基于vue开发的以太坊开源HD钱包vuethwallet源码分析

- Author : parsons

导读:随着区块链越来越火热，数字货币钱包也随之被人重视。数字钱包的发展历程是从最初比特币的非确定性钱包，到确定性钱包，一直到我们现在最广为使用HD分层确定性钱包，而今天要分析的钱包就是现在最为流行的HD分层确定性钱包

HD钱包的英文全称是：Hierarchical Deterministic 之所以叫分层确定性钱包是因为私钥的衍生结构是树状结构，父密钥可以衍生一系列子密钥，每个子密钥又可以衍生出一系列孙密钥，以此类推，无限衍生。在创建钱包或者备份钱包的时候都会看到一堆英文单词（或者中文汉字），这些词就是助记词。

0x01 钱包介绍

钱包操作流程gif

1.1 功能

- Vuethwallet => 普通钱包,随机生成助记
- Seed Wallet => 自定义助记词钱包
- Import Wallet => 导入钱包(导入Keysotre)
- Value Transaction => 导入钱包并进行交易

1.2 特色

> 该钱包的特点就是简单，只具备一个数字钱包最基本的功能:密钥管理(私钥的 生成 + 导入 + 备份)和交易。

- 密钥生成主要使用原生的bip-39库生成助记词(可自定义助记词或随机助记词)然后进一步生成私钥

> 因为使用的是原生的库进行私钥生成，私钥生成过程会比较接触底层

- 备份使用Keystore进行保存

> 以太坊的 keystore文件Linux 系统存储在/home_path/.ethereum/keystore或者Windows系统存储在C:\Users\Appdata/Roaming/Ethereum/keystore）是你独有的、用于签署交易的以太坊私钥的加密文件。如果你丢失了这个文件，你就丢失了私钥，意味着你失去了签署交易的能力，意味着你的资金被永久的锁定在了你的账户里。

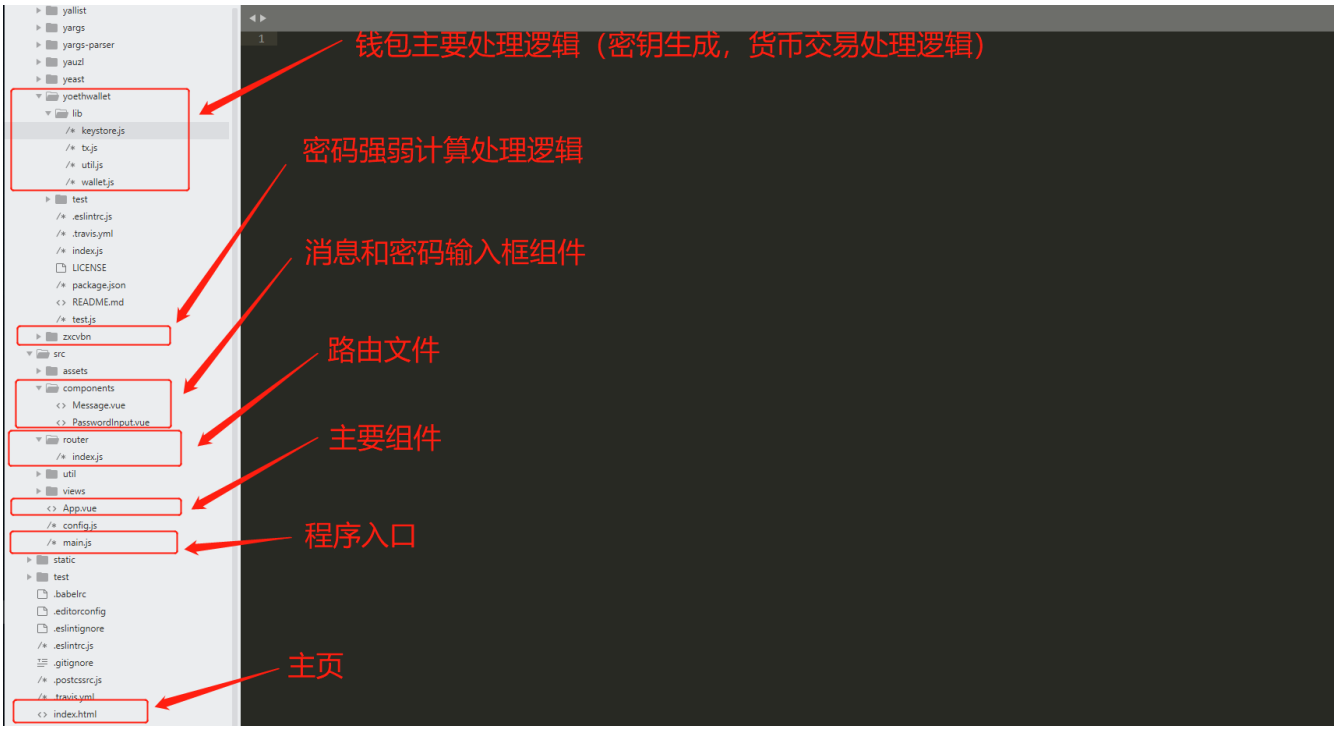
这里是一个Keystore例子

```
{
    "crypto" : {
        "cipher" : "aes-128-ctr",
        "cipherparams" : {
            "iv" : "83dbcc02d8ccb40e466191a123791e0e"
        },
        "ciphertext" : "d172bf743a674da9cdad04534d56926ef8358534d458fffccd4e6ad2fbde479c",
        "kdf" : "scrypt",
        "kdfparams" : {
            "dklen" : 32,
            "n" : 262144,
            "r" : 1,
            "p" : 8,
            "salt" : "ab0c7876052600dd703518d6fc3fe8984592145b591fc8fb5c6d43190334ba19"
        },
        "mac" : "2103ac29920d71da29f15d75b4a16dbe95cfd7ff8faea1056c33131d846e3097"
    },
    "id" : "3198bc9c-6672-5ab3-d995-4942343ae5b6",
    "version" : 3
}
```

