ENS源码分析

ENS (https://learnblockchain.cn/tags/ENS)
以太坊 (https://learnblockchain.cn/tags/%E4%BB%A5%E5%A4%AA%E5%9D%8A)
DApp (https://learnblockchain.cn/tags/DApp)

> 2021年ENS (https://learnblockchain.cn/article/2487)大火，有很多用户用户赚了不菲的空投，甚至部分用户赚了上千万。但是ENS到底是怎么实现的？技术细节有什么？目前笔者在中文网站暂未发现从技术角度进行全面讲解的...

前言

2021年ENS大火，有很多用户用户赚了不菲的空投，甚至部分用户赚了上千万。

但是ENS到底是怎么实现的？技术细节有什么？

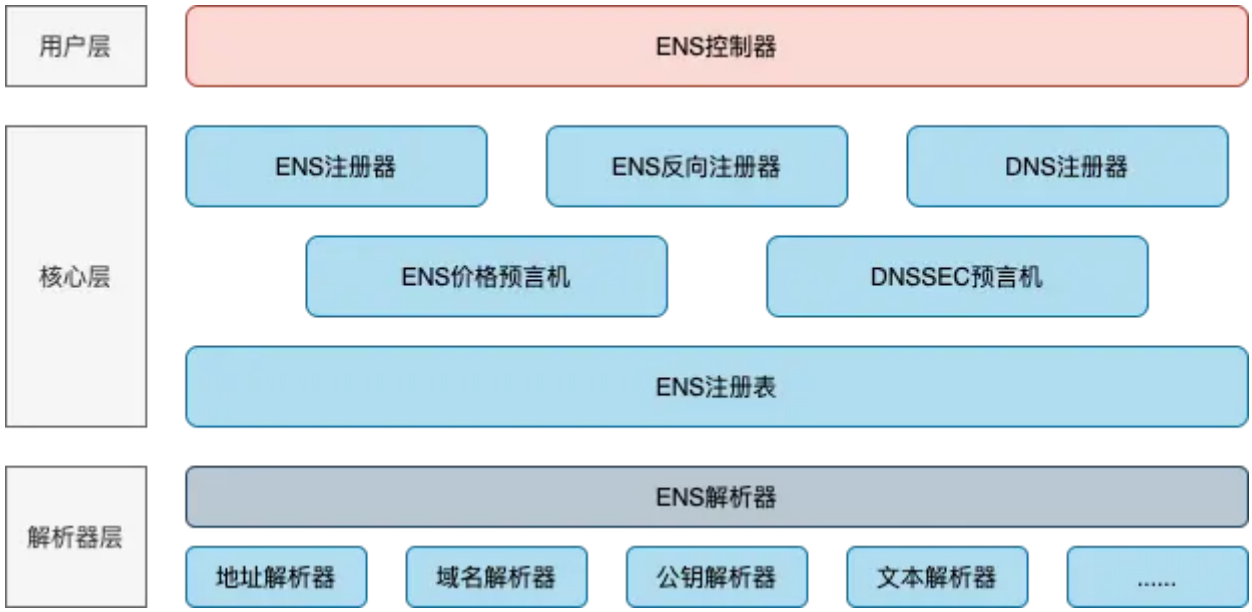
目前笔者在中文网站暂未发现从技术角度进行全面讲解的文章，因此尝试从源码的角度来分析下。

以下仅代表自身感悟，如有错漏之处还请指正。

注：本文所参考的合约地址为ens-contracts (https://github.com/ensdomains/ens-contracts/tree/fb88681d476e8cabc642781262043b8d521fefd5)

