# 交易所对接流程

## 钱包准备

Windows钱包使用介绍：

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）
3. QT方式启动，双击图标即可。
4. 命令行方式：首先进入到windows钱包所在的文件目录，使用命令启动钱包：

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

Linux钱包使用介绍：

1. 下载地址为：<https://github.com/Achain-Dev/Achain_linux>
2. 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>

1. 编译成功后，启动命令：

Achain --rpcuser admin --rpcpassword 123456 --httpdendpoint 127.0.0.1:8299 --server --data-dir c:\user\_data\config

Docker 版本钱包使用介绍

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。

### 钱包同步

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：

第一个是blockchain\_head\_block\_num，这个是目前的块号。

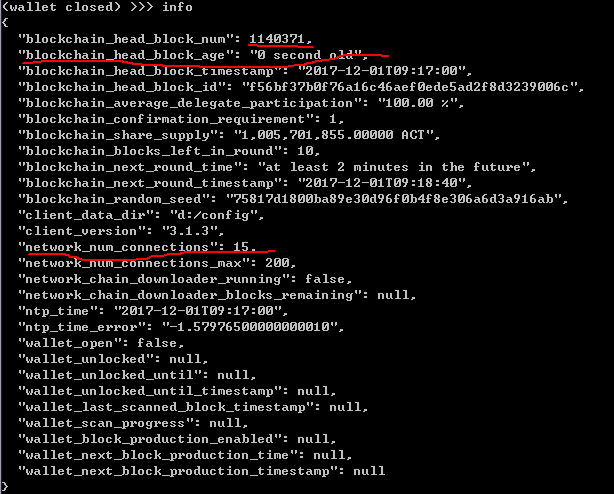
第二个是blockchain\_head\_block\_age，这个是目前块产生的时间。我们产块时间为10秒，因此如果这个值大于10秒，那么块还没有同步结束。

第三个参数是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进行rpc调用。

例如：



### 创建钱包及账户

第一步：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：

C:\Users\fanyukun\AppData\Local\Temp\1513651597(1).png

第二步：创建账户。

目前对接的交易所管理用户有两种方式：

第一种：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。

控制台下输入：wallet\_set\_automatic\_backups false

第二种：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。好处：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。

C:\Users\fanyukun\AppData\Local\Temp\1513651780(1).png

## 获取交易（扫块方式）

1. 获取链区块

调用blockchain\_get\_block\_count命令获取目前最新的块号，

使用命令blockchain\_get\_block依次查询每个区块，参数为块号。

注意：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：



注意：trx\_id来自区块上的user\_transaction\_ids字段

重点解释一下：ori\_trx\_id和result\_trx\_id，仅仅是合约调用中会出现，ACT转账中没有。

（浏览器上交易id是：ACT是trx\_id，合约调用是：ori\_trx\_id）。

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。

而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trx\_id。

2、查询区块上交易（ACT交易）

使用上面user\_transaction\_ids字段中的trx\_id作为参数，

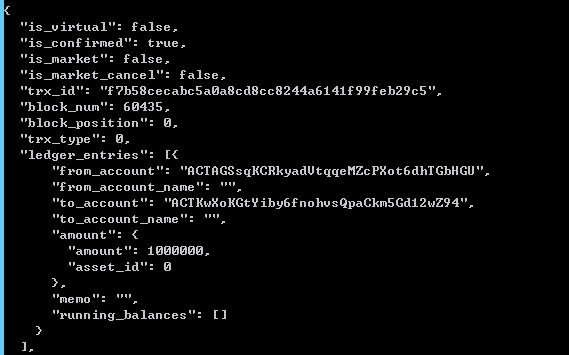
调用RPC方法：blockchain\_get\_pretty\_transaction获取本次交易的入账和出账方。

from\_account:转出地址，

to\_account :转入地址，

amount:交易金额的105倍。

例如：



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：



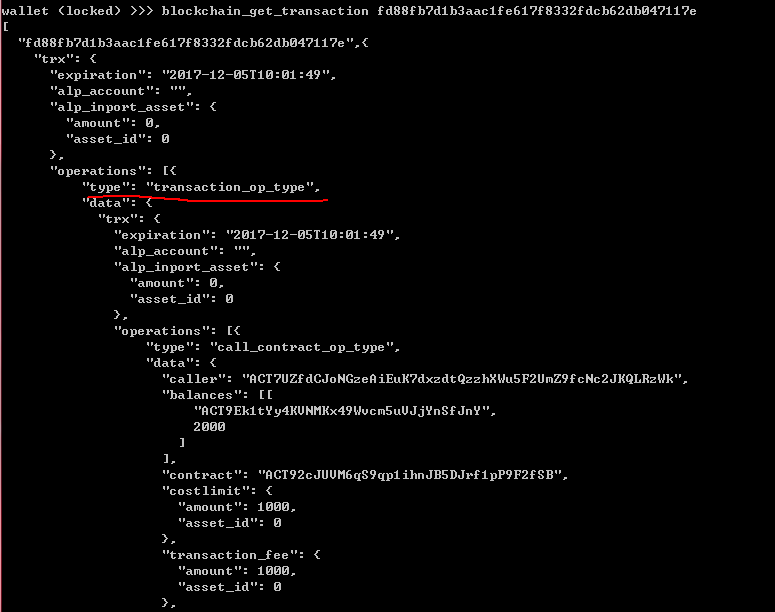
1. **查询区块上交易（合约Token交易）**

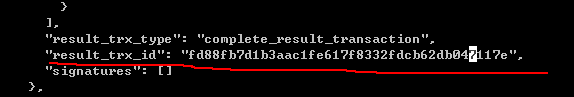
使用上面user\_transaction\_ids字段中的trx\_id作为参数，

