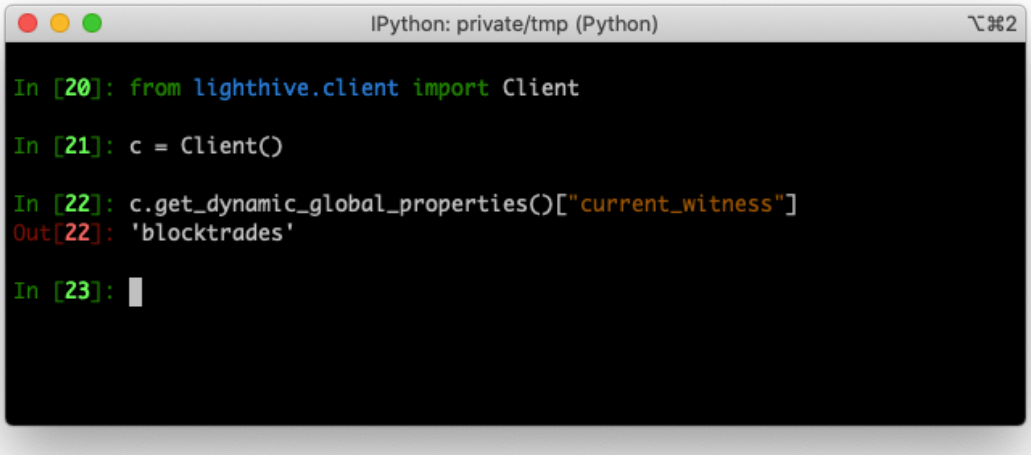
lighthive Documentation

emre yilmaz

Feb 15, 2023

Contents

lighthive is a **light** python client to interact with the HIVE blockchain. It's simple and stupid. It doesn't interfere the process between the developer and the HIVE node.



CHAPTER 1

Features

- No hard-coded methods. All potential future appbase methods are automatically supported.
- Retry and Failover support for node errors and timeouts. See *Retry and Failover*.

CHAPTER 2

Limitations

- No support for pre-appbase nodes.
- No magic methods and encapsulation over well-known blockchain objects. (Comment, Post, Account, etc.)

CHAPTER 3

Installation

lighthive requires python3.6 and above. Even though it's easy to make it compatible with lower versions, it's doesn't have support by design to keep the library simple.

You can install the library by typing to your console:

```
$ (sudo) pip install lighthive
```

After that, you can continue with *Getting Started*.

CHAPTER 4

Documentation Pages

4.1 Getting Started

Client class is the primary class you will work with.

```
from lighthive.client import Client

client = Client()
```

Appbase nodes support different api namespaces.

Client class uses **condenser_api** as default. Follow the official developer portal's api definitions to explore available methods.

4.1.1 Automatic Node Selection

If you prefer, you can pass `automatic_node_selection` flag True to the `Client`.

That way, lighthive requests a `get_dynamic_global_properties` call to the each defined node, and sorts nodes by their response time.

```
2021-12-30 17:20:28,515 lighthive    INFO     Measurements:
https://rpc.ausbit.dev: 0.12 [s]
https://api.openhive.network: 0.12 [s]
https://hive-api.arcange.eu: 0.12 [s]
https://hived.emre.sh: 0.14 [s]
https://api.deathwing.me: 0.15 [s]
https://rpc.ecency.com: 0.16 [s]
https://api.hive.blue: 0.19 [s]
https://api.pharesim.me: 0.28 [s]
https://api.hive.blog: 0.46 [s]
https://techcoderx.com: 0.77 [s]
```

(continues on next page)

(continued from previous page)

```
2021-12-30 17:20:28,516 lighthive    INFO     Automatic node selection took  0.81␣
↪seconds.
2021-12-30 17:20:28,516 lighthive    INFO     Node set as https://rpc.ausbit.dev
```

Since it's a time-consuming operation, this is an optional flag, and it's default is False.

