Marshal()

函数原型：

```
func (bz HexBytes) Marshal() ([]byte, error)
```

实现标准的二进制序列化接口。

3.2.3 func Unmarshal()

函数原型：

```
func (bz *HexBytes) Unmarshal(data []byte) error
```

实现标准的二进制反序列化接口，将 `bz` 的值设置为 `data` 的值。

3.2.4 func MarshalJSON()

函数原型：

```
func (bz HexBytes) MarshalJSON() ([]byte, error)
```

实现标准的 JSON 序列化接口。

3.2.5 func UnmarshalJSON()

函数原型：

```
func (bz *HexBytes) UnmarshalJSON(data []byte) error
```

实现标准的 JSON 反序列化接口，将 `bz` 的值设置为 JSON字符串 `data` 的值。

3.2.6 func Bytes()

函数原型：

```
func (bz HexBytes) Bytes() []byte
```

实现标准的获取字节切片接口。

3.2.7 func String()

函数原型：

```
func (bz HexBytes) String() string
```

实现标准的获取字符串表达接口，将 `bz` 转换成全部大写的十六进制字符串。

3.3 type Address

SDK提供的描述账户地址的标准类型。

3.3.1 definition

```
type Address = string
```

3.4 type Hash

SDK提供的描述哈希数据的标准类型。

3.4.1 definition

```
type Hash = HexBytes
```

3.5 type PubKey

SDK提供的描述公钥数据的标准类型。

3.5.1 definition

```
type PubKey = HexBytes
```

3.6 type KVPair

SDK提供的描述键/值对的标准类型。

3.6.1 definition

```
type KVPair struct {
    Key   []byte `protobuf:"bytes,1,opt,name=key,proto3" json:"key,omitempty"`
    Value []byte `protobuf:"bytes,2,opt,name=value,proto3" json:"value,omitempty"`
}
```

4 sdk/bn

代码包 `blockchain/smcsdk/sdk/bn` 封装了一个处理大数的类 `Number`，进行加减乘除操作时不必考虑溢出的问题。

4.1 functions

本章描述程序包 `sdk/bn` 提供的关于类 `Number` 简便构造函数。

4.1.1 func N()

函数原型：

```
func N(x int64) Number
```

将 `int64` 类型的 `x` 转换成 `Number` 类型对象并返回。

4.1.2 func N1()

函数原型：

```
func N1(b int64, d int64) Number
```

根据传入的 `b` 和 `d`，计算 `b * d` 的结果并返回 `Number` 类型对象。

4.1.3 func N2()

函数原型：

```
func N2(b int64, d1, d2 int64) Number
```

根据传入的 `b`，`d1`，`d2`，计算 `b * d1 * d2` 的结果并返回 `Number` 类型对象。

4.1.4 func NB()

函数原型：

```
func NB(x *big.Int) Number
```

将 `big.Int` 类型的大整数 `x` 转换成 `Number` 类型对象并返回。

4.1.5 func NBS()

函数原型：

```
func NBS(x []byte) Number
```

将按大端表示无符号大整数的字节切片 `x` 转换成 `Number` 类型对象并返回。

4.1.6 func NSBS()

函数原型：

```
func NSBS(x []byte) Number
```

将按大端表示带符号大整数的字节切片 `x` 转换成 `Number` 类型对象并返回。

4.1.7 func NBytes()

函数原型：

```
func NBytes(x []byte) Number
```

将按大端表示无符号大整数的字节切片 `x` 转换成 `Number` 类型对象并返回。

4.1.8 func NSBytes()

函数原型：

```
func NSBytes(x []byte) Number
```

将按大端表示带符号大整数的字节切片 `x` 转换成 `Number` 类型对象并返回。

4.1.9 func NString()

函数原型：

```
func NS(s string) Number
```

将按十进制字符串表示的大整数 `s` 转换成 `Number` 类型对象并返回，如果解析失败将返回 `0`。

4.1.10 func NStringHex()

函数原型：

```
func NS(s string) Number
```

将以 `0x` 或 `0X` 开头的十六进制字符串表示的无符号大整数 `s` 转换成 `Number` 类型对象并返回，如果解析失败将返回 `0`。

4.1.11 func NewNumber()

函数原型：

```
func NewNumber(x int64) Number
```

将 `int64` 类型的大整数 `x` 转换成 `Number` 类型对象并返回。

4.1.12 func NewNumberStringBase()

函数原型：

```
func NewNumberStringBase(s string, base int) Number
```

将字符串表示的大整数 `s` 转换成 `Number` 类型对象并返回，字符串按给定的基数 `base` 进行解析，如果解析失败将返回 `0`。

基数 `base` 必须是 `0` 或者 `2` 到 `MaxBase` 之间的整数。如果 `base` 为 `0`，字符串 `s` 的前缀决定实际的转换基数：前缀为 `"0x"`、`"0X"` 表示十六进制；前缀 `"0b"`、`"0B"` 表示二进制；前缀 `"0"` 表示八进制；其它都自动采用十进制作为基数。

针对 `<= 36` 的基数，大写和小写字母表达相同的数，字母 `'a'` 到 `'z'` 和 `'A'` 到 `'Z'` 都表达数值 `10` 到 `35`。

针对 `> 36` 的基数，大写字母 `'A'` 到 `'Z'` 表达数值 `36` 到 `61`。

4.1.13 func NewNumberBigInt()

函数原型：

```
func NewNumberBigInt(x *big.Int) Number
```

将 `big.Int` 类型的大整数 `x` 转换成 `Number` 类型对象并返回。

4.1.14 func NewNumberLong()

函数原型：

```
func NewNumberLong(b int64, d int64) Number
```

根据传入的 `b` 和 `d`，计算 `b * d` 的结果并返回 `Number` 类型对象。

4.1.15 func NewNumberLongLong()

函数原型：

```
func NewNumberLongLong(b int64, d1, d2 int64) Number
```

根据传入的 `b`，`d1`，`d2`，计算 `b * d1 * d2` 的结果并返回 `Number` 类型对象。

4.2 class Number

本章描述类 `Number` 的成员函数。

4.2.1 func String()

函数原型：

```
func (x Number) String() string
```

实现标准的获取字符串表达接口，将 `x` 转换为十进制字符串。`x` 未设定初始大数值，返回 `nil`。

4.2.2 func Value()

函数原型：

```
func (x Number) Value() *big.Int
```

获取 `x` 的 `big.Int` 的值。`x` 未设定初始大数值，返回 `nil`。

4.2.3 func CmpI()

函数原型：

```
func (x Number) CmpI(y int64) int
```

将 `x` 与 `y` 进行比较，返回 `-1` 代表 `x < y`，`0` 代表 `x == y`，`+1` 代表 `x > y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.4 func Cmp()

函数原型：

```
func (x Number) Cmp(y Number) int
```

将 `x` 与 `y` 进行比较，返回 `-1` 代表 `x < y`，`0` 代表 `x == y`，`+1` 代表 `x > y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.5 func IsZero()

函数原型：

```
func (x Number) IsZero() bool
```

判断 `x` 是否为 `0`。`x` 等于 `0` 返回 `true`。`x` 未设定初始大数值将触发 `panic`。

4.2.6 func IsPositive()

函数原型：

```
func (x Number) IsPositive() bool
```

判断 `x` 是否为正整数(不包含 `0`)。`x` 为正数返回 `true`。`x` 未设定初始大数值将触发 `panic`。

4.2.7 func IsNegative()

函数原型：

```
func (x Number) IsNegative() bool
```

判断 `x` 是否为负整数。`x` 为负数返回 `true`。`x` 未设定初始大数值将触发 `panic`。

4.2.8 func IsEquall()

函数原型：

```
func (x Number) IsEqualI(y int64) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x == y`，`false` 代表 `x != y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.9 func IsEqual()

函数原型：

```
func (x Number) IsEqual(y Number) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x == y`，`false` 代表 `x != y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.10 func IsGreaterThanI()

