Cybex 接口说明

调用方法

- Cybex中任意节点包含两种API接口，所有API数据使用JSON-RPC2.0形式。
- 可以选择使用HTTP或Websockets方式进行访问。

  其中HTTP方式仅允许只读的查询数据库，如：

```
$ curl --data '{"jsonrpc": "2.0", "method": "call", "params": [0, "get_account
s", [["1.2.0"]]], "id": 1}' http://127.0.0.1:8090/rpc
{"id":1,"result":[{"id":"1.2.0","annotations":[],"membership_expiration_date":
"1969-12-31T23:59:59","registrar":"1.2.0","referrer":"1.2.0","lifetime_referre
r":"1.2.0","network_fee_percentage":2000,"lifetime_referrer_fee_percentage":80
00,"referrer_rewards_percentage":0,"name":"committee-account","owner":{"weight
_threshold":1,"account_auths":[],"key_auths":[],"address_auths":[]},"active":{
"weight_threshold":6,"account_auths":[["1.2.5",1],["1.2.6",1],["1.2.7",1],["1.
2.8",1],["1.2.9",1],["1.2.10",1],["1.2.11",1],["1.2.12",1],["1.2.13",1],["1.2.
14",1]],"key_auths":[],"address_auths":[]},"options":{"memo_key":"GPH111111111
1111111111111111111114T1Anm","voting_account":"1.2.0","num_witness":0,"num_
committee":0,"votes":[],"extensions":[]},"statistics":"2.7.0","whitelisting_ac
counts":[],"blacklisting_accounts":[]}]}
```

  Websocket方式需要登录（链上执行广播需要会话信息）并可执行查询和修改操作，如：

```
$ npm install -g wscat
$ wscat -c ws://127.0.0.1:8090
> {"id":1, "method":"call", "params":[0,"get_accounts",[["1.2.0"]]]}
< {"id":1,"result":[{"id":"1.2.0","annotations":[],"membership_expiration_date
":"1969-12-31T23:59:59","registrar":"1.2.0","referrer":"1.2.0","lifetime_refer
rer":"1.2.0","network_fee_percentage":2000,"lifetime_referrer_fee_percentage":
8000,"referrer_rewards_percentage":0,"name":"committee-account","owner":{"weig
ht_threshold":1,"account_auths":[],"key_auths":[],"address_auths":[]},"active"
:{"weight_threshold":6,"account_auths":[["1.2.5",1],["1.2.6",1],["1.2.7",1],["
1.2.8",1],["1.2.9",1],["1.2.10",1],["1.2.11",1],["1.2.12",1],["1.2.13",1],["1.
2.14",1]],"key_auths":[],"address_auths":[]},"options":{"memo_key":"GPH1111111
1111111111111111111114T1Anm","voting_account":"1.2.0","num_witness":0,"nu
m_committee":0,"votes":[],"extensions":[]},"statistics":"2.7.0","whitelisting_
accounts":[],"blacklisting_accounts":[]}]}
```

- 所有查询操作可以直接访问，所有修改操作需要通过广播交易*broadcast_transaction*方式进行，广播交易时，需要有签名。

- 应用开发可以使用线上已有的cybexjs库来进行。

接口说明

基本说明

- 进行查询时，可以使用curl的普通http形式访问节点端口下的/rpc路径来执行，或直接使用cybexjs-ws库的Apis工具类来进行，该方式使用Websocket方式连接节点。
- Apis使用前需要初始化，传入节点地址等信息。Apis会生成一个实例，并进行Login操作。查询和广播时，使用Apis.instance()获取一个实例，使用该实例的对应方法db_api()/history_api()/network_api()/network_api()获得访问不同类型api的途径，并添加参数进行查询或广播。
- 具体示例可以参考之前传过去的cybex-daemon项目代码，尤其是其中的CybexDaemon类。其中包含了部分查询操作的示例，以及一个performTransaction方法，该方法可以用来完成签名并广播交易。
- 关于广播并执行一个交易，大致的步骤是 **构造交易体 - 向交易体中添加操作 - 添加签名 - 广播交易**，其中需要手动进行的主要是**构造和添加交易中的操作**，其他主要可以由**TransacitonBuilder**类协助执行。
- Cybex中的所有标准数据结构都有唯一ID进行标识。CybexID采用三段式的形式标记（如x.x.xxxx的形式）。
    - 其中第一段为数据大类，目前Cybex中有0/1/2三类ID：
        - **0** 目前仅用来表示撮合成交
        - **1** 表示链上协议中定义的数据结构，用来进行交易和操作的验证，钱包与数据库都需遵循该结构；
        - **2** 表示系统当前运行中所采用的动态数据，不需要在客户端和节点之间进行通讯，仅存在于链上，用于执行业务逻辑。如某一资产现有的资金池状况/智能资产的抵押状况/当前系统的区块数据等。
    - ID第二段表示具体的数据类型，第三段表示在该类型中的序号。
    - 常用的ID类型可以查看 http://docs.bitshares.org/development/blockchain/objects.html#list-of-commonly-used-objects 或查阅代码获得。
- 可以添加的操作列表，以及操作结构的数据说明可以在 https://github.com/CybexDex/cybexjs/blob/master/serializer/src/operations.js 和 https://github.com/CybexDex/cybexjs/blob/master/serializer/src/types.js 查询。
- 目前开发需要查询和广播操作，下面会详细介绍。

