阿拉伯人
积硅步，至千里。微信:wucheng1985

分类：以太坊-区块链

以太坊白皮书

一个下一代智能合约和去中心化应用平台

中本聪的比特币，在2009年的时候，经常被认为是一个在金融和货币领域里比较激进的做法，它作为世界上第一个数字货币，它既没有强大的背景支持，也没有所谓的“内在价值”，更没有集中的发行人或控制者，人们甚至都不知道中本聪这人是谁o(╯□╰)o。然而，另一方面，可以说是更加重要的一面，比特币的底层的区块链技术作为一个去中心化的一致性的基础技术，受到越来越多的技术专家的注意。现在有很多采用区块链技术的山寨比特币，如用区块链上的数字来代表客户的资产和金融工具的彩色币(“colored coins”)，不可替代的资产如姓名币(“Namecoin”)，以及更加复杂的应用，包括让数字资产直接被一段可以执行任意规则的代码来控制的智能合约 (“smart contracts”)，甚至是基于区块链的“去中心化的自治组织（decentralized autonomous organizations）” (DAOs)。以太坊所尝试的是提供一个区块链，它内置了一个完全成熟的图灵完备的程序语言，可以用这个语言来创建“合约”，创建以上所说的任何的系统，还有许多我们还没有想到的，只要简单的使用几行代码就可以构建起逻辑。

目录

- 比特币和现有概念的介绍
  - 历史
  - 比特币一个状态转移系统
  - 挖矿
  - 梅克尔树（Merkle Trees）
  - 山寨区块链应用
  - 脚本 （Scripting）
- 以太坊
  - 以太坊账户
  - 信息和交易
  - 以太坊状态转移函数

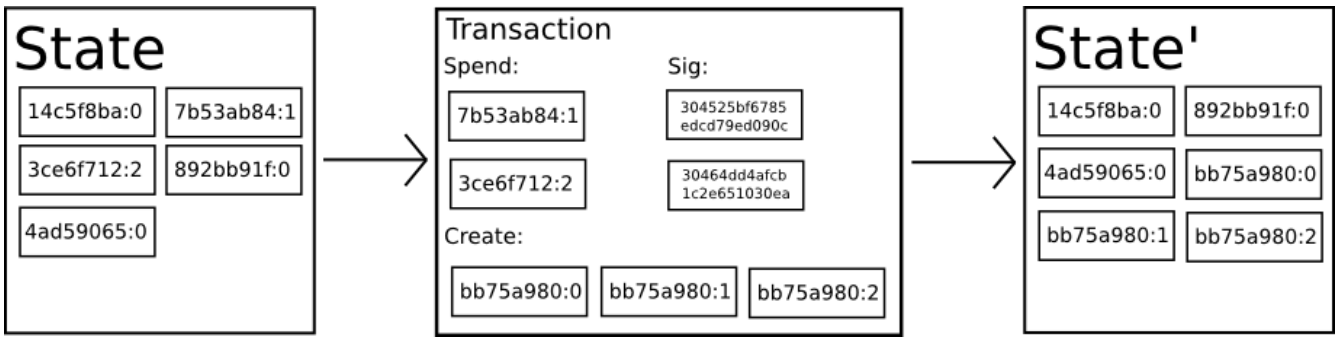
- 代码执行
- 区块链和挖矿
- 应用
  - 令牌系统
  - 金融衍生工具
  - 身份和信用系统
  - 分散的文件存储
  - 去中心的自治组织
  - 更多的应用

比特币和现有概念的介绍

历史

去中心化的货币的概念，和上面所说的备选的应用像产权登记等已经存在了几十年。上世纪80年代和90年代的匿名电子现金协议主要使用了一种被称为“乔姆盲签（Chaumian Blinding）”的加密技术，这种技术为这些新货币提供了很高的隐私保护，但是由于他们的基础协议在很大程序上需要依赖于一个中央中介，因此未能获得支持。1998年，戴伟（Wei Dai）的b-money首次引入了通过解决计算难题和去中心化的共识来创造货币的思想，但是该建议并未给出实现去中心化的共识的具体方法。2005年，芬尼（Hal Finney）引入了“可重复的工作证明”（reusable proofs of work）概念，它同时使用b-money的思想和Adam Back提出的计算困难的哈希现金（Hashcash）难题来创造密码学货币。但是，这种概念再次迷失于理想化，因为它需要可信任的计算作为后端。在2009年，一种去中心化的货币由中本聪在实践中首次得到实现，通过使用公共密钥密码来管理所有权，并通过一种一致性算法来跟踪货币的持有者，这种算法被称为“工作证明”。这“工作证明” 的算法是一种突破，因为它同时解决了2个问题。首先，它提供了一种简单的、有节制的有效的共识算法，允许网络中的节点一起同意比特币总帐状态的一组更新。其次，它提供了一种机制，允许任何节点自由进入共识的处理进程，从而解决了谁来影响共识的政治难题，同时阻止女巫攻击。 这参与共识投票的节点的投票权比重是直接和节点它们自己的算力挂钩的。 从那时起，也一个新的方案被提出，称之为“权益证明”，计算一个节点的权重，与它所拥有的货币占比成比例，而不是计算资源。关于这两种方法的相对优缺点的讨论超出了本文的范围，但是应该注意的是，这两种方法都可以作为加密货币的支柱。

比特币一个状态转移系统



从技术角度来说,这加密货币的账本，如比特币可以被看作为一个状态转移的系统,在这个系统里,有一个包含现在所有已存在的比特币的持有者的状态，并且有一个“状态转移函数”可以使用一个状态和一个交易来产生一个新的状态。在一个标准的银行体系里,这状态是一个资产负债表,当一个交易要求把X$的钱，从A转移到B时， 那么它的状态转移函数就会从A的账户中减去数量为$X的金额，然后在B的账户中增加数量为$X的金额。如果A的账户中没有$X的钱，那么状态转移函数就会返回一个错误。所以，我们可以做如下定义:

//使用一个状态和一个交易才生一个新状态，或者返回错误

```
APPLY(S,TX) -> S’ or ERROR
```

在银行系统中，它可以定义成这样:

```
APPLY({ Alice: $50, Bob: $50 },"send $20 from Alice to Bob") = { Alice: $30, Bob: $70 }
```

或者:

```
APPLY({ Alice: $50, Bob: $50 },"send $70 from Alice to Bob") = ERROR
```

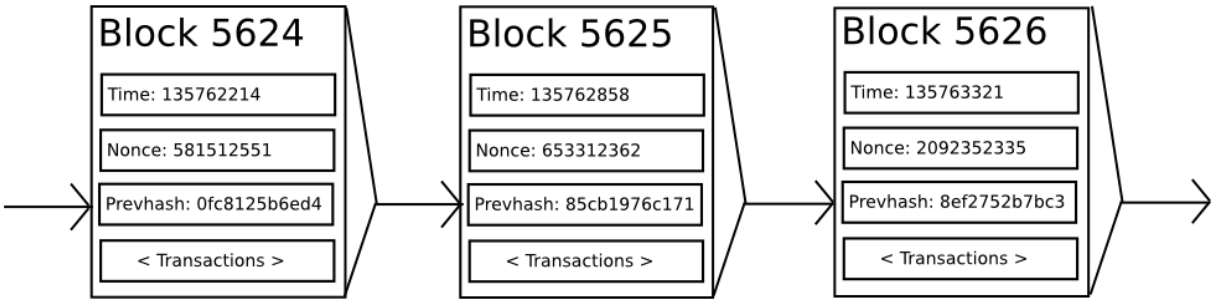
这“状态”在比特币中是指所有的已经被挖出的但是还没消费的硬币的集合 (技术上,“没有被花掉的交易的产出(unspent transaction outputs)”或 UTXO)，每个 UTXO都有一个面值和一个持有者 (持有者是由20个字节组成的地址，其本质是一个加密的公钥). 一个交易包含了一个或多个输入，每一个输入都包含了对一个已存在的UTXO的引用，和用持有者的地址所关联着的私钥来产生的一个加密签名,并且会产生一个或多个输出,每一个输出包含一个用于添加到状态的新的UTXO。

这状态转移函数 APPLY(S,TX) -> S’ 可以被大概的定义如下:

1. 在 TX 中的每个输入：
   - 如果被引用的UTXO不在 S 里，返回一个错误.
   - 如果提供的签名和UTXO的所有者匹配不上, 返回一个错误.
2. 如果所有的输入的 UTXO 的面值的和少于所有输出的UTXO的面值的和，返回一个错误.
3. 返回一个所有输入的UTXO都被移除的，所有输出的UTXO都被加进的新的 S’ .