函数原型：

```
func (x Number) IsGreaterThanI(y int64) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x > y`，`false` 代表 `x <= y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.11 func IsGreaterThan()

函数原型：

```
func (x Number) IsGreaterThan(y Number) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x > y`，`false` 代表 `x <= y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.12 func IsLessThanI()

函数原型：

```
func (x Number) IsLessThanI(y int64) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x < y`，`false` 代表 `x >= y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.13 func IsLessThan()

函数原型：

```
func (x Number) IsLessThan(y Number) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x < y`，`false` 代表 `x >= y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.14 func IsGEI()

函数原型：

```
func (x Number) IsGEI()(y int64) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x >= y`，`false` 代表 `x < y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.15 func IsGE()

函数原型：

```
func (x Number) IsGE(y Number) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x >= y`，`false` 代表 `x < y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.16 func IsLEI()

函数原型：

```
func (x Number) IsLEI(y int64) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x <= y`，`false` 代表 `x > y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.17 func IsLE()

函数原型：

```
func (x Number) IsLE(y Number) bool
```

将 `x` 与 `y` 进行比较，返回 `true` 代表 `x <= y`，`false` 代表 `x > y`。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.18 func AddI()

函数原型：

```
func (x Number) AddI(y int64) Number
```

计算 `x + y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.19 func Add()

函数原型：

```
func (x Number) Add(y Number) Number
```

计算 `x + y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.20 func SubI()

函数原型：

```
func (x Number) SubI(y int64) Number
```

计算 `x - y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.21 func Sub()

函数原型：

```
func (x Number) Sub(y Number) Number
```

计算 `x - y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.22 func MulI()

函数原型：

```
func (x Number) MulI(y int64) Number
```

计算 `x * y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.23 func Mul()

函数原型：

```
func (x Number) Mul(y Number) Number
```

计算 `x * y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.24 func DivI()

函数原型：

```
func (x Number) DivI(y int64) Number
```

计算 `x / y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.25 func Div()

函数原型：

```
func (x Number) Div(y Number) Number
```

计算 `x / y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.26 func ModI()

函数原型：

```
func (x Number) ModI(y int64) Number
```

计算 `x % y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.27 func Mod()

函数原型：

```
func (x Number) Mod(y Number) Number
```

计算 `x % y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.28 func Sq()

函数原型：

```
func (x Number) Sq() Number
```

计算 `x ** 2`，并返回计算结果。`x` 未设定初始大数值将触发 `panic`。

4.2.29 func Sqrt()

函数原型：

```
func (x Number) Sqrt() Number
```

计算 `x` 平方根，并返回计算结果。`x` 未设定初始大数值将触发 `panic`。

4.2.30 func Exp()

函数原型：

```
func (x Number) Exp(y Number) Number
```

计算 `x ** y`，并返回计算结果。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.31 func Lsh()

函数原型：

```
func (x Number) Lsh(n uint) Number
```

将 `x` 向左移 `n` 位，并返回移位结果。`x` 未设定初始大数值将触发 `panic`。

4.2.32 func Rsh()

函数原型：

```
func (x Number) Rsh(n uint) Number
```

将 `x` 向右移 `n` 位，并返回移位结果。`x` 未设定初始大数值将触发 `panic`。

4.2.33 func And()

函数原型：

```
func (x Number) And(y Number) Number
```

按位计算 `x & y`，并返回计算结果（需要注意这里`x`、`y`有可能为负数，采用二进制补码形式进行按位与）。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.34 func Or()

函数原型：

```
func (x Number) Or(y Number) Number
```

按位计算 `x | y`，并返回计算结果（需要注意这里`x`、`y`有可能为负数，采用二进制补码形式进行按位或）。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.35 func Xor()

函数原型：

```
func (x Number) Xor(y Number) Number
```

按位计算 `x ^ y`，并返回计算结果（需要注意这里`x`、`y`有可能为负数，采用二进制补码形式进行按位异或）。`x` 或 `y` 未设定初始大数值将触发 `panic`。

4.2.36 func Not()

函数原型：

```
func (x Number) Not() Number
```

按位计算 `^ x`，并返回计算结果（需要注意这里`x`有可能为负数，采用二进制补码形式进行取反）。`x` 未设定初始大数值将触发 `panic`。

4.2.37 func Bytes()

函数原型：

```
func (x Number) Bytes() []byte
```

实现标准的获取字节切片接口，将 `x` 转换为大端顺序的字节切片（第一字节解为表示符号，负数为`0xFF`，非负数为 `0x00`），并返回转换结果。例如：`380` 将被转换为`0x00017C`；`-380` 将被转换为`0xFF017C`。`x` 未设定初始大数值将触发 `panic`。

4.2.37 func SetBytes()

函数原型：

```
func (x *Number) SetBytes(buf []byte) Number
```

将 `x` 的值设置为一个大端顺序的字节切片（第一字节为 `0xFF` 表示是一个负数，其绝对值从第二字节开始编码，否则整个字节切片表示一个非负整数），并将 `x` 的值返回。例如：`0x00017C` 和 `0x017C` 都将被转换为 `380`；`0xFF017C` 将被转换为 `-380`。

4.2.38 func MarshalJSON()

函数原型：

```
func (x Number) MarshalJSON() (data []byte, err error)
```

实现标准的 JSON 序列化接口。将 `x` 转成简化版的 JSON 字符串，例如字符串 `380`。`x` 未设定初始大数值将触发 `panic`。

4.2.39 func UnmarshalJSON()

函数原型：

```
func (x *Number) UnmarshalJSON(data []byte) error
```

实现标准的 JSON 反序列化接口。将 `x` 的值设为输入的 JSON 字符串对应的大数。支持简化版的 JSON 字符串（例如：字符串 `380`）与结构版的 JSON 字符串（例如：字符串 `{"v":380}`）。

5 sdk/crypto/ed25519

代码包 `blockchain/smcsdk/sdk/crypto/ed25519` 封装了对椭圆曲线 `ed25519` 的简便应用接口。

5.1 functions

5.1.1 func VerifySign()

函数原型：

```
func VerifySign(pubkey, data, sign []byte) bool
```

验证签名，成功返回 `true`。 `pubkey` 为公钥， `data` 为被签名的数据， `sign` 为签名数据。

6 sdk/crypto/sha3

代码包 `blockchain/smcsdk/sdk/crypto/sha3` 封装了对散列算法 `SHA-3` 的简便应用接口。

6.1 functions

6.1.1 func Sum224()

函数原型：

```
func Sum224(datas ...[]byte) []byte
```

使用 `SHA3-224` 算法计算输入的数据表（多项输入参数按输入顺序进行计算）的 224 位散列值并返回计算结果。

6.1.2 func Sum256()

函数原型：

```
func Sum256(datas ...[]byte) []byte
```

使用 `SHA3-256` 算法计算输入的数据表（多项输入参数按输入顺序进行计算）的 256 位散列值并返回计算结果。

6.1.3 func Sum384()

函数原型：

```
func Sum384(datas ...[]byte) []byte
```

使用 `SHA3-384` 算法计算输入的数据表（多项输入参数按输入顺序进行计算）的 384 位散列值并返回计算结果。

6.1.4 func Sum512()

函数原型：

```
func Sum512(datas ...[]byte) []byte
```

使用 `SHA3-512` 算法计算输入的数据表（多项输入参数按输入顺序进行计算）的 512 位散列值并返回计算结果。

7 sdk/jsoniter

代码包 `blockchain/smcsdk/sdk/jsoniter` 封装了对快速处理 JSON 数据的第三方包 `jsoniter` 的简便应用接口。

7.1 functions

7.1.1 func Marshal()

函数原型：

```
func Marshal(v interface{}) ([]byte, error)
```

将输入的对象 `v` 序列化成 JSON 格式的字符串并返回。

7.1.2 func Unmarshal()

函数原型：