- 交易使用web3.jsAPI进行交易

> web3.js是以太坊提供的一个Javascript库，它封装了以太坊的JSON RPC API，提供了一系列与区块链交互的Javascript对象和函数，包括查看网络状态，查看本地账户、查看交易和区块、发送交易、编译/部署智能合约、调用智能合约等
>
> > 该钱包交易功能比较简单，没有一些完备钱包所具有的功能比如余额查询，Gas价格查询等,但足以学习所用

0x02 钱包的代码架构



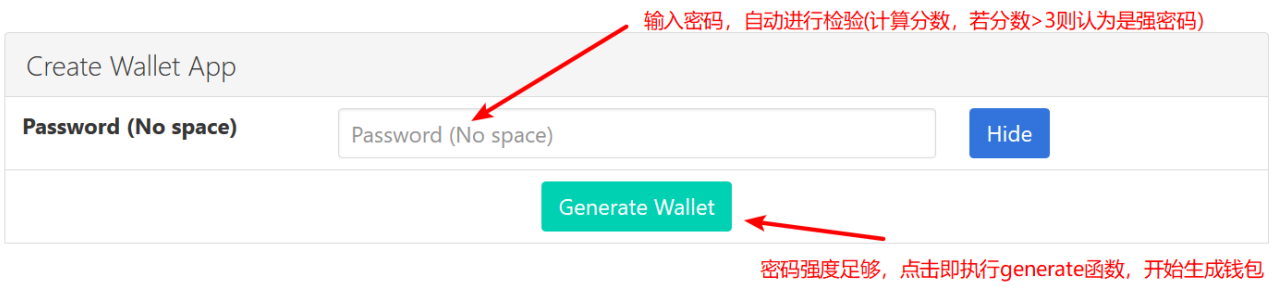
主要文件

> 本文主要对钱包创建以及交易处理部分进行详细分析,其他代码有兴趣可自行深入

1. index.html ==> 主页,只有一个大框架,并无实际内容,靠js进行填充
2. /src/views/main.js ==> 入口主程序,将App.vue插入到index.html
3. /src/views/App.vue ==> 主要组件,响应用户点击并路由到相应的页面
4. /src/view/Wallet.vue ==> 创建普通钱包
5. /src/view/WalletSeed.vue ==> 创建带seed钱包
6. /src/view/ImportKeystore.vue ==> 导入钱包(Keystore)
7. /src/view/ValueTransaction.vue ==> 交易处理
8. /util/confirmedTransaction.js ==> 交易确认逻辑
9. /node_modlues/zxcvb ==> 密码强度校验(分数计算)

0x03 代码分析

3.1 /src/view/Wallet.vue ==> 创建普通钱包(随机助记词)