第一个步骤的前半部分防止交易的发送方消费不存在的硬币, 第一步骤的后半部分阻止了交易的发送方使用别人的硬币, 第二个步骤是强制的价值保护。为了使用这种支付方式，协议如下。 假设 Alice 想发送11.7 BTC 给 Bob. 首先, Alice 要找到一组她自己拥有的有效的UTXO ，且总数要至少不低于 11.7 BTC. 实际上, Alice 不会恰巧刚好拥有11.7 BTC; 她得到的最小值是6+4+2=12.，她然后使用这3个输入和2个输出创建了一个交易。 这第一个输出是 11.7 BTC ，是输出Bob,第二个输出是剩下的 0.3 BTC “找零”。如果 Alice 没有要求把这个找零发送到她自己的账户上, 那么这矿工可以要求拥有这个零钱.

挖矿



如果我们有一个可信任的中央服务器, 那么实现这个系统是一件很简单的事情; 就按照需求所描述的去编写代码即可，把状态记录在中央服务器的硬盘上。 然而，与比特币一样，我们试图去建立一个去中心化的货币系统, 所以，我们需要把状态转移系统和一致性系统结合起来，以确保每个人都同意这交易的顺序。比特币的去中心化的一致性处理进程要求网络中的节点连续不断的去尝试对交易进行打包，这些被打成的包就称为“区块”。 这个网络会故意的每隔10分钟左右就创建一个区块, 每一个区块里都包含一个时间戳，一个随机数，一个对上一个区块的引用 ，和从上一个区块开始的所有交易的列表。随着时间的推移，这会创建一个持久的，不断增长的区块链，这个区块链不断的被更新，使其始终代表着最新的比特币总账的状态。

在这个范例中，用来验证一个区块是否有效的算法如下:

1. 检查其引用的上一个区块是否存在并且有效.
2. 检查这个区块的时间戳是否大于上一个区块的时间戳 并且小于2小时之内
3. 检查这区块上的工作证明是否有效.
4. 让 S[0] 成为上一个区块的最末端的状态.

5. 假设 TX 是这个区块的交易列表，且有 n 个交易。 做for循环，把i从0加到到n-1， 设置 S[i+1] = APPLY(S[i],TX[i]) 如果任何一个申请(APPLY)返回错误，则退出并且返回。
6. 返回true,并且把 S[n] 注册成这个区块最末端的状态。

从本质上说，区块中的每一个交易都必须提供一个有效的状态，从交易执行前的标准状态到执行后的一个新的状态。 注意，状态并没有以任何方式编码进区块中;它纯粹是一个被验证节点所记住的抽象，并且它只能用来被从创世区块起的每一个区块进行安全的计算，然后按照顺序的应用在每一个区块中的每一次交易中。此外，请注意矿工把交易打包进区块的顺序是很重要的; 如果一个区块中有2个交易A和B,B花了一个由A创建的UTXO, 那么如果A比B更早的进入区块，那么这个区块将是有效的，不然就是无效的。

在上述列出的验证条件中，“工作证明” 这一明确的条件就是每一个区块的2次SHA256 哈希值, 它作为一个256位的数字,必须小于一个动态调整的目标值, 截止到本文写作的时间，该动态调整的值的大小大约是 2的187次方。这样做的目的是为了让创建区块的算法变难, 从而，阻止幽灵攻击者从对它们有利的角度出来，来对区块链进行整个的改造。因为 SHA256 被设计成一个完全不可预测的伪随机函数, 这创建一个有效区块的唯一的方法只有是不断的尝试和出错， 不断对随机数进行递增，然后查看新的哈希值是否匹配。

按照当前的目标值2的187次方，这个网络在找到一个有效的区块前，必须进行2的69次方次的尝试; 一般来说, 每隔2016个区块，这个目标值就会被网络调整一次 ，因此网络中平均每隔10分钟就会有一些节点产生出一个新的区块。为了补偿这些矿工的计算工作, 每一个区块的矿工有权要求包含一笔发给他们自己的12.5BTC（不知道从哪来的）的交易。另外,如果任何交易，它的总的输入的面值比总的输出要高，这一差额会作为“交易费用”转给矿工。顺便提一下，对矿工的奖励是比特币发行的唯一途径，创世状态中并没有比特币。

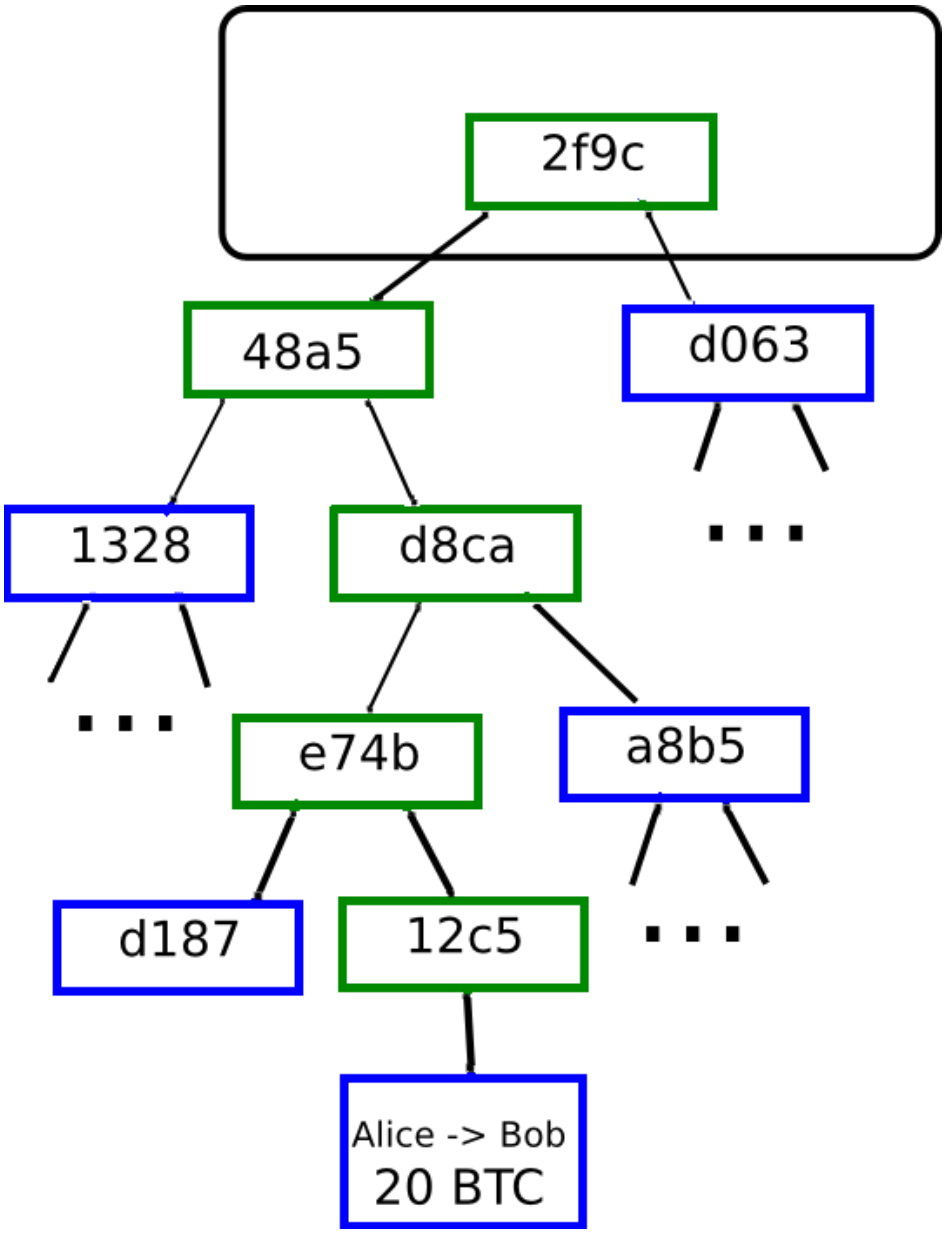
为了更好的理解挖矿的目的,让我们来检测一下，当恶意攻击发生时会发生什么. 由于比特币的底层的加密技术众所周知是安全的,所以这攻击者的目标将是比特币系统中的某个部份，那就是没有被密码直接保护的：交易的次序。攻击者的策略很简单:

1. 发送 100 BTC 到一个商人，以兑换一些商品 (最好是快速交易的数字商品)
2. 等待商品的发货
3. 创建另一个交易，发送同样的 100 BTC 给自己
4. 尝试让网络相信他发给他自己的那个交易是最新出现的。

