NEAR

NEAR Linkdrop

near.org

March 2022

Draven Lu
NEAR技术大使

What is Linkdrop

简单的说，LinkDrop类似于红包的概念。Linkdrop 就是放在一条链接或者二维码中的、人们可以拿出来的数字资产

Linkdrop 不同于传统的空投，后者需要知道接收者的地址，而接收者可能从未听过什么区块链，更没有安装过钱包。

在NEAR Linkdrop中，用户可以通过领取空投直接创建NEAR账户，避免了繁杂的转账手续。

Something useful

体验Linkdrop：https://near-drop-mainnet.onrender.com/

官方Linkdrop地址：https://github.com/near/near-linkdrop

NFT drop：https://github.com/web3gamesofficial/web3games-near-nftdrop

NEAR账户模型

Account ID Rules

- minimum length is 2
- maximum length is 64
- Account ID consists of Account ID parts separated by .
- Account ID part consists of lowercase alphanumeric symbols separated by either _ or -.
- Account ID that is 64 characters long and consists of lowercase hex characters is a specific implicit account ID.

合法Account ID的正则表达式

^(([a-z\d]+[\-_])*[a-z\d]+\.)*([a-z\d]+[\-_])*[a-z\d]+$

NEAR账户模型

Sub-account

- **NEAR的账户模型类似于域名系统，任何账户可以创建自己的子账户，也就是sub-account**

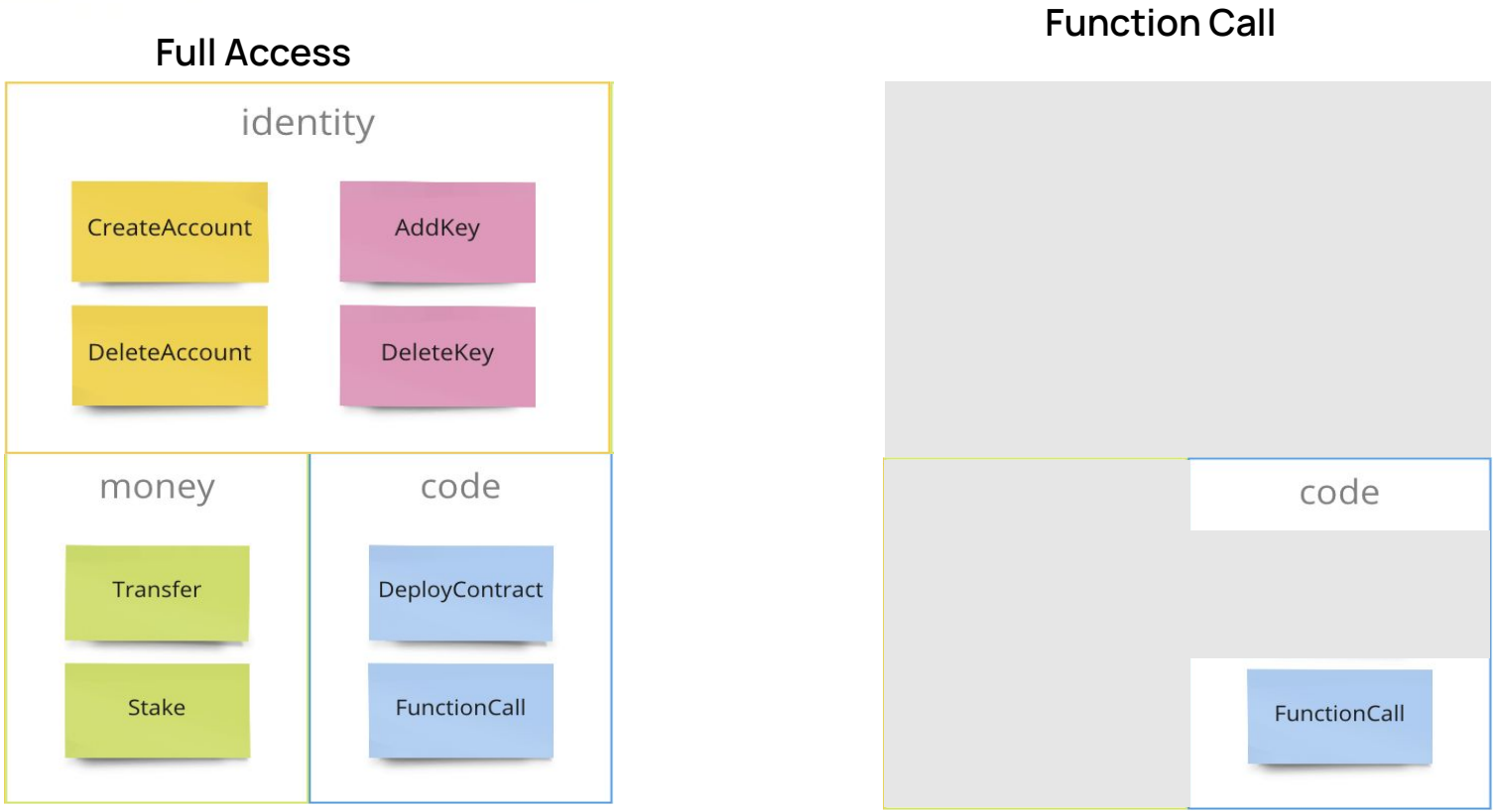
顶级账户

- **形如near,testnet之类的账户为顶级账户**

隐式账户

- **隐式账户和以台坊比特币账户模型很像，都是一组ed25519密钥对，account ID由32字节长度的公钥base58解码后转hex得到**

NEAR账户模型


Full Access
identity
CreateAccount
AddKey
DeleteAccount
DeleteKey
money
Transfer
Stake
code
DeployContract
FunctionCall
Function Call
code
FunctionCall

NEAR账户模型

Keys

On most blockchains, there is one public/private key pair per account. On NEAR, each account can have many key pairs associated with them which we call "Access Keys". There are two types of "Access Keys":