3.1.1 generate ==> 主函数:校验密码强度 + 生成私钥并返回 + 页面渲染 + keystore生成

主要函数

1. wallet.generate => 生成私钥并返回
2. newAddress => 页面渲染+keystore生成

```
generate (callback) {
  if (!this.password) {          /*首先判断用户传入的密码是否为空 */
    this.error = true
    this.msg = 'Please enter password!'
    return
  }
  if (this.score < 3) {
  /* 计算该密码强度,若得分<3 则提示用户修改更加strong的密码
  计算逻辑在  node_modules/zxcvb/  */
    this.error = true
    this.msg = 'Password is not strong, please change!'
    return
  }
  if (!callback || typeof callback !== 'function') {
    callback = function () {}
  }
  let wallet = yoethwallet.wallet
  /* 一切准备就绪,通过加密算法获得私钥
     生成的是HD钱包(bip039生成助记词==> bip032生成根seed ==> bip044定义HDpath) */

wallet.generate('', this.hdPathString, (err, keystore) => {
  /* 开始进行钱包生成，传入一个助记词，返回一个keystore
     参数分析:
     '' ==>助记词,因为创建的是普通钱包,则为空,自动生成随机助记词
     this.hdPathString ==> HD钱包密钥识别符(路径)
     (err,keystore) ==> callback函数,将keystore返回 */
    if (err) {
      console.warn(err.message)
      return
    }
    this.keystore = keystore
    /* 将生成的keystore对象返回 */
    this.newAddress(this.password, callback)
  /*将generate出来的私钥,公钥等信息返回并赋值到页面上绑定的变量并展示给用户,生成keystore文件提供给用户下载*/
```

```
}
}
```

3.1.2 wallet.generate ==> 钱包私钥生成函数

私钥生成步骤

1. bip39.generateMnemonic() ==> 生成助记词(只是将私钥的进行可读化显示,并不是`brainwallets`)
2. bip39.mnemonicToSeedHex(randomSeed) ==> 助记词生成`根seed`
3. HDKey.fromMasterSeed(randomSeed) ==> 根seed生成`私钥`和`chainCode`

```
wallet.generate = function (randomSeed, hdPath, callback) {
 /* 无seed钱包 ==> randomSeed='' */
  try {
    if (!randomSeed || !this.validSeed(randomSeed)) {
      randomSeed = bip39.generateMnemonic();/* 若传入的randomSeed检测到不存在或者非法,
      自动生成一个randomSeed(''为不存在) */
    }
    randomSeed = bip39.mnemonicToSeedHex(randomSeed);/*通过助记词转化为种子,
    用于使用BIP-0032或类似的方法生成确定性钱包*/
    hdPath = (this.validHdPath(hdPath)) ? hdPath : defaultHDPath;
    const hdKey = HDKey.fromMasterSeed(randomSeed);/*通过生成的种子进一步生成私钥*/
    const _hdKey = hdKey.derive(hdPath); /* 利用hdPath导出子私钥*/
    this.fromHdKey(_hdKey, callback); /* 通过私钥导出keysotre(生成一个Keystore对象)*/
  } catch (err) {
    callback(err, null);
  }
```

3.1.2.1 bip39.generateMnemonic()函数分析 ==> 生成助记词

```
助记词生成步骤:
1. 创造一个128到256位的随机顺序(熵,熵的长度=> ENT)
2. 提出SHA256哈希前(ENT/32)位,就可以创造一个随机序列的校验和(校验和的长度为ENT/32)
3. 把校验和加在随机顺序的后面
4. 把顺序分解成11位的不同集合,并用这些集合去和一个预先已经定义的2048个单词字典做对应
5. 生成一个12至24个词的助记词
```

```
/*1. 创造一个128到256位的随机顺序(熵)*/
function generateMnemonic (strength, rng, wordlist) {
  strength = strength || 128 /* 默认生成128位的熵 */
  if (strength % 32 !== 0) throw new TypeError(INVALID_ENTROPY)
  rng = rng || randomBytes

  var hex = rng(strength / 8).toString('hex')/* 熵 */
return entropyToMnemonic(hex, wordlist)/* 通过熵生成助记词 */
  }
```

```
 function entropyToMnemonic (entropyHex, wordlist) {
  wordlist = wordlist || DEFAULT_WORDLIST/* 字典 */

  // 128 <= ENT <= 256/* ENT => 128 <= 熵长度 <= 256) */
```

```
  if (entropyHex.length < 32) throw new TypeError(INVALID_ENTROPY)
  if (entropyHex.length > 64) throw new TypeError(INVALID_ENTROPY)

  // multiple of 4
  if (entropyHex.length % 8 !== 0) throw new TypeError(INVALID_ENTROPY)
  /*2. 提出SHA256哈希前(ENT/32)位,就可以创造一个随机序列的校验和 */
  var entropy = Buffer.from(entropyHex, 'hex')
  var entropyBits = bytesToBinary([].slice.call(entropy))
  var checksumBits = deriveChecksumBits(entropy)

/*3. 把校验和加在随机顺序的后面*/
  var bits = entropyBits + checksumBits

/*4. 把顺序分解成11位的不同集合,并用这些集合去和一个预先已经定义的2048个单词字典做对应
  5. 生成一个12至24个词的助记词 */
  var chunks = bits.match(/(.{1,11})/g)
  var words = chunks.map(function (binary) {
    var index = binaryToByte(binary)
    return wordlist[index]
  })

  return wordlist === JAPANESE_WORDLIST ? words.join('\u3000') : words.join(' ')
}
```

