Comunion Sprint01 文档

李泽辉 *

2020 年 7 月 5 日

Contents

1 登录鉴权流程

本系统使用 metamask 第三方账号作为登录验证识别身份，以此来保护 API 接口，登录鉴权流程如figure1所示

用户点击登陆后，代码如 listing1

```
/**
 * @description 点击连接按钮
 */
async connectAccount() {
  this.loading = true;
  try {
    await this.$store.dispatch('login');
  } catch (error) {}
  this.loading = false;
}
```

Listing 1: 参考文件 HeaderConnect.vue

*王帅（后端），王欣（合约）

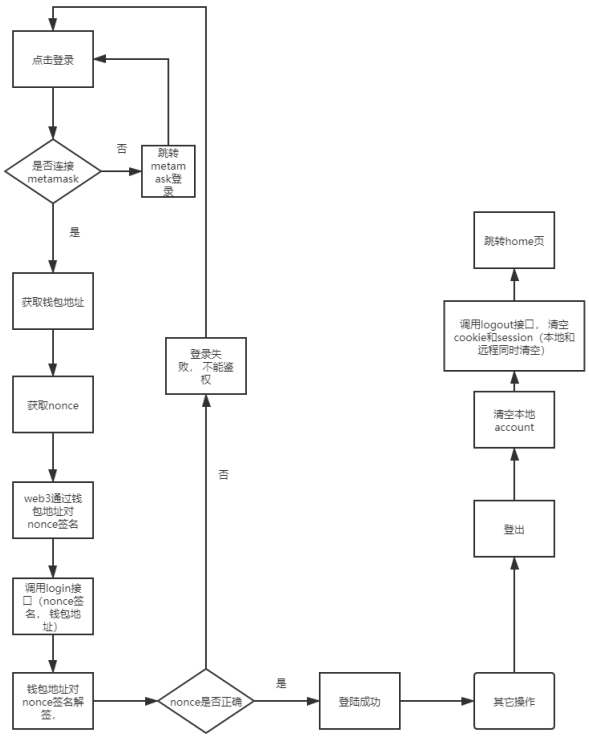

Figure 1: 登录鉴权

dispatch 转入 vuex 判断 metamask 插件是否被引入当前浏览器，如果浏览器没有引入 metamask, 则跳转 metamsk 官网下载页, 如果已安装，则吊起 metamask 登录，登录 metamask 后，前端获取到用户钱包；调用接口获取 nonce, 使用钱包地址对 nonce 签名，发给后端，后端使用钱包地址对 nonce 签名解签，解签结果与后端传给前端的 nonce 一至，则认为用户已经登陆了 comunion 系统，完成身份验证，代码如 listing2

```
/**
  * descriptions 用户登录
  * @param dispatch
  * @param commit
  * @returns {Promise<boolean>}
  */
 async login({ dispatch, commit }) {
   // 检查用户是否安装了metamask
   if (!!window.ethereum && window.ethereum.isMetaMask) {
     try {
       // 调用登录metamask
       const accounts = await ethereum.enable();
       // 取第个账号1
       const account = accounts[0].toLowerCase();
       // 初始化web3
       initWeb3();
       // 获取nonce
       const nonce = await getNonce(account);
       // 获取当前连接网络的ID
       const netWorkId = await web3.eth.net.getId();
       commit('HANDLE_NEW_NETWORK', netWorkId + '');
```

```
22
23          if (nonce) {
24            // 对签名nonce
25            let signature = await web3.eth.personal.sign(nonce, account);
26            // 登录
27            const ret = await login({ publicKey: account, signature });
28            if (!ret) {
29              throw new Error('Error when login, please try agian');
30            }
31            commit('SET_USER', ret);
32            // 注册事件metamask
33            // FIXME: 之前需要先onoff
34            ethereum.autoRefreshOnNetworkChange = true;
35            ethereum.on('chainIdChanged', arg => commit('HANDLE_NEW_CHAIN', arg));
36            ethereum.on('networkChanged', arg => commit('HANDLE_NEW_NETWORK', arg));
37            ethereum.on('accountsChanged', accounts => commit('UPDATE_ACCOUNT', accounts[0]));
38          } else {
39            throw new Error('Error when get nonce, please try again.');
40          }
41          commit('UPDATE_ACCOUNT', account);
42
43          commit('UPDATE_LOGIN_TIME', new Date().getTime());
44
45          /** 不刷新网页的情况下,  个小时自动登出6 */
46          setTimeout(() => {
47            dispatch('logout');
48          }, 2 * 60 * 60 * 1000);
49        } catch (error) {
50          message.error(error?.message || 'Error occured.');
51        }
52        return true;
53      } else {
54        // 打开官网metamask
55        window.open('https://metamask.io/');
56        message.warning('You have not installed metamask.');
57      }
58    },
```

