NEO Name Service 白皮书 1.0

作者：李剑英 刘永新 2018.1

目 录

01

NNS背景介绍

1.1 NNS是什么

NNS（NEO Name Service）是Neo的域名服务，是一个基于Neo区块链的分布式、开源和可扩展的域名系统。旨在将钱包地址、智能合约Hash等人类难以记忆的无规则的字符串用单词短语简写等代替。我们首先提供以“.neo”结尾的域名服务。

通过域名服务，人们再也不用记忆看不懂的地址和Hash，只要知道一个单词或一个短语就能进行转账、使用合约。

NNS可以将域名解析到各种目标。最容易联想到的是Neo的地址(Address)，或者智能合约(ScriptHash)。我们预留了足够的扩展性，可以在NNS不升级的情况下支持更多的解析目标种类。

1.2 为什么需要NNS

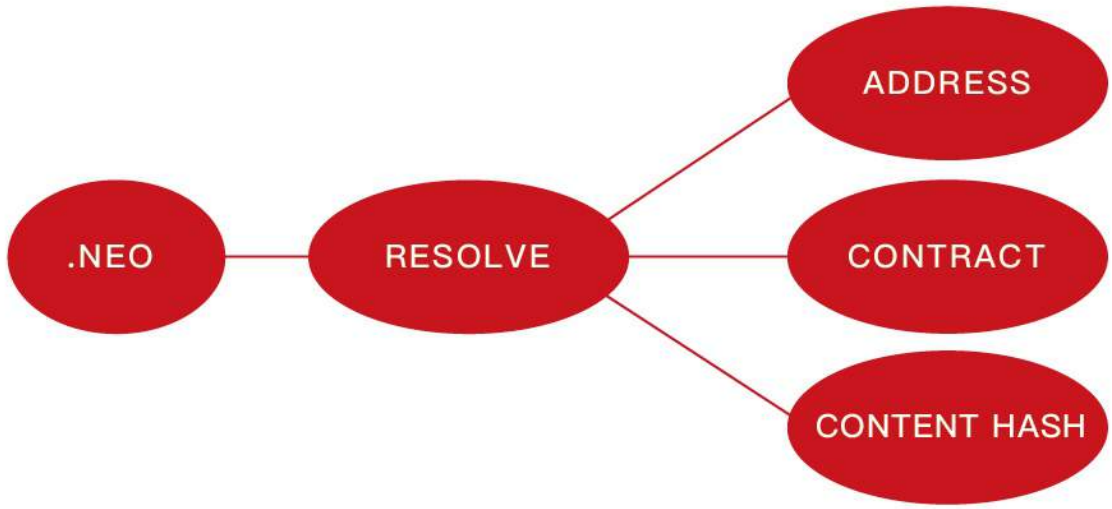
中本聪在设计比特币地址的时候，没有采用程序界常用的base64编码，而是自创了base58编码，去除了几个看起来会产生歧义的字符，如 0 (零), O (大写字母O), l (大写的字母i) and l (小写的字母L)，这体现了中本聪对区块链地址易用性的考虑。然而，区块链地址对于人类还是不够友好，太长，不方便记忆，不容易比较对错。未来随着区块链越来越普及，地址转账的缺点会越来越明显，就像我们今天发邮件很难用一个32位字符串作为一个邮箱账号一样。因此一个别名服务对于区块链系统的易用性有非常大的帮助，例如IPFS上有别名服务IPNS，以太坊上有自己的域名服务ENS，我们认为NEO系统也应该有自己的别名服务，我们称为NEO Name Service(NNS)，NEL社区（NEO中国开发志愿者社区）将实现NNS服务以提升NEO区块链易用性。

1.3 NNS的使用场景

别名服务的最主要使用场景在于别名转账，尤其是那些需要公开自己转账地址并且不常更换地址的账户，例如在ICO时，项目方需要提前公开自己官方帐户地址，但是有可能被黑客修改了官网地址，然而投资人很难发现，如果项目方提前公布一个简短易记的地址别名，别名被修改会很容易被发现，因此可以避免转账地址被黑客修改事件。

一个别名要指向什么样的资源，是可以灵活扩展的，只需要实现相应的解析器即可，除了可以指向一个账户地址，也可以指向一个合约地址，从而实现通过别名和智能合约交互。

区块链作为下一代互联网的基础设施，未来会有越来越多的服务基于区块链构建，例如去中心化云存储服务，而文件寻址是通过文件哈希值唯一标识实现的，我们可以为哈希值取一个容易理解的别名例如文件名，然后将别名映射到文件哈希，从而实现文件寻址，因此别名服务未来可以和NEO上的去中心化文件存储NEOFS结合使用。未来，随着构建在NEO上服务越来越多，NNS还可以为去中心化消息通信、邮件服务等提供别名服务。