- Full Access (Grants full control to the account)
- Function Call (Allows for only non-monetary transaction signing)

Full Access Key

顾名思义就是拥有全部权限的key，例如当使用near login命令登陆near cli时，就会生成一对FullAccessKey保存至本地对~/.near-credentials/目录中。

Full Access Key

FullAccessKey可以执行的actions有：

- Create Account
- Delete Account
- Add Key
- Delete Key
- Deploy Contract
- Function Call
- Transfer Ⓝ
- Stake Ⓝ (for validators)

```
pub enum Action {
    CreateAccount(CreateAccountAction),
    DeployContract(DeployContractAction),
    FunctionCall(FunctionCallAction),
    Transfer(TransferAction),
    Stake(StakeAction),
    AddKey(AddKeyAction),
    DeleteKey(DeleteKeyAction),
    DeleteAccount(DeleteAccountAction),
}
```

Function Call Key

- Function Call Key 在源码中的定义如下：https://github.com/near/nearcore/blob/master/core/primitives-core/src/account.rs
-

```rust
pub struct AccessKey {
    /// Nonce for this access key, used for tx nonce generation. When access key is created, nonce
    /// is set to `(block_height - 1) * 1e6` to avoid tx hash collision on access key re-creation.
    /// See <https://github.com/near/nearcore/issues/3779> for more details.
    pub nonce: Nonce,

    /// Defines permissions for this access key.
    pub permission: AccessKeyPermission,
}

impl AccessKey {
    pub const ACCESS_KEY_NONCE_RANGE_MULTIPLIER: u64 = 1_000_000;

    pub fn full_access() -> Self {
        Self { nonce: 0, permission: AccessKeyPermission::FullAccess }
    }
}
```

