* 官网地址
  + https://ethereum.org/

**下载安装**

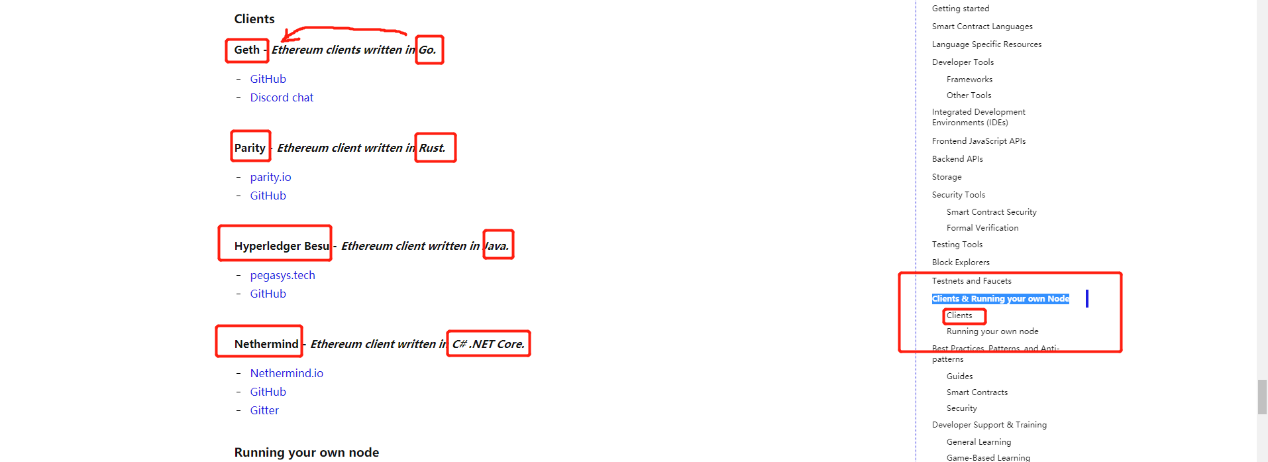
**开发客户端下载**

**官方下载地址：**

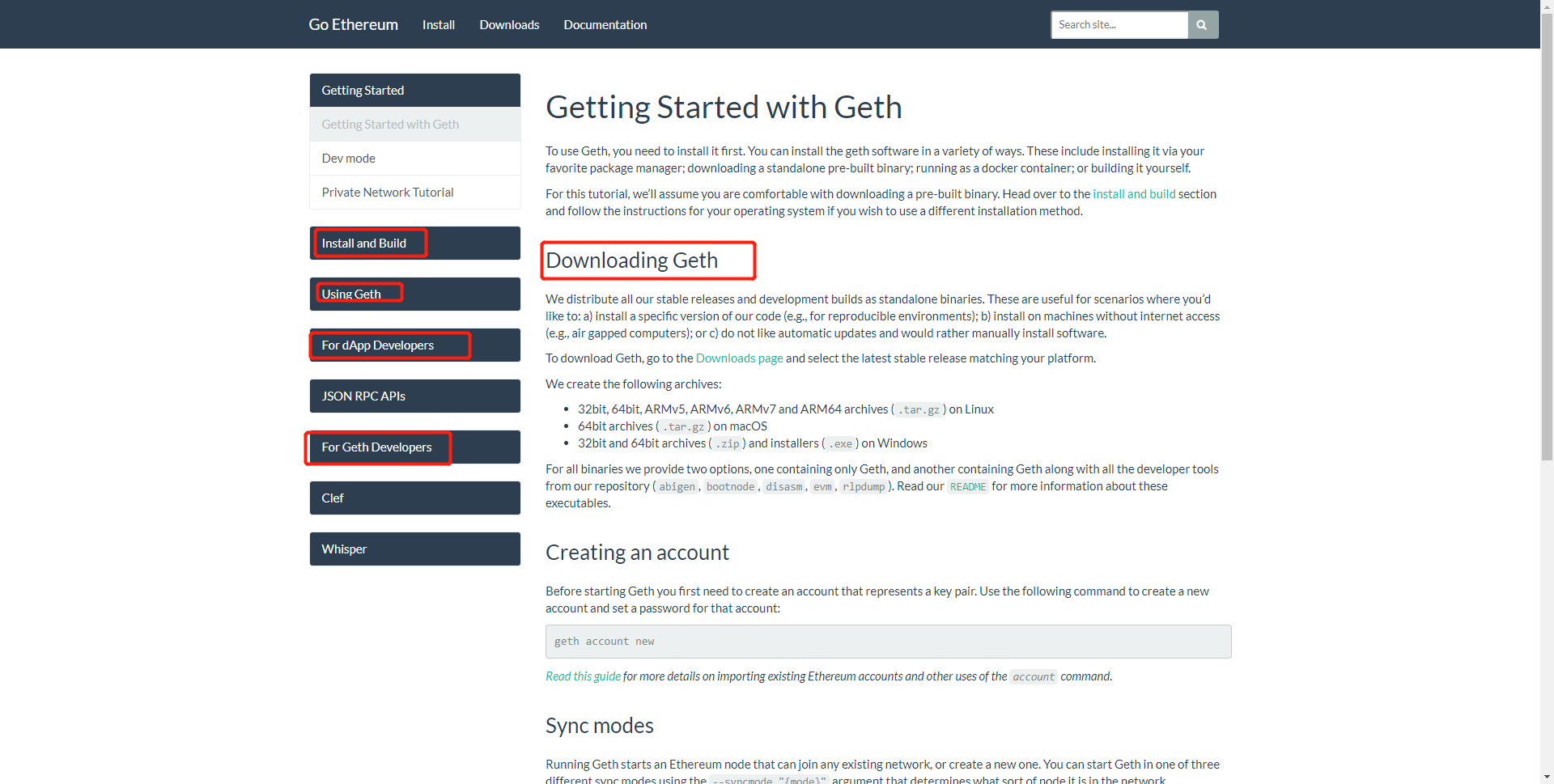
* 地址：
  1. 源代码下载地址：https://github.com/ethereum/go-ethereum
  2. 本地执行文件下载地址：https://geth.ethereum.org/downloads/

1. 从主页的连接【For Developers】进入,里面包含开发的所有教程与资源。包含客户端下载。
2. 在【For Developers】页面右边有个导航，从那儿可以快速找到下载的位置【Clients & Running your own Node】-> 【Client】，
   * 其中下载的版本很多(不同语言实现的客户端），我们选择GO语言实现的客户端：Geth。
     + Github都是源代码版本。需要安装Go语言环境，然后编译安装，编译安装文件是Makefile。
3. 变编译好的安装文件下载地址：
   * https://geth.ethereum.org/downloads/

* 以太坊客户端类型：



1. 本地安装文件下载入口：
   * 在【Clients & Running your own Node】位置找到【Looking for other options?】后，点击进入。
   * 进入Github.com的网站页面，其中找到【Ethereum Clients】节点下的【Geth - Go client】，然后点击进入到官方的连接页面【https://geth.ethereum.org/docs/】。
   * 在【https://geth.ethereum.org/docs/】中，从下载安装，都有详细的步骤。
2. 官方的教程与文档
   1. 下载【Getting Started】
   2. 安装【Install and Build】
   3. geth客户端的管理与操作【Using Geth】
   4. DApp开发【For dApp Developers】
   5. 后台开发【For Geth Developers】



**国内镜像下载地址**

* 下载地址：https://ethfans.org/wikis/Ethereum-Geth-Mirror
  + 还包含各种【汉化】教程与入门。
* 国内爱好者站点：https://ethfans.org/
  + 从菜单【知识库】进入

**下载中需要明白的几个概念**

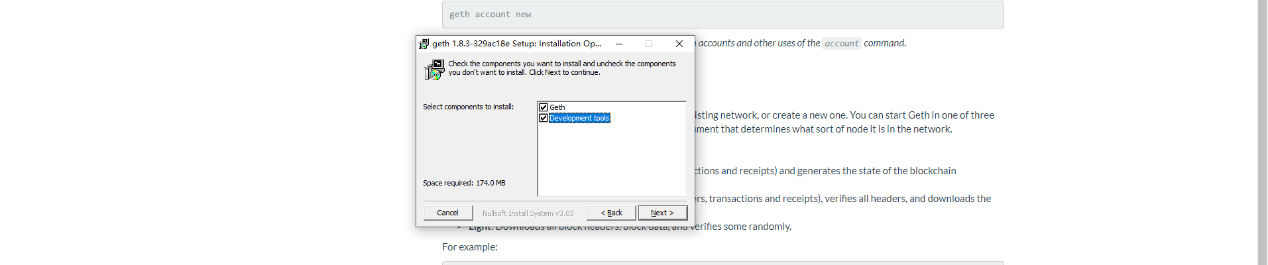
1. Ethereum Wallet：
   * 一个基于GUI的管理平台，被Mist替代。2019年3月已经停止维护，不用下载了安装了。下载Mist即可。
2. Geth
   * 一套API实现，可以通过API实现以太坊访问。
3. Mist
   * 一个基于Web的客户管理应用。开发者可以不需要，想用来同步测试的话，可以下载一个用用。

**安装**

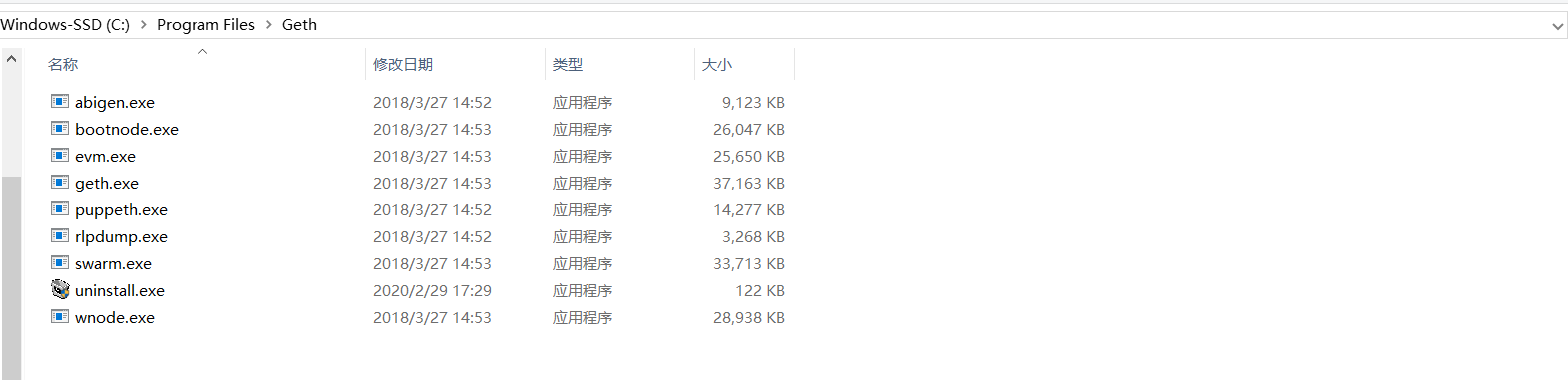
**安装Geth客户端**

* 因为下载的是window安装包，所有按照安装的可视化界面向导，可以很轻松完成安装过程。其中做一些选择即可。

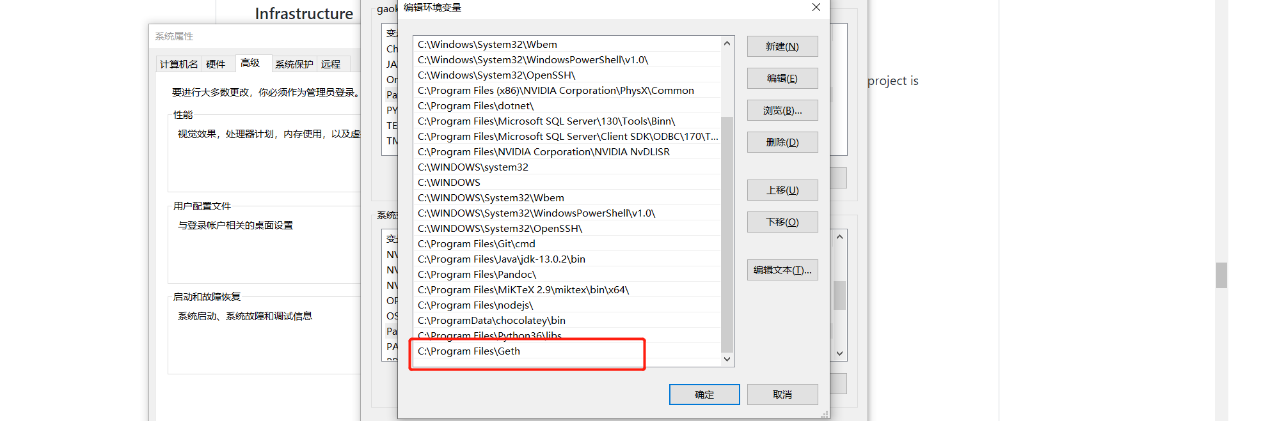
1. 选择安装开发工具



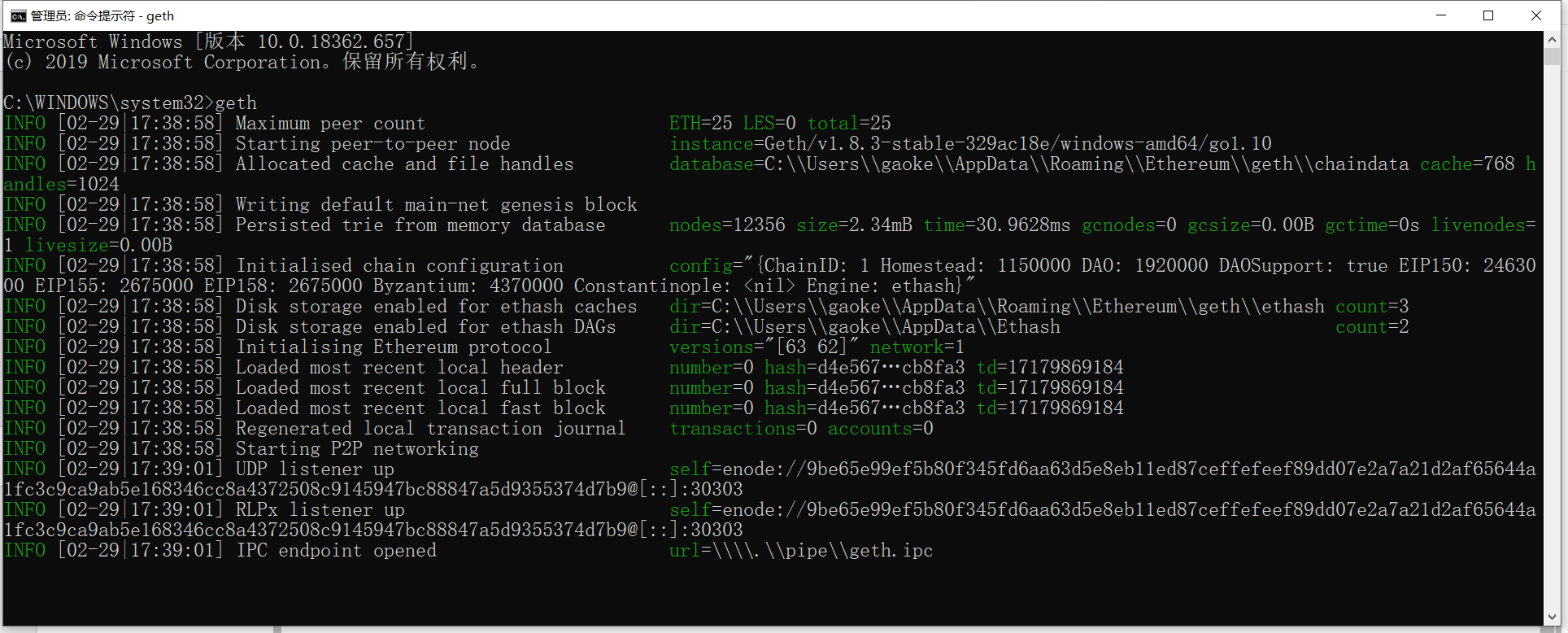
1. 安装完毕的安装目录结构
   * 为了保证安装好的程序在任意位置执行，请配置PATH环境变量。安装默认时配置，最好确认下。



1. 确认安装后，是否配置PATH环境变量



1. 安装测试
   * 在终端直接执行geth命令，可以启动客户端节点。
   * 多个geth节点可以组成私有以太坊。



**Mist安装**

* Mist也是window安装包，所有按照安装的可视化界面向导，可以很轻松完成安装过程。其中做一些选择即可。

1. 安装
   * 按照向导安装即可，安装目录选择默认就行。
   * 其中有两个目录选择：
     1. 安装目录
     2. 区块链数据存放目录
2. 访问连接
   * 安装完毕在开始菜单生成一个【Mist】Web服务快速启动的快捷方式。点击它可以启动一个管理应用Web服务。启动需要下载Node什么的，速度比较慢。



**Mist的下载**

* 官方最新地址：
  + https://github.com/ethereum/mist/releases/

**GETH介绍**

**功能**

1. JavaScript Console：通过后台进行命令操作；
2. Management API：管理相关的API；该API在控制台命令提供应用实现。
   * 账号管理（创建账号、锁定账号、解除锁定等）；
   * 查询账户信息；
   * 查询交易信息；
   * 查询gasPrice；
   * 交易；
   * 挖矿&停止挖矿；
   * 部署智能合约
3. JSON-RPC server：JSON-RPC相关调用API