1.4 NNS和ENS的关系

NNS和ENS具有相同的目标，都是为了提升区块链的易用性，但是基于不同的区块链平台实现，服务于不同的区块链平台。NNS在做系统设计时参考借鉴了ENS的设计，在此对他们表示感谢，同时我们也在ENS基础上做很多创新设计，例如将所有者合约从注册器模块中拆分出来以实现更灵活的所有权控制，在解析方式上，分为了快速解析和完整解析两种方式，在经济模型上引入了一种新型的智能令牌，以实现系统费用的重新分配。

02

NNS系统设计概述

2.1 NNS系统功能

NNS系统有两个作用 一是将beautiful.neo 等人类可读的名称解析为机器使用的标识符，如Neo的地址等。 二是为域名提供描述性数据，比如whois，合约接口描述等。

NNS 和 DNS的目标类似，但是基于区块链架构设计，是去中心化的，服务于区块链网络。NNS使用和DNS一样用点(.)分割的域名称系统，域的所有者对隶属于他的子域名有完全的分配权利。

.neo .gas 这样的根域名由一个称为（注册器Registry）的智能合约管理。一个注册器管理一个根域名，并设定取得其下一级域名所有权的规则。任何人均可遵照对应的注册器设定的规则取得下一级域名的所有权。

2.2 NNS系统架构

NNS有四个系统组件

1. 顶级域名合约（域名根是管理根域名的脚本）
2. 所有者（所有者可以是一个个人账户address，也可以是一个智能合约）
3. 注册器（专门负责给一个域名的子域名分配所有者的智能合约，根域名也会指定一个根域名的注册器）
4. 解析器（负责解析一个域名或者他的子域名）

2.2.1 顶级域名合约

域名根是一个根域名比如.test 所有信息的管理者。无论二级域名 aa.test 还是 三级域名bbb.aa.test，他们的所有者都保存在域名根之中。 域名根以字典的形式保存如下数据

1. 域名的所有者(owner)
2. 域名的注册器(register)
3. 域名的解析器(resolver)
4. 域名的TTL(域名到期时间)

2.2.2 所有者

域名的所有者可以是一个账户地址或者一个智能合约。（ens的设计是拥有域名的智能合约叫做注册器，实际上注册器只是owner的一个特例，我们将域名的所有者和注册器分开了，这个系统会变得更加清晰）

域名的所有者(owner)可以

1.将域名的所有权转移到另一个地址
2.更改注册器，最常见域名注册器为“管理员手动分配子域名”
3.更改解析器

允许所有者是一个智能合约，可以提供多种多样的所有权模式

1.比如双人共有域名，要两人签名才可以转让域名或者更改注册器
2.比如多人共有域名，超过50%人签名才可以转让域名或者更改注册器

如果域名的所有者是一个账户地址，那么用户可以调用注册器的接口管理二级域名。

2.2.3 注册器

（ens的设计是拥有域名的智能合约叫做注册器，实际上注册器只是owner的一个特例，我们将域名的所有者和注册器分开了，这个系统会变得更加清晰。大部分用户并不会去卖自己的二级域名，所以大部分用户无需配置注册器，配置解析器即可）

注册器专门负责将一个域名的子域名重新分配给其他所有者。注册器会调用域名根脚本进行操作。 域名根会检查注册器是否有权限操作此域名。 注册器有两个功能

1.将一个域名的子域名重新分配给其他所有者。
2.查询一个子域名的拥有者是否合法，因为存在三级域名卖掉了，然后二级域名转让给别人这种情况。所以在做完整解析的情况下，解析过程会询问注册器，他的下级域名是不是分配给了指定的所有者，如果没有，则此解析无效。

