目录

[1.环境配置 1](#_Toc98326489)

[1.1 环境配置参考 1](#_Toc98326490)

[1.2 运行所需环境 1](#_Toc98326491)

[1.3 vnt的安装 1](#_Toc98326492)

[1.4 bottle的安装 1](#_Toc98326493)

[1.5 js相关依赖的安装 2](#_Toc98326494)

[2 维特链使用说明 3](#_Toc98326495)

[2.1 搭建本地测试链 3](#_Toc98326496)

[2.2 相关接口的使用 7](#_Toc98326497)

[3 项目设计 8](#_Toc98326498)

[3.1 网站各功能流程 8](#_Toc98326499)

[3.2 智能合约设计 9](#_Toc98326500)

# 1.环境配置

## 环境配置参考

Gvnt文档：

Bottle文档：<https://github.com/vntchain/vnt-documentation/blob/master/smart-contract/compile-contract.md>

## 运行所需环境

Ubuntu版本应为16.04，否则需要docker才能安装bottle

Golang版本应为1.9.2以上

node版本应为v8.11.2

还需要安装gcc和git

为了避免奇怪的错误，建议以上内容全部从官网下载

## vnt的安装

下载及安装教程：<https://github.com/vntchain/go-vnt>

使用cd命令进入到自己的下载的目录

安装make

sudo apt install make

使用下面命令编译gvnt和所有工具:

make all

将gvnt所在路径加入默认路径里

在主目录下

vi .bashrc

在文件末尾添加PATH=$PATH(你的gvnt所在路径)

之后就可以使用gvnt命令了

输入gvnt version查看版本 看看是否成功

## bottle的安装

将``bottle``代码clone到选择的目录

```bash

git clone https://github.com/vntchain/bottle

```

编译``bottle``需要[go](https://golang.org/)编译器，``go``的安装请参考``go``的官方文档

(不要使用手动下载压缩包的方式下载代码，否则会由于缺少文件导致make失败)

同时需要安装依赖包libxml2-dev,xz-utils,gcc

```bash

sudo apt-get install -y libxml2-dev xz-utils gcc

```

然后使用以下命令编译得到``bottle``

```bash

cd bottle

make bottle

```

最后使用以下命令运行bottle

```bash

./build/bin/bottle

```

最后可以在主目录下使用vi .bashrc将bottle的路径添加到环境变量中

## js相关依赖的安装

安装vnt.js: 0.20.7和vnt-kit.js 1.0.0

```

npm install --save https://github.com/vntchain/vnt.js.git

npm install --save https://github.com/vntchain/vnt-kit.js.git

