INNO×NEO 区块链开放夜

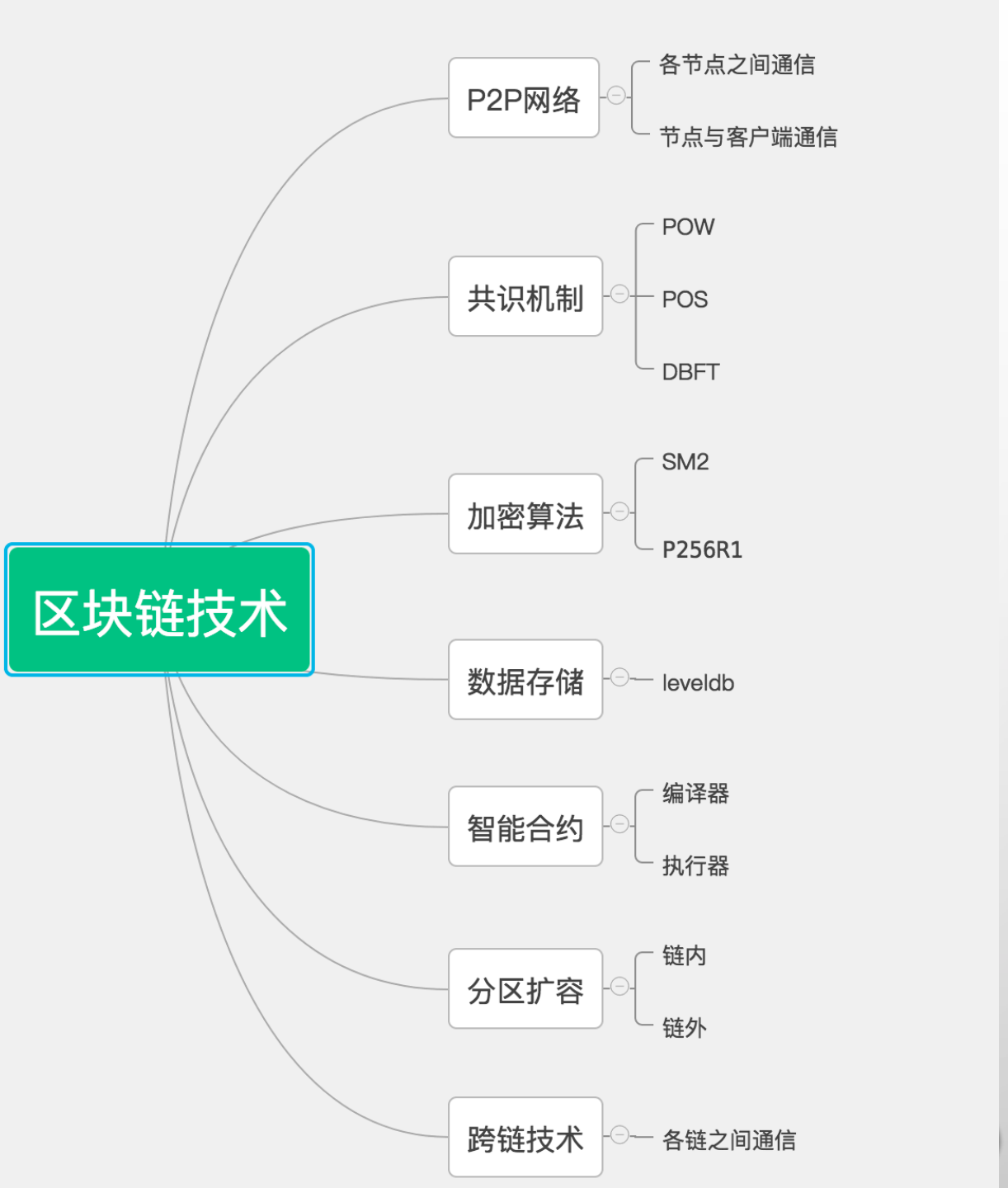
第三期主题：智能合约应用场景和案例分析

智能合约应用场景与案例分析

谈元

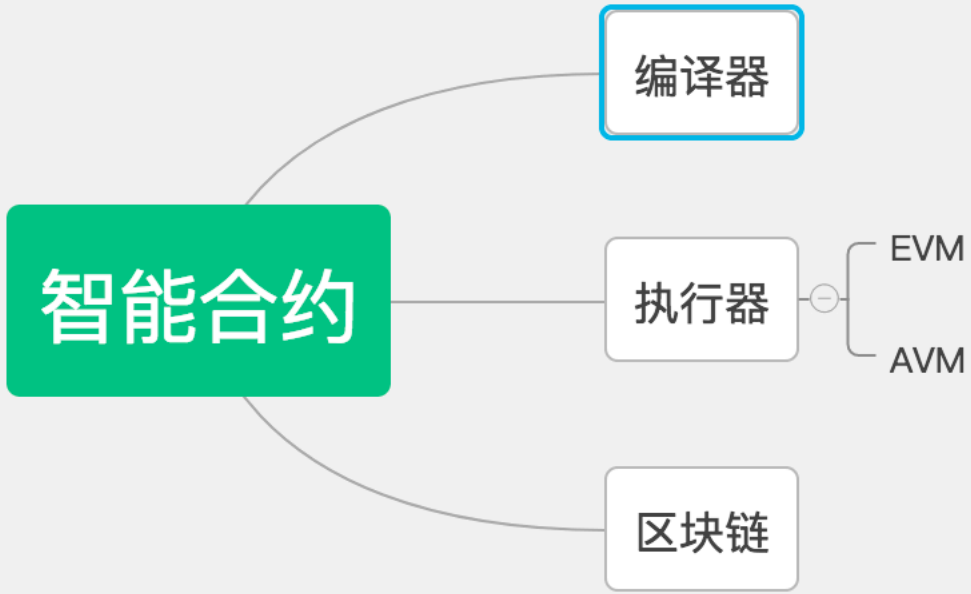
区块链技术

- 区块链本质上是一个去中心化的分布式数据库
- 实现这样的一个分布式账本所需要的技术模块包含：P2P网路、共识机制、加密算法、数据存储


区块链技术
P2P网络
各节点之间通信
节点与客户端通信
共识机制
POW
POS
DBFT
加密算法
SM2
P256R1
数据存储
leveldb
智能合约
编译器
执行器
分区扩容
链内
链外
跨链技术
各链之间通信

智能合约

- 智能合约是运行在区块链上的一段脚本，这段脚本可以实现你想要实现的任意的（理论上）业务逻辑。