Listing 2: 参考文件 user.js

2 startup

startup 的创建是 Comunion 的第一步， 包括创建和上链两步， 点击创建 startup，表单提交，会根据提交的数据创建合约，然后吊起 metamask, 等待签名确认后返回交易 hash，签名请求发出后，把交易 hash 和 startup 的数据一起提交给后端. 流程图如 fig2

Table 1: 上链状态表

	0	1	2	3
startup 状态 (字段：state)	无意义状态	待确认	上链成功	未确认到交易
setting 状态 (字段：settingState)	无意义状态	待确认	上链成功	未确认到交易

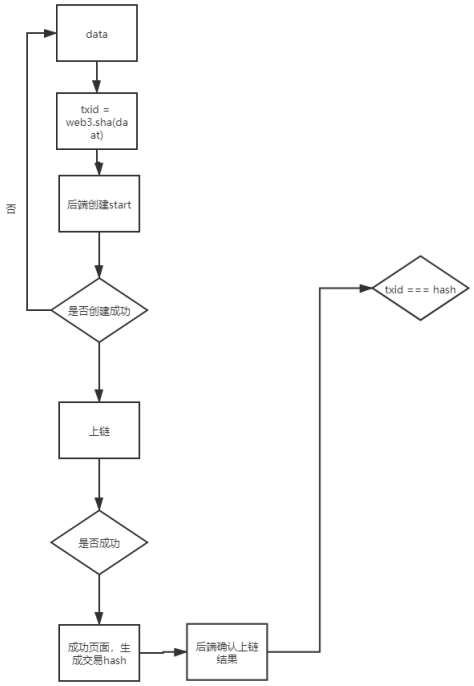

Figure 2: 上链的过程

后端会根据交易 hash 查询上链结果. 更新 startup 的状态,[1]，状态表见 table1. 步骤如下：

1. 提交表单，获取 startup 的 id, 根据合约 abi 创建合约实例
2. 发起交易
3. 交易发出后，返回交易 hash
4. 提交 startup 表单数据和 hash
5. 后端根据 hash 查询上链结果

```
/**
     *@description 提交表单
     */
    onSubmit() {
      if (this.canNotTransaction) {
        return;
      }
      this.$refs.ruleForm.validate(async valid => {
```

[1]由于创建合约实例，需要获取 startup 的 id, 但是创建 startup 需要合约发起交易后的交易 hash, 因此，startup 的 id 在还未创建前，调用接口获取，在创建合约实例和创建 startup 的时候，都要携带 startup id

```
      if (valid) {
        try {
          this.createState = 'creating';
          const startupId = await getPrepareStartupId();
          if (startupId) {
            const id = startupId.id;
            this.ethSendTransaction(this.form, id);
          }
        } catch (e) {
          console.error(e);
        }
      }
    });
  },

  /**
   * @description 发起交易
   * @param formData: 表单数据startup
   * @param : startupId startup id
   * @returns {Promise<void>}
   */
  async ethSendTransaction(formData, startupId) {
    const contractStatpUp = await this.getContractInstance(formData, startupId);
    const codeData = await contractStatpUp.encodeABI();
    const countAll = await web3.eth.getTransactionCount(this.account, 'pending');
    const chainId = await web3.eth.getChainId();

    const tx = {
      from: this.account,
      to: COMUNION_RECEIVER_ACCOUNT,
      data: codeData,
      value: web3.utils.numberToHex(Math.pow(10, 17)),
      nonce: web3.utils.numberToHex(countAll),
      gasPrice: web3.utils.numberToHex(Math.pow(10, 9)),
      gasLimit: web3.utils.numberToHex(183943),
      chainId: chainId
    };
    window.ethereum.sendAsync(
      {
        method: 'eth_sendTransaction',
        params: [tx],
        from: window.ethereum.selectedAddress
      },
      (err, result) => {
        if (err) {
          return console.error(err);
        }
```

```
          const txid = result.result;
          this.createStartUp(formData, startupId, txid);
        }
      );
    },

    /**
     * @description 构建hex, 生成txid
     * @param formData
     * @param startupId
     * @param txid
     */
    async createStartUp(formData, startupId, txid) {
      try {
        if (this.isEdit) {
          // 更新
          const startUp = await updateStartup(this.$router.query.id, { ...formData, txid });
          if (startUp) {
            this.createState = 'successed';
          }
        } else {
          // 后端创建startup
          const startUp = await createStartup({ ...formData, txid, id: startupId });
          if (startUp) {
            this.createState = 'successed';
          }
        }
      } catch (e) {
        console.log('%c\n  e :::----->', 'font-size:30px;background: purple;', e);
      }
    },
```