```
func Unmarshal(bz []byte, v interface{}) error
```

对输入的 JSON 字符串进行解析，并将解析结果存入 `v` 指向的对象。

8 sdk/forx

代码包 `blockchain/smcsdk/sdk/mapx` 封装了对 `for` 循环进行优化处理的应用接口。

8.1 functions

8.1.1 func Range()

函数原型：

```
func Range(args... interface{})
```

智能合约规范中不允许智能合约代码中使用关键字 `for`。

智能合约中需要循环执行的指令必须采用 `Range()` 函数。

`Range()` 函数存在六个版本的使用模型，示例如下：

> 模型一：
>
> ```
> func Range( m map[keyType]valType, f func(key keyType, val valType) )
> func Range( m map[keyType]valType, f func(key keyType, val valType) bool )
> ```
>
> 对映射表对象 m 进行遍历操作，遍历按照映射表键的顺序执行，操作函数为 f，如果 f 的返回值定义为空，则遍历结束以后才可以结束循环，如果 f 的返回值定义为布尔类型，f 返回 false 表示终止执行遍历操作退出循环，f 返回 true 表示继续执行遍历操作。输入参数要求满足如下条件，如果任意一条不满足则发生异常，当发生异常时，将会自动触发SDK进行 panic：
>
> 1. 参数 m 的类型必须是一个映射表；
> 2. 参数 f 必须是一个函数，函数的返回值必须是在空与布尔类型之间二选一；
> 3. 函数 f 的参数必须是两个；
> 4. 函数 f 的第一个参数类型必须是映射表 m 的键对应的类型；
> 5. 函数 f 的第二个参数类型必须是映射表 m 的值对应的类型。
>
> 示例代码如下：
>
> ```
> m := make(map[int]string)
> m[93] = "23231"
> m[13] = "23423423234324"
> m[54] = "3432432423"
> m[23] = "3434545345345"
>
> forx.Range(m, func(k int, v string) {
>     printf("key=%v value=%v\n", k, v)
> })
> ```

> 模型二：

```
func Range( s []valType, f func(i int, val valType) )
func Range( s []valType, f func(i int, val valType) bool )
```

对切片对象 s 进行遍历操作，遍历按照切片顺序从小到大顺序执行，操作函数为 f，如果 f 的返回值定义为空，则遍历结束以后才可以结束循环，如果 f 的返回值定义为布尔类型，f 返回 false 表示终止执行遍历操作结束循环，f 返回 true 表示继续执行遍历操作。输入参数要求满足如下条件，如果任意一条不满足则发生异常，当发生异常时，将会自动触发SDK进行 panic：

1. 参数 m 的类型必须是一个切片；
2. 参数 f 必须是一个函数，函数的返回值必须是在空与布尔类型之间二选一；
3. 函数 f 的参数必须是两个；
4. 函数 f 的第一个参数类型必须是 int 类型，表示元素的索引号，从 0 开始；
5. 函数 f 的第二个参数类型必须是切片 s 的元素值对应的类型。

示例代码如下：

```
s := make([]string)
s = append(s, "23231")
s = append(s, "423792234")
s = append(s, "23232")
s = append(s, "3243455454")

forx.Range(s, func(i int, v string) {
    printf("i=%v value=%v\n", i, v)
})
```

模型三：

```
func Range( n intType, f func(i intType) )
func Range( n intType, f func(i intType) bool )
```

循环执行指定次数，操作函数为 f，如果 f 的返回值定义为空，则执行完指定次数以后才可以结束循环，f 返回 false 表示终止循环操作，f 返回 true 表示继续循环操作。输入参数要求满足如下条件，如果任意一条不满足则发生异常，当发生异常时，将会自动触发SDK进行 panic：

1. 参数 n 的类型必须为表达整数的类型，例如：int、uint、int32等，值必须大于等于 0 的整数；
2. 参数 f 必须是一个函数，函数的返回值必须是在空与布尔类型之间二选一；
3. 函数 f 的参数必须是一个，类型必须于参数 n 的类型相同，表示执行循环的索引号，从 0 开始。

示例代码如下：

```
forx.Range(10, func(i int) {
    printf("i=%v\n", i)
})
```

模型四：

```
func Range( m,n intType, f func(k intType) )
func Range( m,n intType, f func(k intType) bool )
```

根据输入参数遍历 m 到 n 之间(包含 m 和 n)的所有整数，从 m 开始执行，到 n 结束，操作函数为 f，如果 f 的返回值定义为空，则遍历结束以后才可以结束循环，f 返回 false 表示终止执行遍历操作退出循环，f 返回 true 表示继续执行遍历操作。输入参数要求满足如下条件，如果任意一条不满足则发生异常，当发生异常时，将会自动触发SDK进行 panic:

1. 参数 m 和 n 的类型必须为表达整数的同一类型，例如：int、uint、int32等，如果 m 小于 n，则执行顺序为增序，如果 m 大于 n，则执行顺序为降序；
2. 参数 f 必须是一个函数，函数的返回值必须是在空与布尔类型之间二选一；
3. 函数 f 的参数必须是一个，类型必须与参数 m 和 n 的类型相同，表示遍历的整数值，从 m 开始。

示例代码如下:

```
forx.Range(1,100, func(k int) {
    printf("k=%v\n", k)
})
```

模型五:

```
func Range( c func() bool, f func(i int) )
func Range( c func() bool, f func(i int) bool )
```

循环执行函数 f，当控制函数 c 满足返回值为 true 时立即终止循环操作，它控制循环结束的时间，如果 f 的返回值定义为布尔类型，则除了函数 c 可以控制循环结束以外，f 返回 false 表示终止执行循环操作，f 返回 true 表示继续执行循环操作，输入参数要求满足如下条件，如果任意一条不满足则发生异常，当发生异常时，将会自动触发SDK进行 panic:

1. 参数 c 必须是一个函数，函数的返回值必须是布尔类型；
2. 参数 f 必须是一个函数，函数的返回值必须是在空与布尔类型之间二选一；
3. 函数 f 的参数必须是一个，类型必须是 int，表示执行循环次数的索引，从 0 开始。

示例代码如下:

```
toVoter := ballot._voters(to)
forx.range( func() bool {
           return toVoter.delegate != ""
         },
         func(i int) {
           to = toVoter.delegate
           toVoter = ballot._voters(to)
         })
```

模型六:

```
func Range( f func(i int) bool )
```

> 循环执行函数 f，f 返回 false 表示终止执行循环操作，f 返回 true 表示继续执行循环操作。输入参数要求满足如下条件，如果任意一条不满足则发生异常，当发生异常时，将会自动触发SDK进行 panic：
>
> 1. 参数 f 必须是一个函数，函数的返回值必须是布尔类型；
> 2. 函数 f 的参数必须是一个，类型必须是 int，表示执行循环次数的索引，从 0 开始。

示例代码如下：

```
toVoter := ballot._voters(to)
forx.Range( func(i int) bool {
            to = toVoter.delegate
            toVoter = ballot._voters(to)

            if toVoter.delegate != "" {
              return forx.Break
            } else {
              return forx.Continue
            }
          })
```

9 sdk/rlp

代码包 `blockchain/smcsdk/sdk/rlp` 封装了 BCBChain 对交易进行 RLP 编码的操作。源码是基于 `geth` 的开源版本，根据 BCBChain 的需要，添加了对 `bn.Number` 大数的编解码支持，同时添加了对映射表的支持。

10 sdk/ISmartContract

接口 `sdk/ISmartContract` 封装了智能合约上下文获取的途径。

10.1 func Block()

函数原型：

```
func (ISmartContract) Block() IBlock
```

返回对 BCBChain 区块链进行调用时的区块信息。

10.2 func Tx()

函数原型：

```
func (ISmartContract) Tx() ITx
```

返回对 BCBChain 区块链进行调用的交易信息。

10.3 func Message()

函数原型：