```

```

# 这些依赖为以太坊有js工具库，vntchain可以兼容这些工具库

npm install --save ethereumjs-tx@1.3.7

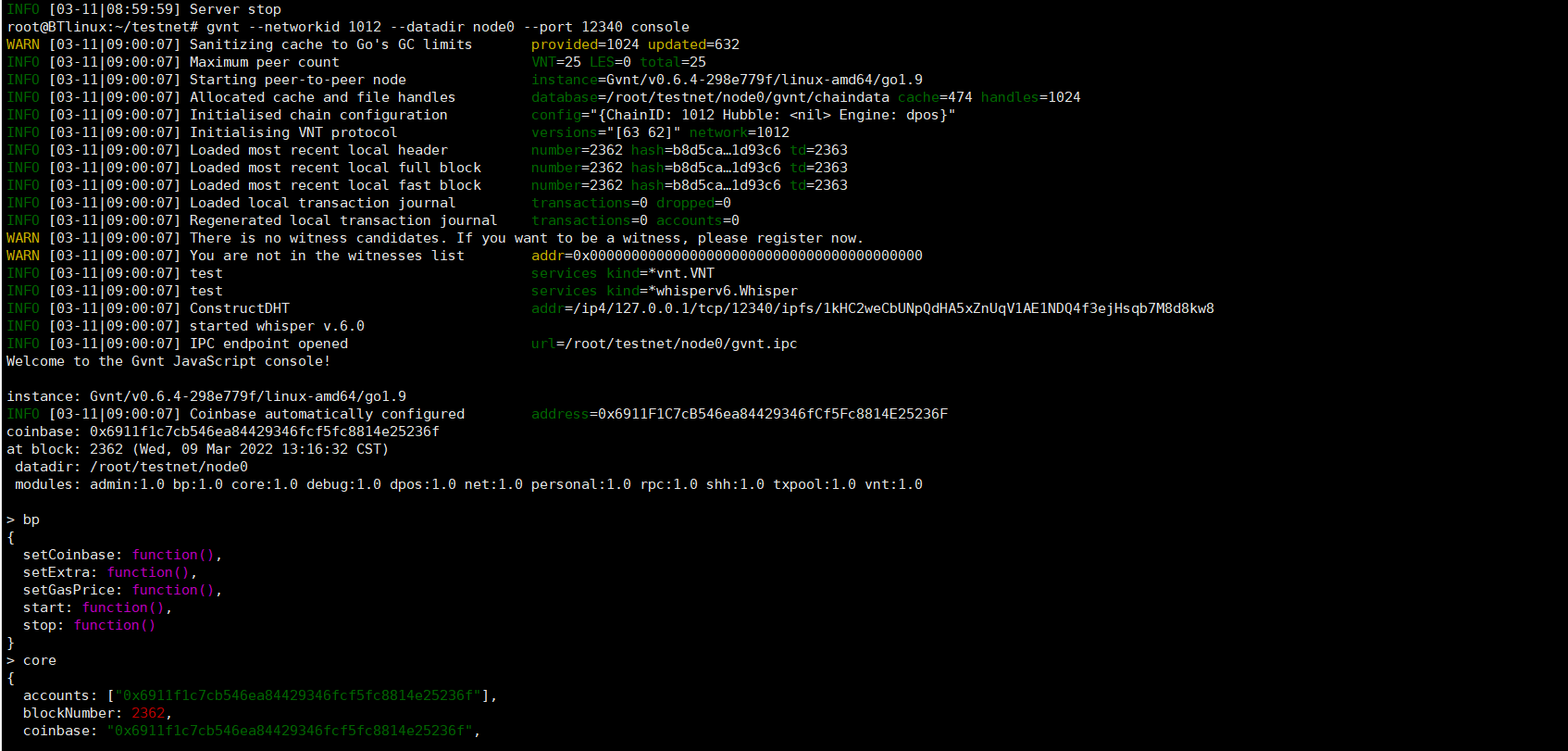
npm install --save ethereumjs-account

```

# 维特链使用说明

## 搭建本地测试链

参考文档为<https://github.com/vntchain/vnt-documentation/blob/master/introduction/set-up-vnt-network/set-up-4-node-vnt-network.md>