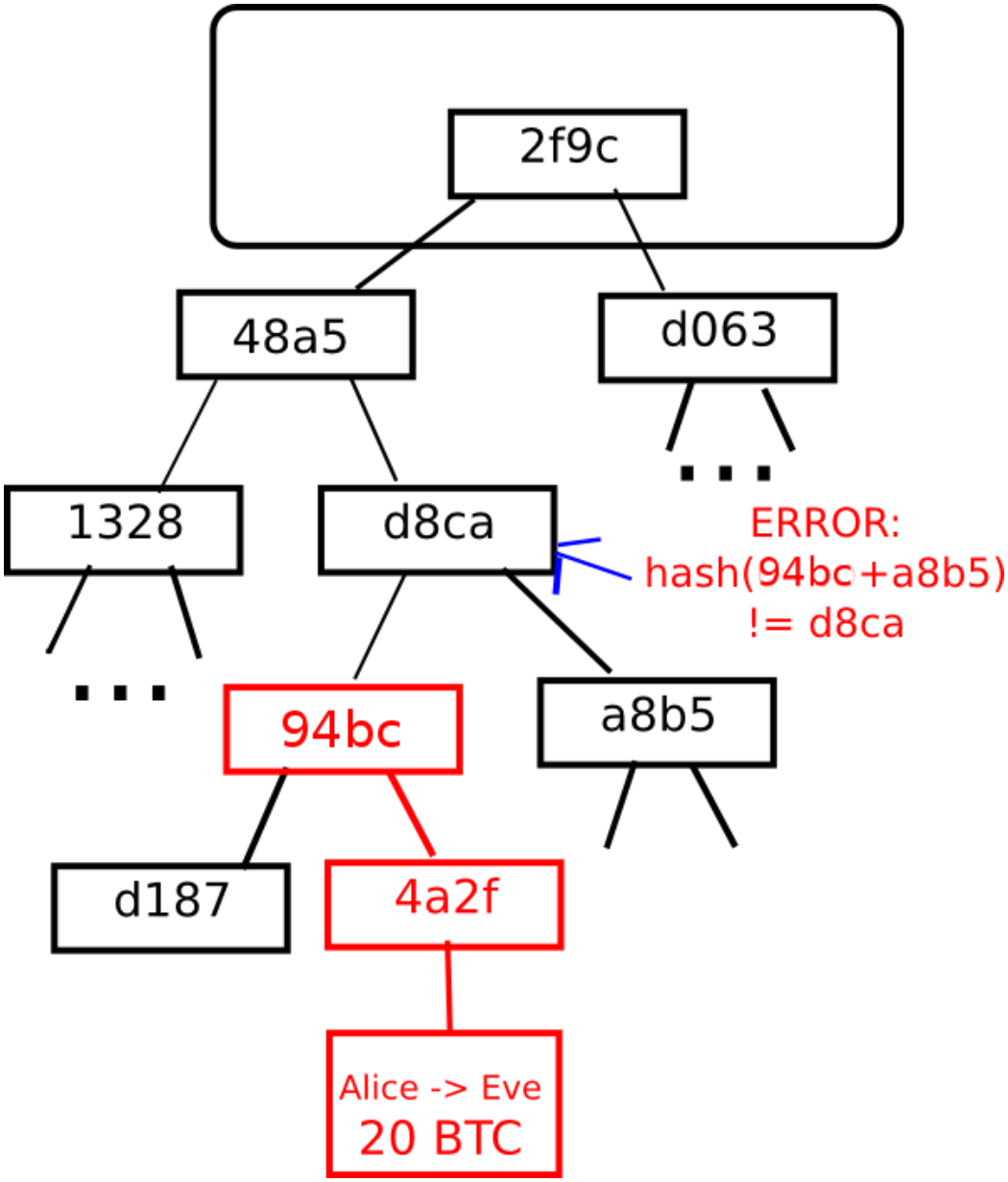
一旦步骤 (1)发生, 几分钟后，一些矿工就会把这个交易打包进一个区块, 声明该区块编号是 270000. 大约一个小时后, 在那个区块之后又会有新增的五个区块添加到区块链中, 这五个区块都间接的指向了那个交易，从而“确认”了它。在这一刻，商家接收到了货款，并且发出了商品; 因为我们假设是一个数字商品，所以攻击者能立刻就收到货。现在攻击者创建另一个交易，向他自己发送这100 BTC 。如果这个攻击者只是简单的释放了这个交易，那么这个交易将不会被执行; 矿工将会尝试运行 APPLY(S,TX) ，注意那个 TX 消耗一个在以太坊状态中已经不存在的UTXO。因此，这个攻击者创建了一个比特币区块链的“分支” , 开始挖取区块270000的另一个版本，这个版本指向同样的区块269999作为一个父亲,但是用这新的交易替换了旧的。因为这区块的数据是不同的 , 这就需要为相关的区块重新做工作证明。 此外,这个攻击者的新的版本的区块270000有一个不同的哈希，所以这已存在的区块 270001 到270005 不会指向它; 因此, 这原来的链和这攻击者的新链是完全独

立的。区块链的规则是：在分支中最长的区块链链将会变成真正的链，所以合法的矿工将会继续在270005这条链上工作，同时攻击者自己一个人单独的工作在新版本的270000这条链上。为了让攻击者他自己的区块成为最长，他需要的计算能力比其他网络的总和还要多（“51%攻击”）。比特币的区块依赖之前所有区块的哈希。一个拥有巨大计算能力的攻击者可以重新设计工作的证明(PoW)，并最终获得大量的比特币。

默克尔树(Merkle Trees)



01: 在默克尔树中只要提供少数的几个节点就可以给出一个分支有效性的证明



02: 试图改变默克尔树的任一部分都将会导致链条上在某处发生不一致的情况

比特币的一个重要特性，这区块是存在一个多级数据结构中的 。一个区块的“哈希值”实际上只是这个区块的头信息的哈希值，一个大约200个字节的数据，其中包含了时间戳，随机数，上一个区块的哈希和一个存储了这个区块中所有交易的称之为默克尔树的数据结构的根哈希。 默克尔树是一种二叉树，包含了一组节点，它们的含有基础信息的树根有大量的叶子节点，一组中间节点，每一个节点都是它的2个子节点的哈希，然后，最终的一个根节点，也是由它的2个子节点的哈希形成，代表着这树的“顶端”。 这个默克尔树的目的是允许在一个区块中的数据能够被零散的传递: 一个节点只能从一个源来下载一个区块的头信息，树的一小部分关联着另一个源 ，并且任然可以保证所有的数据都是正确的。之所以这样做行得通，是因为哈希值都是向上传导的: 如果一个恶意的用户试图在默克尔树的底部替换一个假的交易, 这个更改将导致上面的节点发生变化，然后上面的节点的变化又会导致上上面的节点发生变化，最终改变这个数根节点，因此也改变了这区块的哈希，导致这个协议把它注册成一个完全不同的区块 (几乎可以肯定是一个无效的工作证明).

这默克尔树协议对比特币的长期可持续发展是必不可少的。比特币网络中的一个“完整节点”, 截止到2014年，占用了大约15G的磁盘空间，并且每月正在以10亿字节的速度递增。目前，这对于电脑来说是没有问题的，但是在手机上却是不现实的。在以后的将来，只有商业的和业余爱好者才能参与玩比特币。一个称之为 “简化支付验证（simplified payment verification）” (SPV)的协议 允许另一种类型的节点存在，这种节点称之为 “轻节点（light nodes）”, 其下载区块的头信息,在这区块头信息上验证工作证明，然后只下载与之交易相关的“分支” 。 这使得轻节点只要下载整个区块链的一小部分，就可以安全地确定任何一笔比特币交易的状态和账户的当前余额。

山寨区块链应用

将底层的区块链理念应用到其他概念上的想法也有很长的历史了。在2005年, 尼克萨博提出了这个概念 “用所有权保护产权（secure property titles with owner authority）”, 文档描述了 “复制数据库的新技术”将如何使用一个基于区块链的系统来存储谁拥有土地的注册登记信息， 建立一个精心设计的框架，包括“宅地”，“逆权侵占”和 “地税”等概念。然而，不幸的是，那个时候一个没有有效的复制数据库信息，因此，协议从未付诸实践。然而，在2009年之后，比特币的去中心化的共识一被开发出来，许多山寨应用就迅速涌现出来。

- **Namecoin** – 创建于2010年，名字币是一个去中心化的名字注册数据库 。 在去中心化的协议中，像Tor, Bitcoin 和BitMessage, 这些需要有一些识别账号的方法，这样用户就可以和其他人进行交互了，但是现存在的所有的解决方案中，这个标识符都是一个伪随机的哈希字符串像 `1LW79wp5ZBqaHW1jL5TCiBCrhQYtHagUWy` 。在理想情况下，人们一般都希望拥有一个像 “george” 这样的名字。然而，问题是如果一个用户可以创建一个名为 “george” 的账号的话， 那么其他人也可以使用相同的过程来为自己注册一个名为 “george” 的账号，并且冒充他们。唯一的解决方案只能是先申请原则，第一个人可以注册成功，其他人都注册失败 – 一个完全适用于比特币的共识协议的问题。域名币是利用区块链实现名称注册系统的最早的、最成功的系统。
- **Colored coins** – 彩色币的目的是为人们在比特币区块链上创建自己的数字货币，或者，在更重要的一般意义上的货币 – 数字令牌提供服务。依照彩色币协议，人们可以通过为某一特别的比特币UTXO指定颜色，发行新的货币。该协议递归地将其它UTXO定义为与交易输入UTXO相同的颜色。这就允许用户保持只包含某一特定颜色的UTXO，发送这些UTXO就像发送普通的比特币一样，通过回溯全部的区块链判断收到的UTXO颜色。
- **Metacoins** – 元币的理念是在比特币区块链上创建新的协议，利用比特币的交易保存元币的交易，但是采用了不同的状态转换函数APPLY’。因为元币协议不能阻止比特币区块链上的无效的元币交易，所以增加一个规则如果APPLY’(S,TX)返回错误，这一协议将默认APPLY’(S,TX) = S。这为创建任意的、先进的不能在比特币系统中实现的密码学货币协议提供了一个简单的解决方法，而且开发成本非常低，因为挖矿和网络的问题已经由比特币协议处理好了。