注册器是一个智能合约，可以有不同种类的注册器。

1.先到先得注册器，大家可以自由抢域名。测试网.test后缀域名将采用先到先得注册方式。

2.管理员手动分配注册器，由一个管理员来设置将子域名的所有权如何处理。通常情况下，个人持有的二级域名会通过手动方式分配子域名。

3. 拍卖注册器。测试网.neo后缀的域名及主网.neo后缀域名都会通过抵押拍卖的方式注册。

2.2.4 解析器

NNS最主要的功能，就是完成从域名到解析器的映射。 解析器是一个智能合约，他来完成实际将名字翻译成地址的实际过程。

只要遵循NNS解析器规范的智能合约就可以被配置为解析器。NNS会提供通用的解析器。

如果要增加新的协议类型，在对现有NNS规范没有颠覆性改变的情况下，都可以不需改动NNS系统，直接配置实现。

2.2.5 解析规则

2.2.5.1 域名的存储

NNS中存储的域名为32字节散列值，而不是域名原文的文本。这有几个设计原因：

1.处理过程统一，允许任意长度的域名。

2. 一定程度保留了域名的隐私

3. 将域名转换为散列的算法称为NameHash，我们将在其他的文档资料中对他进行解释。NameHash的定义方式为递归式

```
比如aaa.neo 对应的
hashA = hash256(hash256(".neo") + "aaa")
然后 bbb.aaa.neo对应的
hashB = hash256(hashA+"bbb")
那么 ccc.bbb.aaa.neo 对应的
HashC = hash256(hashB+"ccc")
```

这样的定义方式让我们可以将所有层次的域名，一级，二级到无数级，都扁平化地保存在一个Map<hash256, 解析器> 的数据结构中。

这正是注册器保存域名解析的方法

这个递归计算NameHash的方式，可以用一个函数表达Hash=NameHash(“xxx.xxx.xxx…”);
NameHash实现方法请参考3.2章节。

2.2.5.2 解析过程

用户调用根域名的解析函数进行解析，根域名提供完整和快速两种解析方式。可根据需要调用，也可以直接查询解析器，自行调用。

快速解析方式

快速方式域名根直接查表完整域名的解析器，如果没有，查询父域名的解析器。然后调用解析器解析。

快速方式运算次数少，但可能存在一个漏洞，即为三级域名卖给了别人，解析器存在，但是二级域名已经转让的情况。 此时依然可以正常解析

完整解析方式

完整方式，域名根将从根域名开始，逐层检查所有权和TTL，如果不符合将失败。
运算次数较多，与域名级数线性增长。

2.3 经济模型—无锁定可循环分配令牌NNC

NNS系统会发行一种内置令牌币NEO Name Credit，简称为NNC，
NNC有三个作用：

a) 用于抵押竞拍时的抵押资产。.neo后缀的域名将采用竞拍的方式分配，在投标过程中，谁出具的NNC多，谁将获取域名所有权，投标使用的NNC暂时被锁定，所有权属于域名的所有者，意味着被锁定的NNC会随着域名所有权的转移而转移。
b) 用于域名租金支付。由于竞拍域名只是锁定了NNC，失去了NNC流动性，并不会有其他

损失，因此为了防止投机者恶意竞拍推高域名价格，需要引入租金机制予以调节，即每年或其他固定时间），对域名收取一定租金，作为域名的使用成本。我们会首先开放5个字符以上的二级域名，这时候不会收租金，等到5个字符以下的高价值域名完全开放后会考虑引入租金机制。

c) 系统收入重新分配。在竞拍过程中，系统会收取一定的手续费以防止恶意竞拍，同时系统也会有租金收入，这些收入最终都会根据NNC的持有量按比例返还给NNC持有者。为方便地实现系统收入的重新分配，我们为NEP5令牌增加了币天的概念，NNC令牌持有人只需要隔一段时间手动领一次奖励即可，从而实现NNC令牌的无锁定循环分配。

NNC的发行数量为10亿个，分配比例会在以后的版本中说明。

2.4 域名浏览器

NNS域名浏览器是提供NNS域名查询，拍卖，转让等功能的入口。

2.5 反向解析

NNS将支持反向解析，反向解析将称为验证地址、验证智能合约的一个有效手段。

2.6 路线图

2018年1季度

- 2018.1 正式发布NNS技术白皮书
- 2018.1 完成技术原理测试和验证
- 2018.1.31 在测试网发布包括注册器、解析器的NNS第一阶段测试服务，任何人可以注册未被注册且符合规则的域名

2018年2季度

- 2018.3 发布基于测试网的域名浏览器V1
- 2018.3 在测试网发行NNC
- 2018.4 在测试网发布包含竞标服务的NNS第二阶段测试服务，任何人可以向NEL申请NNC进行竞标测试域名

- 2018.5 发布基于测试网的域名浏览器V2
- 2018.5 在主网发行NNC

2018年3季度

- 2018.6 在主网上发布NNS合约，开放5个字符以上的.neo后缀域名，Neo域名时代来临
- 2018.6 发布基于正式网的域名浏览器

2018年4季度

- 实现域名交易

2019年

- 实现租金机制
- 完全开放.neo后缀域名

03

详细设计

3.1 NNS协议规范

▼协议

通常我们在互联网上使用的url如下

```
http://aaa.bbb.test/xxx
```

其中

1. http是协议(protocol)，NNS服务请求时会把域名和协议分开传递.
2. aaa.bbb.test是域名，NNS服务请求时使用域名的hash
3. xxx是路径，路径不是在dns的层次处理，对于nns也一样，如果有路径，交由其他的方式处理