- 实现智能合约所需要的模块：编译器、执行器、区块链


智能合约
编译器
执行器
EVM
AVM
区块链

编译器

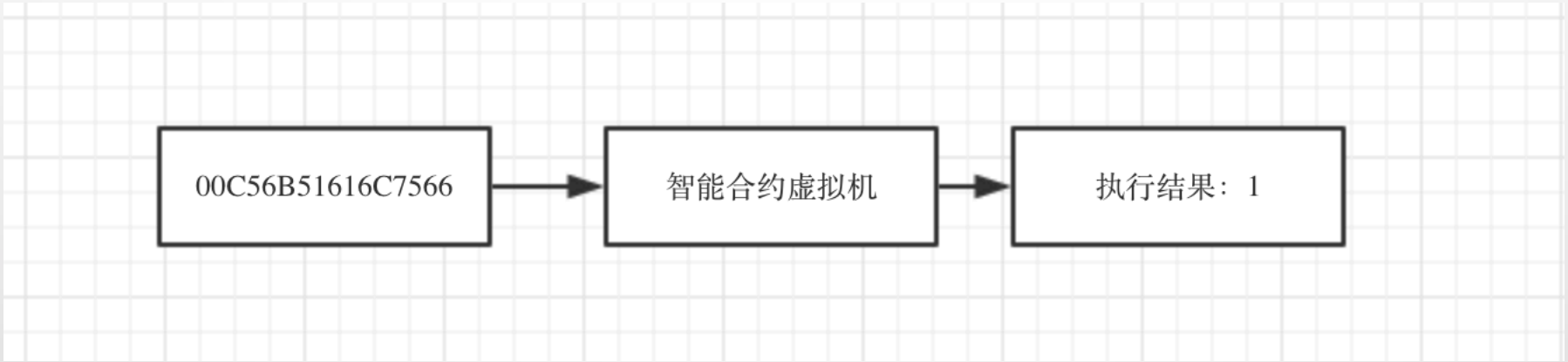
- 智能合约编译器是能将程序编译成VM可识别的二进制码的容器

```
1  using Neo.SmartContract.Framework;
2  using Neo.SmartContract.Framework.Services.Neo;
3
4  class A : FunctionCode
5  {
6      public static int Main()
7      {
8          return 1;
9      }
10 }
```