因此，总得来说，有2中构建共识的方法：构建一个独立的网络，和在比特币之上建立一个协议。 前一种方法，在名字币这样的应用中相当的成功，但很难实现；每一个独立的实现都需要启动一个独立的区块链，以及构建和测试所有必要的状态转移和网络代码。此外,我们预测去中心化技术的应用会以指数级别的速度增

长，绝大多数的应用会因为太小，而不能保护好它们自己的区块链，另外，我们还注意到存在大量去中心化的应用程序,特别是去中心化的自治组织,它们需要相互作用。

基于比特币的方法，有一个缺陷，它没有继承比特币的简化支付验证（SPV）的功能。SPV 可以让比特币使用区块链的深度来做验证；在某种程度上，一旦一个交易的祖先们已经离的足够远了，就可以放心地说，他们就是合法的状态的一部分。基于区块链的元协议，从另一方面说，不能强制区块链不包含用它们自己的协议验证无效的交易。因此，一个完全安全的 SPV 元协议的实现需要从比特币区块链的末端一直扫描到顶端，这样才能确定某些交易的合法性。

目前，基于比特币的元协议的所有“轻”应用的实现都依赖于一个受信任的服务器来提供数据，这可以说是一个非常不理想的，尤其是当一个加密货币的需要消除信任的时候。

脚本

即便是没有任何扩展，这比特币的协议实际上确实促进了一个弱化版的“智能合约”的概念。UTXO 在比特币中不是只被一个公钥持有, 而是还被在一个基于堆栈的程序语言组成的复杂的脚本表达式所持有着。 在这个范例中，一个交易消耗的UTXO必须提供满足脚本的数据。实际上，这最基本的公钥所有权机制也是通过一个脚本来实现的：这个脚本使用一个椭圆曲线签名作为一个输入，验证拥有这个UTXO的交易和地址，如果验证成功，则返回1，不然则返回0。其他，更加复杂的脚本存在于各种复杂的用例中。 例如，你可以构造一个脚本，它要求从给定的三个私钥中，至少要选其中的2个来做签名验证（“多重签名”），这个对公司账本，储蓄账户等来说非常有用。脚本也能用来对解决计算问题的用户支付报酬。人们甚至可以创建这样的脚本“如果你能够提供你已经发送一定数额的的狗币给我的简化确认支付证明，这一比特币就是你的了”，本质上，比特币系统允许不同的密码学货币进行去中心化的兑换。

然而，在比特币中脚本语言有几个重要的限制:

- 缺乏图灵-完备– 那就是说，虽然比特币脚本语言支持的计算方式很多，但是它不是所有的都支持。在主要类别中缺失循环。 这样做的目的是为了防止对交易的验证出现死循环；理论上，它的脚本是可以克服这个障碍的，因为任何的循环都可以通过 if语句重复多次底层代码来模拟，但是这样的脚本运行效率非常低下。
- 值的盲区 – 一个UTXO脚本没有办法提供资金的颗粒度可控的出金操作。比如, 一个预言合约（oracle contract ）的其中一个强大的用例就是一个套期保值的合约，A和B都把价值1000$的BTC放到合约中，30天后，这个合约把价值1000$的BTC发给了A，剩下的发给了B。 这就需要合约要确定1BTC以美元计值多少钱。然而，因为UTXO是不可分割的，为实现此合约，唯一的方法是非常低效地采用许多有不同面值的UTXO（例如有一种2^k的UTXO，其中K可以最大到30)并使预言合约挑出正确的UTXO发送给A和B。
- 状态缺失 – UTXO 要么被使用了，要么没有被使用；这会使得多阶段的合约和脚本没有机会保持任何其他的内部状态。这使得制作多阶段的期权合约、去中心化的交换协议或两阶段加密承诺协议变得困难(对于安全计算奖金来说是必要的)。这也意味着，UTXO只能用于构建简单的、一次性的合约，而不是更复杂的“有状态”的合约，比如去中心化的组织，并且使得元协议难以实现。

- 区块链盲区 -UTXO对某些区块链数据视而不见，比如随机数和之前的区块的哈希。这严重限制了博彩和其他一些类别的应用，因为它剥夺了一种潜在的有价值的随机数的脚本语言。

因此，我们看到了在加密货币之上构建高级应用程序的三种方法：构建一个新的区块链，在比特币之上使用脚本，和在比特币之上构建一个元协议。 构建一个新的区块链可以无限制扩展功能集，但是这样做非常消耗时间。使用脚本很容易实现和标准化，但在其功能上非常有限，而元协议虽然容易，但在可伸缩性方面却存在缺陷。在以太坊中，我们打算建立一个替代性的框架，它提供了更大的开发和更强大的轻客户属性，同时允许应用程序共享一个经济环境和区块链安全。

以太坊

以太坊的目的是创建一种去中心化应用的协议，提供一套对大量的去中心化应用程序非常有用的新方案，特别强调快速开发，对小的和少数人使用的应用也非常安全（小而使用人少的应用容易被51%攻破），以及不同的应用程序之间能够有效的互动。以太坊通过建立在本质上是抽象的基础层来完成这一工作：一个区块链其内置了图灵完备的编程语言，允许任何人编写智能合约和去中心化的应用程序，在这些应用程序中，他们可以创建任意的属于他们自己的规则、交易格式和状态转换函数。名字币的一个简单版本可以用两行代码来编写完成，而其他协议如货币和信用系统则可以用不到20行的代码来构建。智能合约-包含价值而且只有满足某些条件才能打开的加密箱子-也能在我们的平台上构建，并且因为图灵完备性、价值知晓（value-awareness）、区块链知晓（blockchain-awareness）和多状态所增加的力量，远比特币脚本所能提供的智能合约强大得多。

以太坊账户

在以太坊中，状态是由被称为“账户”的对象组成的，每一个账户都有一个20个字节长度的地址 ，账户之间可以直接的进行价值和信息的转移，一个以太坊的账户包含下面4个字段:

- **随机数**, 一个计数器，用以确保每个交易都只会被处理一次
- 账户当前的**以太币额度**
- 账户的**合约代码**, 如果有的话
- 这个账户的 **存储** (默认空)

“以太币” 是以太坊主要的内部加密燃料，并且被用来支付交易的费用。一般情况下，有2种类型的账户: **外部拥有的账户**,被私钥控制的，和**合约账户**, 被合约代码控制的。外部拥有的账户没有代码，用户可以通过一个外部账户来创建和签名一个交易来送一个消息；合约账户中，每次当这个合约收到一个消息的时候，它的代码就会被激活，允许它读取这个消息，并且写入到内部存储中，然后按照一定顺序发送其他的消息或创建合约等。

消息和交易

名词“交易”在以太坊中是指签名的数据包，这个数据包中存储了从外部账户发送的消息，交易包含以下内容:

- 消息的接收者
- 一个可以识别发送者的签名
- 发送方给接收方的以太币的数量
- 一个可选的数据字段
- 一个 STARTGAS 值,表示执行这个交易允许消耗的最大计算步骤
- 一个 GASPRICE 值,表示发送方的每个计算步骤的费用

前面三个是每一个加密货币都有的标准字段。默认情况下第四个数据字段没有任何功能，但是合约可以访问这里的数据；举个例子，如果一个合约是在一个区块链上提供域名注册服务的，那么它就会想把这数据字段中的数据解析成2个字段，第一个字段是域名，第二个字段是域名对应的IP地址。这个合约会从数据字段中读取这些值，然后适当的存储它们。

这个 STARTGAS 和 GASPRICE 字段 是以太坊的反拒绝式攻击用的，非常重要。为了防止在代码中出现意外或敌对的无限循环或其他计算浪费，每个交易都需要设置一个限制，以限制它的计算总步骤是一个明确的值。这计算的基本单位是“汽油（gas）”； 通常，一个计算成本是一个1 滴汽油，但是一些操作需要消耗更多的汽油，因为它们的计算成本更高。在交易数据中每一个字节都有5滴汽油的费用。这样做的目的是为了让攻击者为他们所消耗的每一种资源，包括计算，带宽和存储支付费用；所以消耗网络资源越多，则交易成本就越大。