Function Call Key

```rust
/// Grants limited permission to make transactions with FunctionCallActions
/// The permission can limit the allowed balance to be spent on the prepaid gas.
/// It also restrict the account ID of the receiver for this function call.
/// It also can restrict the method name for the allowed function calls.
#[cfg_attr(feature = "deepsize_feature", derive(deepsize::DeepSizeOf))]
#[derive(
    BorshSerialize, BorshDeserialize, Serialize, Deserialize, PartialEq, Eq, Hash, Clone, Debug,
)]
pub struct FunctionCallPermission {
    /// Allowance is a balance limit to use by this access key to pay for function call gas and
    /// transaction fees. When this access key is used, both account balance and the allowance is
    /// decreased by the same value.
    /// `None` means unlimited allowance.
    /// NOTE: To change or increase the allowance, the old access key needs to be deleted and a new
    /// access key should be created.
    #[serde(with = "option_u128_dec_format")]
    pub allowance: Option<Balance>,

    // This isn't an AccountId because already existing records in testnet genesis have invalid
    // values for this field (see: https://github.com/near/nearcore/pull/4621#issuecomment-892099860)
    // we accomodate those by using a string, allowing us to read and parse genesis.
    /// The access key only allows transactions with the given receiver's account id.
    pub receiver_id: String,

    /// A list of method names that can be used. The access key only allows transactions with the
    /// function call of one of the given method names.
    /// Empty list means any method name can be used.
    pub method_names: Vec<String>,
}
```

Function Call Key

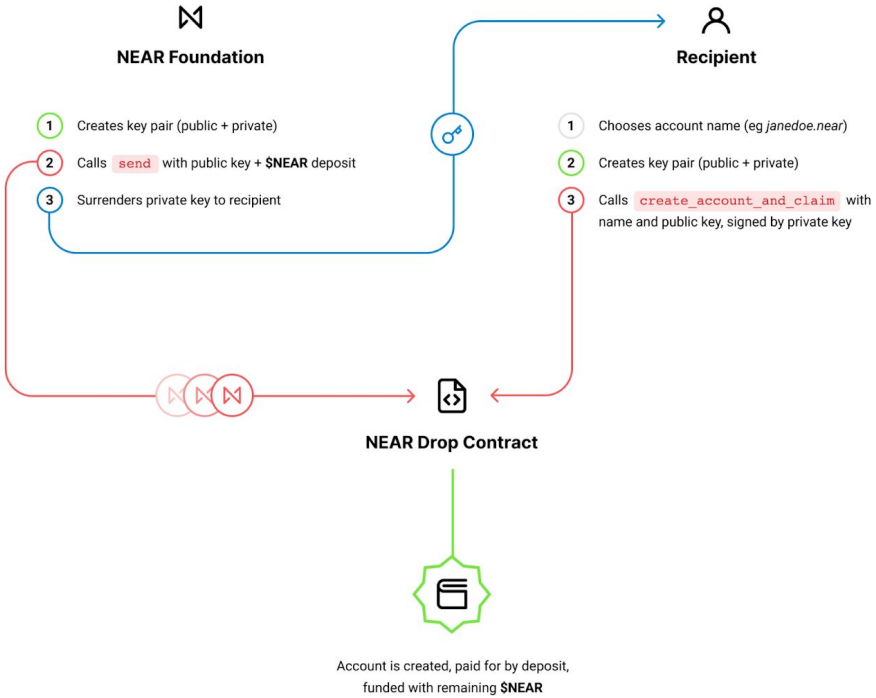
Function Call Key则故名思议是只能执行一些不可支付的合 约函数调用的密钥对，你有多重方式添加function call key，比如说使用 **near api js** 的 **WalletConnection**，使用**near cli**的 **add-key**命令，以及在合 约中使用**env::current_account_id()).add_access_key方法。**