4.1.2 Examples

Get Dynamic Global Properties

```
props = client.get_dynamic_global_properties()

print(props)
```

Get Current Reserve Ratio

```
ratio = c('witness_api').get_reserve_ratio()

print(ratio)
```

Get @emrebeyler's account history

```
history = c.get_account_history("emrebeyler", 1000, 10)

for op in history:
    print(op)
```

Get top 100 witness list

```
witness_list = client.get_witnesses_by_vote(None, 100)

print(witness_list)
```

It's the same convention for every api type and every call on appbase nodes.

Important: Since, api_type is set when the client instance is called, it is not thread-safe to share Client instances between threads.

4.1.3 Optional parameters of Client

Even though, you don't need to pass any parameters to the `Client`, you have some options to choose.

`__init__(self, nodes=None, keys=None, connect_timeout=3, read_timeout=30, loglevel=logging.ERROR, chain=None)`

param nodes A list of appbase nodes. (Defaults: "https://api.hive.blog", "https://api.hivekings.com",

"https://anyx.io")

param keys A list of private keys.

param connect_timeout Integer. Connect timeout for nodes. (Default:3 seconds.)

param read_timeout Integer. Read timeout for nodes. (Default: 30 seconds.)

param loglevel Integer. (Ex: logging.DEBUG)

param chain String. The blockhain we're working with. (Default: HIVE)

param automatic_node_selection Bool. True/False (Default: False)

See *Broadcasting Transactions* to find out how to broadcast transactions into the blockchain.

4.2 Retry and Failover

The workflow on retry and failover:

- Send a request to the RPC node
- If the node returns an HTTP status between *400 and 600 or had a timeout, retry the request up to times.
- Sleep time between cycles has an exponential backoff.
- If the node still can't respond, switch tho next available node until exhausting the node list.
- If all nodes are down or giving errors, and the system is out of options, the original exception is raised.

4.3 Broadcasting Transactions

Since lighthive supports transaction signing out of the box, you only need to define the operations you want to broadcast.

A typical transaction on HIVE blockchain consists of these fields:

```
{
    "ref_block_num": "..",
    "ref_block_prefix": "..",
    "expiration": "..",
    "operations": [OperationObject, ],
    "extensions": [],
    "signatures": [Signature1,],
}
```

As a library user, you don't need to build the all these information yourself. Since all keys except `operations` can be generated automatically, lighthive only asks for a list of operations.

4.3.1 Example: Account Witness Vote

```
from lighthive.client import Client
from lighthive.datastructures import Operation

c = Client(
    keys=["<private_key>",])

op = Operation('account_witness_vote', {
        'account': '<your_account>',
        'witness': 'emrebeyler',
        'approve': True,
```

(continues on next page)

(continued from previous page)

```
    })

c.broadcast(op)
```

4.3.2 Example: Voting for a Post

This will vote with a %1. Percent / 100 = Weight. If you want to downvote, use negative weight.

```
from lighthive.client import Client
from lighthive.datastructures import Operation

client = Client(
    keys=["<private_key>"]
)

op = Operation('vote', {
    "voter": "emrebeyler",
    "author": "emrebeyler",
    "permlink": "re-hitenkmr-actifit-ios-app-development-contribution-
↪20180816t105311829z",
    "weight": 100,
})

client.broadcast(op)
```

4.3.3 Example: Creating a Post (main Comment)

```
import json

from lighthive.client import Client
from lighthive.datastructures import Operation

client = Client(
    keys=["<posting_key>"]
)

post = Operation('comment', {
    "parent_author": None,
    "parent_permlink": "peakd",
    "author": "emrebeyler",
    "permlink": "api-hive-is-down",
    "title": "app.hive.com is down",
    "body": "Body of the post",
    "json_metadata": json.dumps({"tags": "peakd hive lighthive"})
})

resp =client.broadcast(post)

print(resp)
```

Posts are actually Comment objects and same with replies. This example creates a main comment (Post) on the blockchain.

Notes:

- parent_author should be None for posts.
- parent_permlink should be the first tag you use in the post.