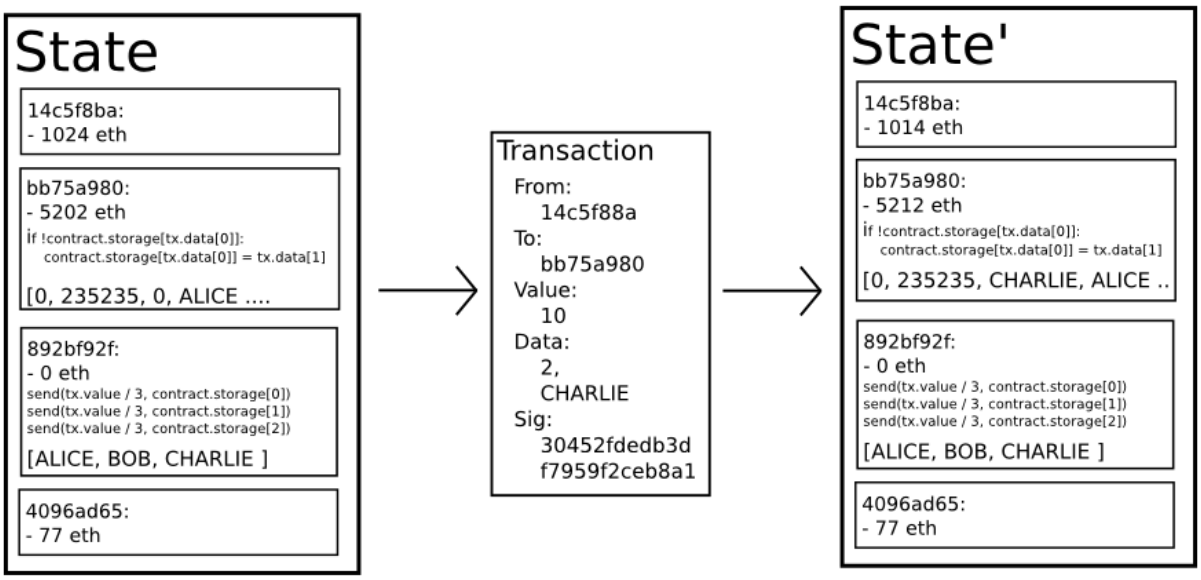
消息

合约有能力向其他合约发生“消息”。消息是虚拟的对象，它从来不会序列号，而且只存在于以太坊的执行环境中。一个消息包含以下内容：

- 消息的发送者 (隐式)
- 消息的接收者
- 与消息一起传送的以太币的数量
- 一个可选的数据字段
- 一个 STARTGAS 值

从本质上说,一个消息就像一个交易,只是它是由一个合约产生的而不是一个外部用户。一个正在执行代码的合约，当执行到 CALL 代码时，会产生并执行一个消息。就像一个交易，一个消息会导致接收方的账户运行它的代码。因此，合约之间可以互相发生关系。

以太坊状态转移函数



以太坊的状态转移函数 APPLY(S,TX) -> S’ 可以被定义成下面的:

1. 检查这个交易是不是合法的 ,签名是不是合法的, 这随机数是不是匹配这个发送者的账户，如果答案是否定的，那返回错误。
2. 用 STARTGAS * GASPRICE 计算交易的费用，并且从签名中确定这个发送者的地址。 从发送者的余额中减去费用，并且增加发送者的随机值。如果余额不够，则返回错误。
3. 初始化 GAS = STARTGAS , 并根据这交易中的字节数拿走一定量的汽油。
4. 把交易的值从发送的账户转移到接收者的账户。如果接收者的账户还不存在，就创建一个。如果这个接收者的账户是一个合约，那么就运行合约的代码直到完成，或者报汽油消耗光的异常。
5. 如果值转移失败了，因为发送者没有足够多的余额，或代码执行消耗光了汽油，恢复除了支付的费用外的所有的状态，并且把这个费用添加到矿工的账户上。
6. 另外,把所有剩下的汽油发退还给发送者，然后把用于支付费用的汽油发送给矿工。

举例，假设合约的代码是这样的:

```
if !self.storage[calldataload(0)]:
    self.storage[calldataload(0)] = calldataload(32)
```

注意，真实的合约代码是用底层的EVM代码编写的；这个列子是用一个叫Serpent 的高级语言写的。假设这个合约的存储开始是空的，并且发送了一个交易，其中包含10个以太币，2000个汽油，汽油价格是0.001比

特币，和64字节的数据，其中0-31字节代表数字2,32-63字节代表字符串 CHARLIE 。在这个案例中，这状态转移函数的处理如下：

1. 检查者交易是否有效并且格式完好。
2. 检查者交易的发送者是否至少有 2000 * 0.001 = 2 以太币。如果有，则从发送者的账户中减去2以太币。
3. 初始化 汽油（gas）= 2000;假设这个交易是170个字节长度并且每个字节的费用是5，那么减去850，汽油还剩1150。
4. 从发送者的账户减去10 个以太币，并且添加到合约的账户中。
5. 运行合约的代码. 在这里例子中:检查合约的存储的第2个索引是否已经被使用，注意到它没有，然后就把这存储的第二个索引的值设置为 CHARLIE . 假设这个操作消耗了187个汽油，那么剩下的汽油总量是 1150 – 187 = 963
6. 把 963 * 0.001 = 0.963 以太币加到发送者的账户，然后反正结果状态。

如果交易的接收端没有合约，那么这总的交易费用就简单的等于汽油的价格乘以这个交易的字节长度，与交易一起发送的数据字段的数据将无关重要。

注意，在恢复这个方面，消息和交易的处理方式是相同的： 如果一个消息执行消耗光了汽油，那么这消息的执行和其他被触发的执行都会被恢复，但是父类的执行不需要恢复。

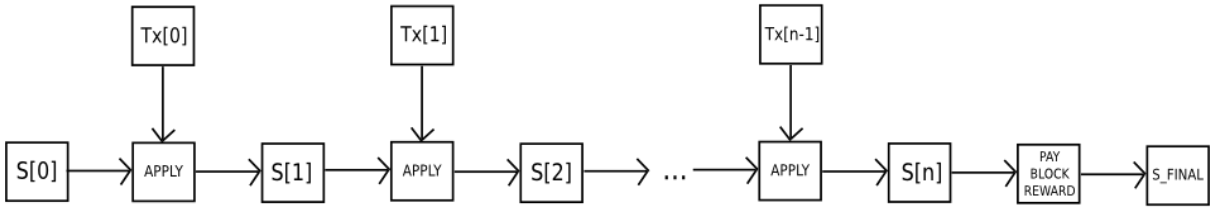
代码执行

以太坊的合约代码是用底层的基于堆栈的字节码语言来编写的，被称为“以太坊虚拟机代码” or “EVM 代码”。这代码有一系列的字节组成，其中每一个字节都标识一个操作。在一般情况下,代码的执行是一个无限循环，直到代码运行结束或遇到错误，或检测到 STOP 或 RETURN 指令。这些操作有三种类型的空间可用于存储数据：

- **堆栈**, 一种后进先出的容器
- **内存**, 一种无线扩展的字节数组
- 合约的持久化 **存储**, 一种键值对的方式. 不像堆栈和内存,计算结束后将会被重置, 存储将长期保存。

代码还可以访问值（以太币），发送者，传入的消息的数据，以及区块的头信息，并且也可以返回一组字节数组当做输出。

区块链和挖矿



这以太坊的区块链和比特币的区块链有很多相似的地方，也有很多不同的地方。这个以太坊和比特币在区块链体系中最重要的不同点是：以太坊的区块同时包含了交易列表和最近区块的状态。除此之外，2个其他的值，区块的编号和难度值也存在在区块中。以太坊中最基本的区块验证算法如下：

1. 检查上一个区块是否存在和其有效性。
2. 检测这区块的时间戳，是不是比上一个区块的大，并且小于15分钟
3. 检查这区块编号，难度值，交易根（transaction root），叔根（uncle root）和汽油限制是否有效
4. 检查这区块的工作证明是否有效
5. 把 S[0] 设置成上一个区块的末端的状态
6. 让 TX 成为这区块的交易列表，如果有 n 个交易。则做for循环 For i in 0...n-1，设置 S[i+1] = APPLY(S[i],TX[i]).如果任何一个应用发生错误，或这区块中汽油的总的消耗达到了 GASLIMIT，则返回一个错误.
7. 让 S_FINAL 等于 S[n]，但是把支付给矿工的奖励添加到这区块里。
8. 检查这个状态 S_FINAL 的默克尔树树根是不是和区块头信息中所提供的状态根是一样的。如果是，则区块有效，不然则无效。

乍看上去，这种方法似乎效率很低，因为它需要将整个状态存储在每个块中，但在现实中，效率应该与比特币相当。原因在于，状态存储在树结构中，并且每个块后，只需要修改树的一小部分。此外，由于所有的状态信息都是最后一个区块的一部分，所以不需要存储整个区块链的历史——这一策略，如果它可以应用于比特币，那么它的磁盘空间将节省5-20倍。