概述

ens的整个技术架构大致类似于下图

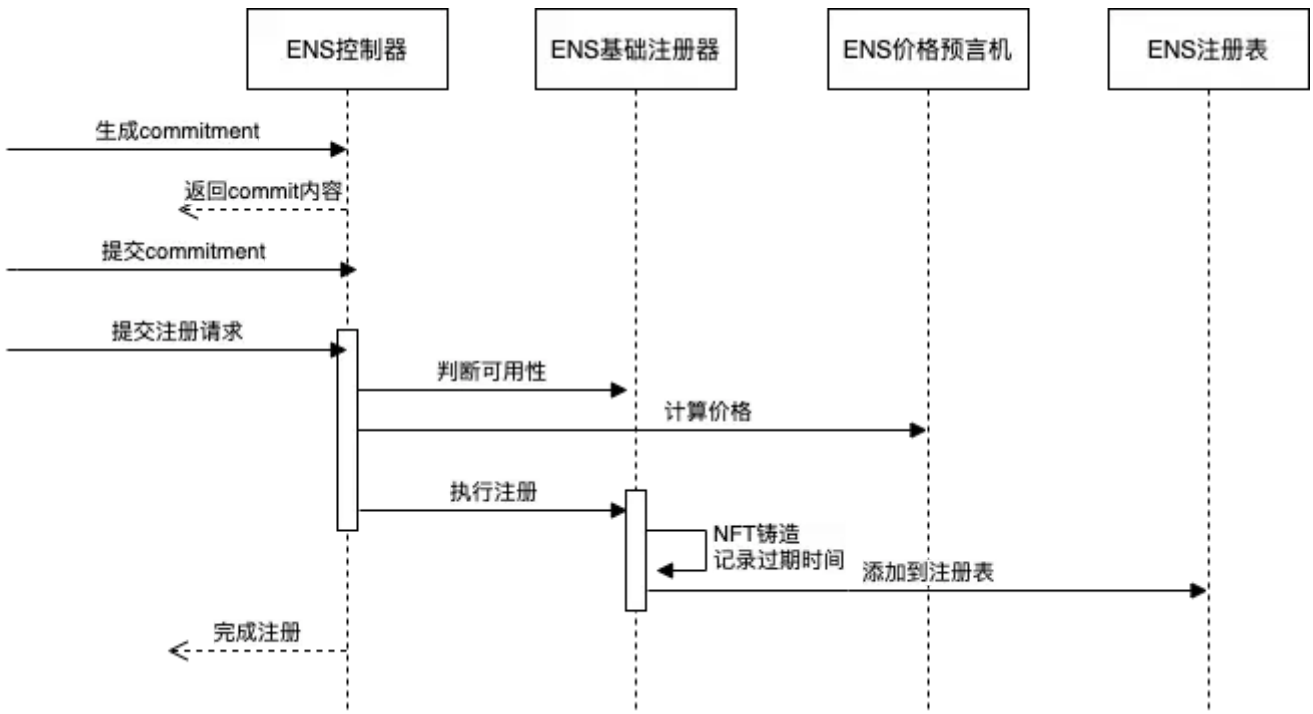


如图所示，ens可以从功能上进行以下划分：

1. 用户层：注册ens的入口，通过外观模式，对请求进行相关校验，转发到下层进行实际处理
2. 核心层：ens的核心功能模块，包括ens的注册表（ens和owner对应关系）、ens注册器（注册一个实际的ens域名）、ens反向注册器（通过address反向获取ens域名）、ens注册价格计算、dns注册关联等功能
3. 解析器层：解析ens域名的实际处理层，支持将ens解析为地址、公钥、文本等。

注册

注册各个模块交互流程图如下：



为了防止域名抢注情况，ens使用了『请求–提交』二阶段注册模式。

用户在申请域名时，首先根据『待申请域名』和『秘密值（随机数）』生成commitment，然后提交至ENS控制器。

在一分钟以后，将域名的『注册请求』和『秘密值（随机数）』一起提交到控制器，完成ens域名的注册。

commitment处理

生成commitment是一个计算哈希的过程，核心实现如下代码：

```
// 创建commitment
    function makeCommitment(string memory name, address owner, bytes32 secret) pure
public returns(bytes32) {
        return makeCommitmentWithConfig(name, owner, secret, address(0), address(0));
    }

    function makeCommitmentWithConfig(string memory name, address owner, bytes32 secret,
address resolver, address addr) pure public returns(bytes32) {
                // 计算ens域名（不带.eth）的哈希
        bytes32 label = keccak256(bytes(name));
        if (resolver == address(0) && addr == address(0)) {
            // 将ens域名（不带.eth）哈希值、域名申请者地址、秘密值（随机数）组合计算哈希
            return keccak256(abi.encodePacked(label, owner, secret));
        }
        require(resolver != address(0));
        return keccak256(abi.encodePacked(label, owner, resolver, addr, secret));
    }
    // commit 过程
    function commit(bytes32 commitment) public {
        // maxCommitmentAge commitment最大有效时间，是24h
                // 如果之前已经存在过该commitment，需要保证前commitment已超过最大有效期，即24小时
                // 如果不存在该commitment，则该要求必定满足
        require(commitments[commitment] + maxCommitmentAge < block.timestamp);
        commitments[commitment] = block.timestamp;
    }
```