创建一个文件夹

cd ~

mkdir testnet

进入到这个文件夹，输入命令创建node0的账户（密码自己设）

gvnt account new --datadir node0

用同样的方法生成结点node1 node2 node3 把每次的地址粘贴到一个文本文件里以后写json的时候用

**生成结点p2p地址**

p2p地址中包含了IP和端口号，我们在启动节点的时候需要指定节点间通信的端口号，节点0到节点4分别分配端口号:12340，12341，12342，12343。

使用如下命令启动节点0，并获取p2p地址。

➜ gvnt --datadir node0 --port 12340 console

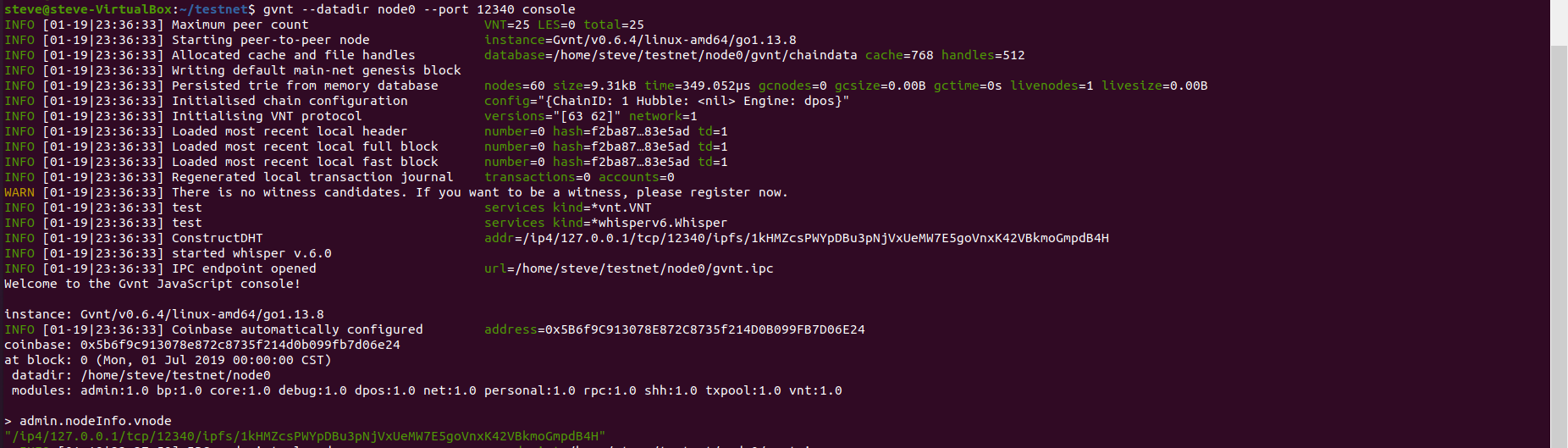
// 省略很多输出

> admin.nodeInfo.vnode

"/ip4/127.0.0.1/tcp/12340/ipfs/1kHcch6yuBCgC5nPPSK3Yp7Es4c4eenxAeK167pYwUvNjRo"