**GETH命令**

1. 以太坊的访问使用两种方式：
   * 通过API实现应用访问操作；
   * 通过geth命令实现终端操作与操作；
2. 获取geth的使用帮助



**配置geth私有网络节点**

**创世文件**

* 该文件的配置使用模板即可，需要可以自己根据需求编写。
  + TOML配置文件，文件格式可以使用json格式，如下例。

{

"config": {

"chainId": 10,

"homesteadBlock": 0,

"eip155Block": 0,

"eip158Block": 0

},

"coinbase" : "0x0000000000000000000000000000000000000000",

"difficulty" : "0x2000",

"extraData" : "",

"gasLimit" : "0xffffffff",

"nonce" : "0x0000000000000042",

"mixhash" : "0x0000000000000000000000000000000000000000000000000000000000000000",

"parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",

"timestamp" : "0x00",

"alloc": {

"08a58f09194e403d02a1928a7bf78646cfc260b0": {

"balance": "0x200000000000000000000000000000000000000000000000000000000000000"

},

"87366ef81db496edd0ea2055ca605e8686eec1e6": {

"balance": "0x200000000000000000000000000000000000000000000000000000000000000"

}

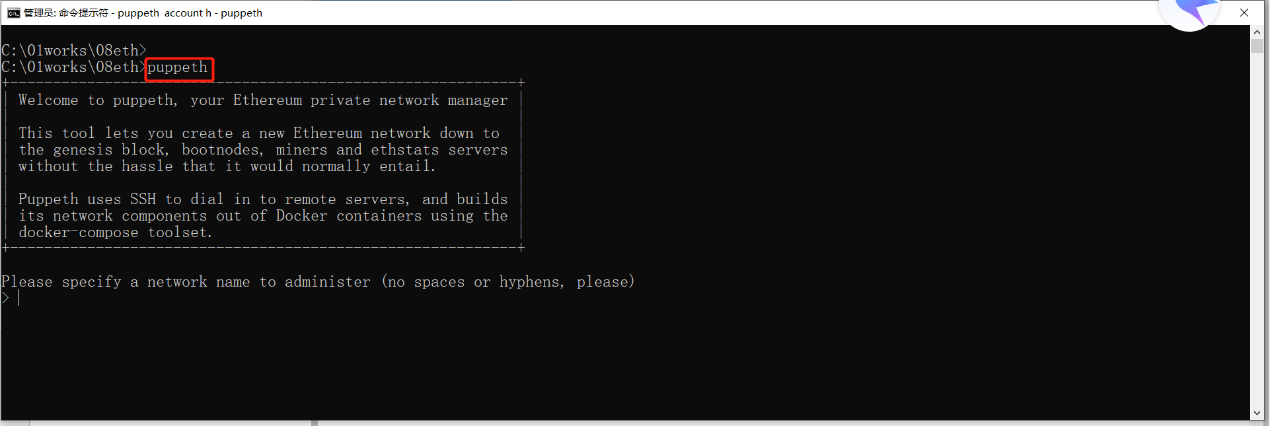
}

}

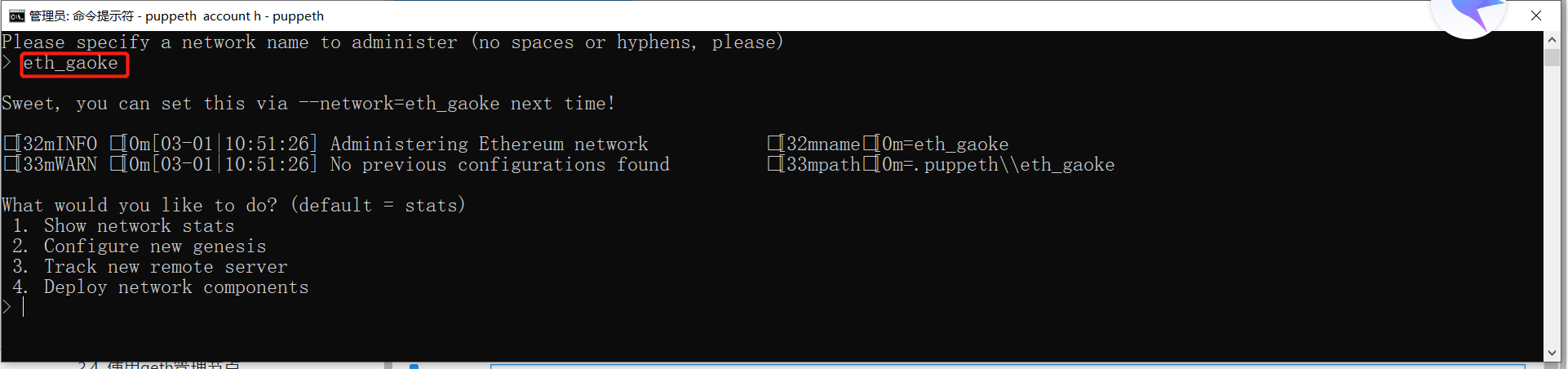
**puppeth工具与创世文件**

* 创世文件的配置不支持通用规范的原因是：区块链不太建议关注私有网络，所以建议使用shell脚本来配置。
* 由于大量的需求，所以在Geth 1.6开始重新设计了的整个配置和标志处理，最后实现了对这个非常需要的特性的支持。配置文件使用TOML格式，可以设置的字段将一对一映射到ethereum内部使用的配置结构。某些字段已被省略，以防止敏感数据在配置文件中循环。
* 但是正确配置字一个创世文件是一件痛苦的事情，所以Geth 1.6开始，安装文件附带了一个命令，来向导式的创建创世文件。
  + 使用dumpconfig导出创世配置文件，然后可以通过geth--config yourfile.toml。

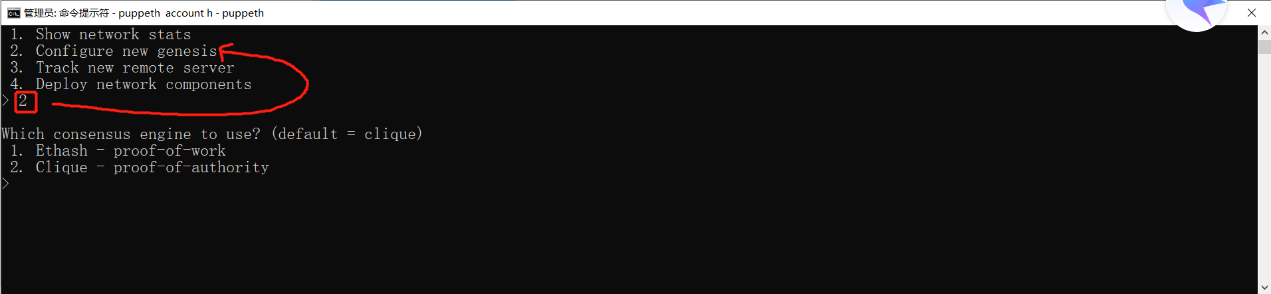
1. 启动命令



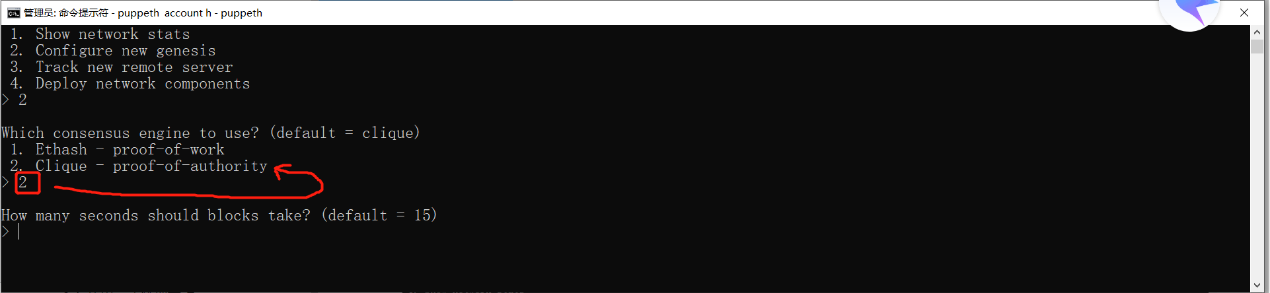
1. 设置网络名



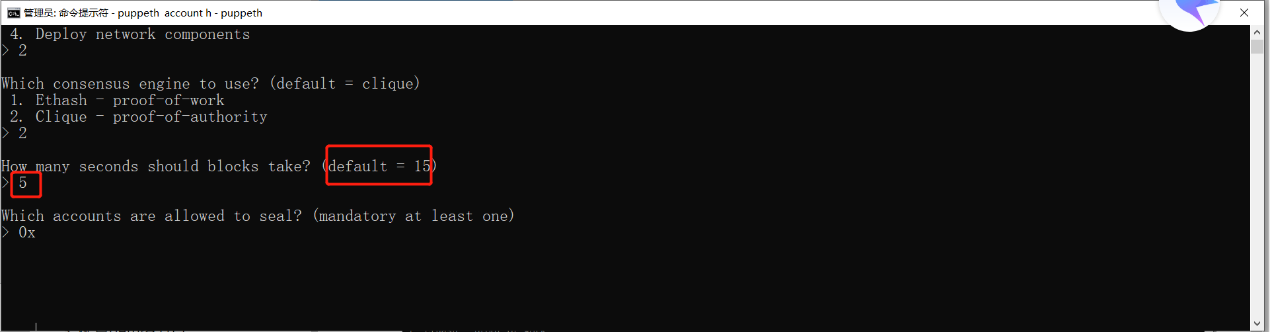
1. 选择操作功能：【2. Configure new genesis】



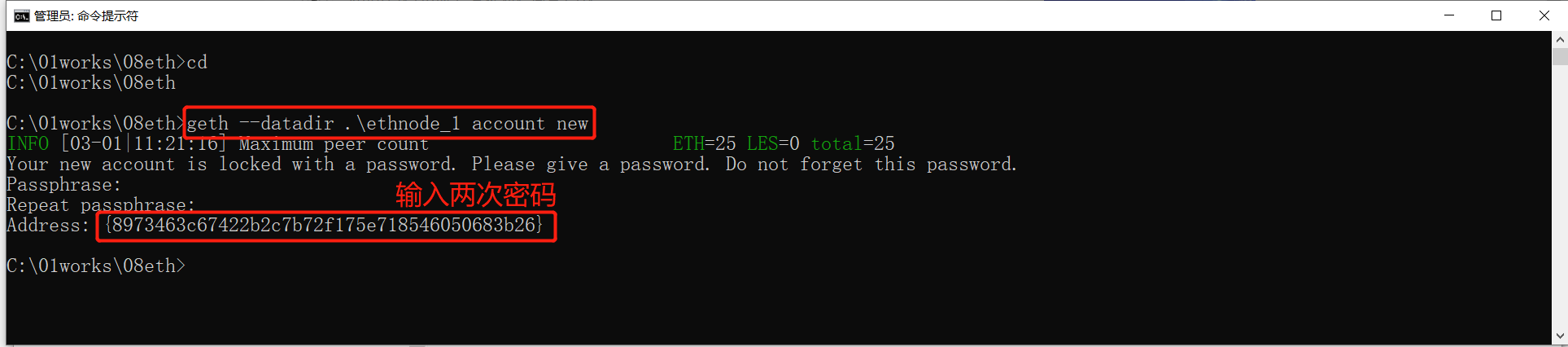
1. 选择共识引擎[consensus engine]
   * 选择默认2- 权威证明（通过预先设定的权威节点来负责达成共识 ，不消耗算力，一般用于私有链测试开发）。
   * 另外一个是工作证明（通过算力达成共识，以太坊的使用这种方式））。

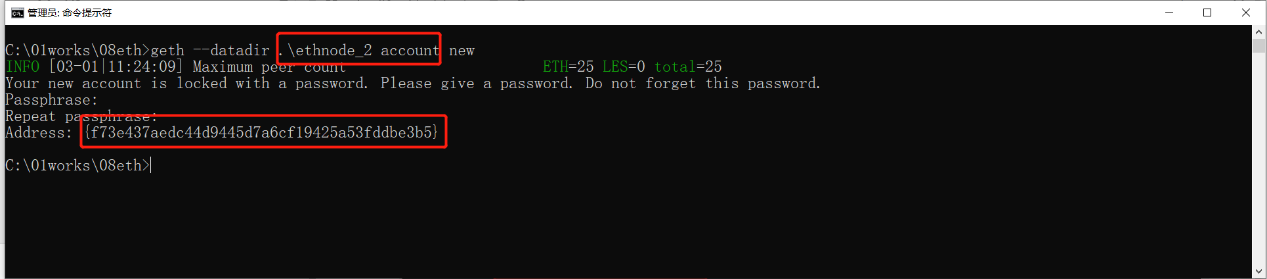


1. 设置出块时间
   * 默认15秒
   * 我们设置为5秒



1. 设置权威账号，用来生成块（至少一个）
   * 账号需要使用geth工具产生。
   * 使用geth创建账号，记住自己的账号信息。
     + 我们这里创建两个,在不同目录节点
       - gaoke2020
       - seu2020