即生成commitment，是将 `hash(ens name)` 、 `address` 、 `secret` 组合后求哈希，结果中包含了ens name和secret，也不会泄露原始信息。

然后用户执行申请过程，就是将commitment记录到区块链的过程。

验证commitment主要包含两个操作：

1. 根据提交的secret判断commitment是否正确
2. commitment提交时间在预期范围内

```
// 根据提交的secret计算commitment
bytes32 commitment = makeCommitmentWithConfig(name, owner, secret, resolver, addr);
// 消耗commitment，也就是验证commitment逻辑
uint cost = _consumeCommitment(name, duration, commitment);
...
function _consumeCommitment(string memory name, uint duration, bytes32 commitment)
internal returns (uint256) {

  // minCommitmentAge commitment最短有效时间，是1min
  // 如果刚提交完commitment就执行注册，该值会大于当前区块时间，不能注册通过
  require(commitments[commitment] + minCommitmentAge <= block.timestamp);

  // maxCommitmentAge commitment最大有效时间，是24h
  // 如果该commitment提交时间过早，该值会小于当前区块时间，不能注册通过
  require(commitments[commitment] + maxCommitmentAge > block.timestamp);
  // 保证该域名可用：1. 域名长度大于3；2. 该域名还未被注册或已超出保留时间（90天）
  require(available(name));
  // 验证通过，删除该commitment
        delete(commitments[commitment]);
  ...
}
```

价格计算

注册ens域名包含不同的价格，目前在StablePriceOracle定义中，不同的域名长度价格不同。

```
function price(string calldata name, uint expires, uint duration) external view override
returns(uint) {
        // 计算待注册域名成都
        uint len = name.strlen();
        if(len > rentPrices.length) {
            len = rentPrices.length;
        }
        require(len > 0);
        // 计算域名注册时长*域名单价
        uint basePrice = rentPrices[len - 1].mul(duration);
        // 域名附加费用_premium价格目前定义为0
        basePrice = basePrice.add(_premium(name, expires, duration));
                                // 将价格转换为eth价格
        return attoUSDToWei(basePrice);
    }
    function attoUSDToWei(uint amount) internal view returns(uint) {
                // 通过预言机获取最新的eth/usd价格
        uint ethPrice = uint(usdOracle.latestAnswer());
        // 计算应当支付多少eth
        return amount.mul(1e8).div(ethPrice);
    }
```

可以发现，价格计算较为简单，

1. 获取待注册的域名长度，进而计算一共需要支付多少usd
2. 通过预言机合约，获取当前的eth/usd价格汇率
3. 计算当前需要支付多少eth

注册

注册包含两种类型：1. 为ens域名设置解析器；2. 使用默认的解析器

```
// 全新注册入口
    function register(string calldata name, address owner, uint duration, bytes32 secret)
external payable {
      registerWithConfig(name, owner, duration, secret, address(0), address(0));
    }
    // 具体注册逻辑
    function registerWithConfig(string memory name, address owner, uint duration, bytes32
secret, address resolver, address addr) public payable {
        // 校验commitment逻辑
        bytes32 commitment = makeCommitmentWithConfig(name, owner, secret, resolver,
addr);
        uint cost = _consumeCommitment(name, duration, commitment);
                                // 计算域名的哈希
        bytes32 label = keccak256(bytes(name));
        // 计算tokenId，方便铸造nft
        uint256 tokenId = uint256(label);

        uint expires;
        // 如果要设置新的解析器，执行设定解析器的逻辑
        if(resolver != address(0)) {
            // 临时设置域名的拥有者为合约地址，方便后续设置解析器
            expires = base.register(tokenId, address(this), duration);

            // 计算域名哈希（包含.eth）
            bytes32 nodehash = keccak256(abi.encodePacked(base.baseNode(), label));

            // 设置用户的指定解析器
            base.ens().setResolver(nodehash, resolver);

            // 配置解析器，将ens和指定的addr对应
            if (addr != address(0)) {
                Resolver(resolver).setAddr(nodehash, addr);
            }

            // 把ens拥有权转移给用户
            base.reclaim(tokenId, owner);
            // nft转移
            base.transferFrom(address(this), owner, tokenId);
        } else {
            // 不设定解析器，使用默认的解析器
            require(addr == address(0));
            expires = base.register(tokenId, owner, duration);
        }
        // 发起事件通知
        emit NameRegistered(name, label, owner, cost, expires);

        // 钱付多了，返还多的钱
        if(msg.value > cost) {
```

```
            payable(msg.sender).transfer(msg.value - cost);
        }
    }
```