```
func (ISmartContract) Message() IMessage
```

返回针对本智能合约调用的消息信息。

10.4 func Helper()

函数原型：

```
func (ISmartContract) Helper() IHelper
```

返回封装了SDK所有辅助功能的对象。

11 sdk/IBlock

接口 `sdk/IBlock` 封装了智能合约获取到的区块信息。

在智能合约调用时，`IBlock` 接口在两个不同阶段对应的区块信息是有区别的。

> 在checkTx阶段，当前智能合约的调用还没有取得共识，此时调用 ISmartContract::Block() 获得的为区块链上最后一个区块之上（区块高度加1）生成的一个模拟区块的信息（可能是一个空区块，总之这个区块与当前所执行的智能合约调用没有任何关系）。

> 在deliverTx阶段，此次调用智能合约的交易已经在区块链上达成共识，并写入到最新一个区块中，此时调用 ISmartContract::Block() 获得的为区块链上最新一个实时区块的信息（不可能是一个空区块，这个区块中至少包含一笔对本次智能合约调用的交易信息）。

11.1 func ChainID()

函数原型：

```
func (IBlock) ChainID() string
```

返回当前区块链的链标识。

11.2 func BlockHash()

函数原型：

```
func (IBlock) BlockHash() types.Hash
```

返回当前区块的区块哈希。

11.3 func Height()

函数原型：

```
func (IBlock) Height() int64
```

返回当前区块的高度。

11.4 func Time()

函数原型：

```
func (IBlock) Time() int64
```

返回当前区块生成时的时间戳信息，结果为相对于 `1970-1-1 00:00:00` 过去的秒数。

11.5 func Now()

函数原型：

```
func (IBlock) Now() bn.Number
```

对 `Time()` 函数的别称。返回当前区块生成时的时间戳信息，结果为相对于 `1970-1-1 00:00:00` 过去的秒数。

11.6 func NumTxs()

函数原型：

```
func (IBlock) NumTxs() int32
```

返回当前区块收录的交易笔数。

11.7 func DataHash()

函数原型：

```
func (IBlock) DataHash() types.Hash
```

返回当前区块的数据哈希。

11.8 func ProposerAddress()

函数原型：

```
func (IBlock) ProposerAddress() types.Address
```

返回当前区块的提案者地址。

11.9 func RewardAddress()

函数原型：

```
func (IBlock) RewardAddress() types.Address
```

返回当前区块的提案者接收出块奖励的地址。

11.10 func RandomNumber()

函数原型：

```
func (IBlock) RandomNumber() types.HexBytes
```

返回当前区块的区块随机数。

11.11 func Version()

函数原型：

```
func (IBlock) Version() string
```

返回当前区块提案者出块时的软件版本。

11.12 func LastBlockHash()

函数原型：

```
func (IBlock) LastBlockHash() types.Hash
```

返回上一区块的区块哈希。

11.13 func LastCommitHash()

函数原型：

```
func (IBlock) LastCommitHash() types.Hash
```

返回上一区块的确认哈希。

11.14 func LastAppHash()

函数原型：

```
func (IBlock) LastAppHash() types.Hash
```

返回上一区块的应用层哈希。

11.15 func LastFee()

函数原型：

```
func (IBlock) LastFee() int64
```

返回上一区块的手续费（单位：`cong`）。

12 sdk/ITx

接口 `sdk/ITx` 封装了对 BCBChain 区块链进行调用的交易信息。

12.1 func Note()

函数原型：

```
func (ITx) Note() string
```

返回当前交易的备注信息。

12.2 func GasLimit()

函数原型：

```
func (ITx) GasLimit() int64
```

返回当前交易的最大燃料限制数量。

12.3 func GasLeft()

函数原型：

```
func (ITx) GasLeft() int64
```

返回当前交易的剩余燃料数量（已经预扣除对当前智能合约调用需要的燃料数量）。

12.4 func Signer()

函数原型：

```
func (ITx) Signer() IAccount
```

返回当前交易签名人(发起人)的账户对象。

13 sdk/IMessage

接口 `sdk/IMessage` 封装了对某个智能合约方法的一次调用相关的所有信息。在同一笔交易当中可以级联多次对不同智能合约方法进行调用的消息。

13.1 func Contract()

函数原型：

```
func (IMessage) Contract() types.Address
```

返回当前消息调用的智能合约地址。

13.2 func MethodID()

函数原型：

```
func (IMessage) MethodID() string
```

返回当前消息调用的智能合约方法ID（采用十六进制字符串进行表示）。

13.3 func Items()

函数原型：

```
func (IMessage) Items() []types.HexBytes
```

返回当前消息调用的每个参数数据字段的原始信息。当发生跨智能合约调用时，在被调用合约内部，`Items()` 获得的数据为 `nil`。

13.4 func GasPrice()

函数原型：

```
func (IMessage) GasPrice() int64
```

返回当前消息调用的燃料价格（单位：`cong`）。

13.5 func Sender()

函数原型：

```
func (IMessage) Sender() IAccount
```

返回当前消息调用发起人的账户对象。第一层的消息调用，消息发起人就是交易的签名人；当发生跨合约调用时，在被调用合约内部，`Sender()` 返回的是发起合约的账户对象。

13.6 func Payer()

函数原型：

```
func (IMessage) Payer() IAccount
```

返回支付当前消息手续费的账户对象。

##

13.7 func Origins()

函数原型：

```
func (IMessage) Origins() []types.Address
```

返回消息完整的调用链。在不是进行跨合约调用时，`Origins()` 返回包含本次交易发起者的账户地址的切片。当发生跨智能合约调用时，`Origins()` 返回表达调用的合约链，在被调用合约内部，`Origins()` 返回的地址列表中最后一个为本次调用发起合约的合约地址。

13.8 func InputReceipts()

函数原型：

```
func (IMessage) InputReceipts() []types.KVPair
```

在级联和跨合约消息调用时，前一个消息调用输出的收据将作为后一个消息调用的输入，本函数返回输入到本次消息调用的收据列表。

13.9 func GetTransferToMe()

函数原型：

```
func (IMessage) GetTransferToMe() []*std.Transfer
```

在前一个消息调用输出的收据中查询并返回向当前智能合约进行转账的所有收据，如果找不到则返回 `nil`。转账收据定义如下：

```
package std

type Transfer struct {
    Token types.Address `json:"token"` // 代币地址
    From  types.Address `json:"from"`  // 资金转出方账户地址
    To    types.Address `json:"to"`    // 资金接收方账户地址
    Value bn.Number     `json:"value"` // 转账金额（单位：```cong```）
}
```

14 sdk/IAccount

接口 `sdk/IAccount` 封装了对某个账户地址的访问接口。

14.1 func Address()

函数原型：

```
func (IAccount) Address() types.Address
```

返回账户对象的账户地址。

14.2 func PubKey()

函数原型：

```
func (IAccount) PubKey() types.PubKey
```

返回账户对象的公钥数据（可能为空）。

14.3 func Balance()

函数原型：

```
func (IAccount) Balance() bn.Number
```

在发行代币的合约中，返回账户对象在本合约所发行的代币子账户的资金余额（单位：`cong`）。在非代币合约中，返回 `0`。

14.4 func BalanceOfToken()

函数原型：

```
func (IAccount) BalanceOfToken(token types.Address) bn.Number
```

给定代币地址，返回账户对象在指定代币子账户的资金余额（单位：`cong`）。如果指定的代币地址不是一个代币，直接返回余额为 `0`。

14.5 func BalanceOfName()

函数原型：

```
func (IAccount) BalanceOfName(name string) bn.Number
```

给定代币名称，返回账户对象在指定代币子账户的资金余额（单位：`cong`）。如果指定的代币名称不是一个代币，直接返回余额为 `0`。

14.6 func BalanceOfSymbol()

函数原型：