Listing 3: 参考文件 New.vue

3 setting

setting 的设置，是 comunion 的第二步，点击 setting 后，可以查看到，刚才创建的 startup 状态，如果此时后端通过交易 hash 在链上查询到了交易，则 startup 的状态是 waitting setting, 点击这个 startup 就可以开始 setting 了。

setting 设置完成后，提交数据，会根据提交的数据创建合约，然后吊起 metamask, 等待签名确认后返回交易 hash，签名请求发出后，把交易 hash 和 startup 的数据一起提交给后端，后端会根据交易 hash 查询上链结果，与 start up 的创建类似！步骤如下：

1. 提交表单 setting 表单，根据合约 abi 创建合约实例
2. 发起交易

3. 交易发出后，返回交易 hash

4. 提交 setting 表单数据和 hash, 后端创建 setting

5. 后端根据 hash 查询上链 setting 结果, 待后端确认到上链后，在 home 页面，可以看到完成后的自己创建的 comunion

```
async onOk() {
    // save data
    const body = {
      ...this.form.finance,
      ...{ ...this.form.governance }
    };
    // 时间转小时
    body.voteMinDurationHours = body.minDuration.days * 24 + body.minDuration.hours;
    body.voteMaxDurationHours = body.maxDuration.days * 24 + body.maxDuration.hours;
    delete body.maxDuration;
    delete body.minDuration;

    body.voteTokenLimit = body.voteTokenLimit ? body.voteTokenLimit : -1;

    const id = this.$route.params.id;
    this.ethSendTransaction(body, id);
    // 关闭loading
    this.$refs.launch.loading = false;
  },

  /**
   * @description 发起交易
   * @param formData
   * @param id
   * @returns {Promise<void>}
   */
  async ethSendTransaction(formData, id) {
    const contractStatpUp = await this.getContractInstance(formData, id);
    const codeData = await contractStatpUp.encodeABI();
    const countAll = await web3.eth.getTransactionCount(this.account, 'pending');
    const chainId = await web3.eth.getChainId();

    const tx = {
      from: this.account,
      to: COMMUNION_SETTING_RECEIVE_ACCOUNT,
      data: codeData,
      value: 0,
      nonce: web3.utils.numberToHex(countAll),
      gasPrice: web3.utils.numberToHex(Math.pow(10, 9)),
      gasLimit: web3.utils.numberToHex(183943),
      chainId: chainId
```

```
    };
    window.ethereum.sendAsync(
      {
        method: 'eth_sendTransaction',
        params: [tx],
        from: window.ethereum.selectedAddress
      },
      (err, result) => {
        if (err) {
          return console.error(err);
        }
        const txid = result.result;
        this.createSetting(formData, txid);
      }
    );
  },

  /**
   * @description 获取合约实例
   * @returns {Promise<*>}
   */
  async getContractInstance(formData, id) {
    const data = JSON.parse(JSON.stringify(formData));
    const contract = new web3.eth.Contract(settingAbi, COMMUNION_SETTING_RECEIVE_ACCOUNT);
    const walletAddrs = data.walletAddrs.map(item => item.addr);
    let voteAssignAddrs = [];
    data.voteAssignAddrs.forEach(item => {
      voteAssignAddrs.push(item || '0x0000000000000000000000000000000000000000');
    });
    /** 发起合约 */
    const contractSetting = await contract.methods.newSetting(
      id,
      data.tokenName,
      data.tokenSymbol,
      data.tokenAddr,
      walletAddrs,
      data.voteType,
      // POS
      data.voteTokenLimit.toString(),
      // Founder assign
      voteAssignAddrs,
      data.voteSupportPercent.toString(),
      data.voteMinApprovalPercent.toString(),
      data.voteMinDurationHours.toString(),
      data.voteMaxDurationHours.toString()
    );
    return contractSetting;
```

```
    },
```