分接口说明

该部分以使用JS辅助库为例进行说明，其他方式会后续补充。

注册用户

注册用户是一个典型的需要广播来修改链上信息的交易，用以创建链上新的用户。其他需要广播的交易步骤也与此类似。

```
    let tr = new TransactionBuilder(); // 构造一个交易体，TransactionBuilder可从cybexjs库中获得。
    tr.add_type_operation("account_create", { // 向交易中添加所需广播的操作，操作名称为 account_create.
      fee: {
          amount: 0,
        asset_id: "1.3.0"
      }, // 该笔交易的手续费字段。
      registrar: chain_registrar.get("id"), // 注册人ID
      referrer: chain_referrer.get("id"), // 引荐人ID
      referrer_percent: referrer_percent, // 引荐人返利额度
      name: new_account_name, // 用户名
      owner: { // 用户的Owner权限设置
        weight_threshold: 1, // Owner权限的域限
        account_auths: [], // 占有权限的用户列表
        key_auths: [[owner_pubkey, 1]], // 占有权限的公钥列表
        address_auths: [] // 占有权限的地址列表
      },
      active: { // 活跃权限配置
        weight_threshold: 1,
        account_auths: [],
        key_auths: [[active_pubkey, 1]],
        address_auths: []
      },
      options: { //其他配置
        memo_key: active_pubkey, // Memo公钥，其他人向该用户发送Memo时，将使用该公钥进行加密
        voting_account: "1.2.5", // 默认投票代理
        num_witness: 0, // 初始投票的证人数量
        num_committee: 0, // 初始投票的委员会数量
        votes: [] //初始投票的ID集合
      }
    });
    tr.add_signer(privKey); // 向交易添加签名，传入私钥
    await tr.broadcast(); // 广播交易，异步操作
```

字段说明：

- 本接口数据结构可查询 https://github.com/NebulaCybexDEX/cybex-core/blob/master/libraries/chain/include/graphene/chain/protocol/account.hpp 获得基本说明。
- **fee** 所有需要广播的交易都需要缴纳手续费，也都需要该字段。消耗的手续费会部分燃烧，燃烧的手续费进入系统预算池。
    - `asset_id` 用来指定用来缴纳手续费的资产，默认是1.3.0，即核心资产(CYB)。

- `amount` 用来填写缴纳手续费数量，如果填 0，则在广播时链上会自动计算最小所需的手续费金额并扣除。一旦手续费不足，交易广播会失败。如果需要在事前获取交易费细节，可以调用*database_api*中的*get_required_fees*接口获取该交易的所需手续费。

注意：

- 手续费可以填写0或高于最低手续费的任意值，多出的手续费会被系统销毁。
- 链上所有费用的计数使用整数，即*amount*形式，其价值*value*/*price*遵循 `value = amount / 10^precision` 的计算，*precision*为资产的精度。

- **registrar/referrer** 注册人和引荐人ID，注册人用来缴纳此次注册所需的手续费，引荐人用来参与此用户未来所有操作时缴纳的手续费分红。

- **referrer_percent** 被注册用户的每笔手续费，除系统收取并燃烧外，应该由注册人和推荐人作为收益获取。该字段则用来规定收益在注册人与引荐人之间的分成比例，即多少百分比的收益分给引荐人，其他则分给注册人。