添加function access key的参数如下：

- `accountId` is the account you are adding the key to
- `--contract-id` is the contract you are allowing methods to be called on
- `--method-names` are optional and if omitted, all methods of the `--contract-id` can be called.
- `--allowance` is the amount of
- PublicKey 只有通过该公钥对应私钥签名的交易才可以通过权限校验，格式上就是常规的ed25519密钥对，可以使用多种方法生成一对合法的keypair

About Linkdrop

Linkdrop的具体实现



LinkDrop : accounts

```rust
#[near_bindgen]
#[derive(Default, BorshDeserialize, BorshSerialize)]
2 implementations
pub struct LinkDrop {
    pub accounts: Map<PublicKey, Balance>,
}
```

LinkDrop : Send

```rust
/// Allows given public key to claim sent balance.
/// Takes ACCESS_KEY_ALLOWANCE as fee from deposit to cover account creation via an access key.
#[payable]
pub fn send(&mut self, public_key: Base58PublicKey) -> Promise {
    assert!(
        env::attached_deposit() > ACCESS_KEY_ALLOWANCE,
        "Attached deposit must be greater than ACCESS_KEY_ALLOWANCE"
    );
    let pk: Vec<u8> = public_key.into();
    let value: u128 = self.accounts.get(key: &pk).unwrap_or(default: 0);
    self.accounts.insert(
        key: &pk,
        value: &(value + env::attached_deposit() - ACCESS_KEY_ALLOWANCE),
    );
    Promise::new(account_id: env::current_account_id()).add_access_key(
        public_key: pk,
        ACCESS_KEY_ALLOWANCE,
        receiver_id: env::current_account_id(),
        method_names: b"claim,create_account_and_claim".to_vec(),
    )
}
```

LinkDrop : Claim

```rust
/// Claim tokens for specific account that are attached to the public key this tx is signed with.
pub fn claim(&mut self, account_id: AccountId) -> Promise {
    assert_eq!(
        env::predecessor_account_id(),
        env::current_account_id(),
        "Claim only can come from this account"
    );
    assert!(
        env::is_valid_account_id(account_id.as_bytes()),
        "Invalid account id"
    );
    let amount: u128 = self: &mut LinkDrop
        .accounts: Map<Vec<u8>, u128>
        .remove(key: &env::signer_account_pk()): Option<u128>
        .expect(msg: "Unexpected public key");
    Promise::new(account_id: env::current_account_id()).delete_key(public_key: env::signer_account_pk());
    Promise::new(account_id).transfer(amount)
}
```

LinkDrop : create_account_and_claim

```rust
/// Create new account and and claim tokens to it.
pub fn create_account_and_claim(
    &mut self,
    new_account_id: AccountId,
    new_public_key: Base58PublicKey,
) -> Promise {
    assert_eq!(
        env::predecessor_account_id(),
        env::current_account_id(),
        "Create account and claim only can come from this account"
    );
    assert!(
        env::is_valid_account_id(new_account_id.as_bytes()),
        "Invalid account id"
    );
    let amount: u128 = self: &mut LinkDrop
        .accounts: Map<Vec<u8>, u128>
        .remove(key: &env::signer_account_pk()): Option<u128>
        .expect(msg: "Unexpected public key");
    Promise::new(new_account_id): Promise
        .create_account(): Promise
        .add_full_access_key(public_key: new_public_key.into()): Promise
        .transfer(amount): Promise
        .then(ext_self::on_account_created_and_claimed(
            amount.into(),
            &env::current_account_id(),
            NO_DEPOSIT,
            ON_CREATE_ACCOUNT_CALLBACK_GAS,
        ))
}
```

Linkdrop Proxy

- 为了解决Linkdrop 中 min allowance过高的问题，一般使用linkdrop proxy去创建和领取linkdrop
- https://github.com/near-apps/linkdrop-proxy

一些有意思的小改造

- 加入一些运气元素
- 一次性发放多个红包
- 确保每个人领到红包的金额均匀分布在一定区间内

Thank you