Domain names

Purpose of the cyber.domain smart contract development

The `cyber.domain` smart contract is designed to create and handle domain names, create or delete a link of domain names to accounts, change domain name owners, as well as to handle a purchase of domain names at auction.

The `cyber.domain` smart contract includes the following actions:

- actions used to acquire a domain name at auction: checkwin, biddomain, biddmrefund and newdomain.
- internal domain actions: passdomain, linkdomain, unlinkdomain and newusername. Although a code of these actions is in the blockchain core, they are called up via smart contract.
- the action declarenames to declare the names used in transactions.

Requirements for domain names

Domain names in CyberWay are formed in accordance with rules and procedures of the Domain Name System (DNS). The following requirements are imposed on the structure of domain names:

- a total number of characters in domain name should not exceed 253 pcs.
- a domain name consists of individual *parts*, separated by the symbol «dot».
- the «dot» characters should not stand side by side in any domain name.
- a number of characters in a separate *part* of domain name should not exceed 63 pcs.
- the valid characters in the domain name are alphanumeric, plus «hyphen».
- the capital letters in the domain name are unacceptable.
- a symbol «hyphen» should not be at the beginning or end of any domain name *part*.
- the further right *part* of domain name must contain at least one alphabetic character. A presence of numeric characters only is not allowed.

Requirements for user names

The following requirements are imposed on a user name structure:

- a total number of characters in a user name should not exceed 32 pcs.
- a user name can consist of individual *parts* separated by the symbol «dot».
- two «dot» symbols near each other are not allowed.
- the valid characters in a user name should be alphanumeric, a symbol «hyphen» could be used as well.
- the capital letters in a user name are unacceptable.
- a symbol «hyphen» should not be at the beginning or end of any user name *part*.

Domain name auction in the smart contract

The actions `checkwin`, `biddomain`, `biddmrefund` and `newdomain` are used to purchase a domain name at auction. The procedure for buying a domain name at the auction is the same as the procedure of buying an account name. The following rules apply to the procedure:

- the bids for the purchase of any domain name are accepted at auction at any time.
- the largest bid is used to determine only one domain name that can be purchased at the moment.
- a domain name is considered to be purchased at auction if the following conditions are met:
  - at least a day has passed after betting on the current domain name;
  - at least one day has passed since previous redemption of any domain name.
- upon completion of auction, a token transfer of the winner is not returned. The winner can take advantage of the following opportunities:
  - to create her/his own domain name using the operation `newdomain` and become its owner.
  - to create subdomain names from it by adding the «dot» symbol and a name to the domain name on the left (for example, an owner of the domain `golos.io` can create subdomains such as `api.golos.io`, `ws.golos.io` and etc). In this case the only direct inheritance of domain names is allowed. It means that if the owner has created a domain for the second level, a domain for the third level can’t be created.

> **Note:**
> The *account names* are formed by adding parts **to the right** (for example, `cyber.msig`, `cyber.domain` can be formed from `cyber`).

checkwin

The `checkwin` action is used to register a domain name owner (a winner) at auction. This action does not require a special call and is called automatically when either `biddomain` or `biddmrefund` is performing.

The `checkwin` action has the following form:

```
[[eosio::action]] void checkwin();
```

This action has no input parameters and can be performed by any account.

biddomain

The `biddomain` action allows an account to bid at the auction. The action has the following form:

```
[[eosio::action]] void biddomain(
    name bidder,
    domain_name name,
    asset bid
);
```

Parameters:

- `bidder` — an account name which bids at the action.
- `name` — a domain name (a string value in accordance with the requirements) on which the bid is done.
- `bid` — a bid (in system tokens, `asset` structure).