使用control-D关闭客户端。

将每次的节点p2p地址（以ip4开头）粘贴到文本文件里以后写json的时候用



其它结点同样方法

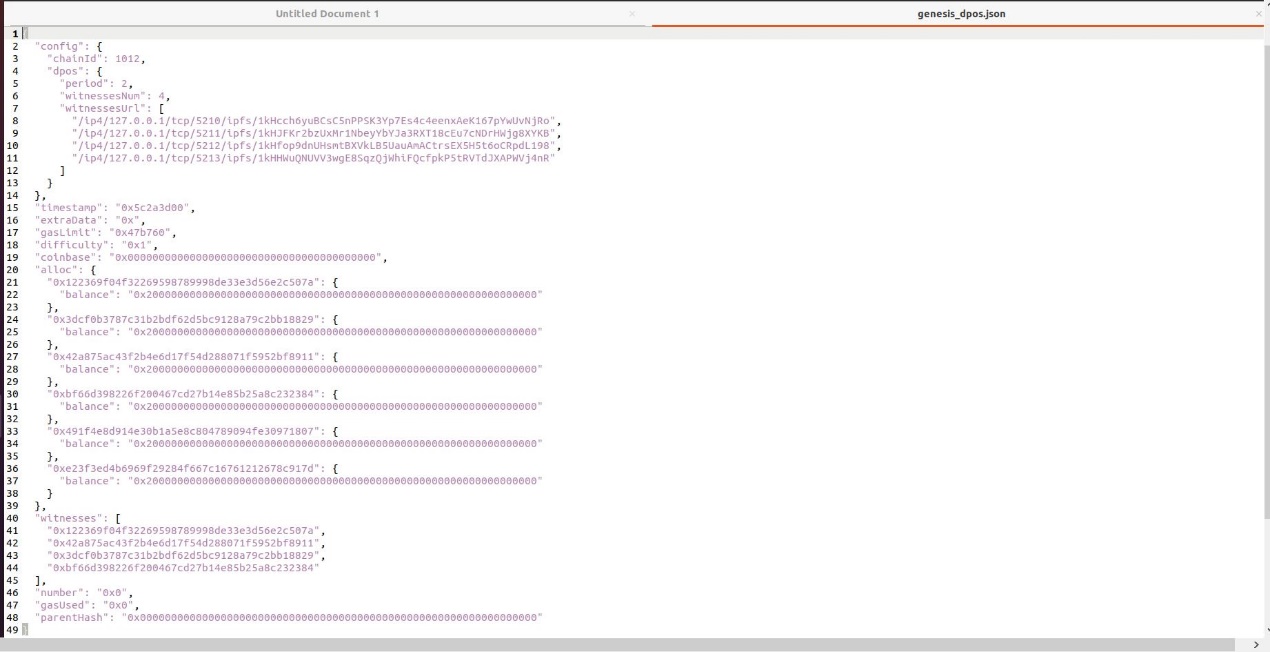
移除节点0的临时数据gvnt目录：

rm -rf node0/gvnt

可以使用上述命令，清理节点1，2，3的临时数据，否则使用配置文件初始化节点时会失败。

**创建创世块配置文件dpos.json**

在自己下载vnt代码的文件夹里找到genesis\_dpos.json，其内容如下



将witnessesUrl和witnesses里面的内容都改成刚刚粘在文本文件里的账号和p2p地址（witness别忘了0x）

把修改后的文件保存到testnet目录，命名为dpos.json

（“alloc”为分配初始虚拟货币，将这部分的节点账号也改成自己网络节点的账号,否则会因为账号缺少余额而无法部署智能合约）

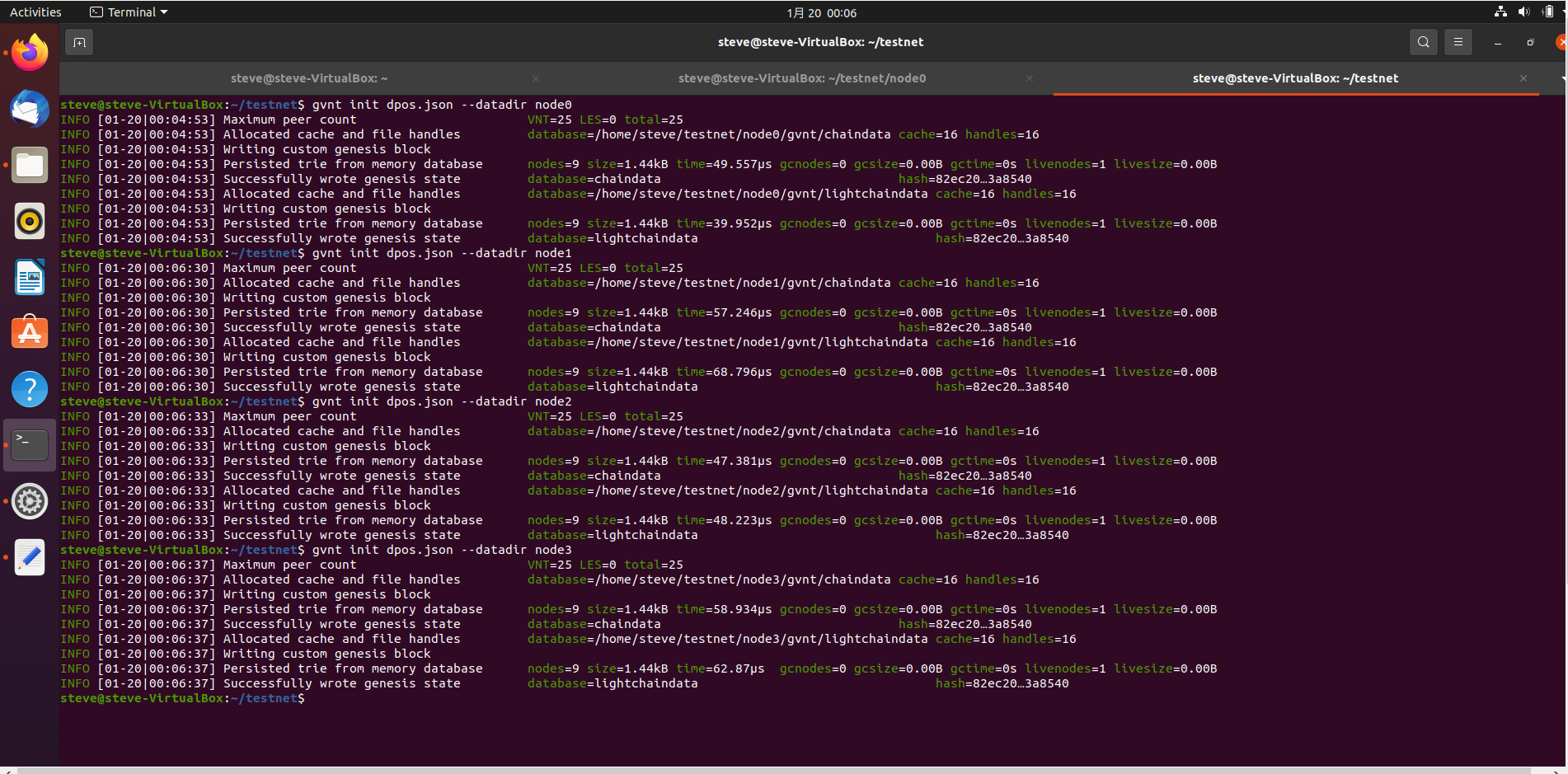
**使用dpos.json初始化每个节点**

gvnt init dpos.json --datadir node0

gvnt init dpos.json --datadir node1

gvnt init dpos.json --datadir node2

gvnt init dpos.json --datadir node3



启动bootnode节点

VNT网络需要使用bootnode自动建立，其中一个初始见证人节点可作为bootnode，其他节点和该节点建立连接，就会和剩余节点建立连接

gvnt --networkid 1012 --datadir node0 --port 12340 console

之后再打开一个命令行窗口，输入之后的命令

启动剩余初始见证人节点