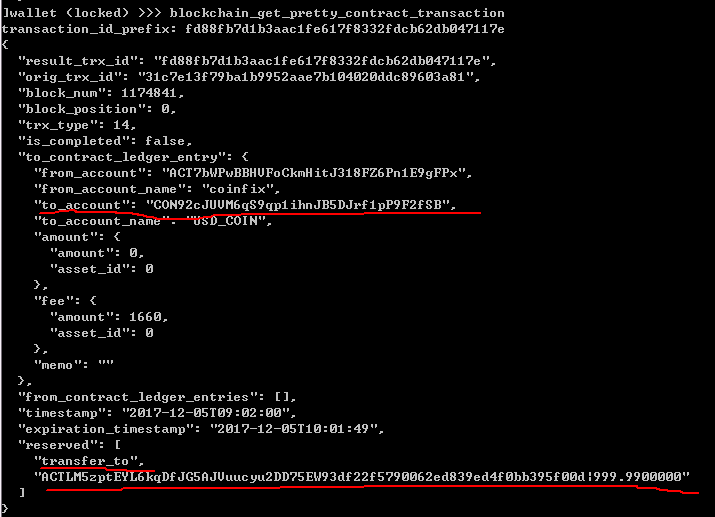
调用RPC方法：blockchain\_get\_transaction获取本次交易的入账和出账方。

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。

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。







3、合约Token获取地址余额

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。

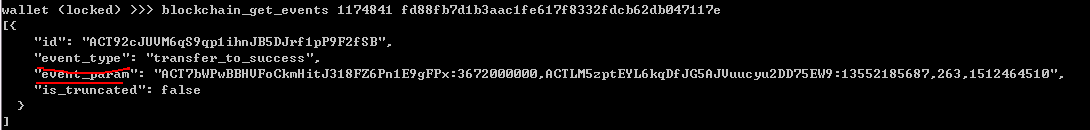
获取 event\_type和event\_param。

其中，event\_type区分转账是否成功。

如果结果为transfer\_to\_success，这这笔交易转账成功，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。



4、ACT地址余额查询

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。

5、合约转账提现

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：

第一个参数为合约id，以“CON”开头。

第二个参数为调用合约的账户，即转出账户。

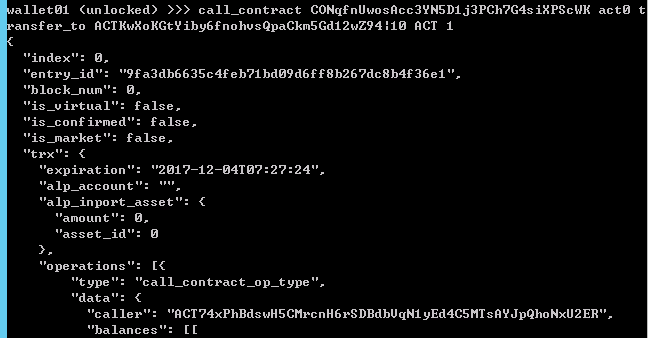
第三个参数为调用方法，这里由于是提现，因此我们写“transfer\_to”。

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

第五个参数写死为“ACT”。

第六个参数可以写死为“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。



**call\_contract调用方法**

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。

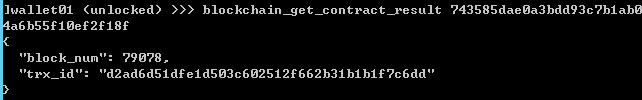
调用blockchain\_get\_contract\_result，参数为刚刚拿到的entry\_id。

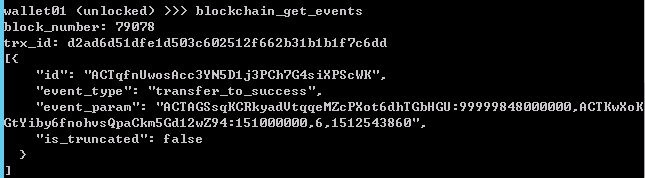
获得结果。结果中有两个参数，第一个是交易发生的块号，第二个参数是trx\_id，将这两个结果保存，然后调用方法blockchain\_get\_events，第一个参数是block\_num，第二个参数是前一步获得的trx\_id。这个方法的两个参数也就是上一步获得的两个参数。

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，应等待出块后再查询，否则查询结果很可能就是空的。出块时间为10秒。





6、ACT转出提现

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。