▼NNS的协议使用字符串定义

以下均为暂定

http协议

```
http协议指向一个string,表示一个互联网地址
```

addr协议

```
addr协议指向一个string，表示一个NEO address，形如
AdzQq1DmnHq86yyDUkU3jKdHwLUe2MLAVv
```

script协议

```
script协议指向一个byte[],表示一个NEO ScriptHash，形如
0xf3b1c99910babe5c23d0b4fd0104ee84ffeec2a5
```

同一域名的不同协议做不同处理

```
http://abc.test 可以指向 http://www.163.com
addr://abc.test 可以指向 AdzQq1DmnHq86yyDUkU3jKdHwLUe2MLAVv
script://abc.test 可以指向 0xf3b1c99910babe5c23d0b4fd0104ee84ffeec2a5
```

3.2 NameHash算法详解

▼Namehash

NNS中存储的域名为32字节散列值，而不是域名原文的文本。这有几个设计原因：

1. 处理过程统一，允许任意长度的域名。
2. 一定程度保留了域名的隐私
3. 将域名转换为散列的算法称为NameHash

▼域名domain与名字数组domainarray 与协议

通常我们在互联网上使用的url如下

```
http://aaa.bbb.test/xxx
```

其中

1.http是协议(protocol)，NNS服务请求时会把域名和协议分开传递.
2.aaa.bbb.test是域名，NNS服务请求时使用域名的hash
3.xxx是路径，路径不是在dns的层次处理，对于nns也一样，如果有路径，交由其他的方式处理

NNS服务使用的不是域名，而是域名的名字数组，这样处理起来更加直接

域名 aaa.bb.test 转成字节数组就是["test","bb","aa"]

你可以这样调用解析

NNS.ResolveFull("http",["test","bb","aa"]);
交由合约去计算出namehash

▼NameHash算法

NameHash算法是将域名转成DomainArray以后，逐级连接计算hash的方法，代码如下

```
//域名转hash算法
static byte[] nameHash(string domain)
{
    return SmartContract.Sha256(domain.AsByteArray());
```

```
}
static byte[] nameHashSub(byte[] roothash, string subdomain)
{
    var domain = SmartContract.Sha256(subdomain.AsByteArray()).Concat(roothash);
    return SmartContract.Sha256(domain);
}
static byte[] nameHashArray(string[] domainarray)
{
    byte[] hash = nameHash(domainarray[0]);
    for (var i = 1; i < domainarray.Length; i++)
    {
        hash = nameHashSub(hash, domainarray[i]);
    }
    return hash;
}
```

▼快速解析

完整的解析传入整个DomainArray,由智能合约去逐个检查一层层解析。计算NameHash的过程也可以挪到客户端算好，再传入智能合约。 调用方式如下

```
//查询 http://aaa.bbb.test
var hash = nameHashArray(["test","bbb"]);//可以客户端计算
NNS.Resolve("http",hash,"aaa");//调用智能合约
```

或者

```
//查询 http://bbb.test
var hash = nameHashArray(["test","bbb"]);//可以客户端计算
NNS.Resolve("http",hash,"");//调用智能合约
```

你也许会考虑查询 aaa.bbb.test 的过程为什么不是这样

```
//查询 http://aaa.bbb.test
var hash = nameHashArray(["test","bbb","aaa"]);//可以客户端计算
```

```
NNS.Resolve("http",hash,"");//调用智能合约
```

我们要考虑aaa.bb.test 是否拥有一个独立的解析器，如果aaa.bb.test被卖给了别人，他指定了一个独立的解析器，这样是可以查询到的。 如果aaa.bb.test 并没有独立的解析器，他是有bb.test的解析器来解析。 那么这样就无法查询到