默认解析器

首先分析下使用默认解析器的逻辑：

```
// 注册逻辑入口
    function register(uint256 id, address owner, uint duration) external override
returns(uint) {
      return _register(id, owner, duration, true);
    }
    function _register(uint256 id, address owner, uint duration, bool updateRegistry)
internal live onlyController returns(uint) {
        // 需要保证该ens域名可用
        require(available(id));
        // 防止申请时间过长等导致的数据溢出
        require(block.timestamp + duration + GRACE_PERIOD > block.timestamp +
GRACE_PERIOD); // Prevent future overflow
        // 记录该域名的过期时间
        // 域名是否可用也是通过expires判断
        expiries[id] = block.timestamp + duration;
        // nft相关逻辑，之前被人拥有，重新铸造
        if(_exists(id)) {
            _burn(id);
        }
        _mint(owner, id);
        // 由于是全新注册域名，需要更新注册表
        if(updateRegistry) {
            ens.setSubnodeOwner(baseNode, bytes32(id), owner);
        }

        emit NameRegistered(id, owner, block.timestamp + duration);
        // 返回域名过期时间
        return block.timestamp + duration;
    }
```

从以上逻辑可知，注册域名的逻辑如下

1. 域名统一使用了expiries map维护，记录域名的过期时间
2. 在ens注册表中记录该域名的拥有者信息
3. 由于ens兼容ERC721，所以注册一个ens域名，也会铸造一个NFT

自定义解析器

如果用户指定了解析器，相比于使用默认的解析器，逻辑要稍微复杂一些

1. 将域名注册给合约自身，以便于有权限设置解析器
2. 合约自身作为owner，设置注册表中的解析器信息
3. 用户若指定配置解析器，则在解析器中设定ens<–>addr关系
4. 设置域名的拥有者为用户指定的owner
5. 涉及到NFT的操作，就执行NFT的转移操作

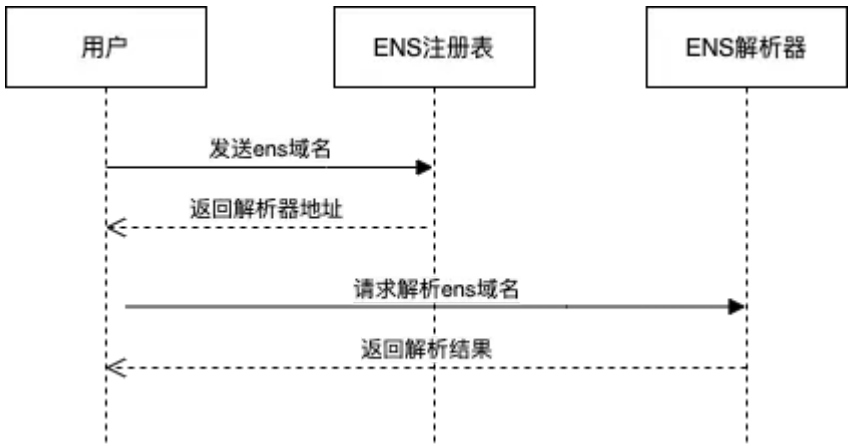
可以发现，为了实现自定义解析器，需要临时赋予合约ens拥有权，执行设置解析器的操作，执行完成后，再重新授予用户。