To perform this action the `cyber.domain` contract account should have the rights of the `bidder` account to call `transfer` action in `cyber.token` contract. If a bid with a bigger value appears at the auction, the previous bid is returned to the `bidder` account. The refunds are made automatically by internal calling `biddmrefund` from `biddomain`.

biddmrefund

The action `biddmrefund` is used to return a bid which was made at the auction to purchase a domain name if a higher bid is made for the same domain name. The action has the following form:

```
[[eosio::action]] void biddmrefund(
    name bidder,
    domain_name name
);
```

Parameters:

- `bidder` — the account name to which the funds are returned from the auction.
- `name` — the domain name for which the bid was made.

In case of exceptional situations (a network failure or an error in the node's operation), the `biddmrefund` action can be called apart from calling `biddomain` (non-standard calling `biddmrefund`).

newdomain

The `newdomain` action is used to create a new domain name. The action has the following form:

```
[[eosio::action]] void newdomain(
    eosio::name creator,
    domain_name name
);
```

Parameters:

- `creator` — account name that creates a domain name.
- `name` — domain name being created.

The `newdomain` action can be used:

- to create domain name by the system account `cyber.domain` (without an auction).
- to create domain name by a winner of the domain name auction.
- to create a sub-domain name by the owner of a direct parent domain.

The smart contract account `cyber.domain` must be privileged or have a permission to transfer funds from `cyber.names` (in case of a refund). Upon completion of this action, the `creator` account becomes the owner of the created domain name.

Username operations

newusername

The `newusername` action is used to create a user name. The `newusername` declaration has the following form:

```
[[eosio::action]] void newusername(
    name creator,
    name owner,
    username name
);
```

Parameters:

- `creator` — an account name in scope of which the user name is created.
- `owner` — an account name that is to be the domain name owner.
- `name` — a string representation of the user name to be created.

The transaction containing the operation `newusername` must be signed by `creator` account.

Domain operations

passdomain

The `passdomain` action is used to transfer a domain name from one account to another one (to change the domain name owner). The `passdomain` declaration has the following form:

```
[[eosio::action]] void passdomain(
    name from,
    name to,
    domain_name name
);
```

Parameters:

- `from` — an account from which a domain name is transferred and which owns it.
- `to` — an account to which the domain name is transferred and which will be a new domain name owner.
- `name` — a string representation of the domain name to be transferred.

The transaction requires a signature from the account `from` that is the domain name owner.

linkdomain

The `linkdomain` action is used to link a domain name to account name. After linking, the account can be found by domain name instead of account name. The `linkdomain` declaration has the following form:

```
[[eosio::action]] void linkdomain(
    name owner,
    name to,
    domain_name name
);
```

Parameters:

- `owner` — an account that is a domain name owner.
- `to` — an account account to which the domain name is linked.
- `name` — a string representation of the domain name to be linked.

unlinkdomain

The `unlinkdomain` action is used to remove domain name link from account name (unlinking a domain name from the account name). The `unlinkdomain` declaration has the following form:

```
[[eosio::action]] void unlinkdomain(
    name owner,
    domain_name name
);
```

Parameters:

- `owner` — an account that is a domain name owner.
- `name` — a string representation of the domain name to be unlinked.

After performing `unlinkdomain`, the linked domain name will not point to the account.

Examples of using the linkdomain, unlinkdomain and newusername

Let an account on which the contract is installed has a hard-to-remembered name `ajhgsd23qw`. To eliminate this drawback, this account buys the `hello` domain name and links it to its contract using `linkdomain`. It is easier for users to remember such domain as `@hello` and send transfers to this name. In contrast to sending transfers to the account name `ajhgsd23qw` the domain name `@hello` will be converted to `ajhgsd23qw` and the `declarenames` action will be added to the transaction indicating the used domain names.

To stop using the domain name `@hello` the account has to call the `unlinkdomain` action. After that, the domain name `@hello` will not be converted to `ajhgsd23qw` and sending a transfer to it will be impossible.

The `ajhgsd23qw` account in its environment can create usernames. For example, thic account can take the name `admin` for owan use, and can assign the short name `t` for the contract `cyber.token`. In this case, the name `admin@@ajhgsd23qw` (note the double `@@`) will be resolved in the name `ajhgsd23qw`, and `t@@ajhgsd23qw` in `cyber.token`. When adding a domain name, it is possible to use `admin@hello` and `t@hello` respectively.