```
func (IAccount) BalanceOfSymbol(symbol string) bn.Number
```

给定代币符号，返回账户对象在指定代币子账户的资金余额（单位：`cong`）。如果指定的代币符号不是一个代币，直接返回余额为 `0`。

14.7 func Transfer()

函数原型：

```
func (IAccount) Transfer(to types.Address, value bn.Number)
```

在代币合约中，将账户对象在本合约所发行的代币子账户资金转给接收地址为 `to` 的账户（资金单位：`cong`），在这里账户对象包括消息调用的发起人或当前合约账户。

满足如下任意一条则发生异常，当发生异常时，将会自动触发SDK进行 `panic`：

```
1. 参数 to 不是一个地址;
2. 参数 to 与账户地址相同;
3. 参数 value 小于等于 0;
4. 账户的余额不够支付 value 的值;
5. 如果当前合约没有发行过代币（因为该函数的参数中没有指定转移哪种代币）;
6. 如果当前合约发行过代币，账户既不是消息调用的发起人也不是当前智能合约的合约账户。
```

14.8 func TransferByToken()

函数原型：

```
func (IAccount) TransferByToken(token types.Address, to types.Address, value bn.Number)
```

如果指定的代币地址为本合约所发行的代币，将账户对象在本合约所发行的代币子账户资金转给接收地址为 `to` 的账户（资金单位：`cong`），在这里账户对象包括消息调用的发起人或当前合约账户。

如果指定的代币地址不是本合约所发行的代币，将账户对象的指定代币子账户资金转给接收地址为 `to` 的账户（资金单位：`cong`），在这里账户对象只能为当前合约账户。

满足如下任意一条则发生异常，当发生异常时，将会自动触发SDK进行 `panic`：

```
1. 参数 token 不是一个地址;
2. 参数 token 不是一个代币;
3. 参数 to 不是一个地址;
4. 参数 to 与账户地址相同;
5. 参数 value 小于等于 0;
6. 账户的余额不够支付 value 的值;
7. 如果当前合约没有发行过代币，账户不是当前智能合约的合约账户;
8. 如果当前合约发行过代币，账户不是消息发送者或当前智能合约的合约账户者。
```

14.9 func TransferByName()

函数原型：

```
func (IAccount) TransferByName(name string, to types.Address, value bn.Number)
```

如果指定的代币名称为本合约所发行的代币，将账户对象在本合约所发行的代币子账户资金转给接收地址为 `to` 的账户（资金单位：`cong`），在这里账户对象包括消息调用的发起人或当前合约账户。

如果指定的代币地址不是本合约所发行的代币，将账户对象的指定代币子账户资金转给接收地址为 `to` 的账户（资金单位：`cong`），在这里账户对象只能为当前合约账户。

满足如下任意一条则发生异常，当发生异常时，将会自动触发SDK进行 `panic`：

```
1. 参数 name 不是一个代币的名称;
2. 参数 to 不是一个地址;
3. 参数 to 与账户地址相同;
4. 参数 value 小于等于 0;
5. 账户的余额不够支付 value 的值;
6. 如果当前合约没有发行过代币，账户不是当前智能合约的合约账户;
7. 如果当前合约发行过代币，账户不是消息发送者或当前智能合约的合约账户者。
```

14.10 func TransferBySymbol()

函数原型：

```
func (IAccount) TransferBySymbol(symbol string, to types.Address, value bn.Number)
```

如果指定的代币符号为本合约所发行的代币，将账户对象在本合约所发行的代币子账户资金转给接收地址为 `to` 的账户（资金单位：`cong`），在这里账户对象包括消息调用的发起人或当前合约账户。

如果指定的代币地址不是本合约所发行的代币，将账户对象的指定代币子账户资金转给接收地址为 `to` 的账户（资金单位：`cong`），在这里账户对象只能为当前合约账户。

满足如下任意一条则发生异常，当发生异常时，将会自动触发SDK进行 `panic`：

```
1. 参数 symbol 不是一个代币的符号;
2. 参数 to 不是一个地址;
3. 参数 to 与账户地址相同;
4. 参数 value 小于等于 0;
5. 账户的余额不够支付 value 的值;
6. 当前合约没有发行过代币，账户不是当前智能合约的合约账户;
7. 如果当前合约发行过代币，账户不是消息发送者或当前智能合约的合约账户者。
```

15 sdk/IContract

接口 `sdk/IContract` 封装了某个智能合约的所有信息以及操作智能合约的接口。

15.1 func Address()

函数原型：

```
func (IContract) Address() types.Address
```

返回智能合约的地址。

15.2 func Account()

函数原型：

```
func (IContract) Account() IAccount
```

返回智能合约的账户对象。

智能合约的账户地址可用于接收转给合约的资金，智能合约账户地址的计算只与合约名称和组织ID相关，合约升级后合约地址发生变化但不会影响合约的账户地址，合约的账户地址没有私钥与之对应，合约账户上的资金只能通过合约进行操纵。

15.3 func Owner()

函数原型：

```
func (IContract) Owner() IAccount
```

返回智能合约的拥有者的外部账户对象（外部账户地址有私钥与之对应）。

15.4 func Name()

函数原型：

```
func (IContract) Name() string
```

返回智能合约名称。

15.5 func Version()

函数原型：

```
func (IContract) Version() string
```

返回智能合约版本号。

15.6 func CodeHash()

函数原型：

```
func (IContract) CodeHash() types.Hash
```

返回智能合约代码所对应的哈希。

15.7 func EffectHeigt()

函数原型：

```
func (IContract) EffectHeight() int64
```

返回智能合约开始生效时的区块高度。

15.8 func LoseHeight()

函数原型：

```
func (IContract) LoseHeight() int64
```

返回智能合约开始失效时的区块高度，0表示没有失效。

15.9 func KeyPrefix()

函数原型：

```
func (IContract) KeyPrefix() string
```

返回智能合约内所能访问的状态数据库KEY值的前缀（格式为`/xxx/yyy`），可用于进行智能合约之间数据的隔离保护。

15.10 func Methods()

函数原型：

```
func (IContract) Methods() []std.Method
```

返回智能合约所有公开方法的详细信息（可被用于级联消息调用合约）。`std.Method`结构定义如下：

```
package std

type Method struct {
    MethodID    string    // 方法ID
    Gas         int64     // 方法在调用时需要消耗的燃料
    ProtoType   string    // 方法原型
}
```

15.11 func Interfaces()

函数原型：

```
func (IContract) Interfaces() []std.Method
```

返回智能合约所有公开接口（可被用于跨合约调用）的详细信息。

15.12 func Mine()

函数原型：

```
func (IContract) Mine() []std.Method
```

返回智能合约公开的挖矿接口的详细信息。

15.13 func Token()

函数原型：

```
func (IContract) Token() types.Address
```

返回智能合约所注册的代币地址，如果合约没有注册过代币，返回空地址。

15.14 func OrgID()

函数原型：

```
func (IContract) OrgID() string
```

返回智能合约所属的组织标识。

15.15 func SetOwner()

函数原型：

```
func (IContract) SetOwner( newOwner types.Address)
```

设置智能合约新的拥有者。`newOwner` 为智能合约新的拥有者的外部账户地址。函数 `SetOwner()` 只能由智能合约的原拥有者调用。如果智能合约发行了代币，该操作同时会将代币的拥有者设置给新的拥有者。

满足如下任意一条则发生异常，当发生异常时，将会自动触发SDK进行 `panic`：

```
1. 消息调用者不是合约的原拥有者;
2. 参数 newOwner 不是一个地址;
3. 参数 newOwner 是合约的原拥有者;
4. 参数 newOwner 是合约的地址;
5. 参数 newOwner 是合约的账户地址。
```

16 sdk/IToken

接口 `sdk/IToken` 封装了某个代币的所有信息以及操作代币的接口。

16.1 func Address()

函数原型：

