

交易所对接流程

1	钱包使用.....	2
1.1	Windows 钱包介绍.....	2
1.2	Linux 钱包介绍.....	2
1.3	Docker 版本钱包介绍.....	2
1.4	同步区块.....	2
1.5	创建钱包.....	4
1.6	创建账户.....	4
2	获取交易历史.....	4
2.1	扫块步骤.....	4
2.2	ori_trx_id 和 result_trx_id.....	5
3	ACT 交易.....	5
3.1	查询区块上 ACT 交易.....	6
3.2	ACT 余额查询.....	7
3.3	ACT 转账提现.....	7
4	合约 Token 交易.....	7
4.1	查询区块上合约 Token 交易.....	8
4.2	合约 Token 余额.....	10
4.3	合约 Token 转账提现.....	11

1 钱包使用

1.1 Windows 钱包介绍

1.钱包下载地址为官网：<https://www.achain.com/home.html>

2.启动钱包（Windows 包括两种：QT 和命令行）

QT 方式启动，双击图标即可。

命令启动钱包，进入到 windows 钱包所在的文件目录，执行命令：

```
Achain-c.exe --rpcuser admin --rpcpassword 123456 --httpdendpoint 127.0.0.1:8299 --server --data --dir d:\config
```

1.2 Linux 钱包介绍

下载地址为：https://github.com/Achain-Dev/Achain_linux

Linux 需要编译代码，具体请参考文档：

https://github.com/Achain-Dev/Achain_linux/blob/master/linux_installation_guide

编译成功后，启动命令：

```
Achain --rpcuser admin --rpcpassword 123456 --httpdendpoint 127.0.0.1:8299 --server
```

1.3 Docker 版本钱包介绍

1、下载地址：<https://github.com/Achain-Dev/Docker>

2、安装启动参考：<https://github.com/Achain-Dev/Docker/blob/master/README.md>

相关参数介绍：

1、rpcuser :启动调用 rpc 的 username；

2、rpcpassword :启动调用 rpc 的 password；

3、httpdentpoint :是使用 http 调用的 ip 地址 （Docker 可以不用设置）

4、data-dir 是钱包同步数据存放的路径。

1.4 同步区块

在钱包启动之后，钱包会自动连接上节点，开始更新区块信息。

此时不要使用 RPC 方法调用钱包，直至同步到最新的区块。

可以通过控制台，调用 info 命令查看钱包同步的情况。

主要关注三个参数：

1.blockchain_head_block_num，这个是目前块号。

2.blockchain_head_block_age，这个是目前块产生的时间。我们产块时间为 10 秒，因此如果这个值大于 10 秒，那么块还没有同步结束。

3.network_num_connection，这个是当前的连接数。如果这个为 0，则块不会同步，需要及时解决。

注意：钱包在同步的过程中，调用 rpc 命令会有延迟或者报错情况。

请在钱包同步之后再继续进行 rpc 调用。

例如：

```
(wallet closed) >>> info
{
  "blockchain_head_block_num": 1140371,
  "blockchain_head_block_age": "0 second old",
  "blockchain_head_block_timestamp": "2017-12-01T09:17:00",
  "blockchain_head_block_id": "f56bf37b0f76a16c46aef0ede5ad2f8d3239006c",
  "blockchain_average_delegate_participation": "100.00 %",
  "blockchain_confirmation_requirement": 1,
  "blockchain_share_supply": "1,005,701,855.00000 ACT",
  "blockchain_blocks_left_in_round": 10,
  "blockchain_next_round_time": "at least 2 minutes in the future",
  "blockchain_next_round_timestamp": "2017-12-01T09:18:40",
  "blockchain_random_seed": "75817d1800ba89e30d96f0b4f8e306a6d3a916ab",
  "client_data_dir": "d:/config",
  "client_version": "3.1.3",
  "network_num_connections": 15,
  "network_num_connections_max": 200,
  "network_chain_downloader_running": false,
  "network_chain_downloader_blocks_remaining": null,
  "ntp_time": "2017-12-01T09:17:00",
  "ntp_time_error": "-1.57976500000000010",
  "wallet_open": false,
  "wallet_unlocked": null,
  "wallet_unlocked_until": null,
  "wallet_unlocked_until_timestamp": null,
  "wallet_last_scanned_block_timestamp": null,
  "wallet_scan_progress": null,
  "wallet_block_production_enabled": null,
  "wallet_next_block_production_time": null,
  "wallet_next_block_production_timestamp": null
}
```