3.1.2.2 bip39.mnemonicToSeedHex(randomSeed)函数分析 ==> 助记词生成根seed

步骤:
1.若用户想使用密码来保护助记符,则可以提供password参数,该钱包默认为''
2.用带有用作密码的助记符句子(以UTF-8 NFKD表示)的PBKDF2函数和用作盐的字符串"助记符"+密码(再次以UTF-8 NFKD表示).
迭代计数设置为2048,HMAC-SHA512用作伪随机函数.派生密钥的长度为512位(64字节).

```
function mnemonicToSeed (mnemonic, password) {
  /* 步骤1 password = '' 默认不保护*/
  var mnemonicBuffer = Buffer.from(unorm.nfkd(mnemonic), 'utf8')
  var saltBuffer = Buffer.from(salt(unorm.nfkd(password)), 'utf8')

  /*步骤2*/
  return pbkdf2(mnemonicBuffer, saltBuffer, 2048, 64, 'sha512')
}
```

3.1.2.3 HDKey.fromMasterSeed(randomSeed)函数分析 ==> 根seed生成私钥和chainCode

```
    HDKey.fromMasterSeed = function (seedBuffer, versions) {
      var I = crypto.createHmac('sha512', MASTER_SECRET).update(seedBuffer).digest()
      /*HMAC-SHA512(512bit输出)单向哈希函数*/
      var IL = I.slice(0, 32) /* 获取私钥 ==> 256bit*/
      var IR = I.slice(32) /* 获取链编码(chain code) ==> 256bit */
      var hdkey = new HDKey(versions)
      hdkey.chainCode = IR
      hdkey.privateKey = IL
```