```
func (IToken) Address() types.Address
```

返回代币的地址。

16.2 func Owner()

函数原型：

```
func (IToken) Owner() IAccount
```

返回代币拥有者的外部账户对象。

16.3 func Name()

函数原型：

```
func (IToken) Name() string
```

返回代币的名称。

16.4 func Symbol()

函数原型：

```
func (IToken) Symbol() string
```

返回代币的符号。

16.5 func TotalSupply()

函数原型：

```
func (IToken) TotalSupply() Number
```

返回代币的总供应量（单位：`cong`）。

16.6 func AddSupplyEnabled()

函数原型：

```
func (IToken) AddSupplyEnabled() bool
```

返回代币是否允许增发。

16.7 func BurnEnabled()

函数原型：

```
func (IToken) BurnEnabled() bool
```

返回代币是否允许燃烧。

16.8 func GasPrice()

函数原型：

```
func (IToken) GasPrice() int64
```

返回代币的燃料价格（单位：`cong`）。

16.9 func SetTotalSupply()

函数原型：

```
func (IToken) SetTotalSupply(newTotalSupply bn.Number)
```

设置代币新的总供应量。`newTotalSupply` 为代币新的总供应量（单位：`cong`），同时更新合约拥有者在该代币子账户上的余额。

满足如下任意一条则发生异常，当发生异常时，将会自动触发SDK进行 `panic`：

```
1. 消息调用者不是合约的拥有者;
2. 参数 newTotalSupply 小于 1000000000(cong);
3. 参数 newTotalSupply 大于原供应量，但代币不允许增发;
4. 参数 newTotalSupply 小于原供应量，但代币不允许燃烧;
5. 合约拥有者在该代币子账户上的余额在更新后小于0。
```

16.10 func SetGasPrice()

函数原型：

```
func (IToken) SetGasPrice(newGasPrice int64)
```

设置代币的燃料价格。`newGasPrice` 为代币新的燃料价格（单位：`cong`）。

满足如下任意一条则发生异常，当发生异常时，将会自动触发SDK进行 `panic`：

```
1. 消息调用者不是合约拥有者;
2. 参数 newGasPrice 大于 1000000000(cong);
3. 参数 newGasPrice 小于基础燃料价格。
```

17 sdk/IHelper

接口 `sdk/IHelper` 封装了SDK所有的帮助对象。

17.1 func AccountHelper()

函数原型：

```
func (IHelper) AccountHelper() IAccountHelper
```

返回账户相关的帮助对象。

17.2 func BlockchainHelper()

函数原型：

```
func (IHelper) BlockChainHelper() IBlockChainHelper
```

返回区块链相关的帮助对象。

17.3 func BuildHelper()

函数原型：

```
func (IHelper) BuildHelper() IBuildHelper
```

返回合约构建相关的帮助对象。

17.4 func ContractHelper()

函数原型：

```
func (IHelper) ContractHelper() IContractHelper
```

返回智能合约相关的帮助对象。

17.5 func GenesisHelper()

函数原型：

```
func (IHelper) GenesisHelper() IGenesisHelper
```

返回创世相关的帮助对象。

17.6 func ReceiptHelper()

函数原型：

```
func (IHelper) ReceiptHelper() IReceiptHelper
```

返回收据相关的帮助对象。

17.7 func TokenHelper()

函数原型：

```
func (IHelper) TokenHelper() ITokenHelper
```

返回代币相关的帮助对象。

17.8 func StateHelper()

函数原型：

```
func (IHelper) StateHelper() IStateHelper
```

返回状态存储相关的帮助对象。

18 sdk/IAccountHelper

接口 `sdk/IAccountHelper` 封装了对区块链账户进行访问的帮助对象。

18.1 func AccountOf()

函数原型：

```
func (IAccountHelper) AccountOf(addr types.Address) IAccount
```

根据账户地址构造一个账户对象，用来对账户进行一些操作。`addr` 为账户地址。返回账户对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 参数 addr 不是地址。
```

18.2 func AccountOfPubKey()

函数原型：

```
func (IAccountHelper) AccountOfPubKey(pubkey types.PubKey) IAccount
```

根据账户公钥构造一个账户对象，用来对账户进行一些操作。pubkey 为账户公钥。返回账户对象。

满足如下任意一条则发生错误，当错误时，返回 nil：

```
1. 参数 pubkey 不是公钥（长度不等于32字节）。
```

19 sdk/IBlockChainHelper

接口 `sdk/IBlockChainHelper` 封装了对区块链进行访问的帮助对象。

19.1 func CalcAccountFromPubkey()

函数原型：

```
func (IBlockchainHelper) CalcAccountFromPubkey(pubkey types.PubKey) types.Address
```

根据公钥计算账户地址。`pubkey` 为公钥。返回计算得出的地址。

满足如下任意一条则发生错误，当错误时，返回空地址：

```
1. 参数 pubkey 不是公钥（长度不等于32字节）。
```

19.2 func CalcAccountFromName()

函数原型：

```
func (IBlockchainHelper) CalcAccountFromName(name string, orgID string) types.Address
```

根据合约名称及其所属组织ID计算出合约的账户地址。`name` 为合约名称，`orgID` 为组织ID。返回计算得出的合约账户地址。

19.3 func CalcContractAddress()

函数原型：

```
func (IBlockchainHelper) CalcContractAddress(
                                name string,
                                version string,
                                orgID string) types.Address
```

根据给定的合约参数计算合约地址。`name` 为合约名称。`version` 为合约版本。`orgID` 为合约所属组织的地址。返回计算得出的地址。

19.4 func CalcOrgID()

函数原型：

```
func CalcOrgID(name string) string
```

根据组织名称计算出组织标识。`name` 为组织名称。返回计算得出的组织标识。

19.5 func CheckAddress()

函数原型：

```
func (IBlockchainHelper) CheckAddress(addr types.Address) error
```

校验地址格式的合法性。`addr` 为地址。成功返回 `nil`。

19.6 func GetBlock()

函数原型：

```
func (IBlockChainHelper) GetBlock(height int64) IBlock
```

根据高度读取区块信息。`height` 为指定的区块高度。返回区块信息对象。如果输入的高度小于等于 `0`，则返回 `nil`，如果输入的高度没有保存区块信息或区块数据不正常，则返回一个该高度的区块信息对象，其它参数全部为空值。

20 sdk/IBuildHelper

接口 `sdk/IBuildHelper` 封装了对智能合约进行构建的帮助对象。

20.1 func Build()

函数原型：

```
func (IBuildHelper) Build(meta std.ContractMeta) std.BuildResult
```

构建合约。 `meta` 为输入的合约元数据，`std.ContractMeta` 结构定义如下：

```
package std

type ContractMeta struct {
    Name         string        //合约名称
    ContractAddr types.Address //合约地址
    OrgID        string        //合约所属组织ID
    Version      string        //合约的版本
    EffectHeight int64         //合约生效高度
    LoseHeight   int64         //合约失效高度（固定为0）
    CodeData     []byte        //合约代码压缩包数据
    CodeHash     []byte        //合约代码的哈希
    CodeDevSig   []byte        //合约开发者对合约代码压缩包的签名数据
    CodeOrgSig   []byte        //组织对合约开发者的签名进行的签名数据
}
```

返回构建结果的详细信息。`std.BuildResult` 结构定义如下：

```
package std

type Method struct {
    MethodID     string        // 方法ID，方法ID的计算与方法原型相关
    Gas          int64         // 方法在调用时消耗的燃料
    ProtoType    string        // 方法原型
}

type BuildResult struct {
    Code         uint32
    Error        string
    Methods      []Method      // 公开的方法列表
    Interfaces   []Method      // 公开的接口列表
    Mine         []Method      // 公开的挖矿接口
    OrgCodeHash  []byte        // 组织所有有效合约代码的哈希（编译以后的程序名称）
}
```