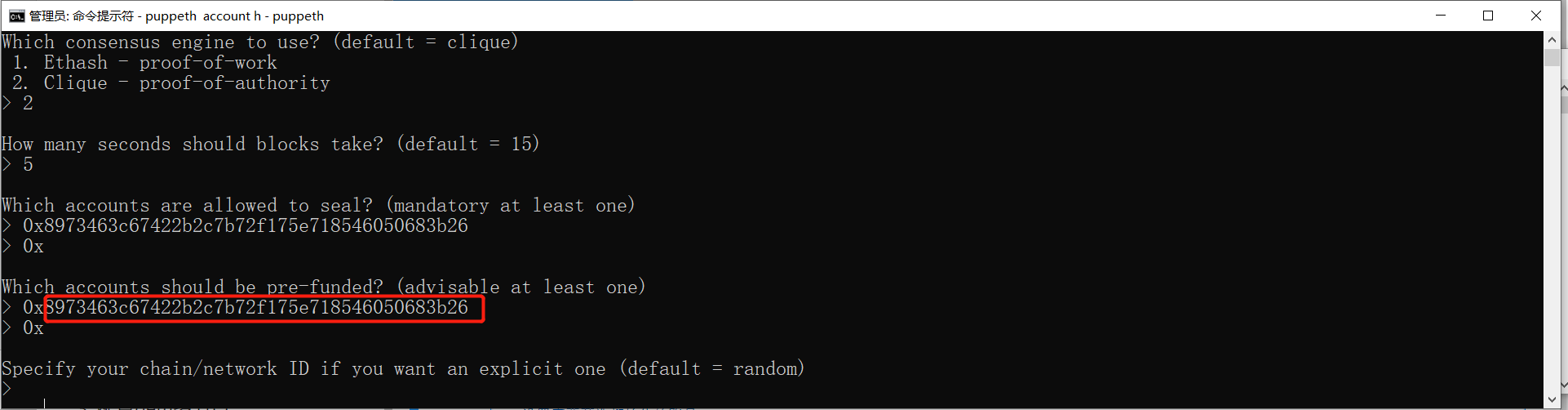




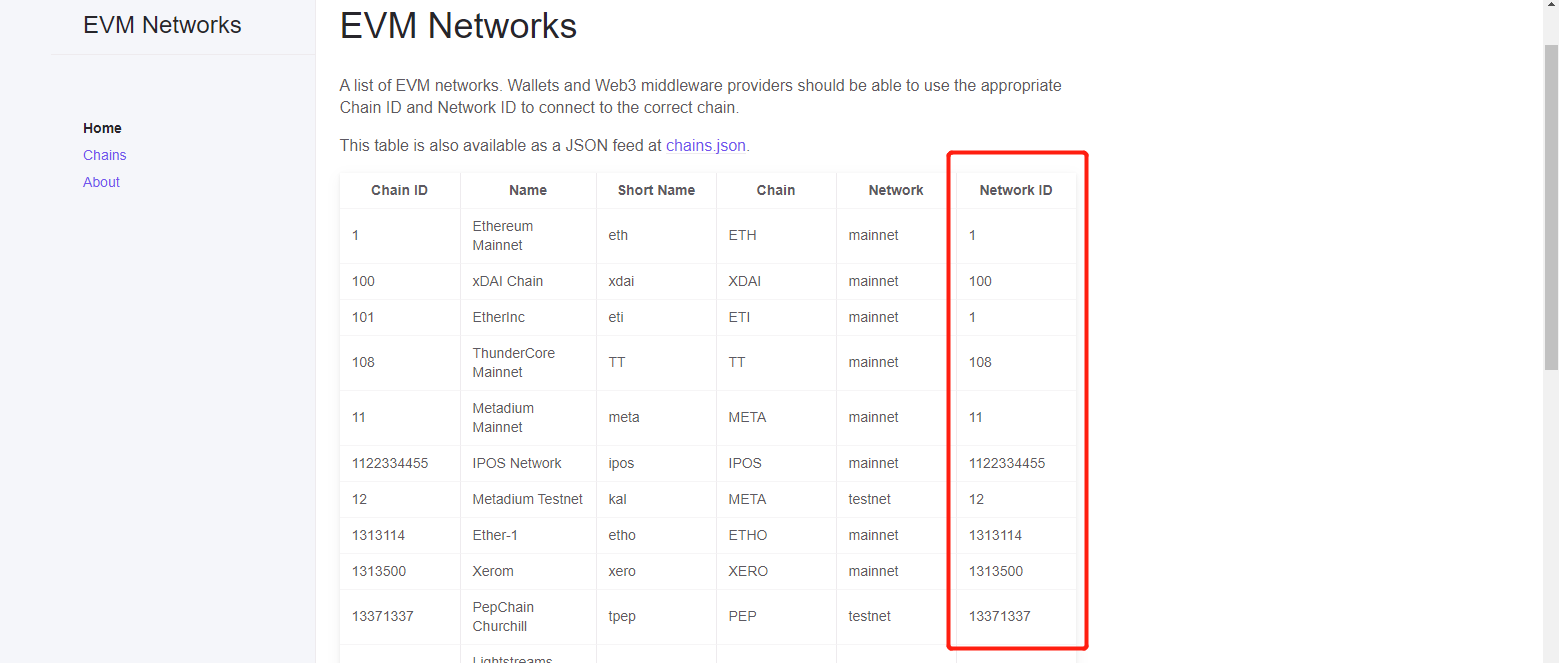
* 继续设置权威账号
  + 设置完毕再次提示输入，直接回车进入下一步骤。

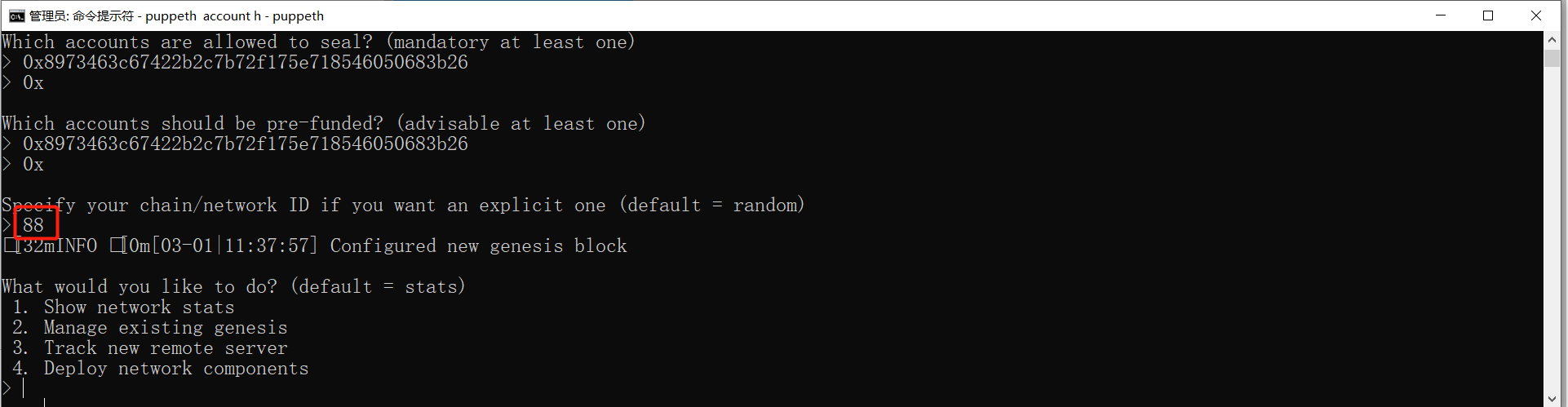


1. 设置需要预选初始化的账号
   * [Which accounts should be pre-funded?]
     + 哪些账户应该预先融资？
     + 我们使用同一个账号。

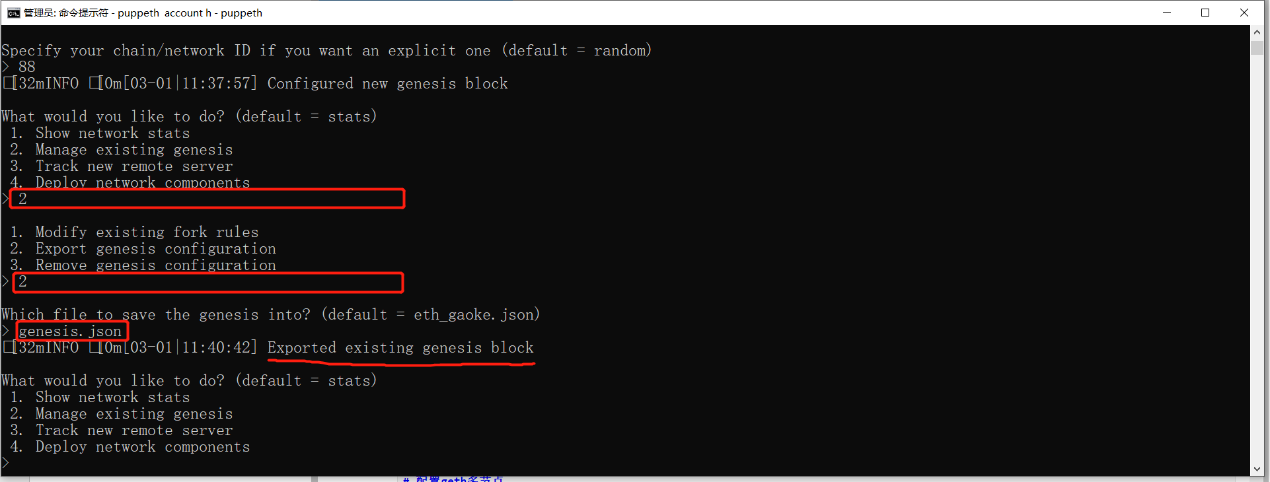


1. 设置网络ID
   * 不能使用1，这是公有链。
   * 公有链上的ID，可以通过如下网址查找收到：https://chainid.network/
   * 我们选择88，缺省【直接回车就是缺省设置】是随机





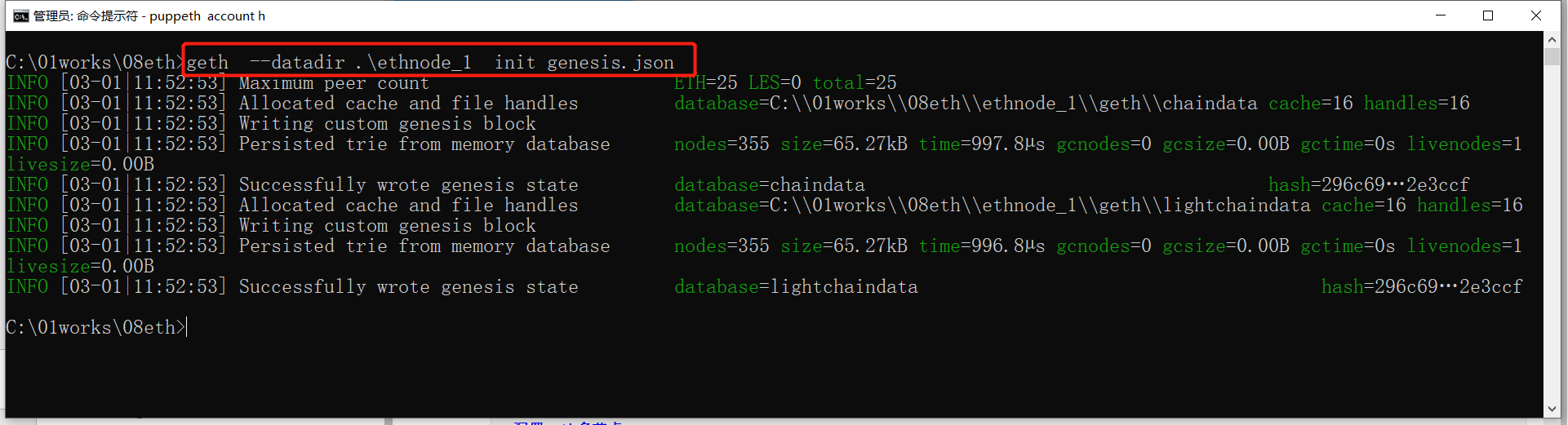
1. 导出创世文件
   * 选择【2. Manage existing genesis】
   * 继续选择【2. Export genesis configuration】
   * 设置导出文件名genesis.json



1. 退出puppeth.exe
   * 暴力退出ctrl + c

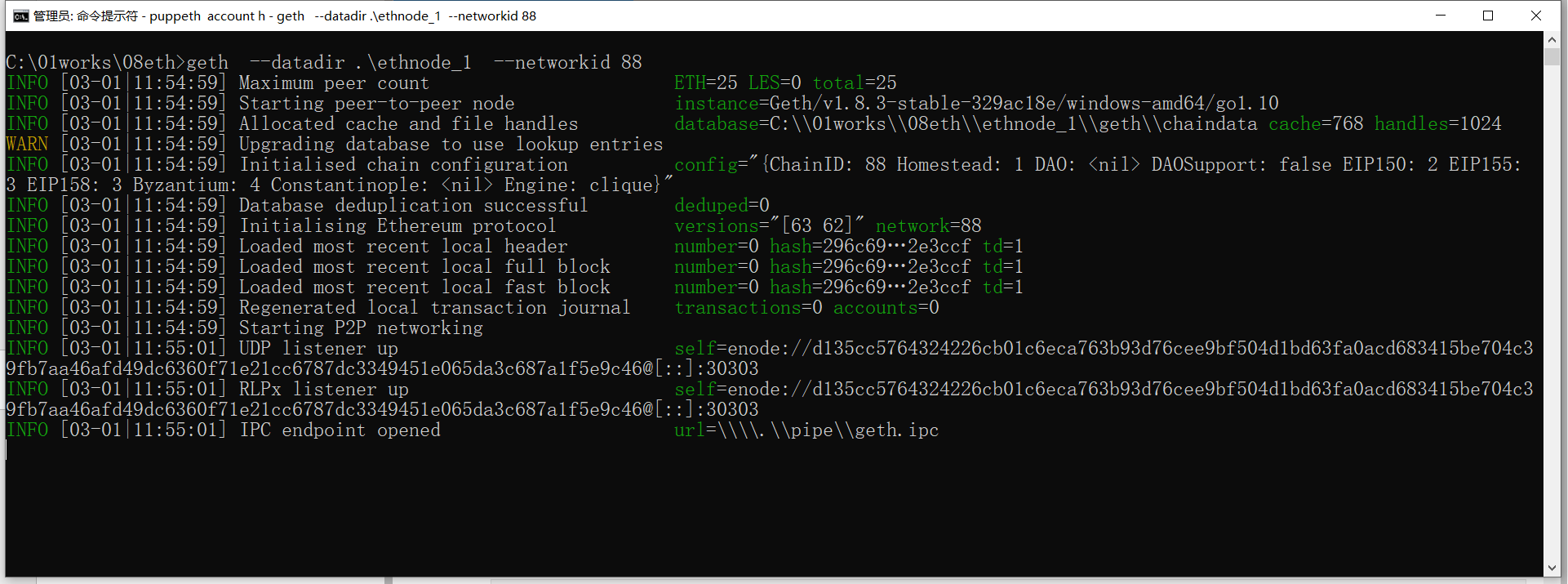
**创建geth节点数据库**

* 命令：
  + geth --datadir 数据库目录 init 创世文件
  + 例子：geth --datadir .\ethnode\_1 init genesis.json



**启动节点服务**

* 简单启动节点：
  + geth --datadir .\ethnode\_1 --networkid 88



* 启动更加详细参数的节点服务
* 命令：
  + geth --identity "enode1" --rpc --rpcport "8121" --rpccorsdomain "\*" --datadir ./enode1/data --port "2001" --nodiscover --rpcaddr "192.168.1.128" --rpcapi "db,eth,net,web3,personal"
    1. identity = 节点识别名
    2. rpc/rpcport/rpccorsdomain/rpcaddr/rpcapi
       - RPC调用的参数
    3. port
       - http调用端口，IP采用本地
    4. nodiscover
       - 不被其他节点自动发现
    5. networkid
       - 一个区块链中的网络ID

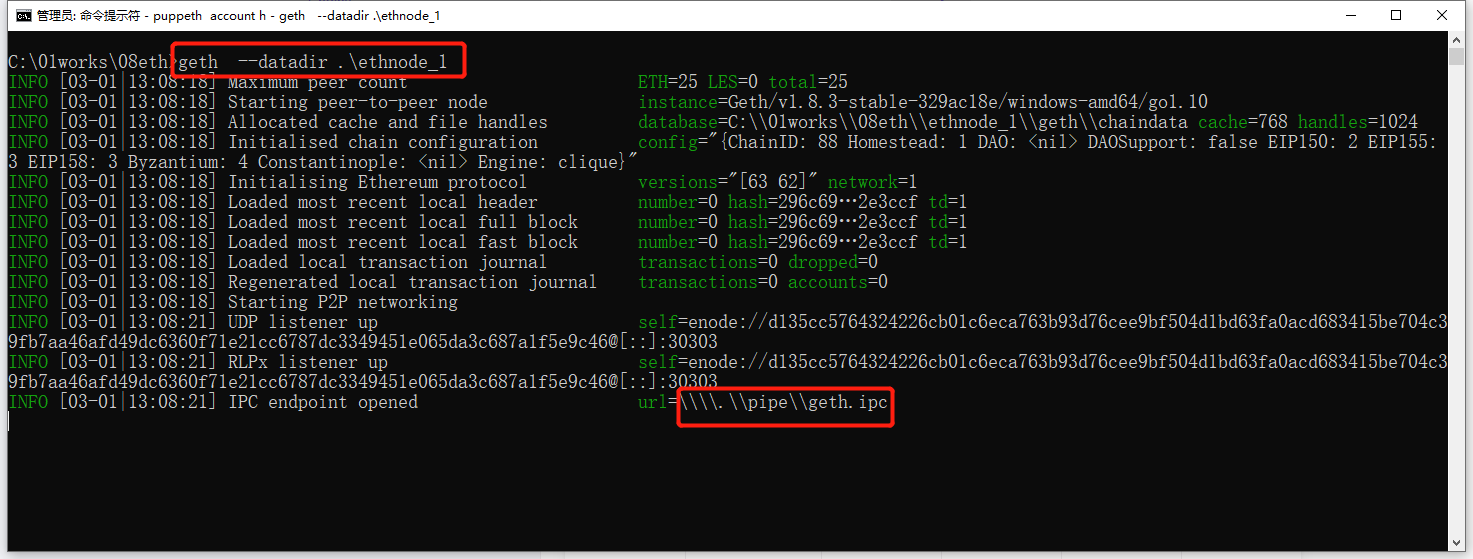
**连接到节点**

**使用attach**

* 通过网络通信连接到已经启动的区块链节点。通信方式一般三种：
  1. pipe管道文件（本地有效，默认）
  2. rpc协议(HTTP)
  3. ws通信（Web Socket）
* 30303是连接端口
  1. 每个服务有自己的端口