00C56B51616C7566

执行器

- 智能合约执行器VIRTUAL MACHINE是一个区块链底层，能运行二进制代码的虚拟机。


00C56B51616C7566
智能合约虚拟机
执行结果：1

智能合约分类

- 智能合约分类：
- 继承自 FUNCTIONCODE 的智能合约
- 继承自 VERIFICATIONCODE 的智能合约

智能合约API

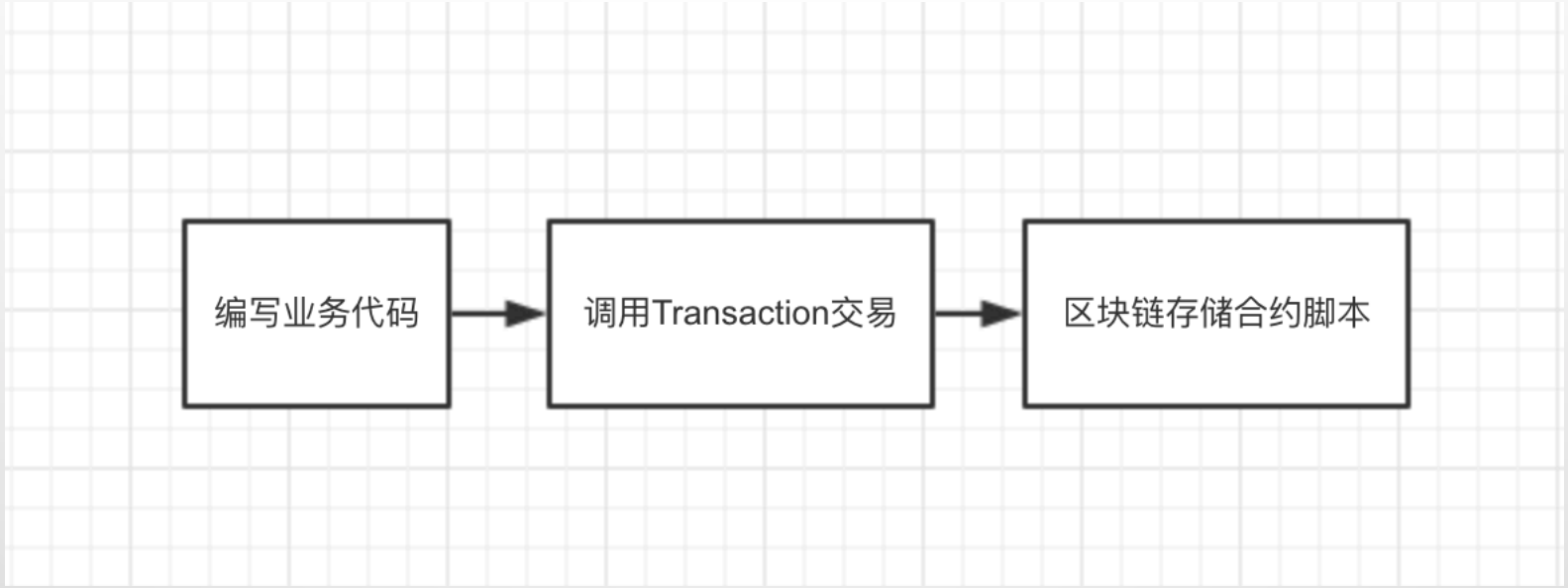
- 链接： HTTP://DOCS.NEO.ORG/ZH-CN/SC/FW/DOTNET/NEO.HTML