3.1.3 newAddress(password,callback) ==> 页面展示 + keystore文件生成

Keystore文件生成步骤

1. 用户传入password,使用密钥生成函数(:kdf => "`scrypt`")计算加密密钥 => encryption key
2. 利用加密密钥对私钥进行加密(:cipher => "`aes-128-ctr`")得到私钥加密后密文 => ciphertext
3. 通过加密密钥(左第二个字节起的16个字节)和ciphertext连接在一起进行哈希散列(`SHA3-256`)计算得出校验值 => mac

```
newAddress (password, callback) {
if (typeof this.keystore.getHexAddress !== 'function') {
return false /* 验证密钥是否生成成功 */
}
let wallet = this.keystore /* keystore对象赋值,以便后续生成keystore文件 */
this.error = false
this.msg = 'Wallet create successfully!' /*密钥生成成功提示信息*/
this.privateKey = wallet.getHexPrivateKey() /* 更新私钥,展示给用户 */
this.address = wallet.getHexAddress(true) /* 更新地址,展示给用户*/
wallet.toV3String(this.password, {}, (err, v3Json) => { /*通过用户传入的password对私钥进行加密,
并生成keystore文件*/
if (err) {
  console.warn(err.message)
  return
}
this.keystoreJson = v3Json /* 传回keystore JSON数据 */
this.keystoreJsonDataLink = encodeURI('data:application/json;charset=utf-8,' +
this.keystoreJson) /* 生成下载文件 */
this.fileName = `${wallet.getV3Filename()}.json` /* 生成文件名字 */
callback()
})
}
--------------------------------------------------------------------------------
/*keystore.prototype.toV3 ==> 通过用户传入的password对私钥进行加密,并生成keystore文件*/
--------------------------------------------------------------------------------
keystore.prototype.toV3 = function (password, options, callback) {
try {
if (!this._privateKey) {
  throw new Error('Please generate wallet with private key.');
}/* 先检查私钥是否生成 */
options = options || {}; /*选项 默认为空 */

const cipherAlgorithm = options.cipher || 'aes-128-ctr'; /*用以加密以太坊私钥的强对称加密算法默
认:aes-128-ctr*/
```

```
const salt = options.salt || crypto.randomBytes(32);/* 盐值 */
const iv = options.iv || crypto.randomBytes(16);/* aes-128-ctr加密所用到的初始化向量 */
const id = uuid.v4({ random: options.uuid || crypto.randomBytes(16) });/*id*/
const kdf = options.kdf || 'scrypt';/*指定密钥派生函数,默认scrypt*/
const kdfparams = {
    /*kdf密钥生成时所需要的参数*/
  dklen: options.dklen || 32,
  salt: salt.toString('hex'),
};
const cb = function (derivedKey) {
    /*callback函数 利用kdf生成完加密密钥之后调用该函数进一步利用加密密钥生成ciphertext*/
  derivedKey = new Buffer(derivedKey);

  let cipher = crypto.createCipheriv(cipherAlgorithm, derivedKey.slice(0, 16), iv);/*准备加
密算法*/

  if (!cipher) {
    callback(new Error('Unsupported cipher algorithm.'), null);
    return;
  }

  const ciphertext = Buffer.concat([ cipher.update(this.privateKey), cipher.final() ]);
  /*利用kdf密钥生成函数生成的加密密钥对私钥加密得到私钥加密密文 ==> ciphertext*/
  const mac = ethUtil.sha3(Buffer.concat([ derivedKey.slice(16, 32), new Buffer(ciphertext,
'hex') ]));
  /*计算校验值mac*/

  const v3 = {
      /*拼接全部生成完毕的数据(即keystore)并返回*/
    version: 3,
    id: id,
    address: this.getHexAddress(),
    crypto: {
      ciphertext: ciphertext.toString('hex'),
      cipherparams: {
        iv: iv.toString('hex'),
      },
      cipher: cipherAlgorithm,
      kdf: kdf,
      kdfparams: kdfparams,
      mac: mac.toString('hex'),
    },
  };

  callback(null, v3);/*返回数据*/
}.bind(this);

if (kdf === 'pbkdf2') {
  kdfparams.c = options.c || 262144;
  kdfparams.prf = 'hmac-sha256';
  crypto.pbkdf2(new Buffer(password), salt, kdfparams.c, kdfparams.dklen, 'sha256',
function (err, derivedKey) {
    if (err) {
```

```
      callback(err, null);
      return;
    }
    cb(derivedKey);
  });
} else if (kdf === 'scrypt') {
    /*kdf ==> 生成加密密钥 */
  const saltUse = util.bufferToArray(salt);

  kdfparams.n = options.n || 262144;
  kdfparams.r = options.r || 8;
  kdfparams.p = options.p || 1;

  scrypt(password, salt, {N: kdfparams.n, r: kdfparams.r, p: kdfparams.p, dklen:
kdfparams.dklen, encoding: 'binary'}, cb);
  /* 利用上面计算出来的各种参数对用户输入的password加密生成加密密钥并调用cb函数生成ciphertext*/
} else {
  throw new Error('Unsupported key derivation function.');
}
} catch (err) {
callback(err, null);
}
```

3.2 /src/view/WalletSeed.vue ==> 创建钱包(自定义助记词)

> 在/Wallet基础上多传入一个Seed参数并增加一个Seed检验参数,基本原理和无seed钱包相同,只是跳过了助记词自动生成的过程,这里不再赘述

```
--------------------------------------------------------------------------------
/*助记词检验,实际上是助记词推导出熵的过程*/
--------------------------------------------------------------------------------
function mnemonicToEntropy (mnemonic, wordlist) {
  wordlist = wordlist || DEFAULT_WORDLIST

  var words = unorm.nfkd(mnemonic).split(' ')
  if (words.length % 3 !== 0) throw new Error(INVALID_MNEMONIC)

  // convert word indices to 11 bit binary strings
  var bits = words.map(function (word) {
    var index = wordlist.indexOf(word)
```

```
    if (index === -1) throw new Error(INVALID_MNEMONIC)

    return lpad(index.toString(2), '0', 11)
  }).join('')

  // split the binary string into ENT/CS
  var dividerIndex = Math.floor(bits.length / 33) * 32
  var entropyBits = bits.slice(0, dividerIndex)
  var checksumBits = bits.slice(dividerIndex)

  // calculate the checksum and compare
  var entropyBytes = entropyBits.match(/(.{1,8})/g).map(binaryToByte)
  if (entropyBytes.length < 16) throw new Error(INVALID_ENTROPY)
  if (entropyBytes.length > 32) throw new Error(INVALID_ENTROPY)
  if (entropyBytes.length % 4 !== 0) throw new Error(INVALID_ENTROPY)

  var entropy = Buffer.from(entropyBytes)
  var newChecksum = deriveChecksumBits(entropy)
  if (newChecksum !== checksumBits) throw new Error(INVALID_CHECKSUM)

  return entropy.toString('hex')
}
```

```
--------------------------------------------------------------------------------
/*传入助记词*/
--------------------------------------------------------------------------------
let wallet = yoethwallet.wallet

wallet.generate(this.randomSeed, '', (err, keystore) => {
                ^^^^^^^^^^^^^^^
if (err) {
  console.warn(err.message)
  return
}
this.keystore = keystore
this.newAddress(this.password, callback)
```