满足如下任意一条则导致失败，当失败时，返回 `BuildResult.Code != types.CodeOK (200)`：

1．当前合约不是合约管理合约；
2．当前合约所属组织不是创世组织；
3．合约编译过程失败。

21 sdk/IContractHelper

接口 `sdk/IContractHelper` 封装了对智能合约进行访问的帮助对象。

21.1 func ContractOfAddress()

函数原型：

```
func (IContractHelper) ContractOfAddress(addr types.Address) IContract
```

根据合约地址构造一个合约对象并读取合约信息，用来对合约进行一些操作。`addr` 为合约地址。返回合约对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 参数 addr 不是地址;
2. 参数 addr 不是合约。
```

21.2 func ContractOfToken()

函数原型：

```
func (IContractHelper) ContractOfToken(tokenAddr types.Address) IContract
```

根据代币地址构造一个针对该代币的合约对象并读取当前有效版本的合约信息，用来对合约进行一些操作。`tokenAddr` 为代币地址。返回合约对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 参数 tokenAddr 不是地址;
2. 参数 tokenAddr 不是代币。
```

21.3 func ContractOfName()

函数原型：

```
func (IContractHelper) ContractOfName(name string) IContract
```

根据合约名称构造一个合约对象并读取合约信息，用来对合约进行一些操作。`name` 为合约名称。返回合约对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 参数 name 为空串;
2. 参数 name 不是合约。
```

22 sdk/IReceiptHelper

接口 `sdk/IReceiptHelper` 封装了对收据进行处理的帮助对象。

22.1 func Emit()

函数原型：

```
func (IReceiptHelper) Emit(interface Interface{})
```

发送一个收据，SDK底层实现会自动将传入的收据对象进行序列化作为本次调用合约的输出收据集中的一员。`interface` 为收据对象。

SDK辅助工具会自动将智能合约定义的收据生成相关代码调用 `Emit()` 函数，合约代码中可以不用直接使用这个函数。示例如下：

```
//@:public:receipt
type receipt interface {
    emitTransferMyCoin(token, from, to types.Address, value bn.Number)
}

func (mc *Mycoin) Transfer(to types.Address, value bn.Number) {
    .
    .
    .
    mc.emitTransferMyCoin(
      mc.sdk.Message().Contract().Address(),
      sender,
      receiver,
      value,
    )
}
```

23 sdk/IGenesisHelper

接口 `sdk/IGenesisHelper` 封装了对创世信息进行访问的帮助对象。

23.1 func ChainID()

函数原型：

```
func (IGenesisHelper) ChainID() string
```

返回创世时指定的区块链ID。

23.2 func OrgID()

函数原型：

```
func (IGenesisHelper) OrgID() string
```

返回创世时指定的创世组织ID。

23.3 func Contracts()

函数原型：

```
func (IGenesisHelper) Contracts() []IContract
```

返回创世时部署的智能合约对象列表。

23.4 func Token()

函数原型：

```
func (IGenesisHelper) Token() IToken
```

返回创世时指定的创世通证对象。

24 sdk/ITokenHelper

接口 `sdk/ITokenHelper` 封装了对 BRC20 标准代币进行访问的帮助对象。

24.1 func RegisterToken()

函数原型：

```
func (ITokenHelper) RegisterToken(name string,
                                  symbol string,
                                  totalSupply Number,
                                  addSupplyEnabled bool,
                                  burnEnabled bool) IToken
```

向 BCBChain 区块链注册一个 `BRC20` 标准代币。`name` 为代币名称，`symbol` 为代币符号，`totalSupply` 为总的供应量（单位：`cong`），`addSupplyEnabled` 为是否允许增发，`burnEnabled` 为是否允许燃烧。注册成功返回对应的代币对象。

满足如下任意一条则注册失败，返回 `nil`：

```
1. 消息调用的发起人不是合约的拥有者;
2. 参数 name 的长度不在 [3,40] 的范围内;
3. 参数 name 对应的代币已经存在;
4. 参数 symbol 的长度不在 [3,20] 的范围内;
5. 参数 symbol 对应的代币已经存在;
6. 参数 totalSupply 小于 1000000000(cong);
7. 合约以前已经注册过一个 BRC20 标准代币;
8. 当前合约未定义标准转账方法或标准转账接口。
```

24.2 func Token()

函数原型：

```
func (ITokenHelper) Token() IToken
```

返回当前合约所注册的代币信息，用来对代币进行一些操作。返回代币对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 当前合约没有注册代币。
```

24.3 func TokenOfAddress()

函数原型：

```
func (ITokenHelper) TokenOfAddress(tokenAddr types.Address) IToken
```

根据代币地址获取代币的信息，用来对代币进行一些操作。`tokenAddr` 为代币地址。返回代币对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 参数 tokenAddr 不是地址;
2. 参数 tokenAddr 不是代币。
```

24.4 func TokenOfName()

函数原型：

```
func (ITokenHelper) TokenOfName(name string) IToken
```

按代币名称获取代币的信息，用来对代币进行一些操作。`name` 为代币名称。返回代币对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 参数 name 不是一个代币的名称。
```

24.5 func TokenOfSymbol()

函数原型：

```
func (ITokenHelper) TokenOfSymbol(symbol string) IToken
```

按代币符号获取代币的信息，用来对代币进行一些操作。`symbol` 为代币符号。返回代币对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 参数 symbol 不是一个代币的符号。
```

24.6 func TokenOfContract()

函数原型：

```
func (ITokenHelper) TokenOfContract(contractAddr types.Address) IToken
```

根据合约地址获取该合约所注册的代币的信息，用来对代币进行一些操作。`contractAddr` 为合约地址。返回代币对象。

满足如下任意一条则发生错误，当错误时，返回 `nil`：

```
1. 参数 contractAddr 不是地址;
2. 参数 contractAddr 不是一个合约;
3. 参数 contractAddr 对应的合约没有注册代币。
```

24.7 func BaseGasPrice()

函数原型：

```
func (ITokenHelper) BaseGasPrice() int64
```

返回基础燃料价格（单位：`cong`）。

25 sdk/IStateHelper

接口 `sdk/IStateHelper` 封装了对存储状态进行访问的帮助对象。

25.1 func Check()

函数原型：

```
func (IStateHelper) Check(key string) bool
```

根据给定的KEY值，检测在智能合约被许可的范围内是否存在KEY值指定的数据。`key` 为KEY值。

25.2 func Get()

函数原型：

```
func (IStateHelper) Get(key string, defaultData Interface{}) Interface{}
func (IStateHelper) GetEx(key string, defaultData Interface{}) Interface{}
```

根据给定的KEY值，在智能合约被许可的范围内读取数据。`key` 为KEY值，`defaultData` 为Value对应存储对象类型的模板或默认值。如果数据不存在则 `Get()` 返回 `nil`， `GetEx()` s返回默认值。

下面是一些对基础类型封装的简便读取函数：

```
func (IStateHelper) GetInt(key string) int
func (IStateHelper) GetInt8(key string) int8
func (IStateHelper) GetInt16(key string) int16
func (IStateHelper) GetInt32(key string) int32
func (IStateHelper) GetInt64(key string) int64
func (IStateHelper) GetUint(key string) uint
func (IStateHelper) GetUint8(key string) uint8
func (IStateHelper) GetUint16(key string) uint16
func (IStateHelper) GetUint32(key string) uint32
func (IStateHelper) GetUint64(key string) uint64
func (IStateHelper) GetByte(key string) byte
func (IStateHelper) GetBool(key string) bool
func (IStateHelper) GetString(key string) string

func (IStateHelper) GetInts(key string) []int
func (IStateHelper) GetInt8s(key string) []int8
func (IStateHelper) GetInt16s(key string) []int16
func (IStateHelper) GetInt32s(key string) []int32
func (IStateHelper) GetInt64s(key string) []int64
func (IStateHelper) GetUints(key string) []uint
func (IStateHelper) GetUint8s(key string) []uint8
```