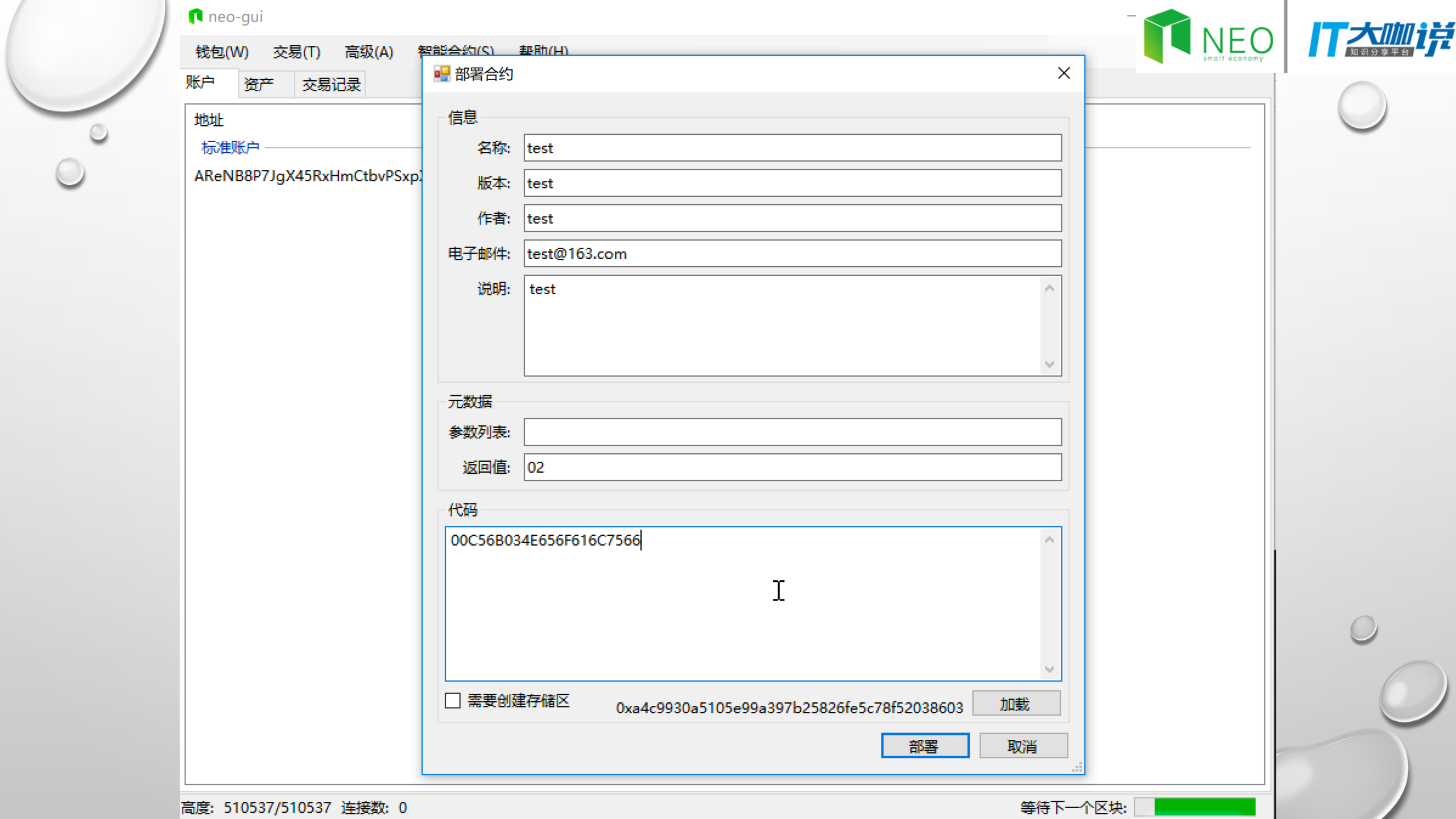
智能合约实战

- 智能合约在区块链上的应用主要包括两个步骤
- DEPLOY TRANSACTION：部署智能合约
- INVOKE TRANSACTION：调用智能合约

部署智能合约

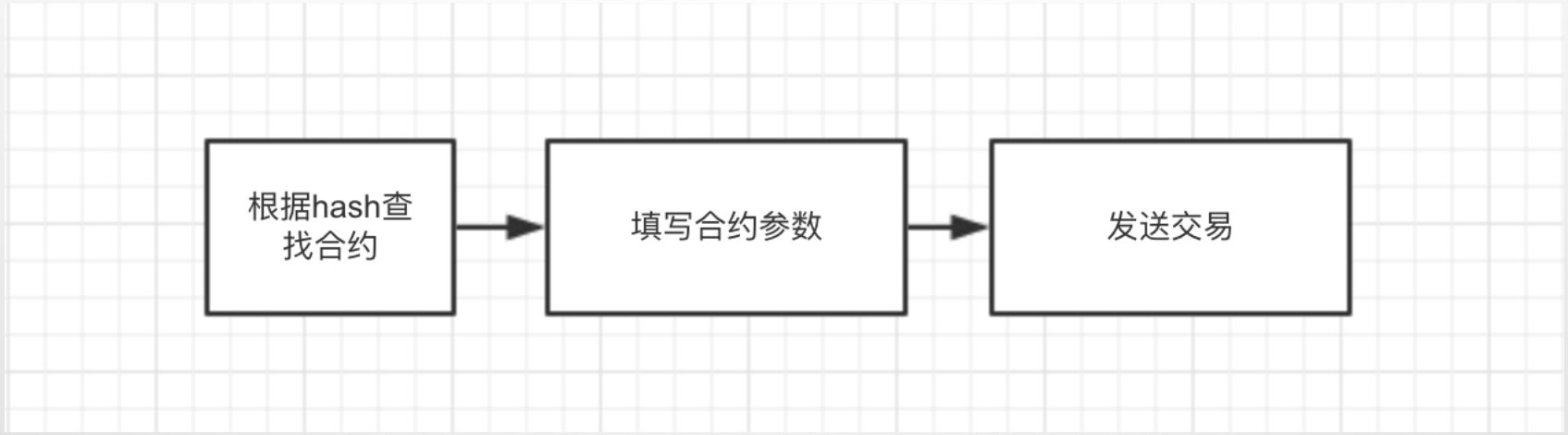
- 将程序通过智能合约编译器生成二进制码
- 发送TRANSACTION到区块链
- 区块链存储智能合约脚本在存储区