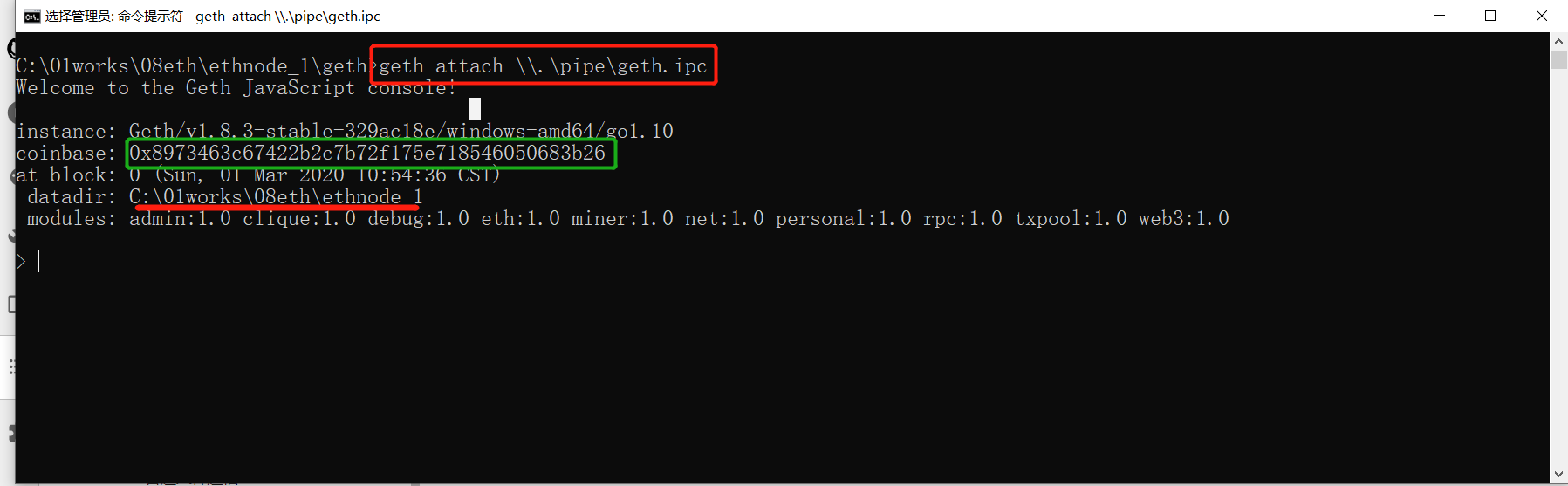
**IPC管道文件通信**

1. 服务器启动参数
   * 不需要什么参数，只要指定数据目录即可。
   * 命令：geth --datadir .\ethnode\_1



1. 连接到服务器

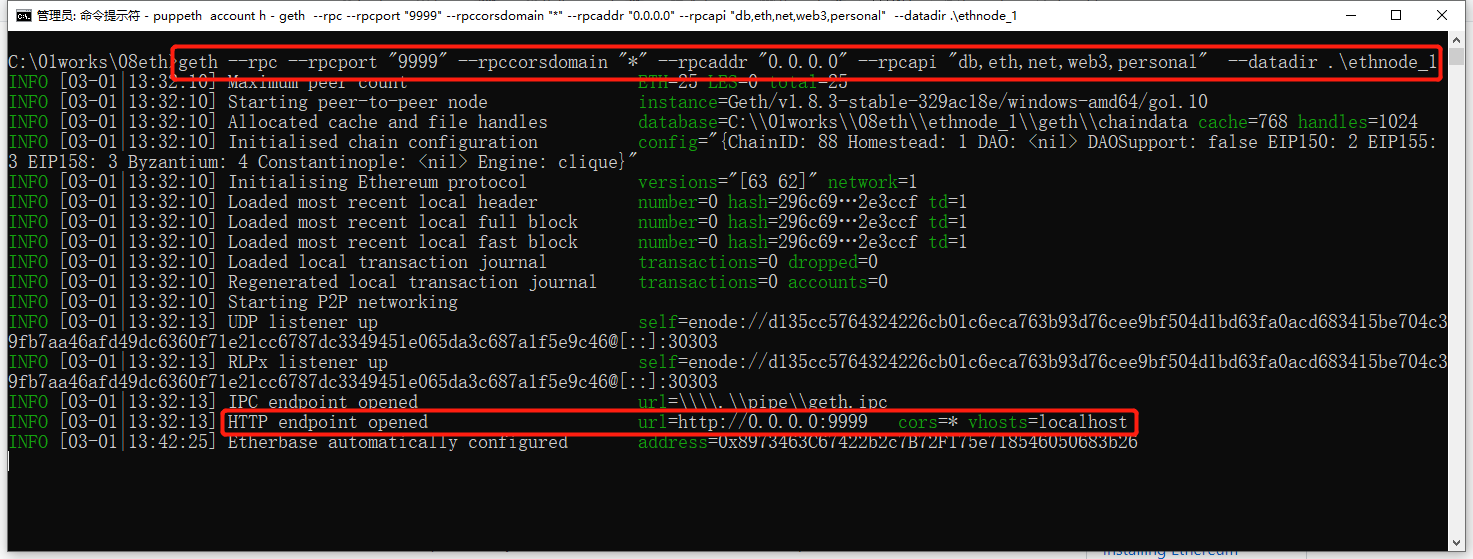
* 命令：
  + geth attach \\.\pipe\geth.ipc
  + 友情提示：按照Unix/Linux与Mac OS思路，找这个文件是找不到地，应该是一个在系统内核按照一定命名规范的管道文件：具体参考window系统编程之IPC通信编程：
    - 管道文件的命名规范：\\.\pipe\管道名



**RPC协议通信**

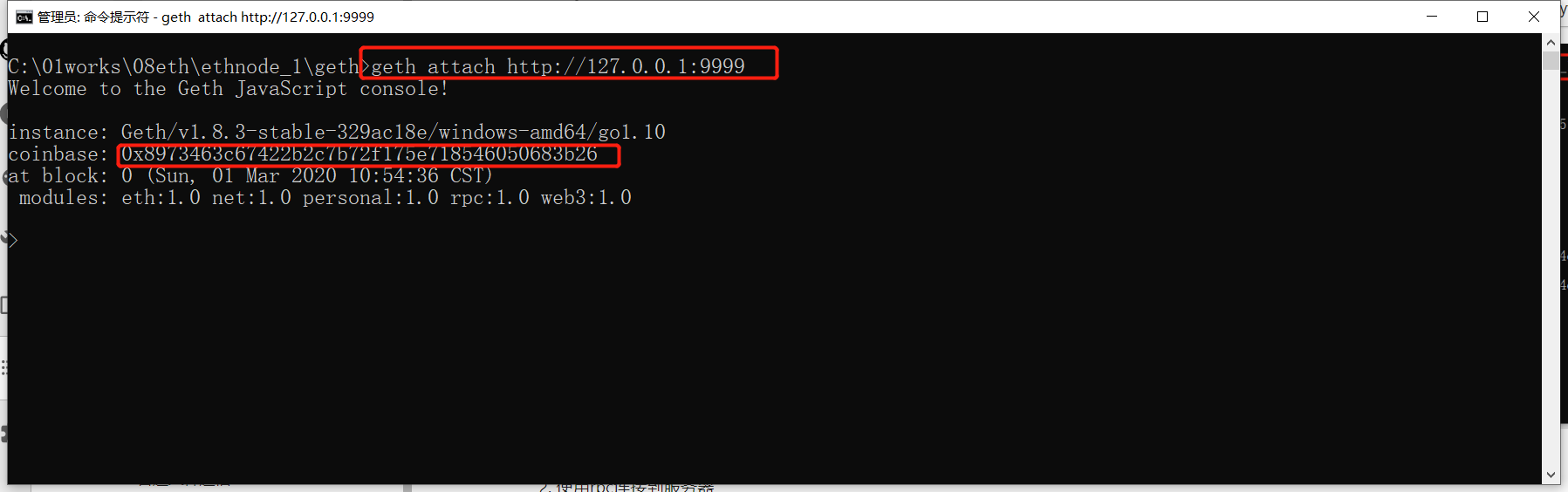
1. 启动服务器

* 命名：geth --rpc --rpcport "8121" --rpccorsdomain "\*" --rpcaddr "0.0.0.0" --rpcapi "db,eth,net,web3,personal" --datadir .\ethnode\_1



1. 使用rpc连接到服务器

* 命令：
  + geth attach http://127.0.0.1:9999



**ws通信**

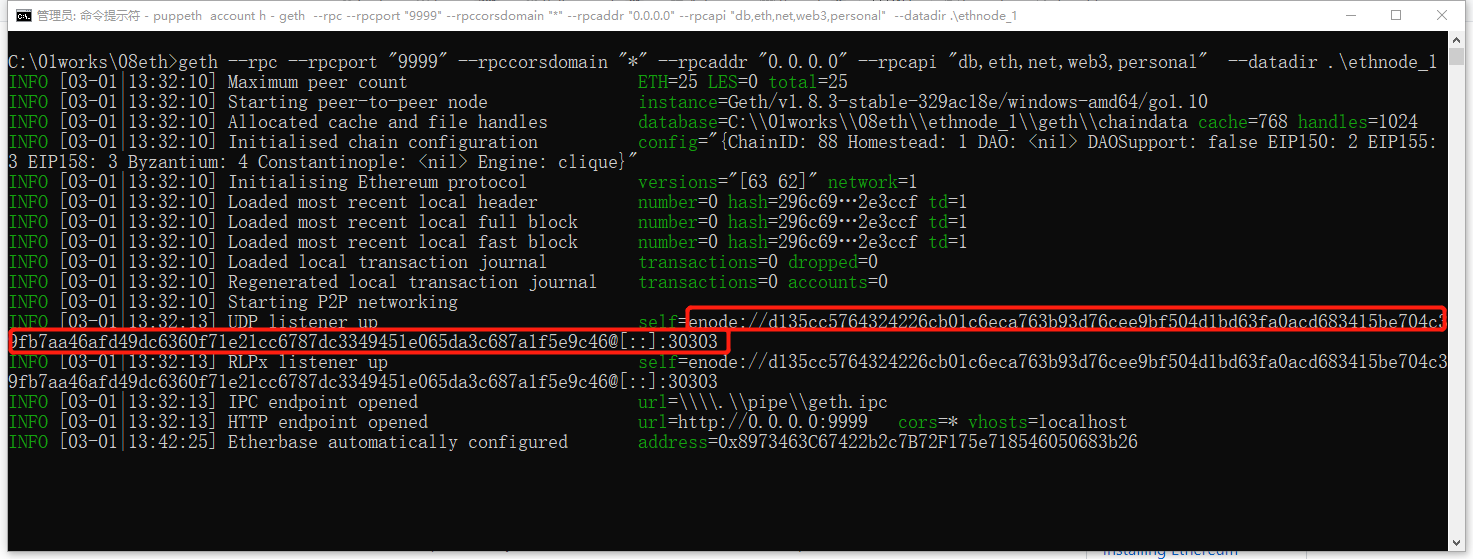
* 说明：这里不做演示。

**使用console**

* 启动服务器自动启动一个console管理交互控制台界面
  + 这种情况后面使用。

**配置geth多节点**

* 多节点配置与前面一样，不过需要注意几个关键点：
  1. 多节点运行要使用rpc通信方式启动私链。同时启动参数中的 nodiscover 参数很重要，它可以避免后启动的节点被先启动的节点自动扫描到，而必须由我们手动去添加节点
  2. 必须使用相同的JSON配置文件初始化创始区块
  3. 注意端口不要冲突：rpcport与port
* 每个节点创建会产生一个唯一的enode



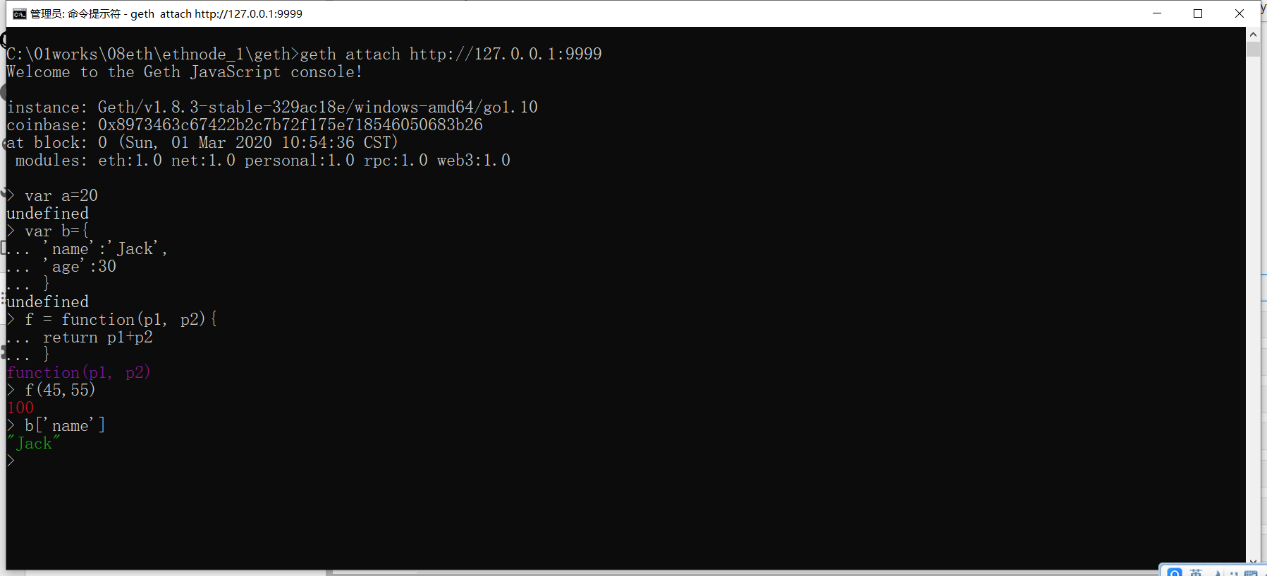
* 多个节点在后面的管理中，通过指令添加到其他节点。

**NodeJS与Javascript基础**

**控制台使用**

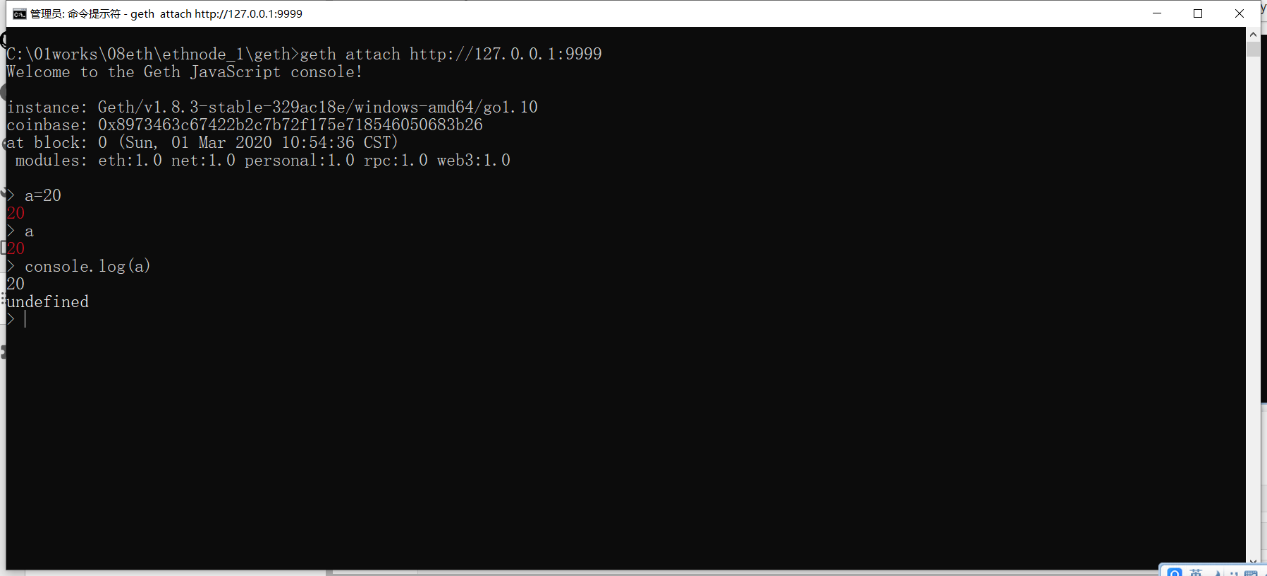
* 连接geth的控制台实际是Javascript交互式编程控制台，一种REPL（Read-Evaluate-Print-Loop）编程方式
  + 使用:exit可以退出
  + 是有个javascript运行环境，支持所有javascript语法。【注意对ES语法的版本】
    - 测试let语法无效；