If you fill parent_author and parent_permlink with actual post information, you will have a reply. (comment)

4.3.4 Example: Creating a transfer

```
from lighthive.client import Client
from lighthive.datastructures import Operation


c = Client(
    keys=["active_key",])

op = Operation('transfer', {
            'from': 'emrebeyler',
            'to': '<receiver_1>',
            'amount': '0.001 HBD',
            'memo': 'test1!'
        })

c.broadcast(ops)
```

4.3.5 Example: Bundling Operations

It's also possible to bundle multiple operations into one transaction:

```
from lighthive.client import Client
from lighthive.datastructures import Operation


c = Client(
    keys=["active_key",])

ops = [
    Operation('transfer', {
        'from': 'emrebeyler',
        'to': '<receiver_1>',
        'amount': '0.001 HBD',
        'memo': 'test1!'
    }),
    Operation('transfer', {
        'from': 'emrebeyler',
        'to': '<receiver_2>',
        'amount': '0.001 HBD',
        'memo': 'test2!'
    }),

]

c.broadcast(ops)
```

4.3.6 Example: Using convert function for HBDs

```
from lighthive.client import Client
from lighthive.datastructures import Operation

client = Client(
    keys=["<active_key>"]
)
client.broadcast(
    Operation('convert', {
        "owner": "emrebeyler,
        "amount": "0.500 HBD",
        "requestid": 1,
    })
)
```

Note: requestid and the owner is unique together.

Important: Since, lighthive doesn't introduce any encapsulation on operations, you are responsible to create operation data yourself. To find out the specs for each operation, you may review the block explorers for raw data or the source code of hiveblocks.

4.3.7 Bonus: Broadcasting Synchronously

This helper function broadcasts the transaction and waits for it to be processed by the network. You can get transaction id as a result.

```
from lighthive.client import Client
from lighthive.datastructures import Operation


c = Client(
    keys=["<active_key>>",])

op = Operation('transfer', {
            'from': '<username>',
            'to': '<username>>',
            'amount': '0.001 HIVE',
            'memo': 'Test'
        })

resp = c.broadcast_sync(op)

print(resp)
```

Result:

```
{
    'id': 'a0a6655bf839f8fc4aa0fbb4c5779835f662045c',
    'block_num': 51151610,
    'trx_num': 6,
    'expired': False
}
```

4.4 Batch RPC Calls

Appbase nodes support multiple RPC calls in one HTTP request. (Maximum is 50.). If you want to take advantage of this:

```
from lighthive.client import Client

c = Client()

c.get_block(24858937, batch=True)
c.get_block(24858938, batch=True)

blocks = c.process_batch()

print(blocks)
```

This will create one request, but you will have two block details.

Important: This feature is not thread-safe. Every instance has a simple queue (list) as their property, and it's flushed every time the `process_batch` is called.

4.5 Helpers

lighthive has a target to define helper classes for well known blockchain objects. This is designed in that way to prevent code repeat on client (library user) side.

It's possible to use lighthive for just a client. However, if you need to get an account's history, or get followers of account, you may use the helpers module.

4.6 Account helper

This class defines an Account in the HIVE blockchain.

```
from lighthive.client import Client
c = Client()
account = c.account('emrebeyler')
```

When you execute that script in your REPL, lighthive makes a RPC call to get the account data from the blockchain. Once you initialized the Account instance, you have access to these helper methods:

4.6.1 Getting account history

Important: With version *0.4.1*, we have started using `account_history_api.account_history`, instead of `condenser_api.account_history`. When you pass operation name into filter and exclude parameters, you need to add an *_operation* suffix to the operation name.