- **name** 被注册用户用户名。Cybex中用户名分为基本用户名和高级用户名，高级用户名手续费较高。

- **active/owner** 用户的活跃/账户权限配置。

  - 对于Cybex上的每一项交易，都需要有一个执行人，交易验签时，需要确认该交易中所包含的签名的阈值总和，达到了执行人权限阈限，交易才可执行。
  - active/owner即是两种不同的权限，其中acitve用来执行日常的交易签名，owner备用，用来调整账户权限本身。
  - 以active权限为例，每个权限可以由三种持有形式，分别是账户形式- `account_auths` 字段，公钥形式- `key_auths` 字段，地址形式- `address_auths` 字段。字段值的类型为Vector<account/key/address, 阈值>， `weight_threshold` 字段为执行操作的有效阈限。执行操作时，会验证每一个签名对应的持有者阈值的加和是否满足阈限。
  - 默认新注册情况下，应该如上例一样，使用由seed生成的公钥作为持有者，生效阈限为1，该公钥签名阈值为1。即，当该用户作为发起者发起交易时，需要一个该publickey对应的签名即可生效。publicKey可以由cybexjs库的PrivateKey生成，具体方式可以参考CybexDaemon类。

- **options** 该字段为账户选项，具体如下

  - `memo_key` : 若某账户B向该用户A转账时，如果需要发送附言，则B先去链上查找用户A的本字段，与发送者的私钥共同对信息进行AES加密。当A收到后，利用 `memo_key` 对应的私钥可以解密信息。
  - `voting_account` : 该用户的投票代理人。1.2.5为默认系统代理，即用户独立投票。
  - `num_witness/num_committee` : 初始建立时投票的见证人/委员会数量。
  - `votes` : 初始建立时的投票对象

用户登录

在Cybex中，用户并没有传统意义上的登录行为，用户和节点之间，并不形成会话关系，不保存彼此状态。所以Cybex登录实际上是一种**伪登录**。

现有Web客户端云账户登录逻辑如下：

用户输入用户名——API获取该用户链上信息并取出active/owner的key_auths——用户输入密码，点击登录——根据用户所输入密码生成私钥及对应公钥——验证用户公私钥对应情况——若有对应，则判断登录成功，订阅该用户，更新用户登录信息。

当用户发起一笔交易时，如前所述构造交易结构体，使用active权限对应私钥进行签名。

获取用户信息可以使用 `get_accounts` / `get_full_accounts` / `get_account_by_name` 接口进行查询。

用户支付

应用中的用户支付下单，将对应Cybex中的**做空单**操作。每一次做空单操作，由两个自操作组成： `call_order_update` 和 `limit_order_create` ，分别对应**通过抵押换取二元资产**和使用所有借入资产**挂出限价单**。示例代码如下：

```
let tr = new TransactionBuilder();
let order = { // 限价单操作
  fee: { // 挂限价单费用
    asset_id: "1.3.0",
    amount: 0
  },
  seller: daemon.daemonAccountInfo.get("id"), // 挂单用户ID
  amount_to_sell: { // 挂单资产数量
    asset_id,
    amount
  },
  min_to_receive: { // 欲换取的最小买入资产数量
    asset_id,
    amount
  },
  expiration: parseInt((Date.now() + expirationDuration) / 1000), // 该挂单失效时间
  fill_or_kill: false, // 是否需要整单撮合
};
tr.add_type_operation("call_order_update", {
  fee: {
    amount: 0,
    asset_id: "1.3.0"
  },
  funding_account: order.seller,  // 抵押借出资产人
  delta_collateral: { // 抵押资产数量
    amount: 1000000000,
    asset_id: "1.3.0"
  },
  delta_debt: order.amount_to_sell, // 借出的资产数量
  expiration: order.getExpiration() // 超时时间
});

tr.add_type_operation("limit_order_create", order.toObject());
tr.add_signer(privKey); // 向交易添加签名，传入私钥
await tr.broadcast(); // 广播交易，异步操作
```

字段说明：

- **seller/funding_account** 下单人ID/资产借出人ID。在做空单中两个ID保持一致。
- **delta_collateral** 进行抵押的资产及数量。用来进行抵押的资产，必须是核心资产或智能资产。
- **amount_to_sell/delta_debt** 希望借入并卖出的资产数量。
- **min_to_receive** 限价单中基于卖出资产数量，想要换取的最少的对标资产的数量。**Cybex中，每个交易对的价格是用两个资产value的比值得到的。**