节点1的启动命令如下，和生成p2p地址的命令相比，需要指定--vntbootnode的p2p地址，即节点0的p2p地址。

gvnt --networkid 1012 --datadir node1 --port 12341 --vntbootnode "/ip4/127.0.0.1/tcp/12340/ipfs/1kHcch6yuBCgC5nPPSK3Yp7Es4c4eenxAeK167pYwUvNjRo" console

使用admin.peers能查看和本节点建立连接的节点，其中<peer.ID 1kHcch6yuBCgC5nPPSK3Yp7Es4c4eenxAeK167pYwUvNjRo>就是节点0的p2p地址，说明他俩已经建立连接。

// 省略启动打印

> admin.peers

[{

caps: null,

id: "<peer.ID 1kHcch6yuBCgC5nPPSK3Yp7Es4c4eenxAeK167pYwUvNjRo>",

name: "",

network: {

inbound: false,

localAddress: "/ip4/192.168.0.104/tcp/12341",

remoteAddress: "/ip4/192.168.0.104/tcp/12340",

static: false,

trusted: false

},

protocols: null

}]

之后在打开一个命令行窗口启动结点2

启动节点2：

gvnt --networkid 1012 --datadir node2 --port 12342 --vntbootnode "/ip4/127.0.0.1/tcp/12340/ipfs/1kHcch6yuBCgC5nPPSK3Yp7Es4c4eenxAeK167pYwUvNjRo" console