1. Javascript语法



1. 使用console对象余直接显示变量值

* 常用console.log函数



**对象查看**

* 查看对象等于是在线帮助，可以知道某个对象的成员函数怎么拼写与使用。

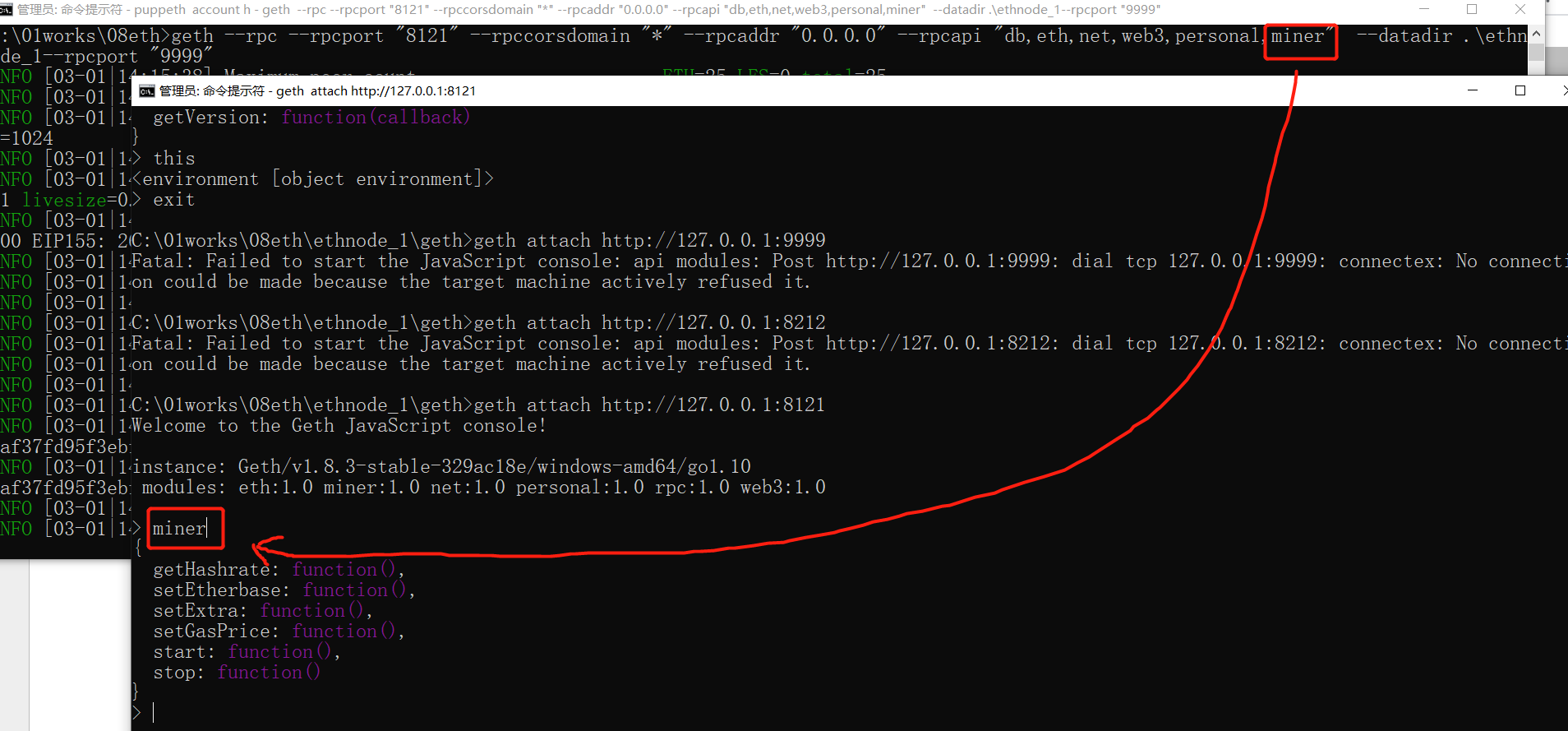
**直接查看**

* 在控制台把变量作为一个语句使用。



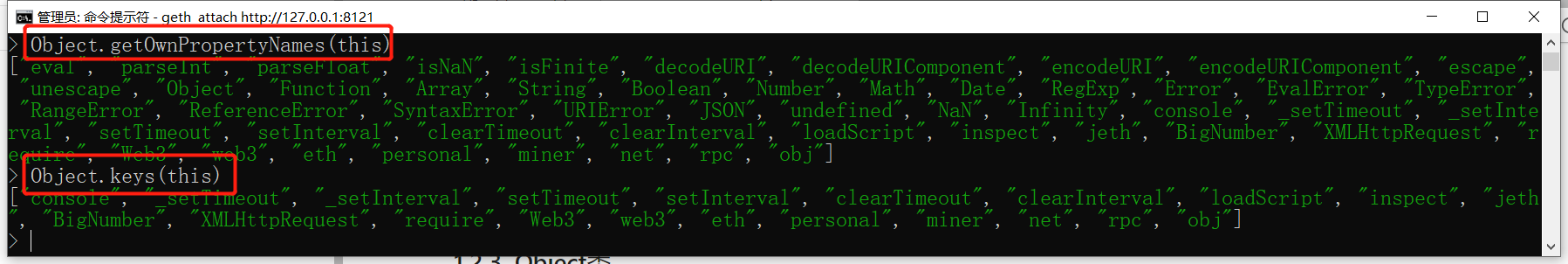
**GETH功能接口开通**

* 通过--rpcapi可以控制某些功能接口是否开通。



**Object类**

* 查看对象使用Object根类的几个静态函数可以查看。
  + Object的标准参考文档：https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global\_Objects/Object
  + Object.keys(对象)
    - 获得对象的实例
  + Object.getOwnPropertyNames(对象)
    - 获得对象的实例名称
* this是当前对象

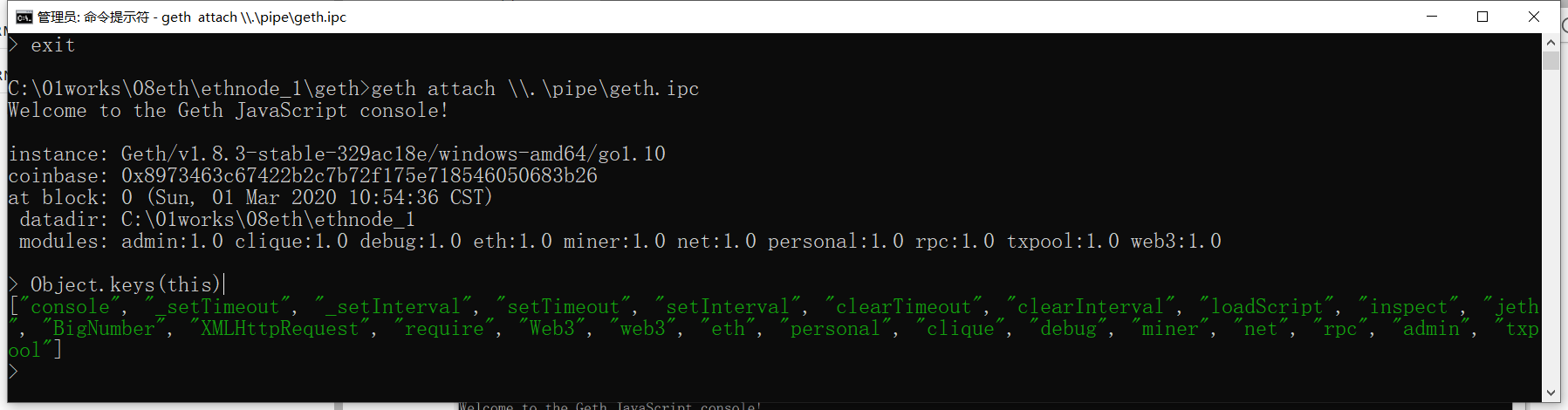


**eth核心对象**

1. RPC默认开放的访问接口如下：
   * "web3", "eth", "personal", "net", "rpc"



1. 使用本地管道登录开放的接口
   * web3", "eth", "personal", "clique", "debug", "miner", "net", "rpc", "admin", "txpool"



**常用核心对象说明**

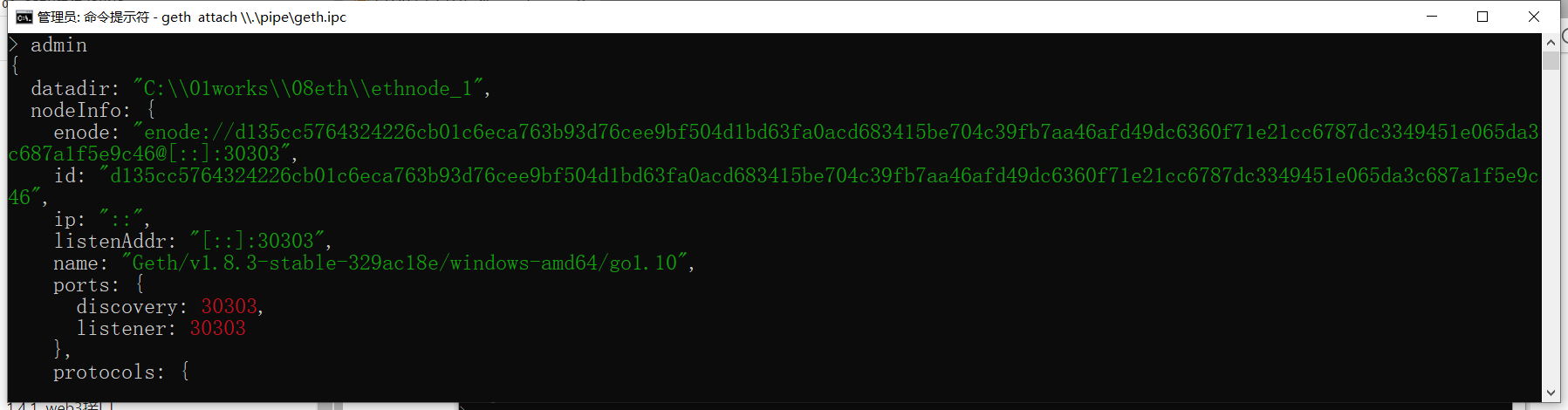
**web3接口**

* 下面的所有接口在web3中都包含。



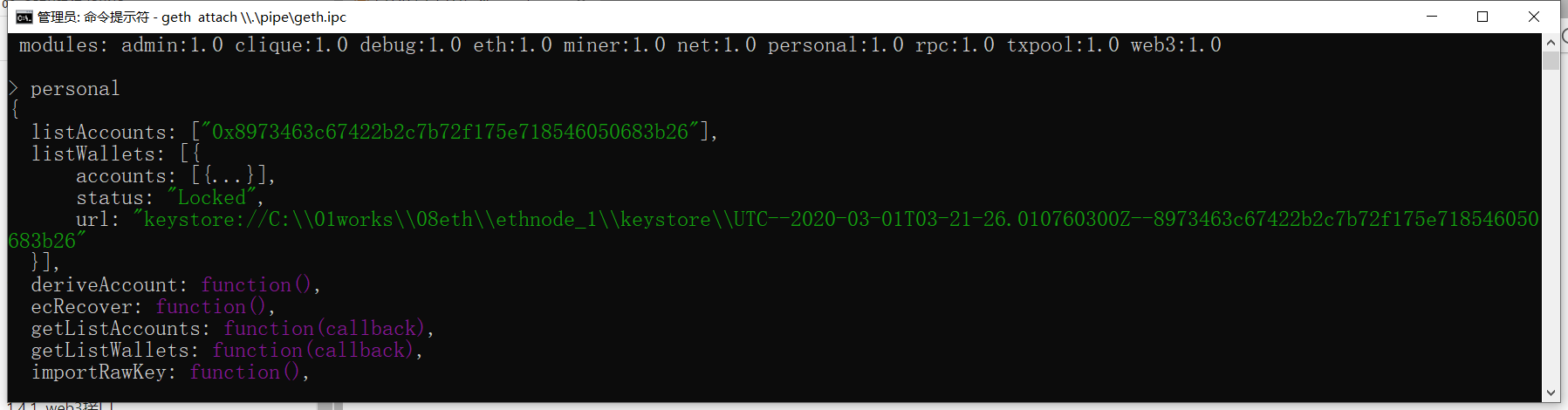
**admin接口**

* 节点信息管理



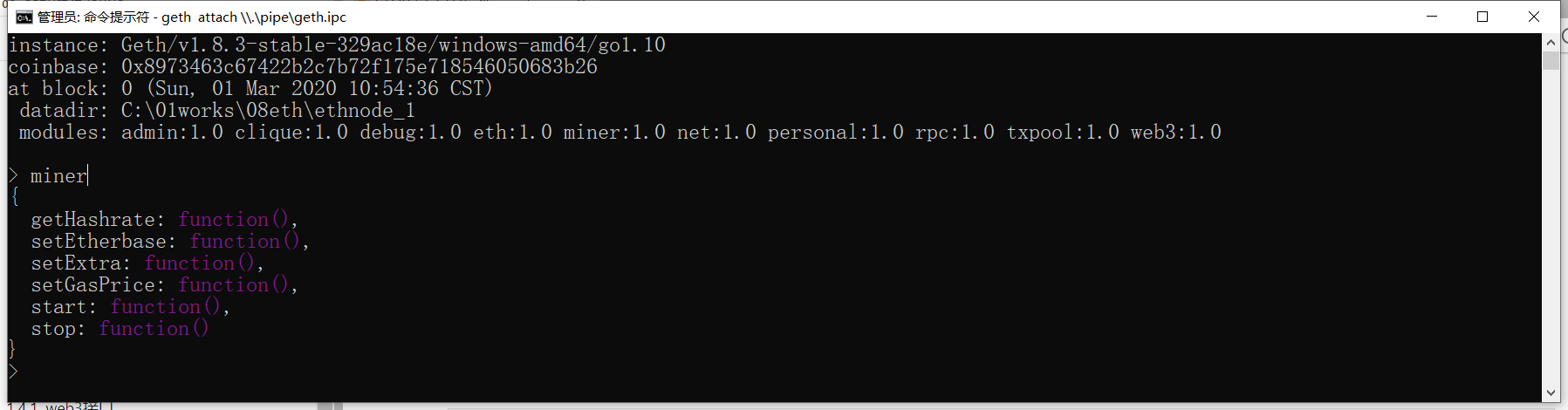
**personal接口**

* 提供个人信息管理：包含账号信息



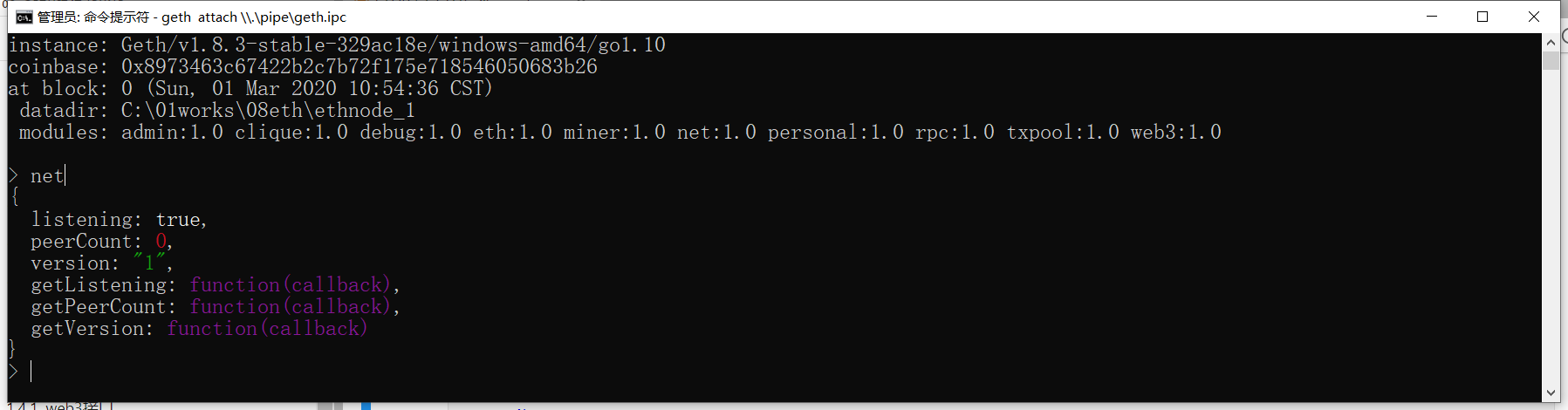
**miner接口**

* 提供挖矿功能



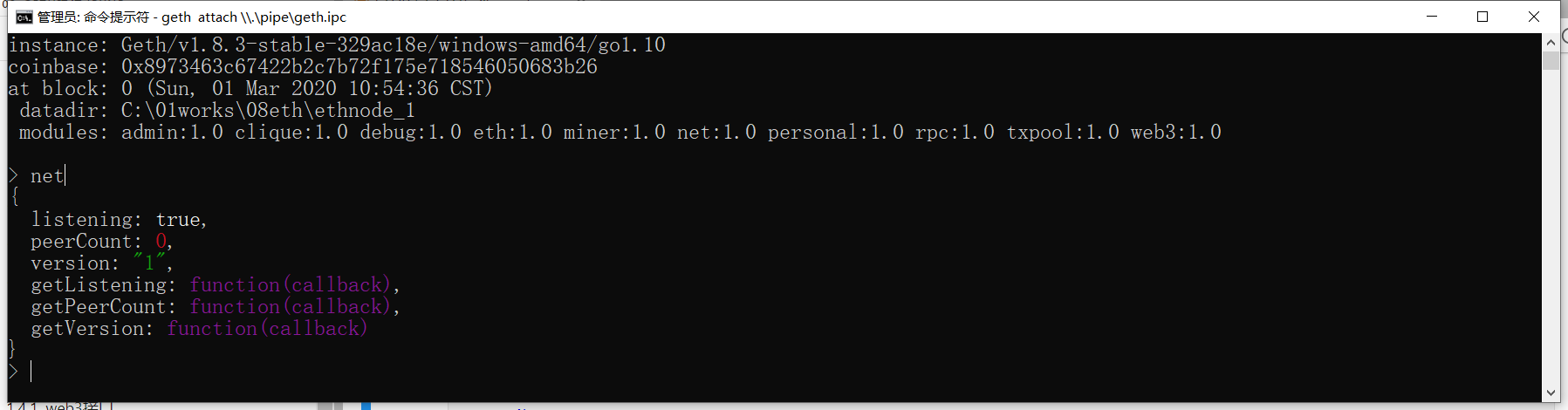
**net接口**

* 提供网络状态访问



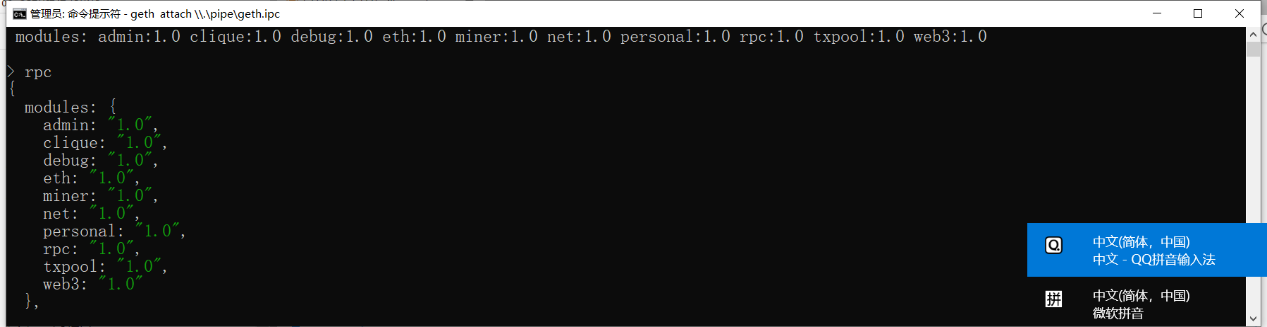
**eth接口**

* 以太坊的所有功能接口：



**rpc接口**

* 列出prc支持从接口机机器版本



**基本操作**

* 操作方式:
  + 控制台方式
    - personal.listAccounts
  + RPC JSON方式
    - {"method": "personal\_listAccounts", "params": []}
* 操作参考文档：
  + https://geth.ethereum.org/docs/rpc/server
  + https://github.com/ethereum/wiki/wiki/JSON-RPC#javascript-api
* 提示：
  + 下面主要以Javascript操作演示为主。