Example:

- Before: `transfer`

- After: `transfer_operation`

Beofre:

With this method, you can traverse entire history of a HIVE account.

history(self, account=None, limit=1000, filter=None, exclude=None, order="desc", only_operation_data=True, start_at=None, stop_at=None):

Parameters

- **account** – (string) The username.
- **limit** – (integer) Batch size per call.
- **filter** – (list:string) Operation types to filter
- **exclude** – (list:s tring) Operation types to exclude
- **order** – (string) asc or desc.
- **only_operation_data** – (bool) If false, returns in the raw format. (Includes transaction information.)
- **start_at** – (datetime.datetime) Starts after that time to process ops.
- **stop_at** – (datetime.datetime) Stops at that time while processing ops.

account_history is an important call for the HIVE applications. A few use cases:

- Getting incoming delegations
- Filtering transfers on spe‘cific accounts
- Getting author, curation rewards

etc.

Example: Get all incoming HIVE of binance account in the last 7 days

```
import datetime

from lighthive.client import Client
from lighthive.helpers.amount import Amount

client = Client()
account = client.account('deepcrypto8')

one_week_ago = datetime.datetime.utcnow() - datetime.timedelta(days=7)
total_hive = 0
for op in account.history(
        stop_at=one_week_ago,
        filter=["transfer_operation"]):
    if op["to"] != "deepcrypto8":
        continue
    total_hive += Amount.from_asset(op["amount"]).amount

print("Total HIVE transfers received", total_hive)
```

4.6.2 Getting account followers

```
from lighthive.client import Client

client = Client()
account = client.account('deepcrypto8')

print(account.followers())
```

Output will be a list of usernames. (string)

4.6.3 Getting account followings

```
from lighthive.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.following())
```

Output will be a list of usernames. (string)

4.6.4 Getting account ignorers (Muters)

```
from lighthive.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.ignorers())
```

4.6.5 Getting account ignorings (Muted list)

```
from lighthive.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.ignorings())
```

4.6.6 Getting voting power

This helper method determines the account's voting power. In default, It considers account's regenerated VP. (Actual VP)

If you want the VP at the time the last vote casted, you can pass consider_regeneration=False.

```
from lighthive.client import Client

client = Client()
```

(continues on next page)

(continued from previous page)

```
account = client.account('emrebeyler')

print(account.vp())
print(account.vp(consider_regeneration=False))
```

4.6.7 Getting resource credits

This helper method determines the account's resource credits in percent. In default, It considers account's regenerated RC. (Actual RC)

If you want the Rc at the time the last vote casted, you can pass consider_regeneration=False.

```
from lighthive.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.rc())
print(account.rc(consider_regeneration=False))
```

4.6.8 Getting account reputation

```
from lighthive.client import Client

client = Client()
account = client.account('emrebeyler')

print(account.reputation())
```

Default precision is 2. You can set it by passing precision=N parameter.

4.7 Amount helper

A simple class to convert "1234.1234 HIVE" kind of values to Decimal.

```
from lighthive.helpers.amount import Amount

amount = Amount("42.5466 HIVE")

print(amount.amount)
print(amount.symbol)
```

4.8 EventListener Helper

EventListener is a helper class to listen specific operations (events) on the blockchain.

Stream blockchain for the incoming transfers related to a specific account

```
from lighthive.helpers.event_listener import EventListener
from lighthive.client import Client

client = Client()
events = EventListener(client)

for transfer in events.on('transfer', filter_by={"to": "emrebeyler"}):
    print(transfer)
```

Stream for incoming vote actions

```
events = EventListener(client)

for witness_vote in events.on('account_witness_vote', filter_by={"witness":
↪"emrebeyler"}):
    print(witness_vote)
```

Conditions via callables

Stream for the comments and posts tagged with utopian-io.

```
from lighthive.client import Client
from lighthive.helpers.event_listener import EventListener

import json

c = Client()
events = EventListener(c)

def filter_tags(comment_body):
    if not comment_body.get("json_metadata"):
        return False

    try:
        tags = json.loads(comment_body["json_metadata"])["tags"]
    except KeyError:
        return False
    return "utopian-io" in tags


for op in events.on("comment", condition=filter_tags):
    print(op)
```

EventListener class also has

- start_block
- end_block

params that you can limit the streaming process into specific blocks.