应用

一般来说，在以太坊上有三种类型的应用。第一种是金融应用，这包括 子货币，金融衍生品，套期保值合约，和一些雇佣合同等。第二类是半金融应用，这里有钱的存在但也有很重的非金钱的方面；最后，还有在线投票和去中心化治理这样的完全的非金融应用。

2017年6月3日 / 以太坊-区块链 / 留下评论

区块链以太坊系列课程

区块链以太坊系列课程

区块链平台以太坊基础入门:
http://edu.51cto.com/course/course_id-8358.html
区块链平台以太坊Solidity智能合约编程详解系列视频课程:
http://edu.51cto.com/course/course_id-8977.html

2017年5月25日 / Solidity、以太坊-区块链 / 留下评论

以太坊 长期节点

EthFans 长期节点 » 论坛 » EthFans | 以太坊爱好者

为了帮助国内的同学们以最快的速度同步以太坊的区块链，EthFans社区建立了一个长期节点，地址为：

```
enode://91922b12115c067005c574844c6bbdb114eb262f90b6355cec89e13b483c3e4669c6d63ec66b6e3ca7a3a462d28edb3c659e9fa05ed4c7234524e582a8816743@120.27.164.92:13333
```

设置为默认连接节点

如果你是geth用户，可以做如下设置，默认连接这个节点：

1. 找到你的data目录，例如linux上默认是 `~/.ethereum`
2. 在data目录里面新建一个 `static-nodes.json` 文件，输入以下内容并保存：

```
["enode://91922b12115c067005c574844c6bbdb114eb262f90b6355cec89e13b483c3e4669c6d63ec66b6e3ca7a3a462d28edb3c659e9fa05ed4c7234524e582a8816743@120.27.164.92:13333"]
```

3. 如常启动geth即可

检查是否连接成功

反馈
建议

1. 通过 geth console 进入控制台，或者通过 geth --ipcpath ~/.ethereum/geth.ipc attach 开控制台挂上当前进程
2. 控制台中输入： admin.peers.forEach(function(p) {console.log(p.network.remoteAddress);})
3. 如果打印出的地址里面包括120.27.164.92, 说明已经连上

2017年4月30日 / 以太坊-区块链 / 留下评论

在区块链平台以太坊上建设民主组织

到目前为止，我们列出的所有合约都是由人类持有并执行的。但是在以太坊的生态系统中并不会歧视机器人。

在这一节中我们要建立一个分散的、民主的组织，存在于区块链上。该组织有一个中央管理者，决定谁是成员和投票规则。

这种特殊的民主制度的运作方式是，它有一个所有者，如管理员，首席执行官或总统的作品。所有者可以向组织添加或删除投票成员。任何成员可提出建议，以以太坊交易的形式来发送以太币或执行一些合约，其他成员可以在支持或反对之间对这项提议进行投票。一旦预定的时间和一定数量的成员投票，建议可以执行：合约会计算选票，如果有足够的选票，它将执行给定的交易。

```
 pragma solidity ^0.4.2;
contract owned {
    address public owner;

    function owned() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        if (msg.sender != owner) throw;
        _;
    }

    function transferOwnership(address newOwner) onlyOwner {
        owner = newOwner;
    }
}

contract tokenRecipient {
    event receivedEther(address sender, uint amount);
    event receivedTokens(address _from, uint256 _value, address _token, bytes
```

```
_extraData);

    function receiveApproval(address _from, uint256 _value, address _token, bytes
_extraData){
        Token t = Token(_token);
        if (!t.transferFrom(_from, this, _value)) throw;
        receivedTokens(_from, _value, _token, _extraData);
    }

    function () payable {
        receivedEther(msg.sender, msg.value);
    }
}

contract Token {
    function transferFrom(address _from, address _to, uint256 _value) returns (bool
success);
}

contract Congress is owned, tokenRecipient {

    /* Contract Variables and events */
    uint public minimumQuorum;
    uint public debatingPeriodInMinutes;
    int public majorityMargin;
    Proposal[] public proposals;
    uint public numProposals;
    mapping (address => uint) public memberId;
    Member[] public members;

    event ProposalAdded(uint proposalID, address recipient, uint amount, string
description);
    event Voted(uint proposalID, bool position, address voter, string justification);
    event ProposalTallied(uint proposalID, int result, uint quorum, bool active);
    event MembershipChanged(address member, bool isMember);
    event ChangeOfRules(uint minimumQuorum, uint debatingPeriodInMinutes, int
majorityMargin);

    struct Proposal {
        address recipient;
        uint amount;
        string description;
        uint votingDeadline;
        bool executed;
        bool proposalPassed;
        uint numberOfVotes;
        int currentResult;
        bytes32 proposalHash;
        Vote[] votes;
        mapping (address => bool) voted;
    }

    struct Member {
```

```
        address member;
        string name;
        uint memberSince;
    }

    struct Vote {
        bool inSupport;
        address voter;
        string justification;
    }

    /* modifier that allows only shareholders to vote and create new proposals */
    modifier onlyMembers {
        if (memberId[msg.sender] == 0)
        throw;
        _;
    }

    /* First time setup */
    function Congress(
        uint minimumQuorumForProposals,
        uint minutesForDebate,
        int marginOfVotesForMajority, address congressLeader
    ) payable {
        changeVotingRules(minimumQuorumForProposals, minutesForDebate,
marginOfVotesForMajority);
        if (congressLeader != 0) owner = congressLeader;
        // It' s necessary to add an empty first member
        addMember(0, '');
        // and let's add the founder, to save a step later
        addMember(owner, 'founder');
    }

    /*make member*/
    function addMember(address targetMember, string memberName) onlyOwner {
        uint id;
        if (memberId[targetMember] == 0) {
           memberId[targetMember] = members.length;
           id = members.length++;
           members[id] = Member({member: targetMember, memberSince: now, name:
memberName});
        } else {
            id = memberId[targetMember];
            Member m = members[id];
        }

        MembershipChanged(targetMember, true);
    }

    function removeMember(address targetMember) onlyOwner {
        if (memberId[targetMember] == 0) throw;

        for (uint i = memberId[targetMember]; i<members.length-1; i++){
```

```
            members[i] = members[i+1];
        }
        delete members[members.length-1];
        members.length--;
    }

    /*change rules*/
    function changeVotingRules(
        uint minimumQuorumForProposals,
        uint minutesForDebate,
        int marginOfVotesForMajority
    ) onlyOwner {
        minimumQuorum = minimumQuorumForProposals;
        debatingPeriodInMinutes = minutesForDebate;
        majorityMargin = marginOfVotesForMajority;

        ChangeOfRules(minimumQuorum, debatingPeriodInMinutes, majorityMargin);
    }

    /* Function to create a new proposal */
    function newProposal(
        address beneficiary,
        uint etherAmount,
        string JobDescription,
        bytes transactionBytecode
    )
        onlyMembers
        returns (uint proposalID)
    {
        proposalID = proposals.length++;
        Proposal p = proposals[proposalID];
        p.recipient = beneficiary;
        p.amount = etherAmount;
        p.description = JobDescription;
        p.proposalHash = sha3(beneficiary, etherAmount, transactionBytecode);
        p.votingDeadline = now + debatingPeriodInMinutes * 1 minutes;
        p.executed = false;
        p.proposalPassed = false;
        p.numberOfVotes = 0;
        ProposalAdded(proposalID, beneficiary, etherAmount, JobDescription);
        numProposals = proposalID+1;

        return proposalID;
    }

    /* function to check if a proposal code matches */
    function checkProposalCode(
        uint proposalNumber,
        address beneficiary,
        uint etherAmount,
        bytes transactionBytecode
    )
        constant
```

```
        returns (bool codeChecksOut)
    {
        Proposal p = proposals[proposalNumber];
        return p.proposalHash == sha3(beneficiary, etherAmount, transactionBytecode);
    }

    function vote(
        uint proposalNumber,
        bool supportsProposal,
        string justificationText
    )
        onlyMembers
        returns (uint voteID)
    {
        Proposal p = proposals[proposalNumber];         // Get the proposal
        if (p.voted[msg.sender] == true) throw;         // If has already voted, cancel
        p.voted[msg.sender] = true;                     // Set this voter as having
voted
        p.numberOfVotes++;                              // Increase the number of votes
        if (supportsProposal) {                         // If they support the proposal
            p.currentResult++;                          // Increase score
        } else {                                        // If they don't
            p.currentResult--;                          // Decrease the score
        }
        // Create a log of this event
        Voted(proposalNumber,  supportsProposal, msg.sender, justificationText);
        return p.numberOfVotes;
    }

    function executeProposal(uint proposalNumber, bytes transactionBytecode) {
        Proposal p = proposals[proposalNumber];
        /* Check if the proposal can be executed:
           - Has the voting deadline arrived?
           - Has it been already executed or is it being executed?
           - Does the transaction code match the proposal?
           - Has a minimum quorum?
        */

        if (now < p.votingDeadline
            || p.executed
            || p.proposalHash != sha3(p.recipient, p.amount, transactionBytecode)
            || p.numberOfVotes < minimumQuorum)
            throw;

        /* execute result */
        /* If difference between support and opposition is larger than margin */
        if (p.currentResult > majorityMargin) {
            // Avoid recursive calling

            p.executed = true;
            if (!p.recipient.call.value(p.amount * 1 ether)(transactionBytecode)) {
                throw;
            }
```

```
            p.proposalPassed = true;
        } else {
            p.proposalPassed = false;
        }
        // Fire Events
        ProposalTallied(proposalNumber, p.currentResult, p.numberOfVotes,
p.proposalPassed);
    }
}
```