1.5 创建钱包

调用命令 `wallet_create` 创建钱包，因为创建的账户都需要保存钱包下面。

命令参数有两个：第一个是钱包名，第二个是钱包密码。

创建完钱包控制台进入解锁状态。

例如：

```
<wallet closed> >>> wallet_create wallet wallet01  
OK
```

1.6 创建账户

目前对接的交易所管理用户有两种方式：

第一种：只建立一个主账户，通过在主账户后拼接 32 位子地址方式（可以选择 UUID，去掉 - ，正好 32 位）来区分用户账户。优点：在转账时，子地址记账，免去了资金归集的麻烦。

因此子地址方式是比较推荐的一种方式。

创建账户的命令为： `wallet_account_create`，参数为用户名。

这个命令返回的参数是账户的地址，如果返回是空或者报错，说明用户名已使用，或者账户名非法。

通过以上步骤我们便完成了钱包的准备工作。

```
wallet01 <unlocked> >>> wallet_account_create act01  
ACTHE6bM2qet3PkBCaG6kooA2xEXz4oEHQ6s
```

第二种：为每一个用户创建一个账户，在钱包下管理账户，需要自己做资金归集。

注意： 采用这种方式需要关闭自动备份，否则日志增长太快。

控制台输入： `wallet_set_automatic_backups false`

2 获取交易历史

2.1 扫块步骤

通过扫块方式，获取交易历史，步骤如下：

- 1.调用 blockchain_get_block_count 命令获取目前最新的块号;
- 2.调用 blockchain_get_block 依次查询每个区块, 参数为块号;
- 3.调用 blockchain_get_pretty_transaction 获取一次交易的入账方和出账方;

如果是合约交易, 调用 blockchain_get_pretty_contract_transaction;

注意: “查询区块”返回的字段"user_transaction_ids": []

如果这个参数为空, 则表示这一个块上没有交易, 可以扫下一块;

如果不为空, 则进行下一步的交易查询操作, user_transaction_ids 中都是交易的 trx_id。

例如:

```
(wallet closed) >>> blockchain_get_block 1140371
{
  "previous": "a9652f371cf938671798f18f1eb7db42e3649d8",
  "block_num": 1140371,
  "timestamp": "2017-12-01T09:17:00",
  "transaction_digest": "c8cf12fe3180ed90ia58a0697a522f1217de72d04529bd255627a4ad6164f0f0",
  "next_secret_hash": "132e0e2b2100e5f43644145440406b938a9a2d6",
  "previous_secret": "5442f81c384968eb9cd2833b56f1590178e2acd",
  "delegate_signature": "1f2195ef7583eb5d2bd4a2372c6b3e119b0daf11e4b3c4f82a6a2de0be21c03693132953a09ada312f3dc2441f291a5522fb18ad4320b2a48dd1b256a547de5dc",
  "user_transaction_ids": [],
  "id": "f56bf37b0f76a16c46aef0de5ad2f8d3239006c",
  "block_size": 166,
  "latency": 0,
  "signee_shares_issued": 0,
  "signee_fees_collected": 0,
  "signee_fees_destroyed": 0,
  "random_seed": "0000000000000000000000000000000000000000000000000000000000000000",
  "processing_time": 0
}
```

注意: trx_id 来自区块上的 user_transaction_ids 字段

(浏览器上交易 id 是: ACT 是 trx_id, 合约调用是: ori_trx_id)

2.2 ori_trx_id 和 result_trx_id

ori_trx_id 和 result_trx_id, 仅仅是合约调用中会出现, ACT 转账中没有。

ori_trx_id 记录的是原始的交易, 也就是只与 ACT 相关的交易方式。因此在调用合约时, ori_trx_id 只记录了交易花了多少 ACT, 手续费是多少 ACT, 调用合约花费上限是多少 ACT。而对于所调用的合约, 仅记录调用了合约的什么方法, 使用的参数是什么, 不关心合约返回的结果。另外, 每次使用调用合约或者转账 rpc 之后, 立即返回的就是 ori_trx_id。

result_trx_id 则记录的是完整的一笔交易。即除了上述的 ACT 交易外, result_trx_id 会记录本次调用的合约是否成功, 合约返回的结果是什么等等。每次扫块所获得的块上的交易 id 就是 result_trx_id。

