Go sdk for CESS network

license Apache2 golang >=1.19 GO reference Go passing

The go sdk implementation of the CESS network, which provides RPC calls, status queries, block transactions and other functions.

Installation

To get the package use the standard:

```
go get -u "github.com/CESSProject/cess-go-sdk"
```

Testing

To run test:

1. Run a CESS node locally.
2. Run the command

   ```
   go test -v
   ```

Documentation & Examples

Please refer to: https://pkg.go.dev/github.com/CESSProject/cess-go-sdk

Usage

Usually, you only care about how to access your data in the CESS network, you need to build such a web service yourself, this sdk will help you quickly realize data access. Note that p2p-go library needs to be used to enable the data transmission.

Create an SDK Client Instance

To create an sdk client, you need to provide some configuration information: your rpc address (if not, use the rpc address disclosed by CESS), your wallet private key, and transaction timeout. Please refer to the following examples:

```
cli, err := New(
    config.CharacterName_Deoss,
  ConnectRpcAddrs([]string{"wss://testnet-rpc0.cess.cloud/ws/", "wss://testnet-rpc1.cess.cloud/ws/"}),
  Mnemonic("xxx xxx ... xxx"),
  TransactionTimeout(time.Duration(time.Second*10)),
)
```

Register as a DeOSS Role

Call the Register method to register. Note that the first parameter specifies that the role you register is deoss, If there is no error, you can view the transaction details through the returned transaction hash.

```
txhash, _, err := cli.Register(cli.GetCharacterName(), cli.GetSignatureAccPulickey(),
"", 0)
```

Process Data According to the Specifications of CESS

Call the ProcessingData method to process your file. You need to specify the file path. The method returns a segment list and the unique identifier hash of the file in the CESS network.

```
segmentInfo, roothash, err := cli.ProcessingData(filepath)
```

Create Storage Order

Before storing data, you also need to create a data storage order, and you need to fill in the roothash and segmentInfo obtained in the previous step, as well as the user account for uploading data, data name, and bucket name.

```
err := cli.GenerateStorageOrder(roothash, segmentInfo, owner, filename, bucketname)
```

Store Data to Storage Nodes

After creating the storage order, wait for one block to find the storage node information allocated in the order. The next step is to store the data to the storage node through the `WriteFileAction` method in the p2p-go library.

After the storage node receives the data, it will automatically report this action. When all the storage nodes in the storage order have all reported, the data is considered to be stored successfully. As long as there is a storage node that does not report, it is regarded as a storage failure. After the timeout period in the order is exceeded, the storage nodes in the order will be reassigned randomly to start a new round of storage. You need to monitor this storage order, and re-give the data block to the newly allocated storage node until the data storage is successful. The `count` in the storage order indicates the number of times your data has been redistributed. An order can redistribute storage up to 5 times. If your data is still not successfully stored after 5 times, you need to re-upload your data.

Fetch Data

Call the `ReadFileAction` method in the p2p-go library to download all data blocks, and then restore the original data through the `ReedSolomon_Restore` method.

Reporting Vulnerability

If you find out any vulnerability, Please send an email to frode@cess.one, we are happy to communicate with you.

License