到此位置，注册一个ens的域名就全部执行完成。

ENS解析

正向解析

解析ens时各个模块交互如下：



可以发现，解析ens域名流程实现了注册表和解析器的解耦，注册表不维护具体内容，具体数据由解析器提供。

首先看下注册表设置和获取的核心逻辑

```
struct Record {
        address owner; // 域名拥有者
        address resolver; // 域名解析器
        uint64 ttl; // 域名解析存活时间
    }
                // ens和记录的映射关系
    mapping (bytes32 => Record) records;
    function setResolver(bytes32 node, address resolver) public virtual override
authorised(node) {
        emit NewResolver(node, resolver);
        // 将解析器添加到records记录中
        records[node].resolver = resolver;
    }
    function resolver(bytes32 node) public virtual override view returns (address) {
        // 根据ens域名，返回解析器地址
        return records[node].resolver;
    }
```

我们以解析以太坊地址为例，查看下解析逻辑

```
uint constant private COIN_TYPE_ETH = 60;
    // 记录ens域名和地址的对应关系
    mapping(bytes32=>mapping(uint=>bytes)) _addresses;
                // 设置地址
    function setAddr(bytes32 node, address a) virtual external authorised(node) {
        setAddr(node, COIN_TYPE_ETH, addressToBytes(a));
    }
    function setAddr(bytes32 node, uint coinType, bytes memory a) virtual public
authorised(node) {
        emit AddressChanged(node, coinType, a);
        if(coinType == COIN_TYPE_ETH) {
            emit AddrChanged(node, bytesToAddress(a));
        }
        // 将地址添加到address映射中
        _addresses[node][coinType] = a;
    }
    // 解析地址
    function addr(bytes32 node) virtual override public view returns (address payable) {
        bytes memory a = addr(node, COIN_TYPE_ETH);
        if(a.length == 0) {
            return payable(0);
        }
        return bytesToAddress(a);
    }
    function addr(bytes32 node, uint coinType) virtual override public view returns(bytes
memory) {
        // 读取address映射关系，获取ens命名对应的地址
        return _addresses[node][coinType];
    }
```

反向解析

反向注册器的功能是实现从以太坊地址到ens域名的解析。类似正向正向注册器支持『.eth』，反向注册器支持的是『.addr.reverse』。

理解了正向解析后，反向解析就比较容易理解：

1. msg.sender地址求hex
2. 在ens注册表中添加 `namehash(hex(msg.sender).addr.reverse)=>owner` 的管理关系，即为反向域名设置 `Record`
3. 在反向解析器中设置 `namehash=>address`

实现的代码逻辑为

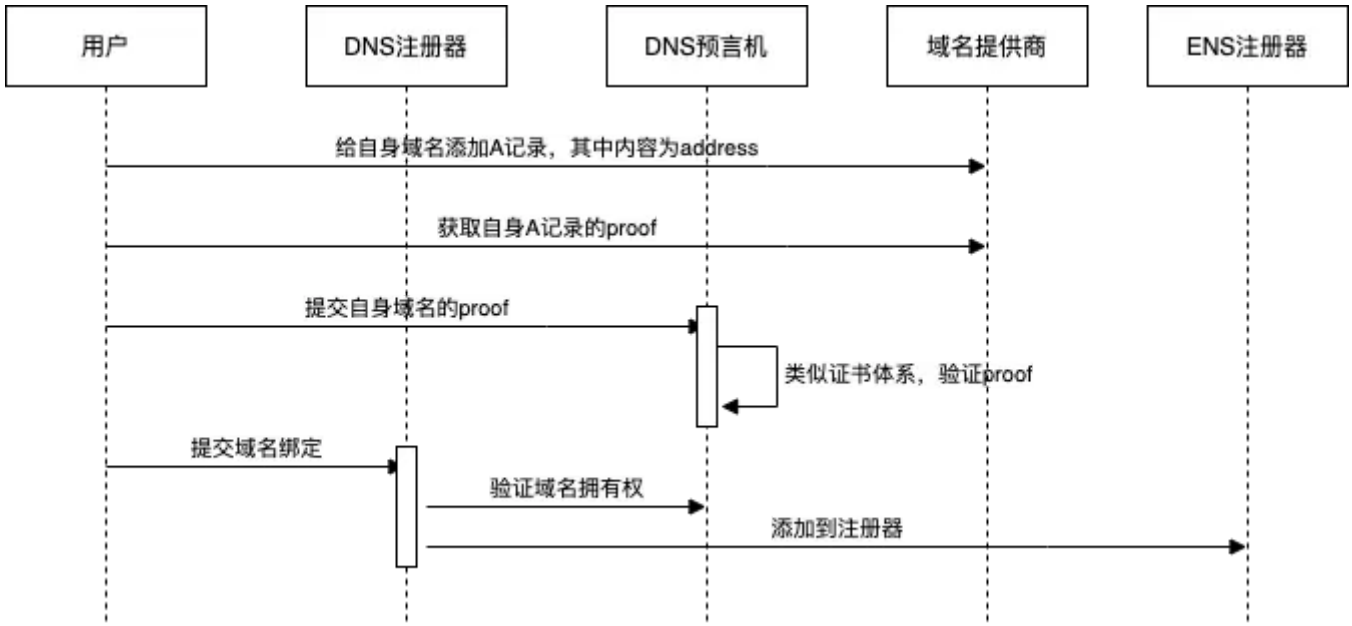
```
// 设置反向域名解析
    function setName(string memory name) public returns (bytes32) {
        // 在ens注册表中添加反向域名的record
        bytes32 node = _claimWithResolver(
            msg.sender,
            address(this),
            address(defaultResolver)
        );
        // 在反向解析器中添加反向域名到ens域名的映射关系
        defaultResolver.setName(node, name);
        return node;
    }
    function _claimWithResolver(
        address addr,
        address owner,
        address resolver
    ) internal returns (bytes32) {
        // 求address的hex编码
        bytes32 label = sha3HexAddress(addr);
        // 计算namehash
        bytes32 node = keccak256(abi.encodePacked(ADDR_REVERSE_NODE, label));
        // 获取当前address有没有设定反向解析器，如果已设置，就判断是否需要更新
        address currentResolver = ens.resolver(node);
        bool shouldUpdateResolver = (resolver != address(0x0) &&
            resolver != currentResolver);
        address newResolver = shouldUpdateResolver ? resolver : currentResolver;
        // 在ens注册表中添加对应关系
        ens.setSubnodeRecord(ADDR_REVERSE_NODE, label, owner, newResolver, 0);

        emit ReverseClaimed(addr, node);

        return node;
    }
```