**personal账号管理**

1. 查看账号

* personal.listAccounts
* eth.accounts

1. 创建账号

**- personal.newAccount()**

1. 账户交易

* personal.sendTransaction(tx, passphrase)
  + tx是一个json的交易信息描述：
    - var tx = {from: "0x391694e7e0b0cce554cb130d723a9d27458f9298", to: "0xafa3f8684e54059998bc3a7b0d2b0da075154d66", value: web3.toWei(1.23, "ether")}
* 例子:

> amount = web3.toWei(1.5, 'ether')

> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1], value:amount})

1. 锁定账户交易

* personal.lockAccount(address)

1. 解锁账户交易

* personal.unlockAccount(address, passphrase, duration)
  + duration: 解锁周期，单位是10秒，，默认30是300秒。设置为0表示知道服务结束。

**miner挖矿操作**

1. start挖矿

* miner.start(number)
  + number是挖矿的线程数。

1. stop停止挖矿

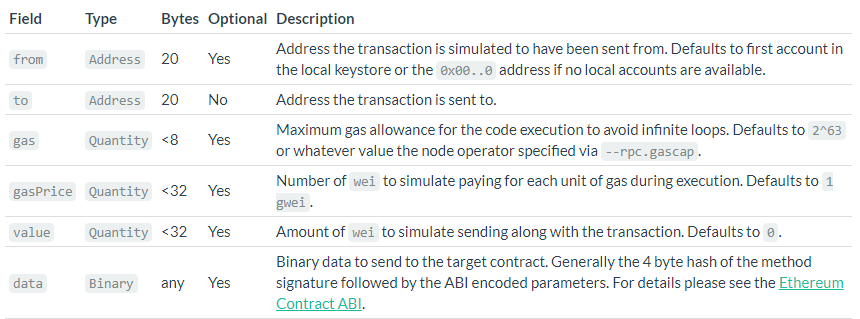
* miner.stop()
  + 返回bool值，表示是否停止成功；

1. setGasPrice设置燃气费
   * miner.setGasPrice(number)
   * 方法用来设置挖矿时交易的最小可接受 gas价格，任何低于此设置值的交易将被排除在挖矿过程之外。
2. setEtherbase设置挖矿基地

* miner.setEtherbase(address)
  + 账户地址。

**eth以太坊操作**

1. 基本信息
   * eth.accounts列表类型的所有账号
   * eth.blockNumber块数；
   * eth.coinbase挖矿基地；
   * eth.gasPrice燃气费价格；
   * eth.mining是否在挖矿；
   * eth.pendingTransactions排队的交易数，返回的是列表类型；
2. 交易调用的参数格式



1. getBalance获取账户余额
   * getBalance(地址)
2. nonce说明：
   * 以太坊中有两种nonce，一种是在区块中的nonce，主要是调整挖矿难度；一种是每笔交易中nonce。
   * 交易中的nonce主要说明交易顺序。防止交易问题。

**admin多节点操作**

1. addPeer添加远程节点

* admin.addPeer(url)
  + url就是服务启动输出的enode
  + 返回bool值
* 例子

> admin.addPeer("enode://a979fb575495b8d6db44f750317d0f4622bf4c2aa3365d6af7c284339968eef29b69ad0dce72a4d8db5ebb4968de0e3bec910127f134779fbcb0cb6d3331163c@52.16.188.185:30303")

**true**

1. peers返回所有的连接节点
   * admin.peers:返回列表

**txtool非标椎RPC调用功能**

1. txpool.content

* 列出当前待包含在下一个块中的所有事务的确切详细信息，以及仅计划在将来执行的事务的详细信息。

1. txpool.inspect

* 列出当前待包含在下一个块中的所有事务的文本摘要，以及仅计划在将来执行的事务的文本摘要。这是专门为开发人员定制的方法，用于快速查看池中的事务并发现任何潜在问题。

1. txpool.status

* 获取当前待包含在下一个块中的事务数，以及仅计划在将来执行的事务数。

**附录**

**以太币的单位**

* Ether币最小的单位是Wei，也是命令行默认的单位, 然后每 1000 进一个单位，依次是
  + kwei (1000 Wei)
  + mwei (1000 KWei)
  + gwei (1000 mwei)
  + szabo (1000 gwei)
  + finney (1000 szabo)
  + ether (1000 finney)
* web3提供两个转换函数
  + web3.toWei(ether)
  + web3.fromWei(wei)

**开发需要的环境**

**安装清单**

1. Node.js
   * 所有环境都依赖Node.js运行环境，包括区块链节点程序geth。
2. geth
   * 区块链节点服务程序。
3. solidity
   * 智能合约的编译器。
4. web3
   * Javascript语言的区块链访问模块。
5. truffle
   * 一个DApp的开发框架（类似Express等框架）
6. Webpack
   * 用来打包Web前端应用。方便使用ES6语法规范编写代码。在truffle创建项目的时候，可以指定使用Webpack打包。
7. 其他：
   * ganache-cli 配套truffle的客户工具，用来提供快捷的开发方式与提升开发效率。
     + ganache-cli提供对账号，以太币等区块链的基本初始化，便于快速开发。

**安装方式与指令**

* 因为整个区块链都是基于Javascript语言，及其Node.js虚拟机环境，所以安装方式都是使用Node.js的包管理器安装。当然两个安装除外：
  1. Node.js：C语言实现的Javascript虚拟机。
  2. geth：Go语言实现的区块链节点程序。
* 确认下已经安装好的Node.js与geth版本
* Node的版本

C:**\W**INDOWS**\s**ystem32>node --version

v12.16.1

* geth的版本

C:**\W**INDOWS**\s**ystem32>geth version

Geth

Version: 1.8.3-stable

Git Commit: 329ac18ef617d0238f71637bffe78f028b0f13f7

Architecture: amd64

Protocol Versions: [63 62]

Network Id: 1

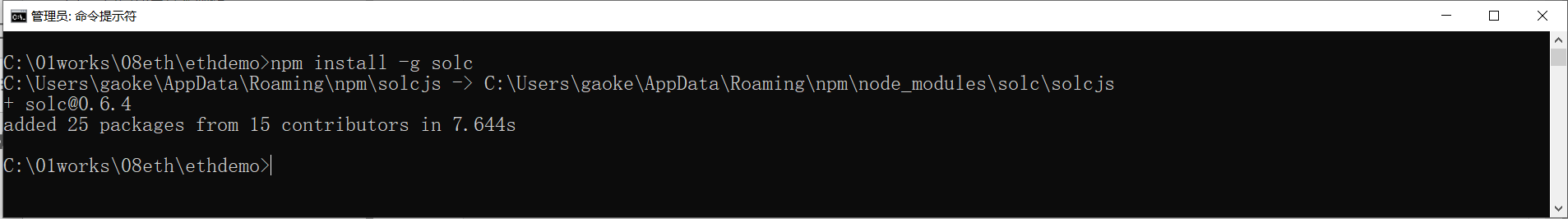
Go Version: go1.10

Operating System: windows

GOPATH=

GOROOT=C:**\g**o

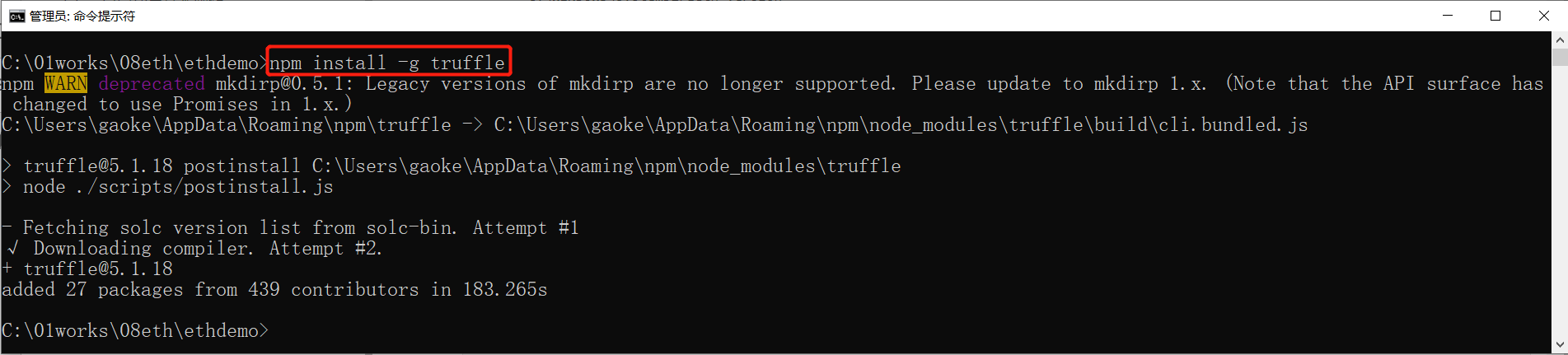
* 安装指令：(记得使用管理员权限)
  1. solidity:npm install –g solc
  2. web3:npm install –g web3
  3. truffle:npm install -g truffle
  4. Webpack:npm install -g webpack
  5. ganache-cli:npm install -g ganache-cli
* 注意：
  + 安装前最好安装一下：npm install node-gyp -g
  + 安装也可以考虑使用项目与局部安装 --save-dev
* solidity安装
  + npm install –g solc



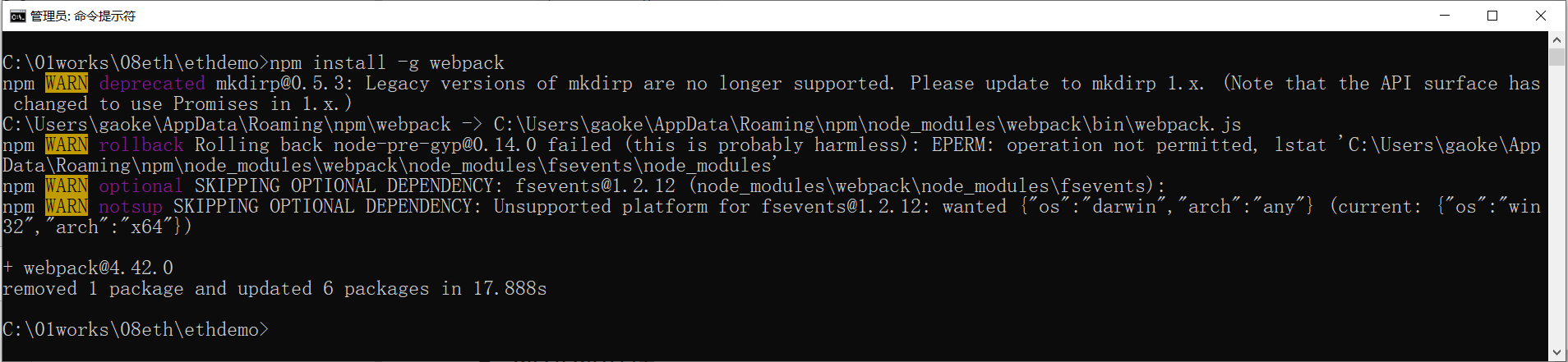
* Web3安装
  + npm install –g web3
  + Window中安装，记得在安装前，使用"vcvars64.bat"初始化一下window的C++编译器环境，因为这个安装要rebuild一些库。