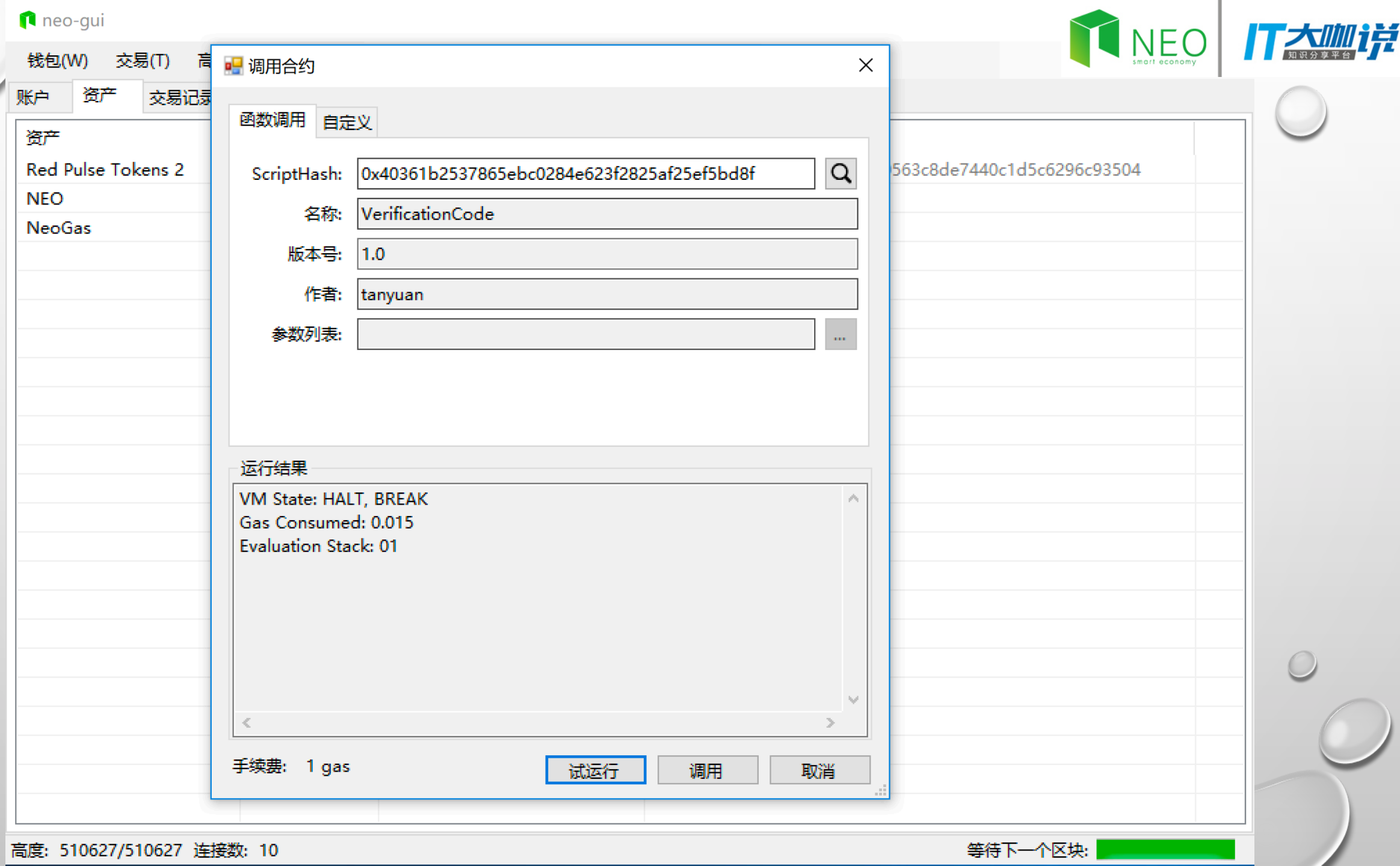

编写业务代码
调用Transaction交易
区块链存储合约脚本


neo-gui
钱包(W)
交易(T)
高级(A)
智能合约(S)
帮助(H)
账户
资产
交易记录
地址
标准账户
AReNB8P7JgX45RxHmCtbvPSxp
部署合约
信息
名称:
test
版本:
test
作者:
test
电子邮件:
test@163.com
说明:
test
元数据
参数列表:
返回值:
02
代码
00C56B034E656F616C7566
需要创建存储区
0xa4c9930a5105e99a397b25826fe5c78f52038603
加载
部署
取消
高度: 510537/510537 连接数: 0
等待下一个区块:

调用智能合约

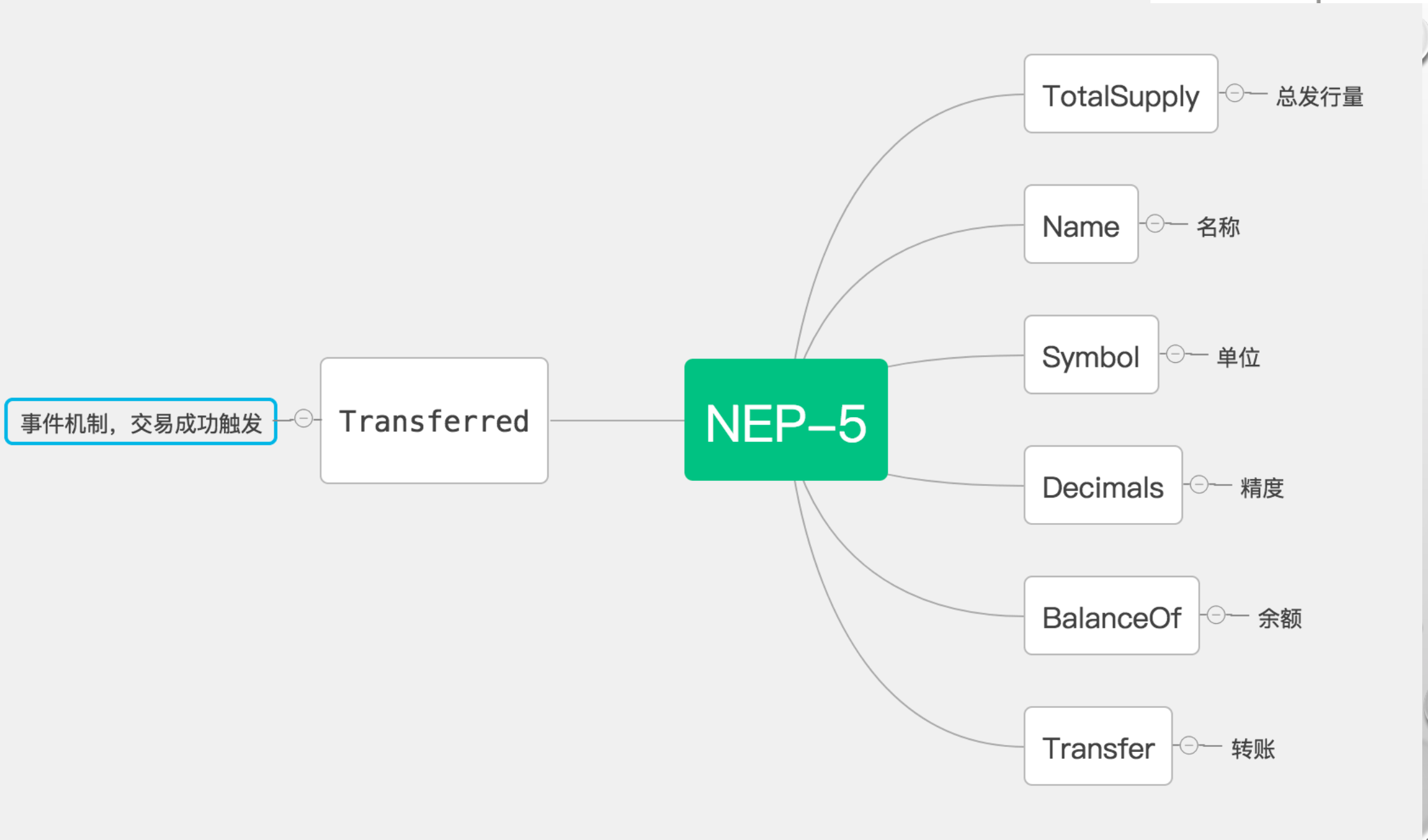
- 根据脚本HASH值查找智能区块链上是否存在该合约
- 填写调用合约的参数
- 发送TRANSACTION交易