而采用第一种查询方式，无论aaa.bb.test 是否拥有一个独立的解析器，都可以查询到。

3.3 顶级域名合约详解

▼顶级域名合约的函数签名

函数签名如下

```
public static object Main(string method, object[] args)
```

部署时采用 参数 0710，返回值 05 的配置

▼顶级域名合约的接口

顶级域名的接口分为三部分

一种为 通用接口，不需要权限验证，所有人都可以调用

一种为 所有者接口，仅接受所有者签名，或者由所有者脚本调用有效

一种为 注册器接口，仅接受由注册器脚本调用有效

▼通用接口

通用接口，不需要权限验证。 代码如下

```
if (method == "rootName")
  return rootName();
if (method == "rootNameHash")
  return rootNameHash();
if (method == "getInfo")
  return getInfo((byte[])args[0]);
if (method == "nameHash")
```

```
    return nameHash((string)args[0]);
if (method == "nameHashSub")
    return nameHashSub((byte[])args[0], (string)args[1]);
if (method == "nameHashArray")
    return nameHashArray((string[])args[0]);
if (method == "resolve")
    return resolve((string)args[0], (byte[])args[1], (string)args[2]);
if (method == "resolveFull")
    return resolveFull((string)args[0], (string[])args[1]);
```

rootName

返回当前顶级域名合约对应的根域名 返回值为string

rootNameHash

返回当前顶级域名合约对应的NameHash 返回值为byte[]

getInfo(byte[] namehash)

返回一个域名的信息 返回值为一个如下数组

```
[
    byte[] owner//所有者
    byte[] register//注册器
    byte[] resolver//解析器
    BigInteger ttl//到期时间
]
```

nameHash(string domain)

将域名的一节转换为NameHash 比如

```
nameHash("test")
nameHash("abc")
```

返回值为byte[]

nameHashSub(byte[] domainhash,string subdomain)

计算子域名的NameHash, 比如

```
var hash = nameHash("test");
var hashsub = nameHashSub(hash,"abc")//计算abc.test 的namehash
```

返回值为byte[]

nameHashArray(string[] nameArray)

计算NameArray的NameHash，aa.bb.cc.test 对应的nameArray是

["test","cc","bb","aa"]

```
var hash = nameHashArray(["test","cc","bb","aa"]);
```

resolve(string protocol,byte[] hash,string or int(0) subdomain)

解析域名

```
第一个参数为协议
    比如http是将域名映射为一个网络地址
    比如addr是将域名映射为一个NEO地址（这可能是最常用的映射）
第二个参数为要解析的域名Hash
第三个参数为要解析的子域名Name
```

应用代码如下

```
var hash = nameHashArray(["test","cc","bb","aa"]);//客户端计算好
resolve("http",hash,0)//合约解析 http://aa.bb.cc.test

or

var hash = nameHashArray(["test","cc","bb");//客户端计算好
resolve("http",hash,“aa")//合约解析 http://aa.bb.cc.test
```

返回类型为byte[]，具体byte[]如何解读，由不同的协议定义，addr协议，byte[]就存的字符串。 协议约定另外撰文。

除了二级域名的解析，必须使用resolve("http",hash,0)的方式，其余的解析建议都使用resolve("http",hash,“aa")的方式。

resolveFull(string protocol,string[] nameArray)

解析域名，完整模式

第一个参数为协议
第二个参数为NameArray

这种解析方式唯一的不同就是会逐级验证一下所有权是否和登记的一致，一般用resolve即可

返回类型同resolve

▼所有者接口

所有者接口全部为 owner_SetXXX(byte[] srcowner,byte[] nnshash,byte[] xxx)的形式。xxx 均是scripthash。

返回值均为 一个byte array [0] 表示失败 [1] 表示成功

所有者接口均接受账户地址直接签名调用和智能合约所有者调用。

如果所有者是智能合约，那么所有者应该自己判断权限，不满足条件，不要发起对顶级域名合约的appcall

▼owner_SetOwner(byte[] srcowner,byte[] nnshash,byte[] newowner)

转让域名所有权，域名的所有者可以是一个账户地址，也可以是一个智能合约。

srcowner 仅在 所有者是账户地址时用来验证签名，他是地址的scripthash
nnshash 是要操作的域名namehash
newowner 是新的所有者的地址的scripthash

▼owner_SetRegister(byte[] srcowner,byte[] nnshash,byte[] newregister)

设置域名注册器合约（域名注册器为一个智能合约）

域名注册器参数形式必须也是0710，返回05

必须实现如下接口

```
public static object Main(string method, object[] args)
```

```
{
    if (method == "getSubOwner")
        return getSubOwner((byte[])args[0], (string)args[1]);
    ...

getSubOwner(byte[] nnshash,string subdomain)
```

任何人可调用注册器的接口检查子域的所有者

对于域名注册器的其他接口形式不做规范，官方提供的注册器会另外撰文说明。

用户自己实现的域名注册器，仅需实现getSubOwner接口

▼owner_SetResolve(byte[] srcowner,byte[] nnshash,byte[] newresolver)

