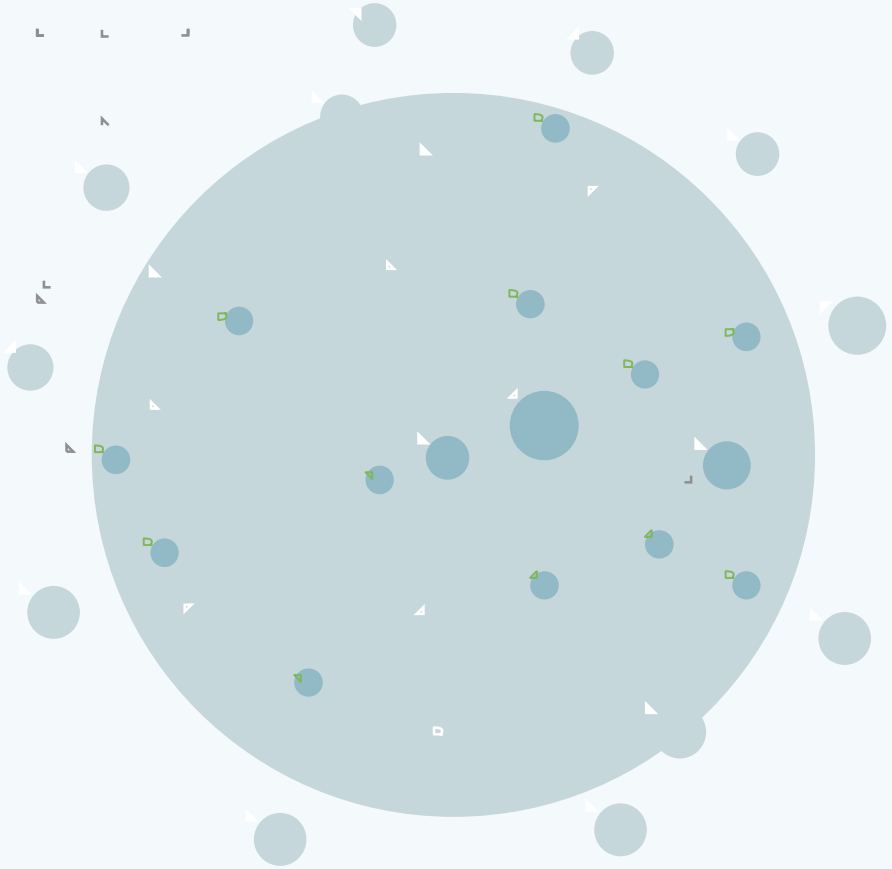
INNO×NEO 区块链开放夜

第三期主题：智能合约应用场景和案例分析

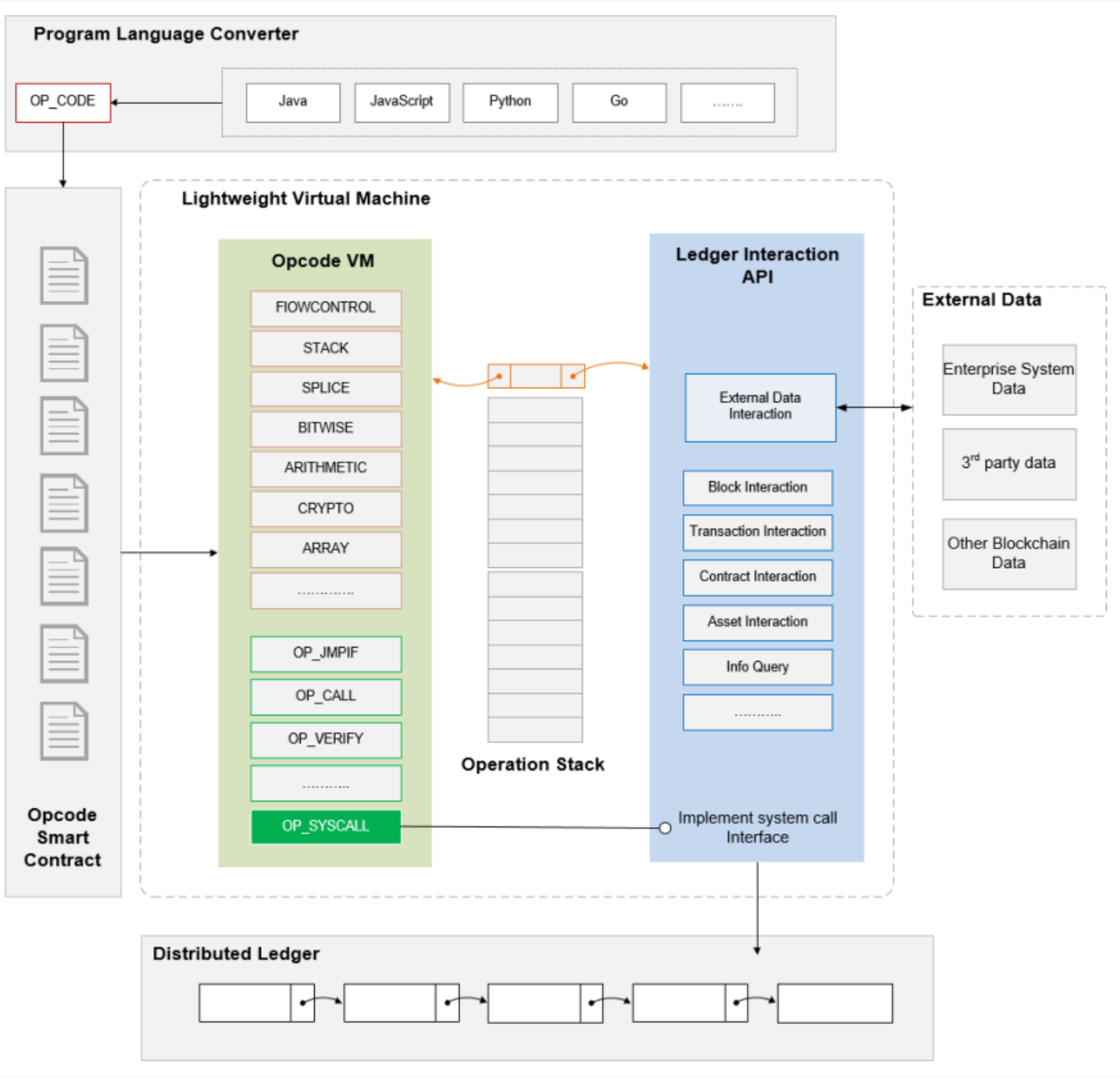
INNOSPACE+

NEO smart economy



智能合约与数字身份验证

分布科技 刘尖奇


Program Language Converter
OP_CODE
Java
JavaScript
Python
Go
.....
Lightweight Virtual Machine
Opcode VM
FlOWCONTROL
STACK
SPLICE
BITWISE
ARITHMETIC
CRYPTO
ARRAY
.....
OP_JMPIF
OP_CALL
OP_VERIFY
.....
OP_SYSCALL
Operation Stack
Ledger Interaction API
External Data Interaction
Block Interaction
Transaction Interaction
Contract Interaction
Asset Interaction
Info Query
.....
Implement system call Interface
External Data
Enterprise System Data
3rd party data
Other Blockchain Data
Opcode Smart Contract
Distributed Ledger

智能合约的类型

VerificationCode 和 FunctionCode

继承自 VerificationCode 的智能合约

- 继承于 VerificationCode 的智能合约是可以在钱包客户端中生成合约地址，如果用户想使用该合约地址里的一笔资产（用户发送了一笔使用该合约地址资产的交易）便会触发该智能合约，合约执行并验证你是否可以动用这笔资产。
- 当该合约返回值为 true 时，验证通过， 用户可以花费这笔资产；
- 当该合约返回值为 false 时，验证失败，用户不可以花费这笔资产，这笔验证失败的交易无法被其它节点广播，也不会被共识节点确认。
- 继承自 VerificationCode 的智能合约的入口点是 Verify 方法，Verify 方法的返回值必须是 bool 类型。