```
func (IStateHelper) GetUint16s(key string) []uint16
func (IStateHelper) GetUint32s(key string) []uint32
func (IStateHelper) GetUint64s(key string) []uint64
func (IStateHelper) GetBytes(key string) []byte
func (IStateHelper) GetBools(key string) []bool
func (IStateHelper) GetStrings(key string) []string
```

以上函数如果从状态数据库中读不到数据，直接返回对应基础类型的默认值。

25.3 func Set()

函数原型：

```
func (IStateHelper) Set(key string, data Interface{})
```

将输入的数据保存到状态数据库智能合约被允许的KEY值下。`key`为KEY值，`data`为要保存的数据对象。

下面是一些对基础类型封装的简便设置函数：

```
func (IStateHelper) SetInt(key string, v int)
func (IStateHelper) SetInt8(key string, v int8)
func (IStateHelper) SetInt16(key string, v int16)
func (IStateHelper) SetInt32(key string, v int32)
func (IStateHelper) SetInt64(key string, v int64)
func (IStateHelper) SetUint(key string, v uint)
func (IStateHelper) SetUint8(key string, v uint8)
func (IStateHelper) SetUint16(key string, v uint16)
func (IStateHelper) SetUint32(key string, v uint32)
func (IStateHelper) SetUint64(key string, v uint64)
func (IStateHelper) SetByte(key string, v byte)
func (IStateHelper) SetBool(key string, v bool)
func (IStateHelper) SetString(key string, v string)

func (IStateHelper) SetInts( key string, v []int )
func (IStateHelper) SetInt8s( key string, v []int8 )
func (IStateHelper) SetInt16s( key string, v []int16 )
func (IStateHelper) SetInt32s( key string, v []int32 )
func (IStateHelper) SetInt64s( key string, v []int64 )
func (IStateHelper) SetUints( key string, v []uint )
func (IStateHelper) SetUint8s( key string, v []uint8 )
func (IStateHelper) SetUint16s( key string, v []uint16 )
func (IStateHelper) SetUint32s( key string, v []uint32 )
func (IStateHelper) SetUint64s( key string, v []uint64 )
func (IStateHelper) SetBytes(key string, v []byte)
func (IStateHelper) SetBools( key string, v []bool )
func (IStateHelper) SetStrings( key string, v []string )
```

将输入的数据保存到状态数据库智能合约被允许的KEY值下。`key`为KEY值，`v`为基础类型数据值。

25.4 func Delete()

函数原型：

```
func (IStateHelper) Delete(key string)
```

在状态数据库中删除 `key` 值对应的数据。

25.5 func Flush()

函数原型：

```
func (IStateHelper) Flush()
```

将对状态数据库的更新刷新到数据库层，不关闭事务。

25.6 func McCheck()

函数原型：

```
func (sh *StateHelper) McCheck(key string) bool
```

根据给定的KEY值，检测在智能合约被许可的范围内是否存在KEY值指定的数据，包括缓存与数据库。`key` 为KEY值，如果存在将被自动加载到缓存中来。

25.7 func McGet()

函数原型：

```
func (IStateHelper) McGet(key string, defaultData Interface{}) Interface{}
func (IStateHelper) McGetEx(key string, defaultData Interface{}) Interface{}
```

根据给定的KEY值，在智能合约被许可的范围内读取数据，并将数据缓存在内存中，在后续智能合约的调用消息中可以直接从内存中读取，而不需要再次访问数据库。`key` 为KEY值，`defaultData` 为Value对应存储对象类型的模板或默认值。如果数据不存在则 `McGet()` 返回 `nil`， `McGetEx()` 返回默认值。

下面是一些对基础类型封装的简便读取函数：

```
func (IStateHelper) McGetInt(key string) int
func (IStateHelper) McGetInt8(key string) int8
func (IStateHelper) McGetInt16(key string) int16
func (IStateHelper) McGetInt32(key string) int32
func (IStateHelper) McGetInt64(key string) int64
```

```
func (IStateHelper) McGetUint(key string) uint
func (IStateHelper) McGetUint8(key string) uint8
func (IStateHelper) McGetUint16(key string) uint16
func (IStateHelper) McGetUint32(key string) uint32
func (IStateHelper) McGetUint64(key string) uint64
func (IStateHelper) McGetByte(key string) byte
func (IStateHelper) McGetBool(key string) bool
func (IStateHelper) McGetString(key string) string

func (IStateHelper) McGetInts(key string) []int
func (IStateHelper) McGetInt8s(key string) []int8
func (IStateHelper) McGetInt16s(key string) []int16
func (IStateHelper) McGetInt32s(key string) []int32
func (IStateHelper) McGetInt64s(key string) []int64
func (IStateHelper) McGetUints(key string) []uint
func (IStateHelper) McGetUint8s(key string) []uint8
func (IStateHelper) McGetUint16s(key string) []uint16
func (IStateHelper) McGetUint32s(key string) []uint32
func (IStateHelper) McGetUint64s(key string) []uint64
func (IStateHelper) McGetBytes(key string) []byte
func (IStateHelper) McGetBools(key string) []bool
func (IStateHelper) McGetStrings(key string) []string
```

以上函数如果从状态数据库中读不到数据，直接返回对应基础类型的默认值。

25.8 func McSet()

函数原型：

```
func (IStateHelper) McSet(key string, data Interface{})
```

将输入的数据保存到状态数据库智能合约被允许的KEY值下，同时更新内存缓存，在后续智能合约的调用消息中可以直接从内存中读取，而不需要再次访问数据库。`key` 为KEY值，`data` 为要保存的数据对象。

下面是一些对基础类型封装的简便设置函数：

```
func (IStateHelper) McSetInt(key string, v int)
func (IStateHelper) McSetInt8(key string, v int8)
func (IStateHelper) McSetInt16(key string, v int16)
func (IStateHelper) McSetInt32(key string, v int32)
func (IStateHelper) McSetInt64(key string, v int64)
func (IStateHelper) McSetUint(key string, v uint)
func (IStateHelper) McSetUint8(key string, v uint8)
func (IStateHelper) McSetUint16(key string, v uint16)
func (IStateHelper) McSetUint32(key string, v uint32)
func (IStateHelper) McSetUint64(key string, v uint64)
func (IStateHelper) McSetByte(key string, v byte)
func (IStateHelper) McSetBool(key string, v bool)
func (IStateHelper) McSetString(key string, v string)
```

```
func (IStateHelper) McSetInts( key string, v []int )
func (IStateHelper) McSetInt8s( key string, v []int8 )
func (IStateHelper) McSetInt16s( key string, v []int16 )
func (IStateHelper) McSetInt32s( key string, v []int32 )
func (IStateHelper) McSetInt64s( key string, v []int64 )
func (IStateHelper) McSetUints( key string, v []uint )
func (IStateHelper) McSetUint8s( key string, v []uint8 )
func (IStateHelper) McSetUint16s( key string, v []uint16 )
func (IStateHelper) McSetUint32s( key string, v []uint32 )
func (IStateHelper) McSetUint64s( key string, v []uint64 )
func (IStateHelper) McSetBytes(key string, v []byte)
func (IStateHelper) McSetBools( key string, v []bool )
func (IStateHelper) McSetStrings( key string, v []string )
```

将输入的数据保存到状态数据库智能合约被允许的KEY值下，同时更新内存缓存，在后续智能合约的调用消息中可以直接从内存中读取，而不需要再次访问数据库。`key` 为KEY值，`v` 为基础类型数据值。

25.9 func McClear()

函数原型：

```
func (IStateHelper) McClear(key string)
```

清除内存缓存中指定的数据（保留状态数据库中的数据）。`key` 为KEY值。

25.10 func McDelete()

函数原型：

```
func (IStateHelper) McDelete(key string)
```

在状态数据库中删除 `key` 值对应的数据，同时从缓存中清除 `key` 值对应的数据。