DNS解析

通过前文的ens正向解析和反向解析分析可知，往注册表添加数据的关键是，**证明自身对数据的所有权**，DNS解析亦如此。

1. 正向解析修改注册表是通过**有且仅有**ENS基础注册器具有『eth』这一baseNode操作权，作为唯一eth域名分配入口，保证分配给用户的域名一定是经过系统控制的
2. 反向解析的输入数据是 `msg.sender` ，除了调用者自身，其他人都不可能给 `msg.sender` 设定反向解析记录

那么DNS解析是怎么操作的呢?



从交互图可以理解DNS解析的交互逻辑

1. 需要去域名提供商手动添加一条A记录，其中域名是 `_ens.{domain}.{suffix}` ，a记录内容是 `a=address`
2. 自身获取A记录的proof，方便DNS预言机验证
3. DNS预言机内置了根公钥和支持的部分域名的证明，按照类似于证书验证体系，即验证树的形式完成proof的验证
4. 有了证明，就可以向DNS注册器提交DNS绑定了
5. 将该DNS记录添加到ENS注册表中

相关代码如下：

```
// 提交证明
    function submitRRSets(RRSetWithSignature[] memory input, bytes calldata _proof)
public override returns (bytes memory) {
        bytes memory proof = _proof;
        for(uint i = 0; i < input.length; i++) {
            proof = _submitRRSet(input[i], proof);
        }
        return proof;
    }
    // 验证并存储证明
    function _submitRRSet(RRSetWithSignature memory input, bytes memory proof) internal
returns (bytes memory) {
        RRUtils.SignedSet memory rrset;
        // 验证证明
        rrset = validateSignedSet(input, proof);

        RRSet storage storedSet = rrsets[keccak256(rrset.name)][rrset.typeCovered];
        if (storedSet.hash != bytes20(0)) {
            // To replace an existing rrset, the signature must be at least as new
            require(RRUtils.serialNumberGte(rrset.inception, storedSet.inception));
        }
        // 存储证明
        rrsets[keccak256(rrset.name)][rrset.typeCovered] = RRSet({
            inception: rrset.inception,
            expiration: rrset.expiration,
            hash: bytes20(keccak256(rrset.data))
        });

        emit RRSetUpdated(rrset.name, rrset.data);

        return rrset.data;
    }
    // 声明域名拥有权
    function claim(bytes memory name, bytes memory proof) public override {
        // 获取要存储到注册表的数据
        (bytes32 rootNode, bytes32 labelHash, address addr) = _claim(name, proof);
        // 添加到ens注册表
        ens.setSubnodeOwner(rootNode, labelHash, addr);
    }
```

其他

1. 注册表权限是如何管理的?