VerificationCode

任何人可用的合约地址

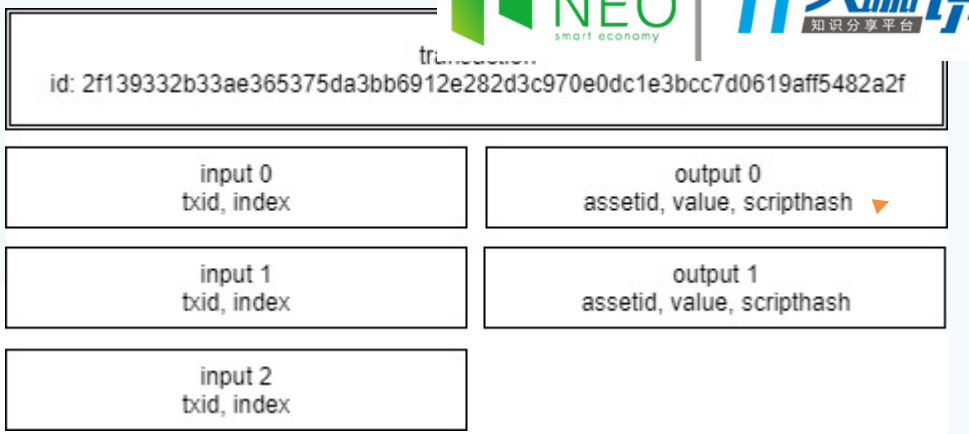
```
using Neo.SmartContract.Framework;
using Neo.SmartContract.Framework.Services.Neo;
using Neo.SmartContract.Framework.Services.System;

namespace Neo.SmartContract
{
    public class Test : VerificationCode
    {
        public static bool Verify(byte[] signature)
        {
            return true;
        }
    }
}
```

合约脚本：52c56b6c766b00527ac461516c766b51527ac46203006c766b51c3616c7566

智能合约参数和返回值

```
/// 表示智能合约的参数类型
public enum ContractParameterType : byte
{
        /// 签名
        Signature = 0,
        Boolean = 1,
        /// 整数
        Integer = 2,
        /// 160位散列值
        Hash160 = 3,
        /// 256 位散列值
        Hash256 = 4,
        /// 字节数组
        ByteArray = 5,
        PublicKey = 6,
        Void = 0xff
}
```

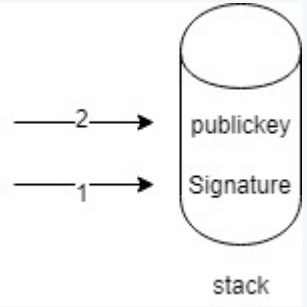
构造交易原始数据



类型：80
版本：00
属性：00
输入：01 2f139332b33ae365375da3bb6912e282d3c970e0dc1e3bcc7d0619aff5482a2f 0000
输出：02 fa76aa81dd4852927ebbdddc11b09e99581bac3306c6384331a52a3f2600ada0 e803000000000000 5f1f7caef4c5712fe63d09d81979ab7b36b6eda1
fa76aa81dd4852927ebbdddc11b09e99581bac3306c6384331a52a3f2600ada0 2823000000000000 082e502f35ec5cf8cc1209d0de00c550578911a7

程序：01
41 403d7a4350b17066ce710eee96e4048b39285db8ff7fd71ecea9502e6695e71c8519f7995aa1a627a87ef3787b0f5f8ce7267851cbbde3a02ac3a093a383824846
23 21039fbb47841f7338c0c654addd6225995642b5b6d492413563f7f8755ba83c0ecdac

交易验证：
1、检查输入的UTXO所属的 scripthash，是否与程序中合约脚本散列的结果一致。
2、使用虚拟机运行程序，如果返回结果为true则鉴权通过，否则失败。



VerificationCode

单签名合约

OPCODE: 21+PUBLICKEY+AC

例如: 21 039fbb47841f7338c0c654addd6225995642b5b6d492413563f7f8755ba83c0ecd ac

标准账户中公钥、合约脚本、合约脚本散列与地址的关系

公钥：039fbb47841f7338c0c654addd6225995642b5b6d492413563f7f8755ba83c0ecd
合约脚本：21039fbb47841f7338c0c654addd6225995642b5b6d492413563f7f8755ba83c0ecdac
合约脚本散列：082e502f35ec5cf8cc1209d0de00c550578911a7
地址：AGX8Xw1HbWGFwozH4tH1ej14FXVdkeTTau

合约脚本 ->合约脚本散列：sha256(), RIPEMD160()
合约脚本散列 ->地址：double sha256() -> checksum, base58(data + checksum[0:4])

http://www.neowallet.net 工具<tab>

VerificationCode

多签名合约

```
OPCODE:5m(signature num) +
       21 + PUBLICKEYA +
       21 + PUBLICKEYB +
       ……
       5n(publickey num) +
       AE
```