再打开一个命令行窗口

启动节点3：

gvnt --networkid 1012 --datadir node3 --port 12343 --vntbootnode "/ip4/127.0.0.1/tcp/12340/ipfs/1kHcch6yuBCgC5nPPSK3Yp7Es4c4eenxAeK167pYwUvNjRo" console

节点3上的peer信息如下，各节点上都会有3个节点，说明我们的测试网络已经搭建起来了：

> admin.peers

[{

caps: null,

id: "<peer.ID 1kHJFKr2bzUnMr1NbeyYbYJa3RXT18cEu7cNDrHWjg8XYKB>",

name: "",

network: {

inbound: false,

localAddress: "/ip4/127.0.0.1/tcp/12343",

remoteAddress: "/ip4/127.0.0.1/tcp/12341",

static: false,

trusted: false

},

protocols: null

}, {

caps: null,

id: "<peer.ID 1kHcch6yuBCgC5nPPSK3Yp7Es4c4eenxAeK167pYwUvNjRo>",

name: "",

network: {

inbound: false,

localAddress: "/ip4/127.0.0.1/tcp/12343",

remoteAddress: "/ip4/127.0.0.1/tcp/12340",

static: false,

trusted: false

},

protocols: null

}, {

caps: null,

id: "<peer.ID 1kHfop9dnUHHmtBXVkLB5UauAmACtrsEX5H5t6oCRpdL198>",

name: "",

network: {

inbound: false,

localAddress: "/ip4/127.0.0.1/tcp/12343",

remoteAddress: "/ip4/127.0.0.1/tcp/12342",

static: false,

trusted: false

},

protocols: null

}]

每次关闭虚拟机都要重新启动结点

打包区块

在每个初始见证人节点上解锁账户、开启共识，可以看到区块不断产生。

personal.unlockAccount(core.coinbase, "账户密码", 3153600000) // 3153600000指解锁的秒数，这段时间之后，账户会自动锁住

bp.start()