根据hash查找合约
填写合约参数
发送交易


neo-gui
钱包(W)
交易(T)
账户
资产
交易记录
资产
Red Pulse Tokens 2
NEO
NeoGas
563c8de7440c1d5c6296c93504
调用合约
函数调用
自定义
ScriptHash:
0x40361b2537865ebc0284e623f2825af25ef5bd8f
名称:
VerificationCode
版本号:
1.0
作者:
tanyuan
参数列表:
...
运行结果
VM State: HALT, BREAK
Gas Consumed: 0.015
Evaluation Stack: 01
手续费:
1 gas
试运行
调用
取消
高度: 510627/510627
连接数: 10
等待下一个区块:

NEP-5 TOKEN STANDARD

- TOKEN STANDARD：代币规范
- 所有的代币发行的智能合约代码需实现该规范
- 客户端和浏览器会根据该规范实现相应的功能


NEP-5
TotalSupply
总发行量
Name
名称
Symbol
单位
Decimals
精度
BalanceOf
余额
Transfer
转账
Transferred
事件机制，交易成功触发

TotalSupply-发行总量

```
public static BigInteger TotalSupply()
{
    return Storage.Get(Storage.CurrentContext, "totalSupply").AsBigInteger();
}
```

Name-名称

```
public static string Name() => "name of the token";
```

Decimals-精度

```
public static byte Decimals() => 8;
```

Symbol-单位

```
public static string Symbol() => "SymbolOfTheToken";
```

BalanceOf-余额

```
public static BigInteger BalanceOf(byte[] address)
{
    return Storage.Get(Storage.CurrentContext, address).AsBigInteger();
}
```

Transferred-事件机制，转账成功触发

```
[DisplayName("transfer")]
public static event Action<byte[], byte[], BigInteger> Transferred;
```

Transfer-转账

```
public static bool Transfer(byte[] from, byte[] to, BigInteger value)
{
    if (value <= 0) return false;
    if (!Runtime.CheckWitness(from)) return false;
    BigInteger from_value = Storage.Get(Storage.CurrentContext, from).AsBigInteger();
    if (from_value < value) return false;
    if (from_value == value)
        Storage.Delete(Storage.CurrentContext, from);
    else
        Storage.Put(Storage.CurrentContext, from, from_value - value);
    BigInteger to_value = Storage.Get(Storage.CurrentContext, to).AsBigInteger();
    Storage.Put(Storage.CurrentContext, to, to_value + value);
    Transferred(from, to, value);
    return true;
}
```