Declarations of the names used in transactions

It is not enough for one domain name to define a user name. The matter is, the user name is linked to both the owner account and the domain account (usually, it is a smart contract) and has a structure of the form `name@domain` . The domain part defines a scope. The accounts with the same name can exist in different scopes. There are several variants that support `username` data and allow a textual representation of a user name to match an account name.

declarenames

The `declarenames` action is used to submit a list of structures with information about the domain names (or user names) used in transactions and their respective accounts. The action has the following form:

```
[[eosio::action]] void declarenames(
    vector<name_info> domains
);
```

Parameter:

- `domains` — an array of descriptions, each description of which is a structure in form of `name_info` .

The `declarenames` action can be called by any account.

The name_info struct declaration

The `name_info` structure is used to check the presence of all elements at the time of executing the procedure, as well as the existence of linking a domain name with the specified account. The `name_info` structure declaration has the following form:

```
struct name_info {
    domain_name domain;
    name account;
    vector<username> users;
};
```

Parameters:

- `domain` — a domain name.
- `account` — an account name matching to the domain name.
- `users` — a list of domain name users.

The input to the `declarenames` declaration is an array of structures. This array should not be empty.

There are no matches for user names because a user name, unlike domain name, does not change its account for the entire time.

The `domains` array should satisfy the following requirements:

- the list should not contain two structures with the same domain name (field `.domain`), except for empty values (“ ”).
- the field `.account` must not contain an empty value.

Notes

- This implementation does not verify that the accounts specified in `declarenames` exist in other actions. The implementation of validation is difficult because the actual validation process requires a lot of resources to read the ABI file format and deserialize the action. A lack of such control causes a certain risk. That is a sender may add more information to the transaction, so the superfluous part of it will not be checked. As a result, it will have to pay extra for overused `bandwidth` resources.
- This implementation does not contain information about positions of the used domain or user names.

Example 1

A transaction may contain an action with several account names as arguments, for example: `someaction(name 1, name 2, name 3, name 4, name 5)` .
If a user sends an action like `someaction(acc1, one@golos, two@golos, acc1, three@golos)` and it converts to (`someaction(acc1, acc1, acc1, acc1, acc3)`), then `declarenames` will contain only used user names or domain names. At the same time, it is impossible to determine the positions where user names were used, for example:

```
            declarenames([{
```

```
            domain:"golos",
            account:N(acc1),
            users:["one","two","three"]
        }])
```

Example 2

Implemented support for the names like `username@@account` (a presence of the symbols «@@» in the name means that the right side is not a domain, but the account name). For example, if a transaction converts view names `alice@@token` , `bob@@token` , `admin@@golos.io` , then the `declareames` declaration will contain the following arguments:

```
        declarenames([
            {domain:"", account:N(golos.io), users:["admin"]},
            {domain:"", account:N(token), users:["alice","bob"]}
        ])
```

In the above case, the `users` in `declarenames` indicate the usernames used, and in the `account` - the scope where these users are registered. Although, the final accounts into which names are resolved are not specified, an explorer (or some other application) can always identify required account via using the pair `username` + `scope` .

Long domain names

When an account is created it is assigned a base32-encoded 8-byte identification name, which is a 12.5-character string. 60 bits are allocated for 12 characters. The remaining 4 bits are reserved for an additional symbol from the set {1, 2, 3, 4, a, b, c, d, e, f, g, h, i, j}.

A user can also assign her/his domain name to an account. The length of each part of the domain name should not exceed 63 characters. The number of domain names assigned to an account can be more than one. Since a fee is charged for storing a domain name, the number of domain names is limited and depends on the user funds.

A domain name can be purchased at auction, and can also be created by the lower-level domain owner.

Adding a domain name to an account allows a user to form domain transactions, the transactions with a link to the domain. Such transaction specify smart contract to be called and operations it performs. Domain name is not an identifier and can be transferred from one account to another.