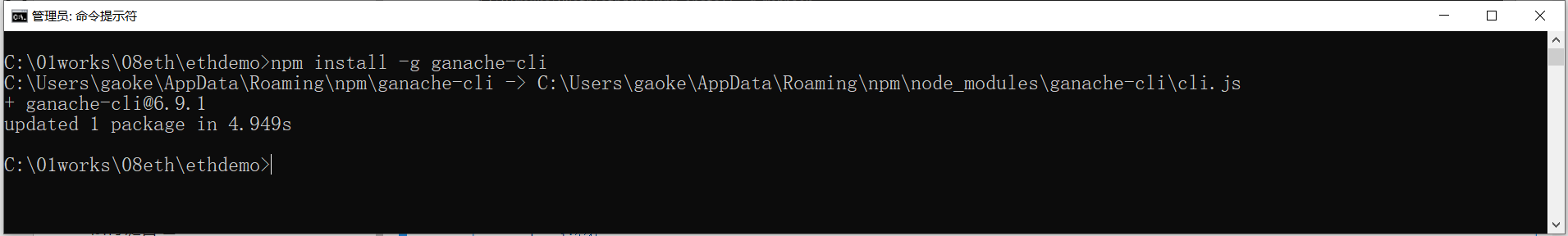
* truffle安装
  + npm install -g truffle



* webpack安装
  + npm install –g webpack

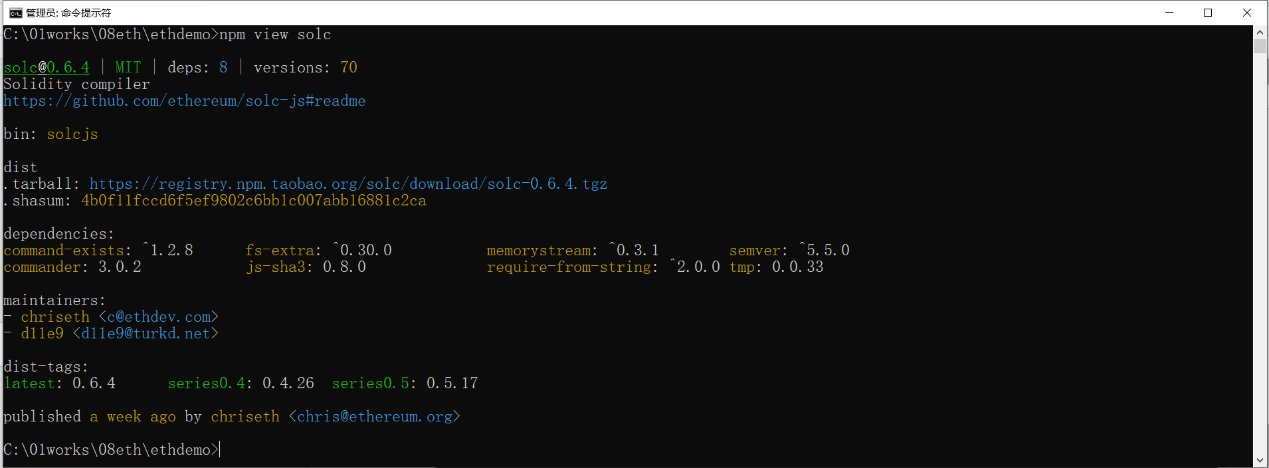


* ganache-cli安装：
  + npm install -g ganache-cli

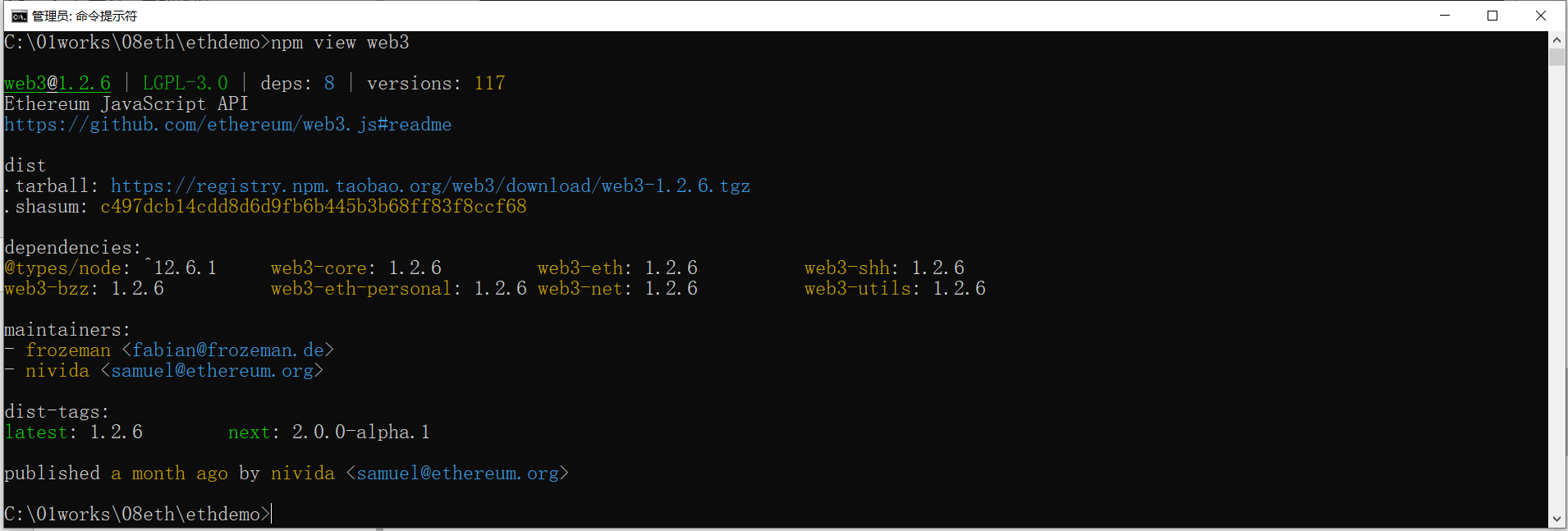


**安装验证**

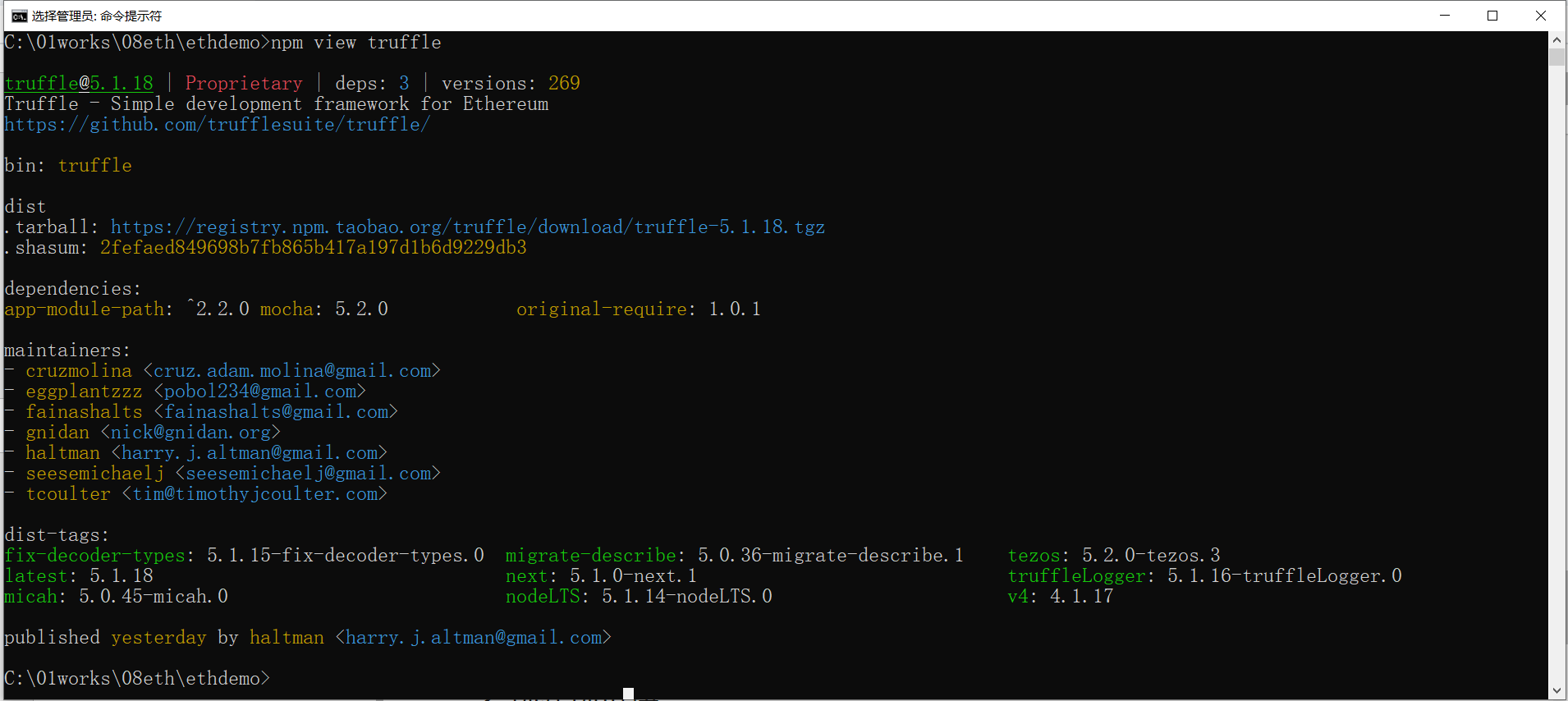
* 验证solidity
  + npm view solc



* 验证web3
  + 指令：npm view web3



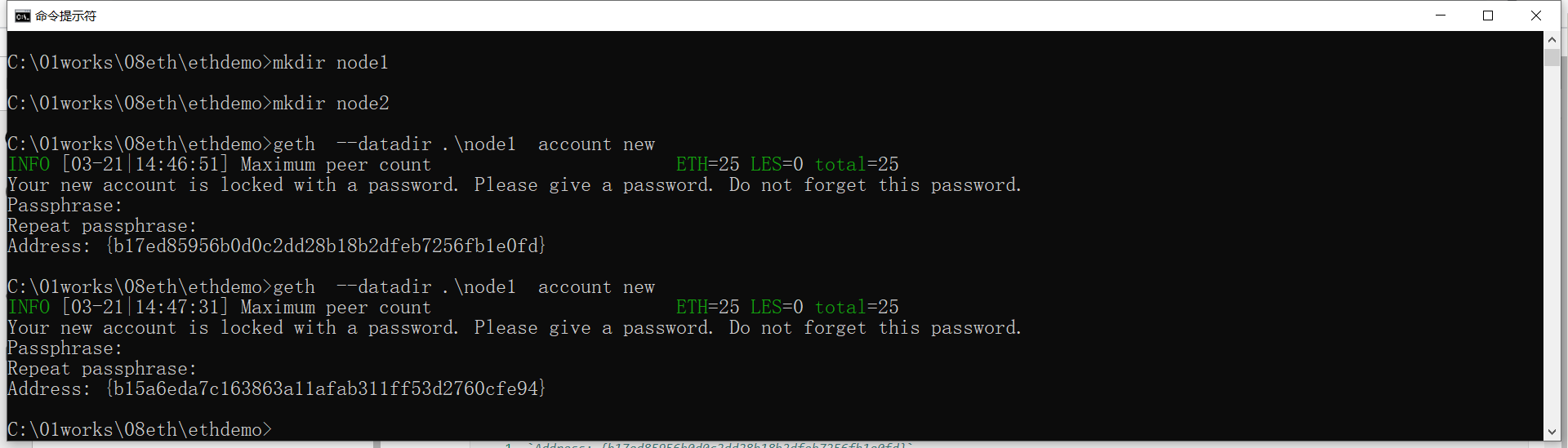
* 验证truffle
  + npm view truffle



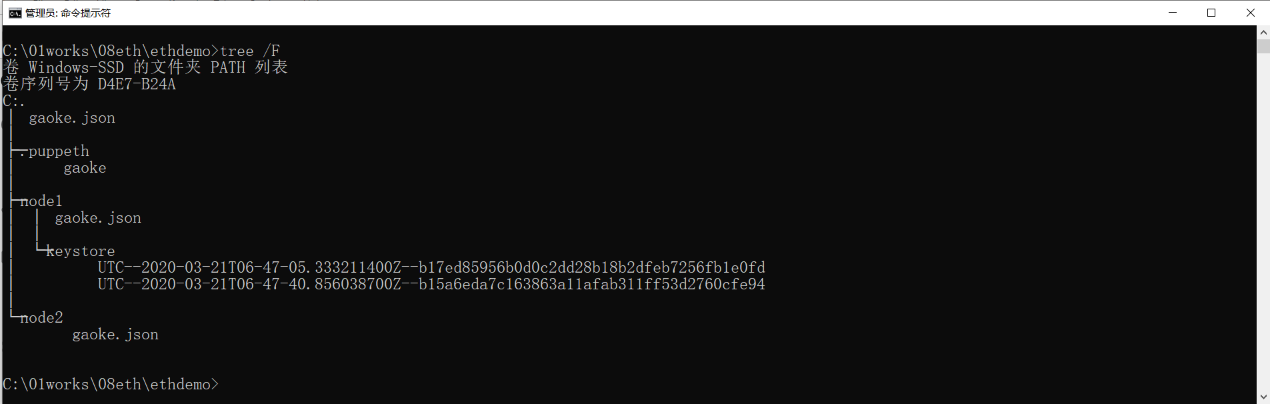
**创建创世块**

**生成创世块配置**

* 使puppeth创建创世配置文件。
  + 命令：puppeth
* 使用puppeth创建创世配置文件的过程，参考我们的上一节内容。
* 这里我们预先创建了两个账号
  1. Address: {b17ed85956b0d0c2dd28b18b2dfeb7256fb1e0fd}
  2. Address: {b15a6eda7c163863a11afab311ff53d2760cfe94}
* 创建2个账号的命令与截图：
  1. geth --datadir .\node1 account new
  2. geth --datadir .\node1 account new



* 创建好的创世文件：
  + gaoke.json（默认使用网络名作为创世文件名）
* 把gaoke.json文件赋值到每个节点的目录下操作。



**节点启动与基本操作**

* 注意：
  + 下面为了方便，使用bat脚本文件操作。所有指令都编写成bat脚本。
  + 注意：账号相关的文件要放到数据目录下哈。

**创建节点**

* 脚本文件：init\_geth.bat
  + ^是字符取消符，放在换行前，等价于取消换行，表示续航符，所有后面不能有其他字符。

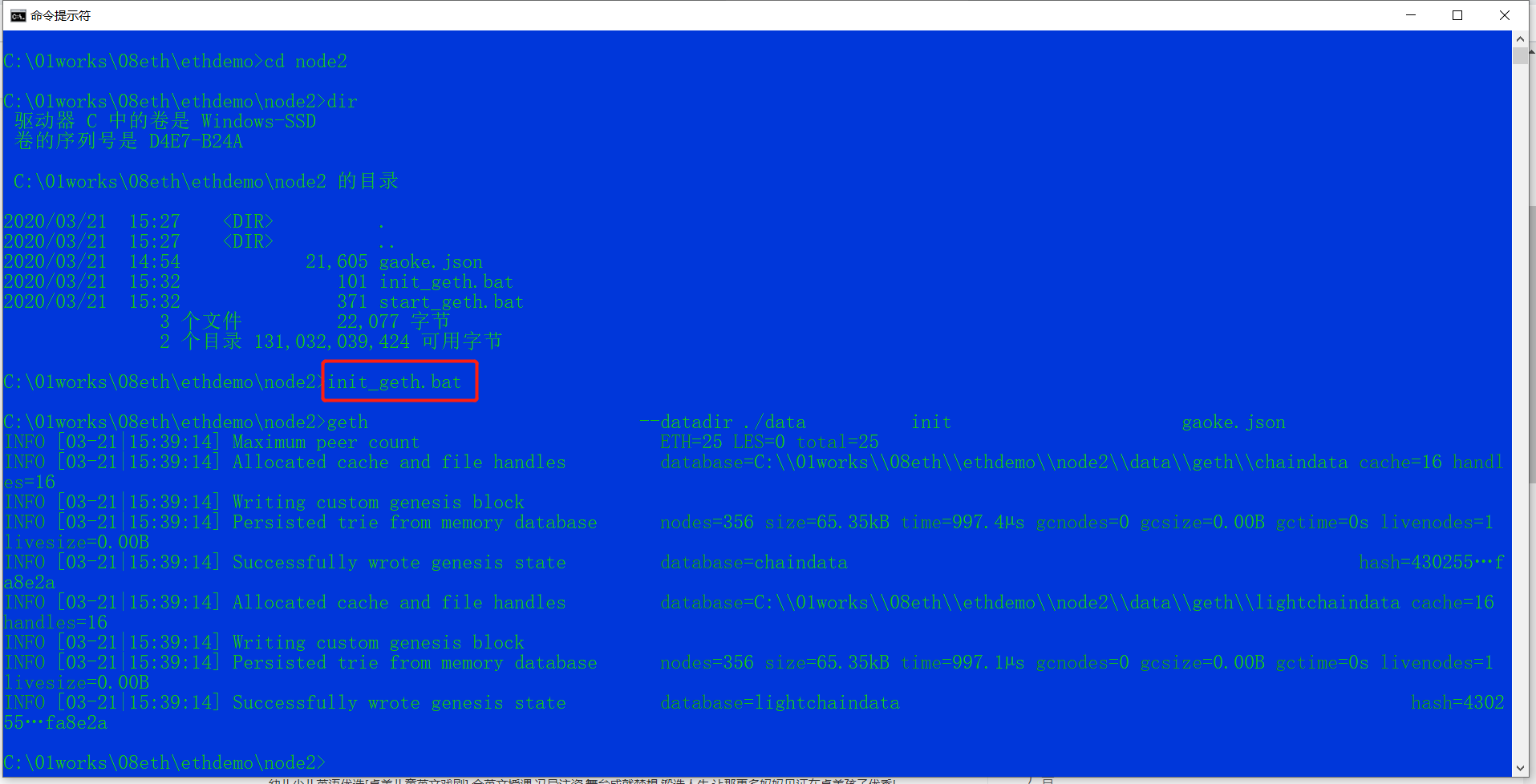
geth **^**

--datadir ./data **^**

init **^**

gaoke.json

* 执行脚本文件创建创世节点



**启动节点**

* 启动节点的脚本：start\_geth.bat

geth **^**

--rpc **^**

--rpcaddr 0.0.0.0 **^**

--rpcport "9999" **^**

--rpccorsdomain "\*" **^**

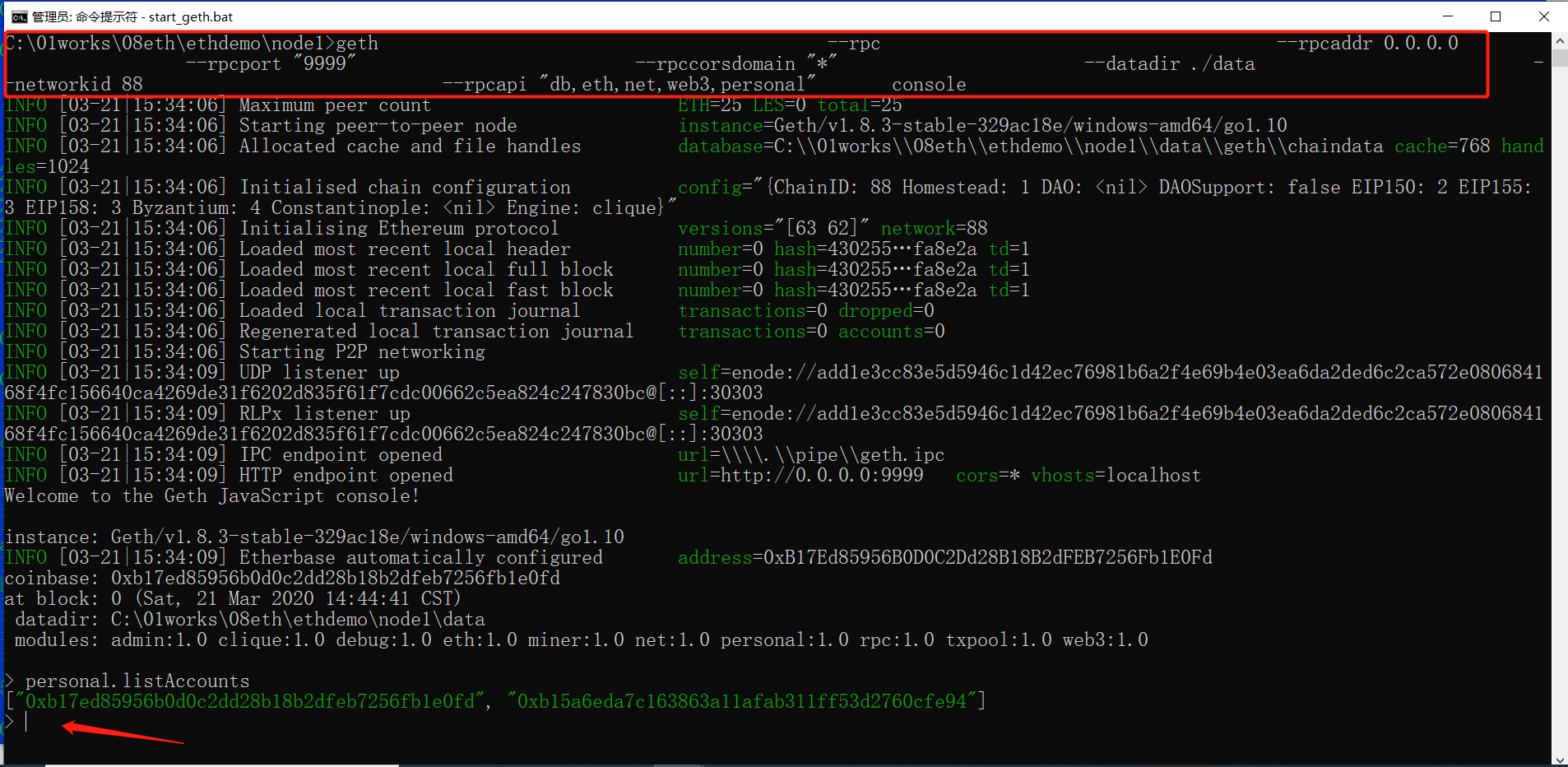
--datadir ./data **^**

--networkid 88 **^**

--rpcapi "db,eth,net,web3,personal" **^**

console

* 注意最后的console会顺带进入交互模式。如果想独占终端，可以不使用console。
  + 可以在其他终端使用命令连接带启动的节点。



* 注意：
  + 启动多个节点，会有三个地方产生冲突：
    1. 整个节点p2p的upd端口30303容易冲突，使用--port指定不同端口。
    2. rpc的端口容易冲突，使用--rpcport指定不同端口。
    3. ipc管道容易冲突，使用--ipcpath指定不同的通信管道。
* 两个节点的典型命令脚本如下：