Listing 4: 参考文件 SettingDetail.vue

A 附录合约 ABI，创建合约实例的时候需要使用

A.1 startup ABI

```
/**
 * @description 合约的abi
 */
export const startupAbi = [
  {
    inputs: [],
    payable: false,
    stateMutability: 'nonpayable',
    type: 'constructor'
  },
  {
    constant: false,
    inputs: [
      {
        internalType: 'address payable',
        name: 'addr',
        type: 'address'
      }
    ],
    name: 'setCoinBase',
    outputs: [],
    payable: false,
    stateMutability: 'nonpayable',
    type: 'function'
  },
  {
    constant: false,
    inputs: [
      {
        internalType: 'string',
        name: 'id',
        type: 'string'
      },
      {
        internalType: 'string',
        name: 'name',
        type: 'string'
      },
```

```
      {
        internalType: 'string',
        name: 'categoryId',
        type: 'string'
      },
      {
        internalType: 'string',
        name: 'mission',
        type: 'string'
      },
      {
        internalType: 'string',
        name: 'descriptionAddr',
        type: 'string'
      }
    ],
    name: 'newStartup',
    outputs: [],
    payable: true,
    stateMutability: 'payable',
    type: 'function'
  },
  {
    constant: true,
    inputs: [
      {
        internalType: 'string',
        name: 'id',
        type: 'string'
      }
    ],
    name: 'getStartup',
    outputs: [
      {
        internalType: 'string',
        name: 'name',
        type: 'string'
      },
      {
        internalType: 'string',
        name: 'categoryId',
        type: 'string'
      },
      {
        internalType: 'string',
        name: 'mission',
        type: 'string'
      },
      {
        internalType: 'string',
        name: 'descriptionAddr',
        type: 'string'
      }
```

```
      },
      {
        internalType: 'string',
        name: 'descriptionAddr',
        type: 'string'
      }
    ],
    payable: false,
    stateMutability: 'view',
    type: 'function'
  }
];


```

Listing 5: (参考文件 startup.js)

A.2 setting ABI

```
 /**
 * @description setting 上链的api
 * @type {{}}
 */
export const settingAbi = [
  {
    inputs: [],
    payable: false,
    stateMutability: 'nonpayable',
    type: 'constructor'
  },
  {
    constant: false,
    inputs: [
      {
        name: 'id',
        type: 'string'
      },
      {
        name: 'tokenName',
        type: 'string'
      },
      {
        name: 'tokenSymbol',
        type: 'string'
      },
      {
```

```
        name: 'tokenAddr',
        type: 'string'
      },
      {
        name: 'walletAddrs',
        type: 'address[]'
      },
      {
        name: 'voteType',
        type: 'string'
      },
      {
        name: 'voteTokenLimit',
        type: 'string'
      },
      {
        name: 'voteAssignAddrs',
        type: 'address[]'
      },
      {
        name: 'voteMSupportPercent',
        type: 'string'
      },
      {
        name: 'voteMinApprovalPercent',
        type: 'string'
      },
      {
        name: 'voteMinDurationHours',
        type: 'string'
      },
      {
        name: 'voteMaxDurationHours',
        type: 'string'
      }
    ],
    name: 'newSetting',
    outputs: [],
    payable: true,
    stateMutability: 'payable',
    type: 'function'
  },
  {
    constant: true,
    inputs: [
      {
        name: 'id',
```

```
        type: 'string'
      }
    ],
    name: 'getTokenSetting',
    outputs: [
      {
        name: 'tokenName',
        type: 'string'
      },
      {
        name: 'tokenSymbol',
        type: 'string'
      },
      {
        name: 'tokenAddr',
        type: 'string'
      },
      {
        name: 'walletAddrs',
        type: 'address[]'
      }
    ],
    payable: false,
    stateMutability: 'view',
    type: 'function'
  },
  {
    constant: true,
    inputs: [
      {
        name: 'id',
        type: 'string'
      }
    ],
    name: 'getVoteSetting',
    outputs: [
      {
        name: 'voteType',
        type: 'string'
      },
      {
        name: 'voteTokenLimit',
        type: 'string'
      },
      {
        name: 'voteAssignAddrs',
        type: 'address[]'
```

```
      },
      {
        name: 'voteMSupportPercent',
        type: 'string'
      },
      {
        name: 'voteMinApprovalPercent',
        type: 'string'
      },
      {
        name: 'voteMinDurationHours',
        type: 'string'
      },
      {
        name: 'voteMaxDurationHours',
        type: 'string'
      }
    ],
    payable: false,
    stateMutability: 'view',
    type: 'function'
  }
];
```

Listing 6: (参考文件 setting.js)