3 ACT 交易

本步骤需要在 1、2 步骤的基础上进行, 请须知。

3.1 查询区块上 ACT 交易

调用 `blockchain_get_block_count` 命令获取目前最新的块号；

调用 `blockchain_get_block` 依次查询每个区块，参数为块号；

使用上面 `user_transaction_ids` 字段中的 `trx_id` 作为参数，调用 RPC 方法：

`blockchain_get_pretty_transaction` 获取本次交易的入账和出账方。

`from_account`: 转出地址，
`to_account` : 转入地址，
`amount`: 交易金额的 10^5 倍。

例如：

```
{
  "is_virtual": false,
  "is_confirmed": true,
  "is_market": false,
  "is_market_cancel": false,
  "trx_id": "f7b58cecabc5a0a8cd8cc8244a6141f99feb29c5",
  "block_num": 60435,
  "block_position": 0,
  "trx_type": 0,
  "ledger_entries": [{
    "from_account": "ACTAGSsqKCRkyadUtqqeMZcPXot6dhTGBHGU",
    "from_account_name": "",
    "to_account": "ACTKwKoKGtYibY6fnohvsQpaCkm5Gd12wZ94",
    "to_account_name": "",
    "amount": {
      "amount": 1000000,
      "asset_id": 0
    },
    "memo": "",
    "running_balances": []
  }
],
  1,
}
```

然后调用 RPC 方法：`blockchain_get_transaction` 获取是否有子地址交易。

这里 `alp_account` 是入账地址，图中是带有子地址的一笔交易，通过字段 `alp_account` 即可获得子地址。

例如：

```

transaction_id_prefix: f7b58cecabc5a0a8cd8cc8244a6141f99feb29c5
[
  "f7b58cecabc5a0a8cd8cc8244a6141f99feb29c5", {
    "trx": {
      "expiration": "2017-12-04T04:17:09",
      "alp_account": "ACTKwXoKgtYibY6fnohvsQpaCkm5Gd12wZ941234567890123456789012",
      "alp_inport_asset": {
        "amount": 1000000,
        "asset_id": 0
      },
      "operations": [{
        "type": "withdraw_op_type",
        "data": {
          "balance_id": "ACTDyHHzqDiiCQCFHyPUzjZu9oqXUgQrZTbe",
          "amount": 1001000,
          "claim_input_data": ""
        }
      }
    ]
  }, {

```

3.2 ACT 余额查询

调用 RPC 方法: blockchain_list_address_balances 参数 (地址)

返回: 将返回的所有 balance 字段值相加, 便是地址上 ACT 余额。

3.3 ACT 转账提现

调用方法 wallet_transfer_to_address 向外转出 ACT。

在调用此方法前, 需要将钱包打开并解锁。这个方法共接收四个参数,

第一个参数是转出的 ACT 数量, 这里为真实数量;

第二个参数写死为 ACT;

第三个参数为转出的账户名;

第四个参数为转出的地址。

这个方法会返回一个交易 id, 此交易 id 是 trx_id。

4 合约 Token 交易

本步骤需要在 1、2 步骤的基础上进行, 请须知。

4.1 查询区块上合约 Token 交易

调用 `blockchain_get_block_count` 命令获取目前最新的块号；

调用 `blockchain_get_block` 依次查询每个区块，参数为块号；

然后执行如下步骤：

1. 使用上面 `user_transaction_ids` 字段中的 `trx_id` 作为参数，调用 RPC 方法：

`blockchain_get_transaction` 获取本次交易的入账和出账方。

使用 `blockchain_get_block` 返回的 `user_transaction_ids` 字段中的 `trx_id` 作为参数；

2. 如果 `operation` 字段中，第一个 `type` 值为 `transaction_op_type`，那么则这一笔交易为合约交易。接下来获取这个结果 `Json` 中的 `result_trx_id` 字段，注意，这个 `result_trx_id` 是 `trx` 这一层 `JsonObject` 的结果，不要取错了。