注册表实现了一个 authorised 验证逻辑，只有域名拥有者或授权操作者才能执行相关写操作。

```
modifier authorised(bytes32 node) {
        address owner = records[node].owner;
        require(owner == msg.sender || operators[owner][msg.sender]);
        _;
    }
```

合约在部署时，默认赋予了合约部署者 `0x0` 的操作权限。合约部署者后续会调用 `setSubnodeOwner` 赋予特定用户操作各个域名的权限，比如 `eth`

```
// 合约部署者，提交node为0x0，label为eth，owner为指定用户，即赋予了特定用户操作eth的权限
    function setSubnodeOwner(bytes32 node, bytes32 label, address owner) public virtual
override authorised(node) returns(bytes32) {
        bytes32 subnode = keccak256(abi.encodePacked(node, label));
        _setOwner(subnode, owner);
        emit NewOwner(node, label, owner);
        return subnode;
    }
```

调用记录可查看etherscan交易(https://etherscan.io/tx/0x057a18943891fc4defd54ff6b18c4fa1e15b822f299f2f08117e4fd11d44f971)

2. 购买eth域名，默认没有配置解析器和ttl?

是的

3. ens的域名是怎么存储的?

在ens注册表中，所有的域名都会进行namehash计算，然后使用bytes32进行存储。

namehash算法定义是

```
def namehash(name):
  if name == '':
    return '\0' * 32
  else:
    label, _, remainder = name.partition('.')
    return sha3(namehash(remainder) + sha3(label))
```

比如有一个域名为 `mysite.swarm` ，则计算方式为

```
node = '\0' * 32
node = sha3(node + sha3('swarm'))
node = sha3(node + sha3('mysite'))
# 计算结果
namehash('') = 0x0000000000000000000000000000000000000000000000000000000000000000
namehash('eth') = 0x93cdeb708b7545dc668eb9280176169d1c33cfd8ed6f04690a0bcc88a93fc4ae
namehash('foo.eth') = 0xde9b09fd7c5f901e23a3f19fecc54828e9c848539801e86591bd9801b019f84f
```

使用这种形式有以下几种原因：

1. 合约不需要处理可读的文本字符串，降低不同编码的影响
2. 从一个域名(bob.eth)的namehash可以推导任意子域名(alice.bob.eth)的namehash
3. 推导过程无需知道或处理原域名(bob.eth)

发表于 2022-02-17 17:33　阅读（ 2007 ）　学分（ 80 ）
分类：DApp (https://learnblockchain.cn/categories/DApp)

你可能感兴趣的文章

Web3系列教程之入门篇---8. Dapp白名单 (https://learnblockchain.cn/article/4409) 48 浏览
Web3系列教程之入门篇---7. 一些需要注意的学习点 (https://learnblockchain.cn/article/4398) 157 浏览
Web3系列教程之入门篇---6. Solidity高级指南 (https://learnblockchain.cn/article/4387) 223 浏览
探索查看以太坊交易池的方法 (https://learnblockchain.cn/article/4386) 256 浏览
Web3系列教程之入门篇---5. 以太坊虚拟机（EVM） (https://learnblockchain.cn/article/4377) 222 浏览
写给Solidity开发者的Solana入门指南 (https://learnblockchain.cn/article/4375) 759 浏览

相关问题

以太坊将死于什么？ (https://learnblockchain.cn/question/3814) 2 回答
BSC链交易如果使用自毁函数节约手续费 (https://learnblockchain.cn/question/3693) 1 回答
登链钱包无法链接Dapp (https://learnblockchain.cn/question/3623) 1 回答
Dapp中使用walletconnect调用合约投资方法 (https://learnblockchain.cn/question/3486) 1 回答
冷钱包中 dapp登录及支付问题 (https://learnblockchain.cn/question/3453) 1 回答
Vue 如何导入自己写好的与钱包交互的函数 (https://learnblockchain.cn/question/3395) 2 回答

2 条评论