设置域名解析器合约（域名解析器为一个智能合约）

域名解析器参数形式必须也是0710，返回05

必须实现如下接口

```
public static byte[] Main(string method, object[] args)
{
    if (method == "resolve")
        return resolve((string)args[0], (byte[])args[1]);
    ...

resolve(string protocol,byte[] nnshash)
```

任何人可调用解析器接口进行解析

对于域名解析器的其它接口形式不做规范，官方提供的解析器会另外撰文说明。

用户自己实现的域名解析器，仅需实现resolve 接口

▼注册器接口

注册器接口由注册器智能合约进行调用，只有一个

▼register_SetSubdomainOwner(byte[] nnshash,string subdomain,-byte[] newowner,BigInteger ttl)

注册一个子域名

nnshash 是要操作的域名namehash
subdomain 是要操作的子域名
newowner 是新的所有者的地址的scripthash
ttl 是域名过期时间（区块高度）
成功返回 [1] ,失败返回 [0]

3.4 所有者合约详解

▼所有者合约工作方式

所有者合约采用Appcall的形式调用顶级域名合约的owner_SetXXX 接口

```
[Appcall("dffbdd534a41dd4c56ba5ccba9dfaaf4f84e1362")]
static extern object rootCall(string method, object[] arr);
```

顶级域名合约会检查调用栈，取出调用它的合约和顶级域名合约管理的所有者进行比对。

所以只有所有者合约可以实现管理

▼所有者合约的意义

用户可以用所有者合约实现复杂的合约所有权模式

比如：

双人共有，双人签名操作。

多人共有，投票操作。

3.5 注册器合约详解

▼注册器合约工作方式

注册合约采用Appcall的形式调用顶级域名合约的register_SetSubdomainOwner接口

```
[Appcall("dffbdd534a41dd4c56ba5ccba9dfaaf4f84e1362")]
static extern object rootCall(string method, object[] arr);
```

顶级域名合约会检查调用栈，取出调用它的合约和顶级域名合约管理的注册器进行比对。

所以只有指定的注册器合约可以实现管理。

▼注册器接口

注册器参数形式必须也是0710，返回05

```
public static object Main(string method, object[] args)
{
    if (method == "getSubOwner")
        return getSubOwner((byte[])args[0], (string)args[1]);
    if (method == "requestSubDomain")
        return requestSubDomain((byte[])args[0], (byte[])args[1], (string)args[2]);
    ...
```

▼getSubOwner(byte[] nnshash,string subdomain)

```
此接口为注册器规范要求，必须实现，完整解析域名时会调用此接口验证权利
nnshash 为域名hash
subdomain 为子域名
返回 byte[] 所有者地址，或者空
```

▼requestSubDomain(byte[] who,byte[] nnshash,string subdomain)

此接口为演示的先到先得注册器使用，用户调用注册器的这个接口申请域名

who谁在申请

nnshash 申请哪个域名

subdomain 申请的子域名

3.6 解析器合约详解

▼解析器合约工作方式

1.解析器自己保存解析信息

2.顶级域名合约会以nep4的方式调用解析器的解析接口获取解析信息。

3.解析器设置解析数据时采用Appcall的形式调用顶级域名合约的getInfo接口来验证域名所有权

```
[Appcall("dffbdd534a41dd4c56ba5ccba9dfaaf4f84e1362")]
static extern object rootCall(string method, object[] arr);
```

任何合约都可以通过 appcall 顶级域名合约getInfo接口的方式来验证域名所有权

▼解析器接口

注册器参数形式必须也是0710，返回05

```
public static byte[] Main(string method, object[] args)
{
    if (method == "resolve")
        return resolve((string)args[0], (byte[])args[1]);
    if (method == "setResolveData")
                return setResolveData((byte[])args[0], (byte[])args[1], (string)args[2],
(string)args[3], (byte[])args[4]);
    ...
```

▼resolve(string protocol,byte[] nnshash)

此接口为解析器规范要求，必须实现，完整解析域名时会调用此接口最终解析

protocol为协议字符串

nnshash为解析的域名

返回byte[] 解析数据

▼setResolveData(byte[] owner,byte[] nnshash,string or int[0] sub-domain,string protocol,byte[] data)