3. 接下来，调用 `blockchain_get_pretty_contract_transaction` 这个方法，参数为上一步获取的 `result_trx_id` 获得结果。首先看 `to_account` 字段是否为你所查询的 `Token` 合约 ID，如果不是就直接跳过。接下来看 `reserved` 字段，第一个参数就是调用的方法，如果是 `transfer_to` 则是合约转账，不是就跳过。第二个参数是 `transfer_to` 方法的参数，用竖线“|”隔开。第一个参数是转到地址，第二个是转账金额。如果要监控是否给自己的地址打合约币，就可以监控这个参数是不是自己的地址。


```
wallet <locked> >>> blockchain_get_transaction fd88fb7d1b3aac1fe617f8332fdb62db047117e
[
  "fd88fb7d1b3aac1fe617f8332fdb62db047117e", {
    "trx": {
      "expiration": "2017-12-05T10:01:49",
      "alp_account": "",
      "alp_inport_asset": {
        "amount": 0,
        "asset_id": 0
      },
    },
    "operations": [ {
      "type": "transaction_op_type",
      "data": {
        "trx": {
          "expiration": "2017-12-05T10:01:49",
          "alp_account": "",
          "alp_inport_asset": {
            "amount": 0,
            "asset_id": 0
          },
        },
        "operations": [ {
          "type": "call_contract_op_type",
          "data": {
            "caller": "ACT7UZfdCJoNGzeAiEuK7dxzdtQzzhXWu5F2UmZ9fcNc2JKQLRzWk",
            "balances": [ [
              "ACT9Ek1tYy4KUNMKx49Wocm5uUJjYnSfJnY",
              2000
            ]
            ],
            "contract": "ACT92cJUUM6q$9qp1ihnJB5DJrf1pP9F2f$B",
            "costlimit": {
              "amount": 1000,
              "asset_id": 0
            },
            "transaction_fee": {
              "amount": 1000,
              "asset_id": 0
            },
          },
        },
      ],
    },
  ],
}
```

```

}
],
"result_trx_type": "complete_result_transaction",
"result_trx_id": "fd88fb7d1b3aac1fe617f8332fdb62db047117e",
"signatures": [ ]
},

```

```

wallet <locked> >>> blockchain_get_pretty_contract_transaction
transaction_id_prefix: fd88fb7d1b3aac1fe617f8332fdb62db047117e
{
  "result_trx_id": "fd88fb7d1b3aac1fe617f8332fdb62db047117e",
  "orig_trx_id": "31c7e13f79ba1b9952aae7b104020ddc89603a81",
  "block_num": 1174841,
  "block_position": 0,
  "trx_type": 14,
  "is_completed": false,
  "to_contract_ledger_entry": {
    "from_account": "ACT7bWPwBBHUFoCkmHitJ318FZ6Pn1E9gFPx",
    "from_account_name": "coinfix",
    "to_account": "CON92cJUUM6qS9qp1ihnJB5DJrf1pP9F2fSB",
    "to_account_name": "USD_COIN",
    "amount": {
      "amount": 0,
      "asset_id": 0
    },
    "fee": {
      "amount": 1660,
      "asset_id": 0
    },
    "memo": ""
  },
  "from_contract_ledger_entries": [],
  "timestamp": "2017-12-05T09:02:00",
  "expiration_timestamp": "2017-12-05T10:01:49",
  "reserved": [
    "transfer_to",
    "ACTLM5zptEYL6kqDfJG5AJUuucyu2DD75EW93df22f5790062ed839ed4f0bb395f00d1999.9900000"
  ]
}

```

4.2 合约 Token 余额

调用方法 `blockchain_get_events`, 参数是交易所在块号: `block_num` 和之前获得的 `trx_id`。

获取 `event_type` 和 `event_param`:

`event_type` 区分转账是否成功, 若为 `transfer_to_success`, 这笔交易转账成功, 否则失败。

`event_param` 则有 4 个参数, 用逗号 “,” 隔开。第一个是转出地址及余额, 用冒号 “:” 隔开, 第二个参数为转入地址及余额, 第三个参数是版本号 (递增), 第四个参数为时间戳。