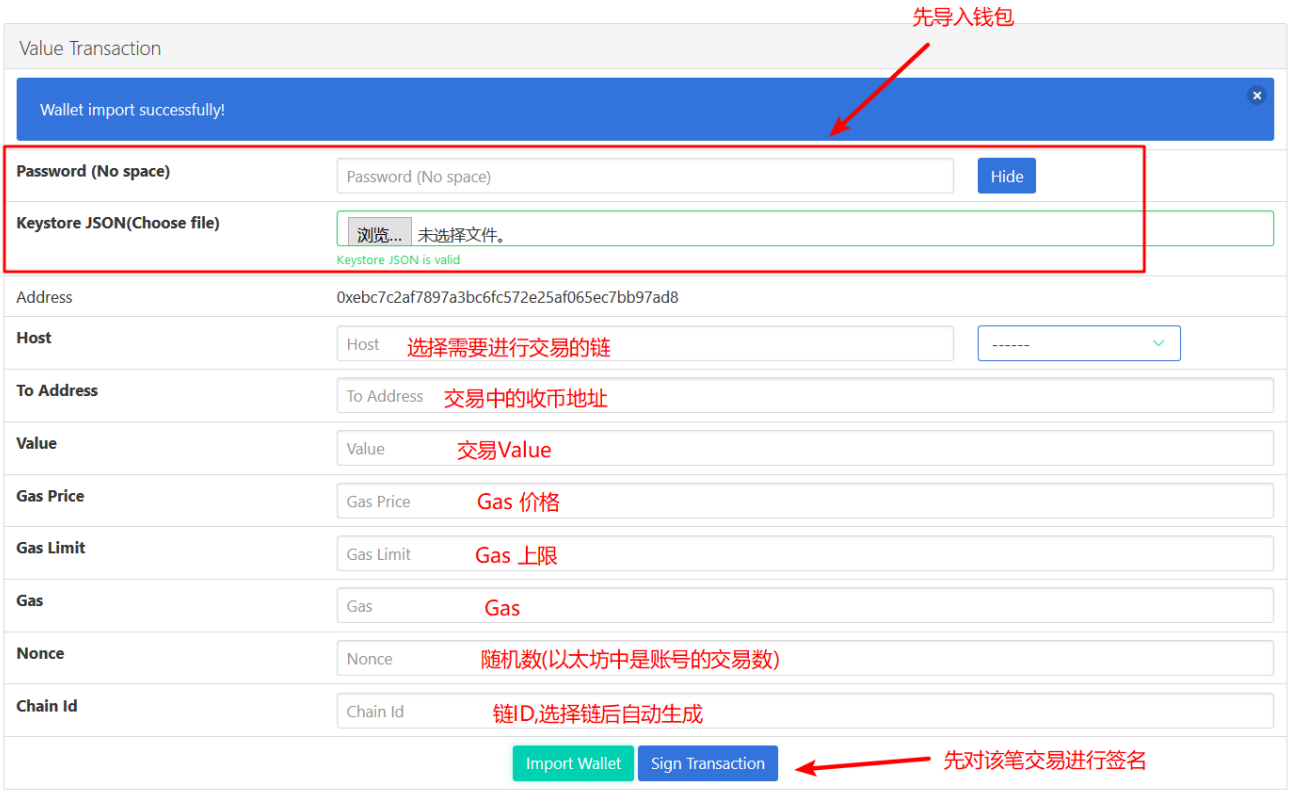
3.3 /src/view/ImportKeystore.vue ==> 导入Keystore 逻辑处理