以上，开发测试网络就搭建起来了。

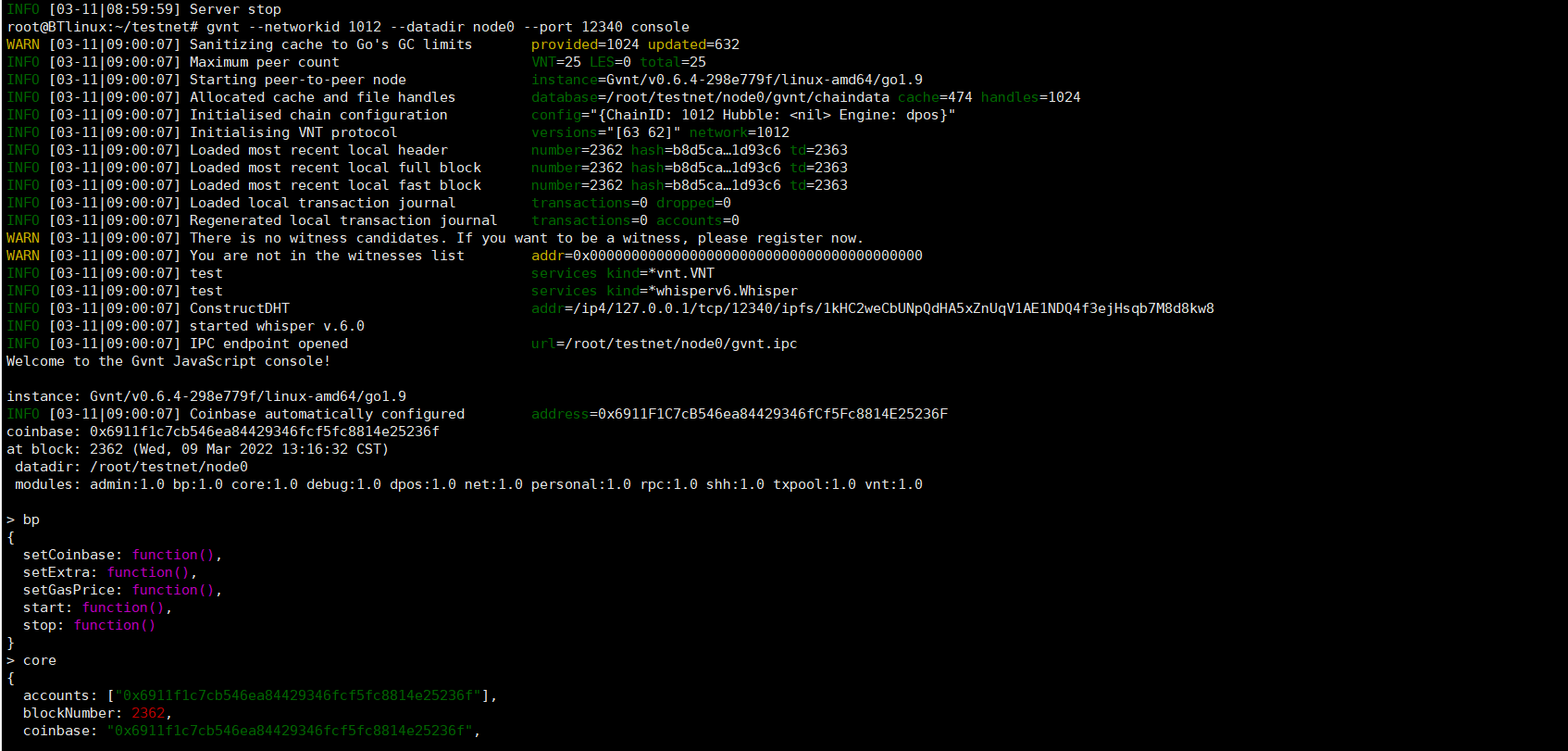
## 相关接口的使用

gvnt为我们提供的接口分别有节点控制台，JSON-RPC和JAVASCRIPT API三种，三者中功能最多的为节点控制台，JSON-RPC包含了节点控制台中的一部分方法，通过JSON-RPC可以实现对这些方法的远程调用，而JAVASCCRIPT API推测应该为JSON-RPC的进一步封装，可以通过编写javascript的方式远程操作节点。