注意: 显示的余额是本次交易发生之后的余额。

```

wallet <locked> >>> blockchain_get_events 1174841 fd88fb7d1b3aac1fe617f8332fdb62db047117e
{
  "id": "ACT92cJUUM6qS9qp1ihnJB5DJrf1pP9F2fSB",
  "event_type": "transfer_to_success",
  "event_param": "ACT7bWPwBBHUFoCkmHitJ318FZ6Pn1E9gFPx:3672000000.ACTLM5zptEYL6kqDfJG5AJUuucyu2DD75EW9:13552185687.263.1512464510",
  "is_truncated": false
}

```

4.3 合约 Token 转账提现

合约 Token 的转账功能全部是通过合约调用来实现的。

因此需要通过 call_contract 方法来实现合约转账。

call_contract 共需要 6 个参数：

第一个为合约 id，以“CON”开头。

第二个为调用合约的账户，即转出账户。

第三个为调用方法，这里由于是提现，因此我们写“transfer_to”。

第四个为调用的方法所需参数，格式：转到地址|转出金额，参数间用竖线“|”隔开。

第五个为“ACT”。

第六个可以为“1”。注意：这是交易手续费上限，可以设置为大于 0.01 的任何数。

注意：转账金额最多支持 5 位小数，如果多于 5 位，后几位会全部忽略。

```
wallet01 <unlocked> >>> call_contract CONqfnUwosAcc3YN5D1j3PCh7G4siXPScWK act0 t
transfer_to ACTKwXoKGtYibY6fnohusQpaCkm5Gd12wZ94!10 ACT 1
{
  "index": 0,
  "entry_id": "9fa3db6635c4feb71bd09d6ff8b267dc8b4f36e1",
  "block_num": 0,
  "is_virtual": false,
  "is_confirmed": false,
  "is_market": false,
  "trx": {
    "expiration": "2017-12-04T07:27:24",
    "alp_account": "",
    "alp_inport_asset": {
      "amount": 0,
      "asset_id": 0
    },
    "operations": [{
      "type": "call_contract_op_type",
      "data": {
        "caller": "ACT74xPhBdsW5CMrcnH6rSDBdbUqN1yEd4C5MTsAYJpQhoNxU2ER",
        "balances": [[
```

在 call_contract 之后，会返回交易的具体信息，这里我们拿到 entry_id，也就是我们的之前所叙述的 ori_trx_id。

调用 blockchain_get_contract_result，参数为刚刚拿到的 entry_id。结果中有两个参数，第一个是交易发生的块号，第二个参数是 trx_id，将这两个结果保存，然后调用方法 blockchain_get_events，第一个参数是 block_num，第二个参数是前一步获得的 trx_id。这个方法两个参数也就是上一步获得的两个参数。

这一步结束我们会获取 event_type 和 event_param。其中，event_type 区分转账是否成功。如果结果为 transfer_to_success，这笔交易转账成功，否则失败。event_param 则有 4 个

参数，用逗号“,”隔开。第一个是转出地址及余额，用冒号“:”隔开，第二个参数为转入地址及余额，第三个参数是版本号（递增），第四个参数为时间戳。

注意：call_contract 结束之后，交易还未出块。因此查询交易具体信息时，请等待出块后再查询，否则查询结果很可能就是空的。出块时间为 10 秒。

```
lwallet01 (unlocked) >>> blockchain_get_contract_result 743585dae0a3bdd93c7b1ab0
4a6b55f10ef2f18f
{
  "block_num": 79078,
  "trx_id": "d2ad6d51dfe1d503c602512f662b31b1b1f7c6dd"
}
```

```
wallet01 (unlocked) >>> blockchain_get_events
block_number: 79078
trx_id: d2ad6d51dfe1d503c602512f662b31b1b1f7c6dd
[
  {
    "id": "ACTqfnUwosAcc3YN5D1j3PCh7G4siXPScWK",
    "event_type": "transfer_to_success",
    "event_param": "ACTAGSsqKCRkyadUtqqeMZcP%ot6dhTGBHGU:99999848000000,ACTKwXoK
GtYibY6fnohvsQpaCkm5Gd12wZ94:151000000,6,1512543860",
    "is_truncated": false
  }
]
```