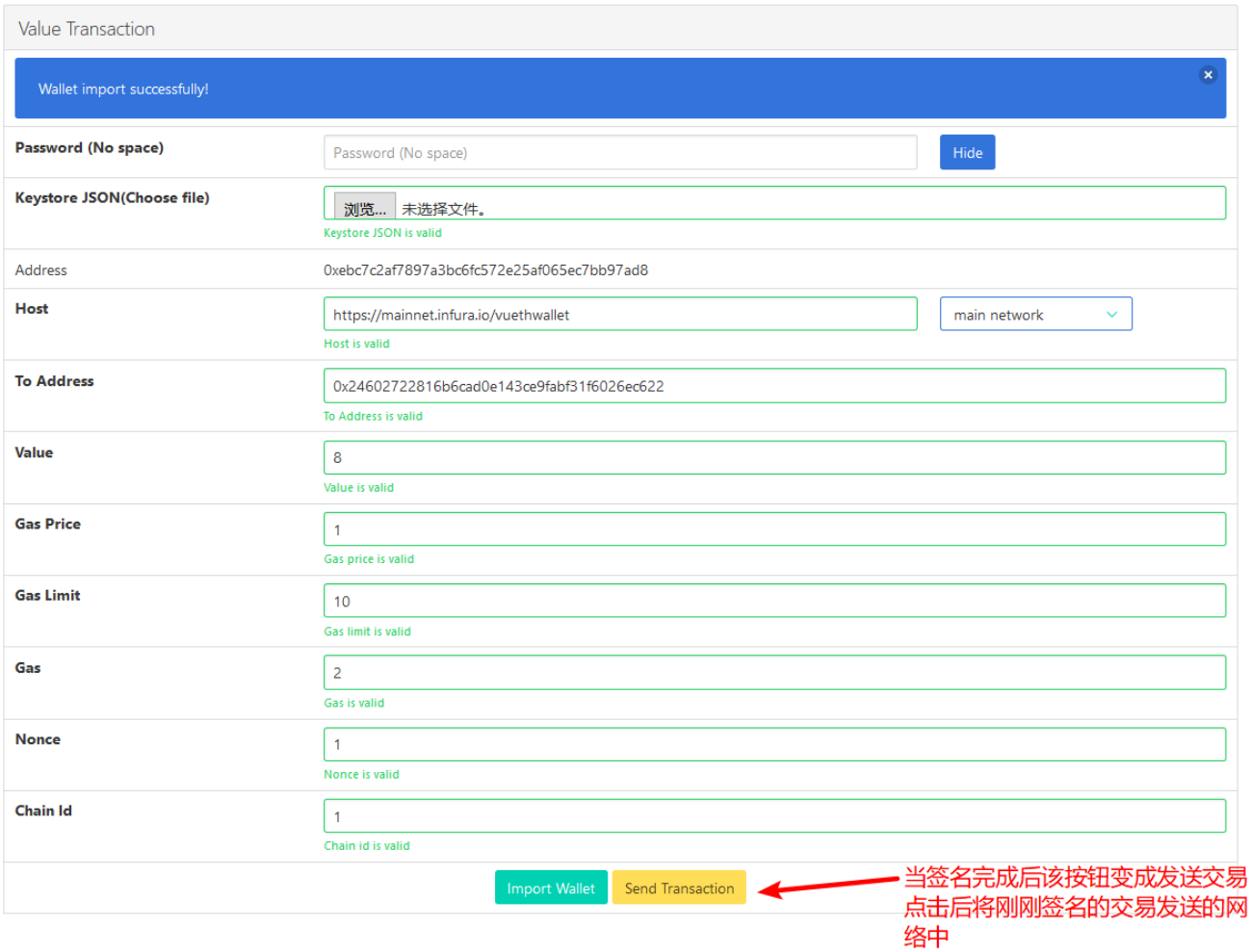
> 生成Keystore是将私钥利用加密密钥(kdf(password) => 加密密钥)将私钥加密的过程,而导入keystore是利用解密密钥(kdf(password) => 解密密钥) 将Keystore还原为私钥的过程,不再赘述

3.4 /src/view/ValueTransaction.vue ==> 交易逻辑处理

> **交易步骤**

1. 先导入Keystore(略)
2. 制造一笔交易并进行签名
3. 发送该笔已经签名的交易到网络
4. 确认交易





3.4.1 签名交易函数分析

```
signTransaction () {
  if (!this.host) {
    this.error = true
    this.msg = 'Please enter host'
    return
  }
  if (!this.toAddress) {
    this.error = true
    this.msg = 'Please enter to address'
    return
  }

  let valueTx = yoethwallet.tx.valueTx({from: this.address, to: this.toAddress, value:
this.val, nonce: this.nonce, gas: this.gas, gasPrice: this.gasPrice, gasLimit:
this.gasLimit, chainId: this.chainId})
  /* 利用用户输入的参数创造一笔交易 */

  valueTx.sign(this.keystore.getPrivateKey())
  /* 用私钥对交易进行签名 */

  this.signedTransaction = '0x' + valueTx.serialize().toString('hex')
  /* 获取已签名数据 */
}
```