此接口为演示的标准解析器所有，所有者（目前还只支持账户地址所有者）可以调用此接口配置解析数据

owner 所有者

nnshash 设置哪个域名

subdomain 设置的子域名（可以传0，如果设置的就是域名解析，非子域名传0）

protocol 协议字符串

data 解析数据

返回[1]表示成功 或者[0]表示失败

3.7 投标竞拍域名注册方式详解

▼竞标服务

竞标服务以确定谁有权注册某一个二级域名为服务目标。服务进程分为四个阶段：开标、投标、揭标、中标。

▼开标

任意未被注册或已过期且不违反域定义的域名都可被任意标准地址（账户）申请开标。一旦开标立即进入投标阶段。

▼投标

投标由开标启动，为期72小时，在这段时间内任意标准地址（账户）可以提交一个加密的报价，并支付NNC保证金。投标人通过发送NNC同时附加其自定义的一组8位任意字符的sha256散列值作为报价用以隐藏真实报价，以防止没有必要的恶性竞争。投标人不足1人，竞标自动结束，域名立即进入可开标状态。

▼揭标

投标过程结束后进入48小时的揭标过程。在此期间，投标人需要提交报价的明文和加密字符串明文，用以验证揭标人的真正出价。揭标后，保证金会扣除部分系统费用后返还。如果投标人没有揭标，视为放弃竞标。揭标人不足1人，竞标自动结束，域名立即进入可开标状态。

▼中标

揭标结束后，中标人需要通过一笔交易正式获得域名的所有权。

关于投标竞拍模式的域名分配规则会在以后版本进一步细化。

▼交易服务

交易服务支持域名登记员发布域名所有权转让邀约，支持固定价格转让和荷兰式减价拍卖方式转让。

3.8 无锁定可循环分配令牌NNC技术实现

NNS的经济系统需要一种资产，故我们设计了一种资产。

NNS的经济系统要求资产总量不变，且拍卖所得和租金消耗均视为销毁。故我们设计的资产具有消耗功能，消耗资产会重新分配。因为销毁和重分配会循环进行，因此我们称为可循环分配令牌。无锁定指的是分配过程不会造成用户资产锁定，下文详述。

▼重新分配机制

我们为令牌使用了销毁接口，主要的销毁方式为

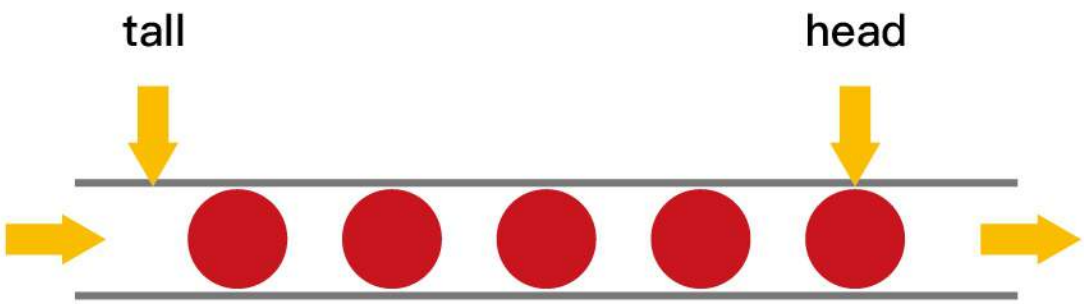
1. 支付租金，系统销毁
2. 二级域名拍卖所得，系统销毁
3. 任何人想要销毁他自己的部分令牌，系统销毁

一旦销毁，就计入销毁池，销毁池的资产会再进入奖池，用户可从奖池中领走资产。

▼无锁定的领奖机制

通常考虑这样的系统，一般如拍卖将系统分为开标、投标、揭标、中标四个阶段，用户资金在投标阶段必须支付入系统,意味锁定，中标后被消耗，或者流标退还，意味解锁。

而NNC令牌，并不划分为：参加领奖等待开奖，领奖几个阶段，不涉及用户资产的锁定。



NNC使用奖池队列的机制，如图，只保持固定数量的奖池（比如五个）

产生超过五个奖池时，最早的奖池被销毁。

配合奖池机制再将用户资产分为两个部分：固定资产和零钱，其中固定资产的持有时间只能增加，而只有固定资产持有时间小于奖池领奖时间才能领奖的模式。

领奖后固定资产持有时间增加，犹如币天被消耗掉，用这种方式防止重复领取。

▼奖池细节

令牌会维持几个奖池，每个奖池产生时把销毁池里的资产全部移入奖池，如果超过最大奖池数，把最早的奖池也销毁，把销毁奖池里剩余的资产也计入新奖池。

奖池数量为确定值，比如每4096块产生一个奖池，最多维持五个奖池，产生第六个奖池时，销毁第一个奖池，并把他的资产也全部放入最新的奖池。（以上数值几个奖池，多久产生一个奖池均为暂定）