VerificationCode

锁仓合约

```
using Neo.SmartContract.Framework;
using Neo.SmartContract.Framework.Services.Neo;
using Neo.SmartContract.Framework.Services.System;

namespace Neo.SmartContract
{
    public class Test : VerificationCode
    {
        public static bool Verify(byte[] signature)
        {
            Header header = Blockchain.GetHeader(Blockchain.GetHeight());
            if (header.Timestamp < 1505304000) // 2017-9-13 20:00
                return false;

            // publickey 03ee283f91d04e8225f6fdee0fe0fb86112a54e5af56f23a40dfdbc4f385a3549a
            return VerifySignature(new byte[] {3, 238, 40, 63, 145, 208, 78, 130, 37, 246,
 253, 238, 15, 224, 251, 134, 17, 42, 84, 229, 175, 86, 242, 58, 64, 223, 219, 196,
 243, 133, 163, 84, 154}, signature);

        }
    }
}
```

继承自 FunctionCode 的智能合约

- 继承于 FunctionCode 的智能合约可以编译好后发布到区块链中供其它用户使用。它相当于编程中的“函数”，即可以被“主函数”（上述的两种智能合约调用方式）调用，也能被“其它函数”（其它继承于 FunctionCode 的智能合约）调用。
- 继承自 FunctionCode 的智能合约的入口点是 Main 方法，返回值可以是 void、int、byte[]
- 本文是对上文的代码进行详述。
- 创建好的智能合约项目默认继承自 FunctionCode 入口点是 Main

FunctionCode

获取区块时间

```
using Neo.SmartContract.Framework;
using Neo.SmartContract.Framework.Services.Neo;
using Neo.SmartContract.Framework.Services.System;

namespace Neo.SmartContract
{
    public class HelloWorld : FunctionCode
    {
        public static int Main()
        {
            Header header = Blockchain.GetHeader(Blockchain.GetHeight());
            return (int)header.Timestamp;
        }
    }
}
```

合约脚本哈希： 0x2685f2a44a4883126a59ca5cd90ecfd340c5ac1d

```
public class BlockBase
{
    /// 区块版本
    public uint Version;
    /// 前一个区块的散列值
    public UInt256 PrevHash;
    /// 该区块中所有交易的Merkle树的根
    public UInt256 MerkleRoot;
    /// 时间戳
    public uint Timestamp;
    /// 区块高度
    public uint Index;
    public ulong ConsensusData;
    /// 下一个区块的记账合约的散列值
    public UInt160 NextConsensus;
    /// 用于验证该区块的脚本
    public Witness Script;
    public UInt256 Hash;
    public int Size;
}
```

FunctionCode

计算A+B

```
using Neo.SmartContract.Framework;
using Neo.SmartContract.Framework.Services.Neo;
using Neo.SmartContract.Framework.Services.System;

namespace Neo.SmartContract
{
    public class HelloWorld : FunctionCode
    {
        public static int Main(int a, int b)
        {
            return a+b;
        }
    }
}
```

合约脚本哈希： 0x9a7eab74e3578976a0c62f7e5387022a99994e96

FunctionCode

第三方调用A+B合约

```
using Neo.SmartContract.Framework;
using Neo.SmartContract.Framework.Services.Neo;
using Neo.SmartContract.Framework.Services.System;

namespace Neo.SmartContract
{
    public class HelloWorld : FunctionCode
    {
        [Appcall("9a7eab74e3578976a0c62f7e5387022a99994e96")]
        public static extern int AnotherContract(int a, int b);
        public static int Main()
        {
            return AnotherContract(1,2);
        }
    }
}
```

合约脚本哈希： 0x2d669affb98d4e7c12a0bfffbeb071d49fa30ce0

FunctionCode

锁仓合约

```
using Neo.SmartContract.Framework;
using Neo.SmartContract.Framework.Services.Neo;
using Neo.SmartContract.Framework.Services.System;

namespace Neo.SmartContract
{
    public class HelloWorld : FunctionCode
    {
        public static bool Main(uint timestamp, byte[] pubkey, byte[] signature)
        {
            Header header = Blockchain.GetHeader(Blockchain.GetHeight());
            if (header.Timestamp < timestamp)
            {
                return false;
            }
            return VerifySignature(pubkey, signature);
        }
    }
}
```

合约脚本哈希： 0x395c71b12759a9751559c5b9d6c29e4c3fcad72b

资源列表

NEO 智能合约介绍
http://docs.neo.org/zh-cn/sc/introduction.html

智能合约环境准备：如何用 C# 编写 NEO 智能合约
http://docs.neo.org/zh-cn/sc/getting-started-csharp.html

智能合约参数和返回值
http://docs.neo.org/zh-cn/sc/tutorial/Parameter.html

感谢您的聆听

Thanks