如何发布

打开钱包，切换到**Contracts** 页面，然后点击 **deploy contract** ，把上面的代码黏贴进**solidity code** 框里，在右边选择**Congress** 合约，你应该可以看到如下的选项：

- **Minimum quorum for proposals** ：一项决议被投票通过所需的最小票数
- **Minutes for debate** ：决议被执行前需要的最小的辩论时间（单位：分钟）
- **Margin of votes for majority** ：设置0，代表简单多数，1：需要投票结果绝对一致

与合约互动

在**“read from contract”**处，你可以看到该合约的所有你可以免费执行的函数，它们只能读取区块链的信息。

在**“Write to contract”** 处，有一系列可以做计算的方法，并且把计算结果保存到区块链上。因此，这些操作会消耗gas.

在与合约交互之前，需要先添加几个投票的成员。在 **“select function”** 的下拉选项中，选择 **“Change Membership”**，

添加一个简单的决议：发送以太币

现在让我们添加第一个决议到我们的合约里，在方法的下拉列表里，选择 **New proposal**.

“beneficiary” 是你想发送以太币的目的地，”etherAmount” 是想发送的以太币的数量，只能是整数。“Job description” 你可以填写一些原因备注等。“Transaction bytecode” 字段留空即可。然后点击执行！

投票也是非常简单！在右侧方法选择下拉框中选择 “vote”，在第一个输入框中输入决议的编号，在“Supports proposal ”中，如果你投赞成票，则点击“Yes”，如果投反对票则留空即可。然后点击”**execute**” 执行你的投票。

当投票时间截止，你就可以执行**“executeProposal”** 函数了。如果决议只是发送以太币，那 “**transactionBytecode**” 留白即可。最后点击”execute” 按钮。

如果出现一个警告 “estimated fee consumption” ,那意味着有一些原因会导致这函数不会被执行，会被突然终止.在本合约中，当在决议讨论时间截止前，会出现这个警告。

添加一个复杂的决议：使用自己的令牌

你可以使用这个民主组织合约执行以太坊上的任何交易。

股东协会

在上一个章节中，我们创建一个合约，它的成员只能是收到主席邀请的人。但是有几个缺点：如果有人想要改变他的主账号呢？如果有些的投票权重比较高呢？

我们可以修改一下这合约的代码，使其绑定一个特殊的令牌，这令牌代表了这合约的股份数量。首先创建一个令牌， **initial supply** 设置成100，**decimals** 设置成0，**symbol**设置成 %。。

下面是股东代码：

```
 pragma solidity ^0.4.2;
/* The token is used as a voting shares */
contract token { mapping (address => uint256) public balanceOf;  }


/* define 'owned' */
contract owned {
    address public owner;

    function owned() {
        owner = msg.sender;
    }

    modifier onlyOwner {
        if (msg.sender != owner) throw;
        _;
    }

    function transferOwnership(address newOwner) onlyOwner {
```

```
        owner = newOwner;
    }
}

contract tokenRecipient {
    event receivedEther(address sender, uint amount);
    event receivedTokens(address _from, uint256 _value, address _token, bytes
_extraData);

    function receiveApproval(address _from, uint256 _value, address _token, bytes
_extraData){
        Token t = Token(_token);
        if (!t.transferFrom(_from, this, _value)) throw;
        receivedTokens(_from, _value, _token, _extraData);
    }

    function () payable {
        receivedEther(msg.sender, msg.value);
    }
}

contract Token {
    function transferFrom(address _from, address _to, uint256 _value) returns (bool
success);
}

/* The democracy contract itself */
contract Association is owned, tokenRecipient {

    /* Contract Variables and events */
    uint public minimumQuorum;
    uint public debatingPeriodInMinutes;
    Proposal[] public proposals;
    uint public numProposals;
    token public sharesTokenAddress;

    event ProposalAdded(uint proposalID, address recipient, uint amount, string
description);
    event Voted(uint proposalID, bool position, address voter);
    event ProposalTallied(uint proposalID, int result, uint quorum, bool active);
    event ChangeOfRules(uint minimumQuorum, uint debatingPeriodInMinutes, address
sharesTokenAddress);

    struct Proposal {
        address recipient;
        uint amount;
        string description;
        uint votingDeadline;
        bool executed;
        bool proposalPassed;
        uint numberOfVotes;
        bytes32 proposalHash;
        Vote[] votes;
```

```
        mapping (address => bool) voted;
    }

    struct Vote {
        bool inSupport;
        address voter;
    }

    /* modifier that allows only shareholders to vote and create new proposals */
    modifier onlyShareholders {
        if (sharesTokenAddress.balanceOf(msg.sender) == 0) throw;
        _;
    }

    /* First time setup */
    function Association(token sharesAddress, uint minimumSharesToPassAVote, uint
minutesForDebate) payable {
        changeVotingRules(sharesAddress, minimumSharesToPassAVote, minutesForDebate);
    }

    /*change rules*/
    function changeVotingRules(token sharesAddress, uint minimumSharesToPassAVote, uint
minutesForDebate) onlyOwner {
        sharesTokenAddress = token(sharesAddress);
        if (minimumSharesToPassAVote == 0 ) minimumSharesToPassAVote = 1;
        minimumQuorum = minimumSharesToPassAVote;
        debatingPeriodInMinutes = minutesForDebate;
        ChangeOfRules(minimumQuorum, debatingPeriodInMinutes, sharesTokenAddress);
    }

    /* Function to create a new proposal */
    function newProposal(
        address beneficiary,
        uint etherAmount,
        string JobDescription,
        bytes transactionBytecode
    )
        onlyShareholders
        returns (uint proposalID)
    {
        proposalID = proposals.length++;
        Proposal p = proposals[proposalID];
        p.recipient = beneficiary;
        p.amount = etherAmount;
        p.description = JobDescription;
        p.proposalHash = sha3(beneficiary, etherAmount, transactionBytecode);
        p.votingDeadline = now + debatingPeriodInMinutes * 1 minutes;
        p.executed = false;
        p.proposalPassed = false;
        p.numberOfVotes = 0;
        ProposalAdded(proposalID, beneficiary, etherAmount, JobDescription);
        numProposals = proposalID+1;
```

```
        return proposalID;
    }

    /* function to check if a proposal code matches */
    function checkProposalCode(
        uint proposalNumber,
        address beneficiary,
        uint etherAmount,
        bytes transactionBytecode
    )
        constant
        returns (bool codeChecksOut)
    {
        Proposal p = proposals[proposalNumber];
        return p.proposalHash == sha3(beneficiary, etherAmount, transactionBytecode);
    }

    /* */
    function vote(uint proposalNumber, bool supportsProposal)
        onlyShareholders
        returns (uint voteID)
    {
        Proposal p = proposals[proposalNumber];
        if (p.voted[msg.sender] == true) throw;

        voteID = p.votes.length++;
        p.votes[voteID] = Vote({inSupport: supportsProposal, voter: msg.sender});
        p.voted[msg.sender] = true;
        p.numberOfVotes = voteID +1;
        Voted(proposalNumber,  supportsProposal, msg.sender);
        return voteID;
    }

    function executeProposal(uint proposalNumber, bytes transactionBytecode) {
        Proposal p = proposals[proposalNumber];
        /* Check if the proposal can be executed */
        if (now < p.votingDeadline  /* has the voting deadline arrived? */
            ||  p.executed        /* has it been already executed? */
            ||  p.proposalHash != sha3(p.recipient, p.amount, transactionBytecode)) /*
Does the transaction code match the proposal? */
            throw;

        /* tally the votes */
        uint quorum = 0;
        uint yea = 0;
        uint nay = 0;

        for (uint i = 0; i <  p.votes.length; ++i) {
            Vote v = p.votes[i];
            uint voteWeight = sharesTokenAddress.balanceOf(v.voter);
            quorum += voteWeight;
            if (v.inSupport) {
                yea += voteWeight;
```

```
            } else {
                nay += voteWeight;
            }
        }

        /* execute result */
        if (quorum <= minimumQuorum) {
            /* Not enough significant voters */
            throw;
        } else if (yea > nay ) {
            /* has quorum and was approved */
            p.executed = true;
            if (!p.recipient.call.value(p.amount * 1 ether)(transactionBytecode)) {
                throw;
            }
            p.proposalPassed = true;
        } else {
            p.proposalPassed = false;
        }
        // Fire Events
        ProposalTallied(proposalNumber, result, quorum, p.proposalPassed);
    }
}
```