**节点控制台**

启动节点的命令后加上console即可启动节点控制台（上文中介绍的方法已启动节点控制台）

启动节点后可以发现这里有很多模块，输入每个模块的名称查看对象中的方法和属性，可以利用这些方法查看节点相关信息或控制节点。



输入vnt直接查看所有模块的方法和属性（因为这些模块应该都是在vnt中）

其中admin都是与管理相关的内容，core为节点相关的内容，personal为账户相关的内容

笔者并没有找到详细介绍控制台模块中的方法的使用方法和参数格式的文章，建议参考JSON-RPC和JAVASCRIPT RPC的使用文章或以太坊中的类似方法。

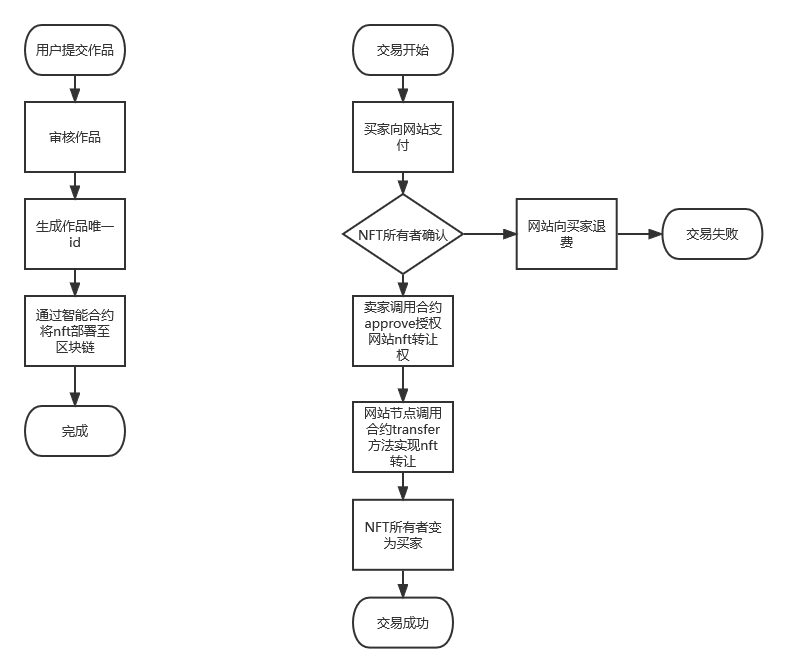
**JSON-RPC**

# 项目设计

## 网站各功能流程

3.1.1用户上传作品

每个用户在区块链网络上拥有一个钱包，用户通过这个钱包发起交易。当网站审核作品通过后，将通过加密算法生成的作品唯一id返回给用户，以用户自己在区块链上的账户的名义调用智能合约中的createNFT方法为自己铸造NFT。

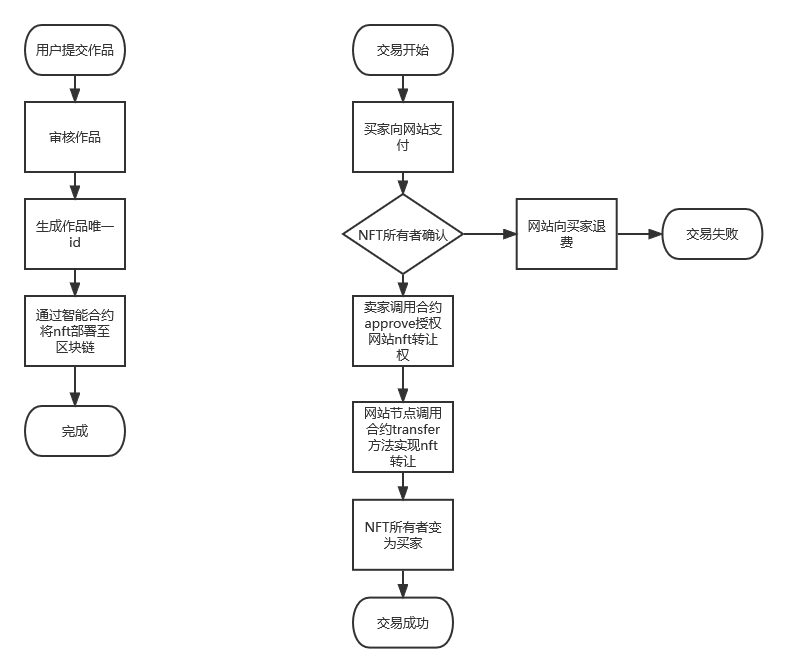


3.1.2用户交易

在用户交易的过程中，买方和卖方没有直接的接触，卖方甚至不知道买方是谁，网站节点在两者之间进行沟通。



如图所示，在购买过程中，买方先向网站支付预先定好的价格，之后网站按照之前的约定将其交付给卖方，卖方确认后将nft的转让权通过调用智能合约中的approve方法交给网站节点，网站节点将nft装让给买家账户。交易的具体流程如下：



## 智能合约设计

uint256 NFTcounter //作为nft的唯一编码，每次加一

mapping (uint256, address) private tokenOwner //key为nft序号 value为所有者地址

mapping(uint256, string)private tokenURI //key为nft序号 value为作品的唯一id

mapping(uint256, bool)private approval //key为nft序号 value为所有者是否授权网站转让

function createNFT(string tokenID) //铸造NFT

function transfer(address ) //只有网站能够调用，且当网站拥有转让权时才能调用成功

function approve() //给予网站节点该nft的转让权，只有nft拥有者才能调用