每个奖池会对应一个块，这个块就是领奖时间，只有持币时间早于领奖时间，才可以领奖。

▼固定资产和零钱的细节

固定资产和零钱，其中固定资产记录一个持有时间。

固定资产和零钱只影响领奖数额，对其余功能没有影响。

固定资产+零钱=用户的资产余额

固定资产没有小数部分，小数部分均计入零钱，下文提到视为固定资产均表示将整数部分视为固定资产，小数部分计入零钱。不再重复说明，统称“视为固定资产”。

用户转账，优先使用零钱，不足部分使用固定资产。

用户转账转出方：固定资产只能减少。

用户转账转入方：固定资产不变，转入金额全部计入零钱。

只有两种方法使固定资产增加。

1. 创建账户（向一个没有nnc令牌的地址转入视为创建账户）
   转入的资产视为固定资产，持有时间为转入块。
2. 领奖
   领奖后将个人资产包括领奖所得，视为固定资产，持有时间为领奖块。

然后用户领奖时只能逐个奖池领取自己固定时间<奖池的部分

领取比率算法为 当前奖池数额/（总发行量-当前奖池数额）

用户在每个奖池里的可领数额为：自己的固定资产*当前奖池领取比率。

以数字来说明，现有奖池3个，分别是 4096，8000，10000块产生的。用户的固定资产100，持有时间7000。则他不能领第一个奖池，只能领第二个和第三个奖池的资金。当前块10500，一旦用户领奖，他的资产持有时间就变成10500，哪个奖池他都不能领了。

比如有一个奖池，内有资金五千三十万，他的领取比率就是50300000/（1000000000-503000000）=0.123587223，用户有100固定资产，则能在此奖池内领取12.3587223个NNC。

▼NNC接口（仅述相比NEP5多出来的接口）

NNC首先符合NEP5规范，NEP规范接口不再赘述

·balanceOfDetail(byte[] address)

返回用户资产细节，有多少固定资产，有多少零钱，总额，固定资产持有块，无需签名，任何人可查。

返回结构体

{

现金数额

固定资产数额

固定资产持有块

余额（固定资产+现金）

}

·use(byte[] address,BigInteger value)

消耗一个账户的若干资产，需账户签名

消耗资产即进入奖池

·getBonus(byte[] address)

指定账户领奖，需账户签名。

领奖后会将该账户总资产视为固定资产并变动该账户的固定资产持有块。

·checkBonus()

检查当前奖池，无需签名

返回Array<BonusInfo>

BonusInfo

{

StartBlock;//领奖块

BonusCount;//该奖池总量

BonusValue;//该奖池剩余

LastIndex;//上一个bonus的id

}

·newBonus ()

产生新奖池，任何人均可调用，但是奖池产生要符合奖池间隔，所以重复调用并无作用。此接口可视为督促产生新奖池。

04

总结

总结

NNS项目是一个完全构建在NEO区块链上的一个智能合约协议层应用，是一个真正意义上的区块链落地应用。NNS的所有服务都是由智能合约提供的，是分布式并且灵活可扩展的，不存在中心化风险。NNS是一次NEO智能合约体系的大规模应用，为了实现解析器的灵活可扩展，我们应用了最新的NEP4动态调用，在经济模型中，我们会设计Vickrey拍卖合约和荷兰式拍卖合约，为了实现系统费用更易重新分配，我们扩展了NEP5令牌标准，增加了币天的概念，从而实现无需锁定令牌就可以实现系统收入的重新分配，将应用令牌和股权融合成一个令牌。

域名服务能够提升区块链的易用性，并且具有丰富的使用场景，会形成围绕域名的生态系统。未来，我们会和NEO生态的客户端合作，让所有的NEO钱包支持通过别名转账，我们也会探索一些新的使用场景，例如和宠物游戏合作，通过NNS为宠物起一个昵称。未来，随着NEO生态的应用越来越多，NNS的域名也会越来越有价值。

05

参考资料

[1]. NEO White Paper: Superconducting Exchange. https://github.com/neo-project/docs/blob/e01d268426a8b5f9b3676cfd03d0b8b83d7711a1/en-us/white-paper.md#highly-scalable-architecture-design, Accessed 2018.

[2]. NEO NEP-5: Token Standard. https://github.com/neo-project/proposals/blob/master/nep-5.mediawiki, Accessed 2018.

[3]. ENS. https://github.com/ethereum/ens,Accessed 2018.

[4]. EIP137 – Ethereum Name Service. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-137.md, Accessed 2018.

[5]. EIP162 – Initial ENS Registrar Specification. https://github.com/ethereum/EIPs/issues/162, Accessed 2018.

NNS

NNS联系方式
邮箱：contact@neons.name
官网：neons.name

Twitter：@NewEconoLab
QQ群：702673768
公众号：NEL新经济实验室