geth **^**

--rpc **^**

--rpcaddr 0.0.0.0 **^**

--rpcport "7777" **^**

--rpccorsdomain "\*" **^**

--datadir ./data **^**

--networkid 88 **^**

--rpcapi "db,eth,net,web3,personal" **^**

--port 30503 **^**

--ipcpath "\\.\pipe\geth2.ipc" **^**

console

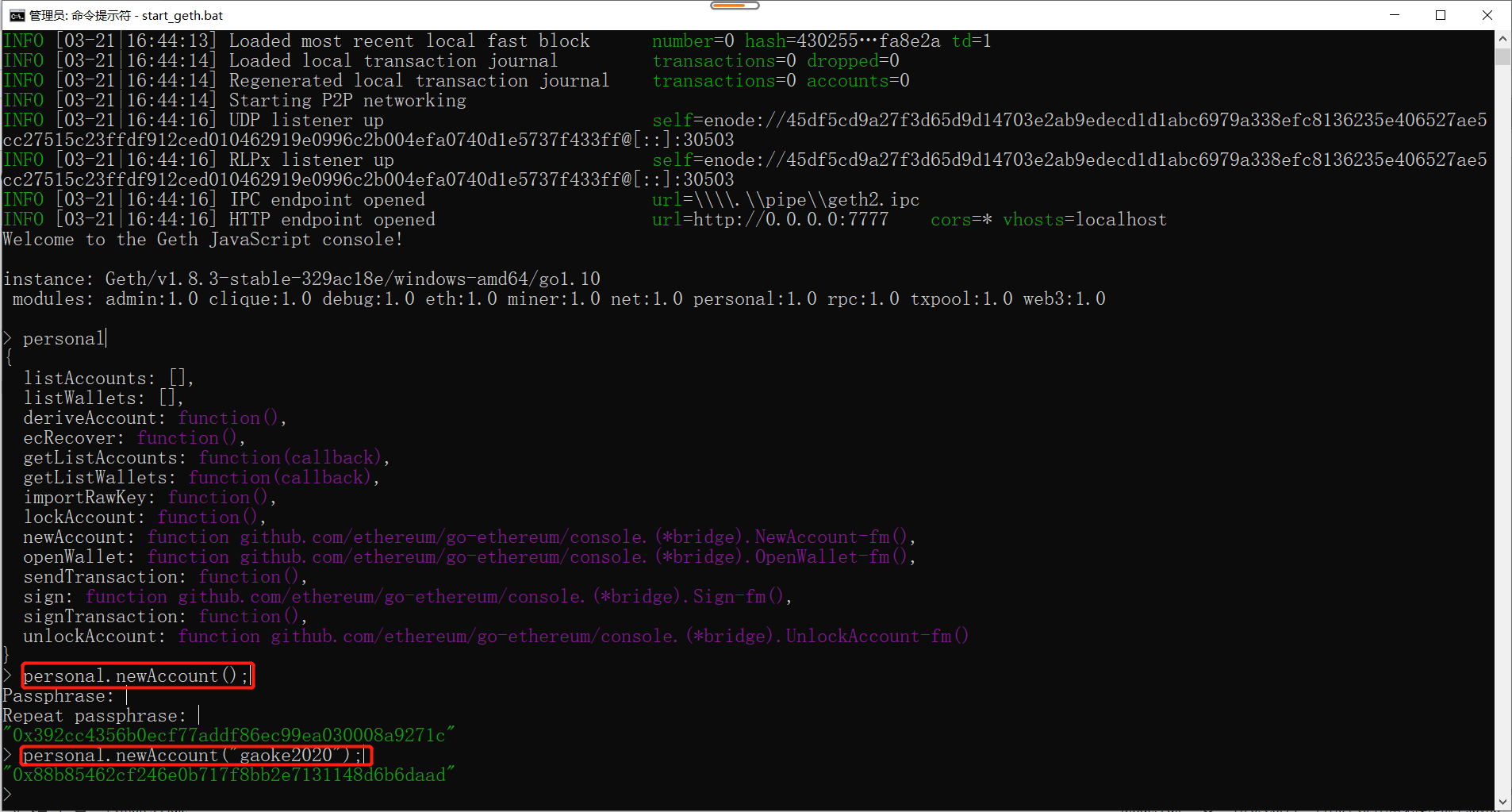
**私有链管理**

**创建账号**

* 创建账号两种方式：
  1. 可以使用geth命令行创建账号；
  2. 使用node.js对象personal对象创建账号；
* personal对象创建方式

personal.newAccount(); *//创建账号，后面会提示输入密码*

personal.newAccount("密码"); *//创建账号，直接填写密码*



**加入节点到私有链**

**使用admin实现节点管理**

* 节点相关的管理都在admin对象。

1. admin.nodeInfo对象包含节点信息

nodeInfo: {

enode: "enode://add1e3cc83e5d5946c1d42ec76981b6a2f4e69b4e03ea6da2ded6c2ca572e080684168f4fc156640ca4269de31f6202d835f61f7cdc00662c5ea824c247830bc@[::]:30303",

id: "add1e3cc83e5d5946c1d42ec76981b6a2f4e69b4e03ea6da2ded6c2ca572e080684168f4fc156640ca4269de31f6202d835f61f7cdc00662c5ea824c247830bc",

ip: "::",

listenAddr: "[::]:30303",

name: "Geth/v1.8.3-stable-329ac18e/windows-amd64/go1.10",

ports: {

discovery: 30303,

listener: 30303

},

protocols: {

eth: {

config: {...},

difficulty: 1,

genesis: "0x4302556c6eabe679a728c5de7f6ddf366094cb431e86ae0bfab2919673fa8e2a",

head: "0x4302556c6eabe679a728c5de7f6ddf366094cb431e86ae0bfab2919673fa8e2a",

network: 88

}

}

},

1. 节点相关属性与函数
   1. peers: []
      * 查看已经添加的节点。
   2. addPeer
      * 添加其他节点到私有链。
      * admin.addPeer(enode)
   3. removePeer
      * 从链删除节点。
      * admin.removePeer(enode)

* 添加节点：

> admin.addPeer("enode://45df5cd9a27f3d65d9d14703e2ab9edecd1d1abc6979a338efc8136235e406527ae5cc27515c23ffdf912ced010462919e0996c2b004efa0740d1e5737f433ff@[::]:30503");

**true**

* 添加节点后的状态：

> admin.peers

[{

caps: ["eth/63"],

id: "45df5cd9a27f3d65d9d14703e2ab9edecd1d1abc6979a338efc8136235e406527ae5cc27515c23ffdf912ced010462919e0996c2b004efa0740d1e5737f433ff",

name: "Geth/v1.8.3-stable-329ac18e/windows-amd64/go1.10",

network: {

inbound: **false**,

localAddress: "127.0.0.1:54994",

remoteAddress: "127.0.0.1:30503",

**static**: **true**,

trusted: **false**

},

protocols: {

eth: {

difficulty: 1,

head: "0x4302556c6eabe679a728c5de7f6ddf366094cb431e86ae0bfab2919673fa8e2a",

version: 63

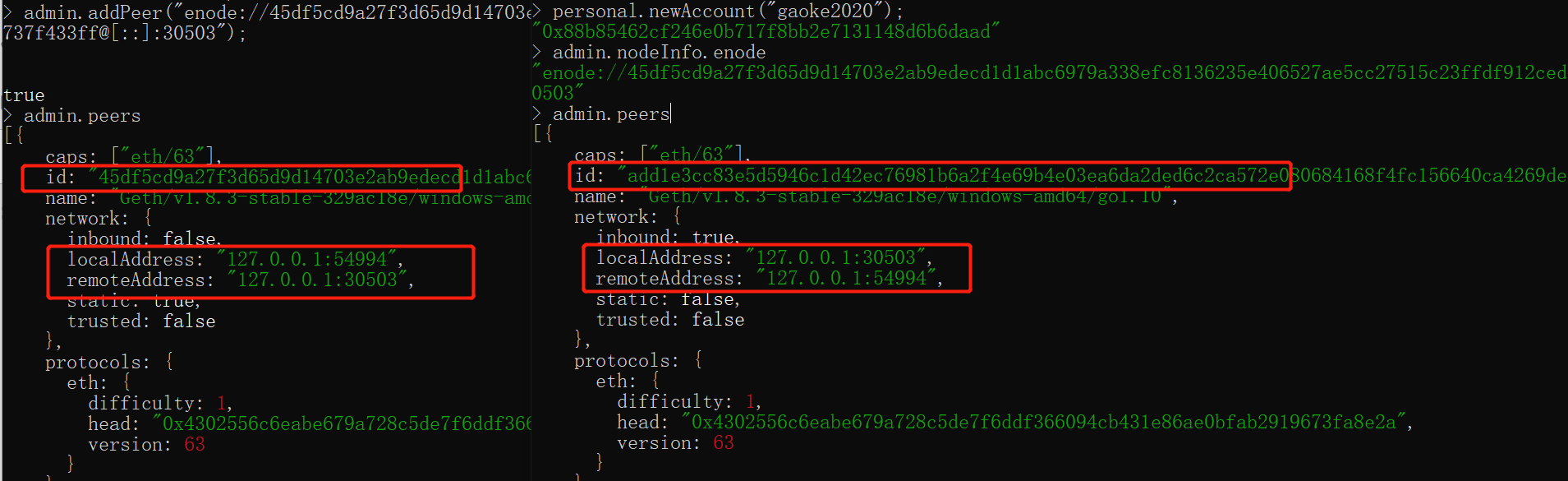
}

}

}]

>

* 注意：
  + 添加成功，可以在两个节点查看节点的信息。



1. 节点的核心信息：
   * enode：在启动节点的时候可以看见：enode://45df5cd9a27f3d65d9d14703e2ab9edecd1d1abc6979a338efc8136235e406527ae5cc27515c23ffdf912ced010462919e0996c2b004efa0740d1e5737f433ff@[::]:30503
   * enode可以在node控制台查看：
     + admin.nodeInfo.enode

**使用net对象查看节点的网络状态**

> net

{

listening: **true**,

peerCount: 1,

version: "88",

getListening: **function**(callback),

getPeerCount: **function**(callback),

getVersion: **function**(callback)

}

>

**静态节点添加**

* 在datadir下的geth目录下创建一个文件：static-nodes.json
  + 并且添加其他节点的enode值，则当前节点启动会自动连接。

**交易**

**交易的条件**

* 解锁账号才能交易：主要是转出账号解锁。
  + personal.unlockAccount(账号，密码，解锁保留的时间长度)
    - 时间周期单位为秒，0表示解锁到geth退出位置。
* 交易需要挖矿才能产生交易执行：
  + miner.start(线程数)；

**交易**

* 交易函数在eth中，由函数sendTransaction提交交易：
  + eth.sendTransaction(交易对象)

- 交易内容使用json格式对象表示： 1. from - String|Number: - 交易发送方账户地址，不设置该字段的话，则使用web3.eth.defaultAccount属性值。 - 可设置为一个地址或本地钱包web3.eth.accounts.wallet中的索引序号 2. to - String: 可选 - 消息的目标地址，对于合约创建交易该字段为null 3. value - Number|String|BN|BigNumber: (optional) - The value transferred for the transaction in wei, also the endowment if it’s a contract-creation transaction. 4. gas - Number: 可选，默认值：待定， - 用于交易的gas总量，未用完的gas会退还 5. gasPrice - Number|String|BN|BigNumber: 可选， - 该交易的gas价格，单位为wei，默认值为web3.eth.gasPrice属性值 6. data - String: 可选， - 可以是包含合约方法数据的ABI字符串，或者是合约创建交易中的初始化代码 7. nonce - Number: 可选， - 使用该字段覆盖使用相同nonce值的挂起交易 ```

* 发起一个交易的例子：

personal.unlockAccount(personal.listAccounts[1], "gaoke2020", 0)

eth.sendTransaction({from: "0xb15a6eda7c163863a11afab311ff53d2760cfe94",to: "0x392cc4356b0ecf77addf86ec99ea030008a9271c", value: "74000000000000000"})

miner.start(3)

* 注意：
  + 其他节点也需要挖矿。
  + 挖矿需要签名权限的账号。（注意以太坊两种账号方式：权限账号与工作量账号）

**删除交易**

**查看交易**

1. 获取块数（账本）
   * blockNumber: 1,
2. 获取块信息
   * eth.getBlock(=块编号);

**智能合约**

**合约编译工具**

C:**\U**sers**\g**aoke>solcjs --help

Usage: solcjs [options]

Options:

-V, --version output the version number

--version Show version and exit.

--optimize Enable bytecode optimizer.

--bin Binary of the contracts in hex.

--abi ABI of the contracts.

--standard-json Turn on Standard JSON Input / Output mode.

-o, --output-dir <output-directory> Output directory **for** the contracts.

-h, --help output usage information

**在geth中配置智能合约编译器**

**编写智能合约**

pragma solidity ^0.6.4;

contract Calc{

*/\**

*区块链存储*

*\*/*

uint count;

*/\**

*执行会写入数据，所以需要transaction的方式执行。*

*\*/*

function add(uint a, uint b) **public** returns(uint){

count++;

**return** a + b;

}

*/\**

*执行不会写入数据，所以允许call的方式执行。*

*\*/*

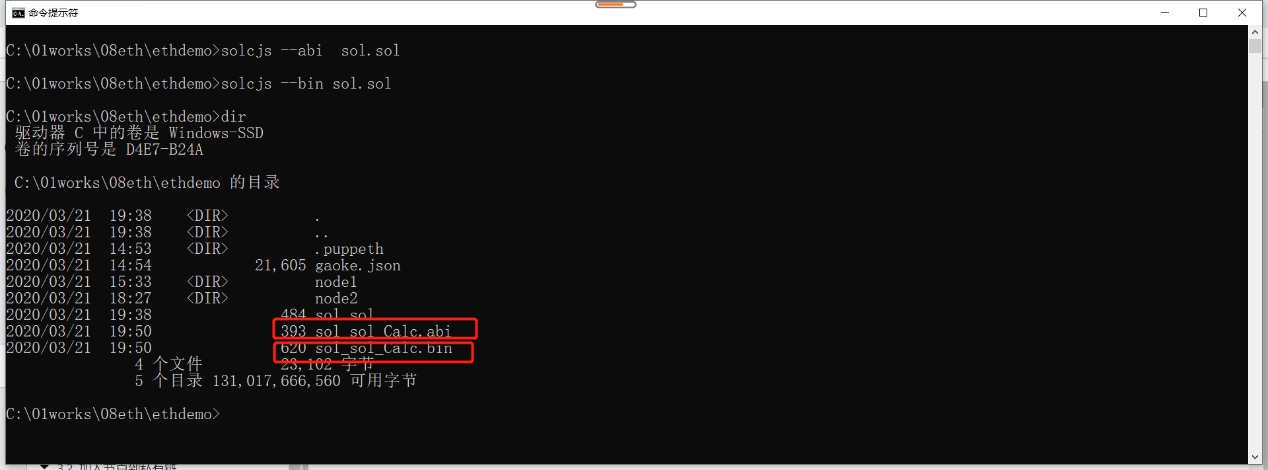
function getCount() **public** view returns (uint){

**return** count;

}

}

**使用solcjs编译智能合约**



**部署智能合约**

eth.sendTransaction({from:eth.accounts[1],gas:"80000",data:""})

INFO [03-21|19:53:34] Submitted contract creation fullhash=0x1555fb27c2c0908178f6230bbe10b6e3a946ec319764f362655c15ea9ffca013 contract=0xC29BFa99d6e07bb10f5225Ee48c59569f7b7D6DC

"0x1555fb27c2c0908178f6230bbe10b6e3a946ec319764f362655c15ea9ffca013"

>

**var** cc = eth.contract([{"inputs":[{"internalType":"uint256","name":"a","type":"uint256"},{"internalType":"uint256","name":"b","type":"uint256"}],"name":"add","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"nonpayable","type":"function"},{"inputs":[],"name":"getCount","outputs":[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"}])

* 访问地址：
  + var adr = cc.at(eth.accounts[1]);
* 调用：
  + adr.函数.call(参数)
* javascript调用例子

**let** Web3 = require("web3");

**let** Tx = require("ethereumjs-tx");

*//合约地址*

**let** contractAddr = "0x35...2827";

*//创建web3对象*

**let** web3 = **new** Web3;

*//连接到ropsten*

**let** INFURA\_api = "https://ropsten.infura.io/v3/1b...4b0";

web3.setProvider(**new** Web3.providers.HttpProvider(INFURA\_api));

*//账户地址*

**let** fromAddr = "0x97...e05B7";

**let** count = web3.eth.getTransactionCount(fromAddr);

**let** gasPrice = web3.eth.gasPrice;

**let** gasLimit = 90000;

**let** rawTransaction = {

"from": fromAddr,

"nonce": web3.toHex(count),

"gasPrice": web3.toHex(gasPrice),

"gasLimit": web3.toHex(gasLimit),

"to": contractAddr,

"value": "0x0",

"data": "0xa3c8e393" *//函数名callme取keccak后的bytes4*

};

*//账户私钥，不包含'0x'字符*

**let** priv\_key = **new** Buffer.from("bb...623ef3ed3c", "hex");

**let** tx = **new** Tx(rawTransaction);

*//用私钥签名交易信息*

tx.sign(priv\_key);

**let** serializedTx = tx.serialize();

*//发送交易*

web3.eth.sendRawTransaction('0x'+serializedTx.toString('hex'),

**function**(err, hash) {

**if** (!err)

console.log(hash);

**else**

console.log(err);

});