3.4.2 发送交易到网络函数分析

```
sendTransaction () {
if (!this.host) {
this.error = true
this.msg = 'Please enter host'
return
}
if (!this.signedTransaction) {
this.error = true
this.msg = 'Please sign transaction first'
return
}
this.send = true

const web3 = this.web3/* 实例化web3对象 */

web3.eth.sendRawTransaction(this.signedTransaction, function (err, txId) {
/*发送一个已经签名的交易,返回一个32字节的16进制格式的交易哈希串=> txId*/
if (err) {
  this.send = false
  this.signedTransaction = ''
  this.error = true
  this.msg = 'Please sign transaction again'
  console.warn(err.message)
  return
}
```

```
this.result = txId

confirmedTransaction(web3, txId, function (err, tx) {
/* 确认交易是否完成 */

  this.send = false
  this.signedTransaction = ''

  if (err) {
    this.error = true
    this.msg = 'Please send transaction again'
    console.warn(err.message)
    return
  }
  console.log('Transaction confirmed')
})
}.bind(this))
```

3.4.3 确认交易函数分析

```
module.exports = exports = function (web3, txId, cb) {
  let confirmed = false /*交易是否确定的flat*/
  let limit = 5 /* 需要确定的次数 */
  let blockNumber = web3.eth.blockNumber/* 当前区块的Number */

  return whilst(
    function () {
      return confirmed === false
    },
    function (callback) {
      web3.eth.getTransaction(txId, function (err, tx) {
        if (err) {
          window.setTimeout(function () {
            callback(err, null)
          }, 1000)
        }
        if (tx && tx.blockNumber !== null) {
          if (blockNumber >= (tx.blockNumber + limit)) {
              /*设定判断条件,当前区块Number >= 交易区块的Number + limit  即认为该区块已经被确认 */
            confirmed = true/*设定Flat=>True*/
            window.setTimeout(function () {
              callback(null, tx)
            }, 1000)
            return
          }
        }
        window.setTimeout(function () {
          callback(null, null)
        }, 1000)
      })
    },
    function (err, tx) {
```

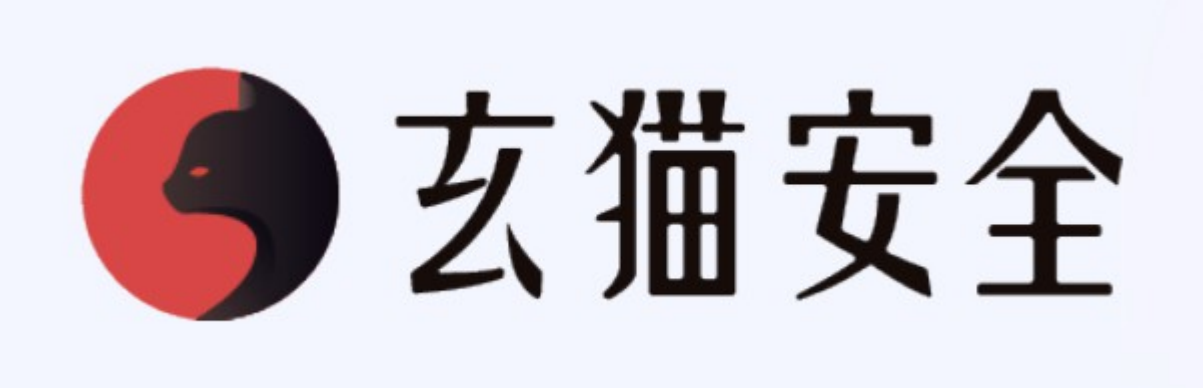
```
      if (err) {
        return cb(err, null)
      }
      if (tx && confirmed) {
        return cb(null, tx)
      }
    }
  )
}
```

> 结语:本文主要对该钱包的一些主要功能(私钥生成,keystore,交易)进行代码分析以及对代码逻辑捋顺。因笔者水平有限，没有太深入底层代码的研究，但通过对该钱包的逻辑理解，若以后要进行更'丰富'钱包开发会对理解钱包开发的各种库的原理有一定的帮助。因行文比较匆忙，若文中有理解不当之处欢迎致邮。

0x04 参考

- https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki
- https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
- http://web3.tryblockchain.org/Web3.js-api-refrence.html#toc_44
- https://en.wikipedia.org/wiki/Key_derivation_function
- https://ethfans.org/posts/what-is-an-ethereum-keystore-file

关于玄猫安全



玄猫区块链安全实验室专注区块链安全领域，致力于提供区块链行业最专业的安全解决方案，团队成员来自于百度、阿里、360等国际顶尖安全团队，已为数十家交易所、电子钱包、智能合约等提供基础安全建设、渗透测试、漏洞挖掘、应急响应等安全服务。

玄猫安全实验室提供专业权威的智能合约审计服务、区块链专项应用评估、区块链平台安全评估等多项服务。

商务合作：Lyon.chen@xuanmao.org