部署和使用

部署方式几乎和上一章节的一样。

https://ethereum.org/dao#the-shareholder-association

2017年3月23日 / 以太坊-区块链 / 留下评论

以太币 之 挖矿 2017-02-06更新

介绍

加密货币就像黄金一样，黄金或贵金属都是稀缺的。唯一一能增加他们总量的方法就是挖矿。这在以太坊中也同样适合，以太币唯一的发行模式就是挖矿。与黄金不大一样的是，挖矿也是为了确保以太坊全网的安全。

挖以太币=保护网络=验证计算

挖矿

以太坊，和所有区块链技术一样，使用一种安全的激励驱动模式。一致性是以选择具有最高总难度的那个区块为基础的。矿工创造区块，其他人检测区块的有效性。在其他的 合法标准里，只是检查一个区块是否包含一个给定难度的工作量证明（PoW）。但是随着以太币的发展，这个有可能被替换成权益证明的模式（PoS:proof of stake）。

挖矿奖励

成功的挖到区块的矿工将获得如下奖励：

1. 一个固定的奖励，5个以太币
2. 在区块内消耗掉的gas的成本,相当于多少以太币取决于当前gas的价格。
3. 如果区块中含有uncle,则每个uncle可以获得1/32的奖励。

挖矿人所消耗的gas由每个交易的发起者支付。

如果想要挖矿，那你需要一个**完全同步好数据**的以太坊客户端，和一个以太坊账号。这账号是用接收挖矿奖励的，通常被称为*coinbase 或etherbase.*

Cpu挖矿

您可以使用您的计算机的中央处理器(CPU)挖以太币。不过，我们推荐使用GPU挖矿,因为GPU的效率是CPU的2个数量级！您可以使用CPU挖私有链中的以太币。

使用GETH

geth客户端默认不会开启挖矿。要开启CPU挖矿模式，你要设置 –mine 命令行参数。-minerthreads参数用于设置并行挖矿的线程数量（默认可设置为cpu的核数）

geth –mine –minerthreads=4

你可以在JavaScript控制台开启或停止挖矿操作。miner.start 接受一个参数，表示挖矿的线程数。

```
> miner.start(8)
true
> miner.stop()
true
```

为了获取到以太币，你必须设置etherbase(或coinbase)的地址。etherbase默认是你主账号的地址。如果你没有一个etherbase地址,geth –mine是不会启动的。

你可以在命令行里设置你的 etherbase:

```
geth --etherbase 1 --mine  2>> geth.log // 1 是索引: 你创建的第二个账号
或
geth --etherbase '0xa4d8e9cae4d04b093aac82e6cd355b6b963fb7ff' --mine 2>> geth.log
```

你也可以在控制台上重置etherbase:

```
miner.setEtherbase(eth.accounts[2])
```

注意,你的etherbase地址并不需要一定是本机上的。

你可以添加一些额外信息（32个字节）到被你挖到的区块上。通常情况下是一个unicode 字符串，所以在这里，你可以把你的名字打上以满足你的小小的虚荣心。

```
miner.setExtra("lalalala")
...
debug.printBlock(131805)
BLOCK(be465b020fdbedc4063756f0912b5a89bbb4735bd1d1df84363e05ade0195cb1): Size: 531.00 B
TD: 643485290485 {
NoNonce: ee48752c3a0bfe3d85339451a5f3f411c21c8170353e450985e1faab0a9ac4cc
Header:
[
...
        Coinbase:           a4d8e9cae4d04b093aac82e6cd355b6b963fb7ff
        Number:             131805
        Extra:              lalalala
...
}
```

你可以用命令miner.hashrate查看你的hashrate（hash运算速率）值，单位H/s。

```
> miner.hashrate
712000
```

你成功挖到一些区块后，你可以查看etherbase帐户的余额，假设你的etherbase 是一个本地账号：

```
> eth.getBalance(eth.coinbase).toNumber();
'34698870000000'
```

为了能让挖到的以太币能够用于你以后发送交易所需支付的gas,你需要把这个账号解锁。

```
> personal.unlockAccount(eth.coinbase)
Password
true
```

反馈
建议

在JavaScript控制台输入下面的代码片段，你可以检查哪块是由一个特定的矿工开采

```
function minedBlocks(lastn, addr) {

addrs = [];

if (!addr) {

addr = eth.coinbase

}

limit = eth.blockNumber – lastn

for (i = eth.blockNumber; i >= limit; i–) {

if (eth.getBlock(i).miner == addr) {

addrs.push(i)

}

}

return addrs

}

//扫描的最新的1000区块并返回你挖到的区块的区块编号（blocknumbers）

minedBlocks(1000, eth.coinbase);

//[352708, 352655, 352559]
```

GPU挖矿

硬件

为了将DAG适当的导入到内存中，每个GPU都需要1-2GB的内存。如果你得到Error GPU mining. GPU memory fragmentation?则说明你的内存不足。GPU挖矿是用OpenCL实现的， AMD的GPU会比同类型的NVIDIA GPU快很多。ASIC和FPGA相对效率低下，因此不推荐使用。获取openCL驱动，根据你的芯片和平台试一下：

- AMD SDK openCL
- NVIDIA CUDA openCL

UBUNTU 配置

您需要Ubuntu 14.04或15.04和fglrx图形驱动程序。也可以使用NVidia驱动程序和其他平台。只要能正常

如果你使用15.04，你可以去 “Software and Updates > Additional Drivers”将其设置为 “Using video drivers for the AMD graphics accelerator from fglrx”。

如果你使用14.04，则去 “Software and Updates > Additional Drivers”，将其设置为 “Using video drivers for the AMD graphics accelerator from fglrx”。

MAC 配置

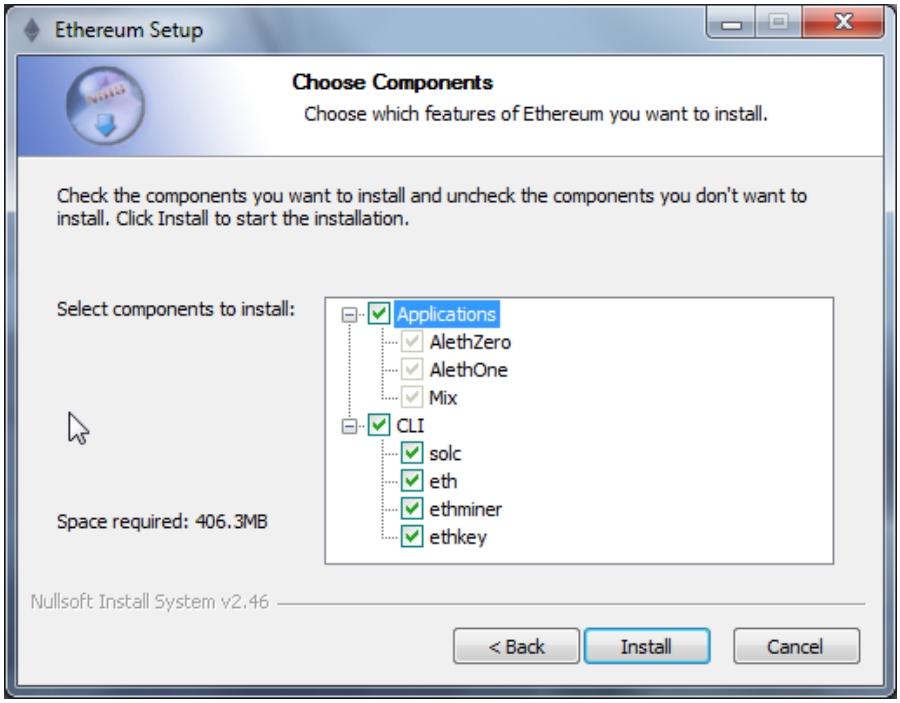
```
wget
http://developer.download.nvidia.com/compute/cuda/7_0/Prod/local_installers/cuda_7.0.29_
mac.pkg
sudo installer -pkg ~/Desktop/cuda_7.0.29_mac.pkg -target /
brew update
brew tap ethereum/ethereum
brew reinstall cpp-ethereum --with-gpu-mining --devel --headless --build-from-source
```

检查你的冷却状态:

aticonfig –adapter=0 –od-gettemperature

WINDOWS 配置

下载最新的Eth++安装程序，在“Choose Components”页面勾选ethminer选项：



在GETH和ETHMINER

```
geth account new // 如果你还没有账号，设置一个ethereum帐户
geth --rpc --rpccorsdomain localhost 2>> geth.log &
ethminer -G  // -G 显卡, -M 执行基准问题测试
tail -f geth.log
```

ethminer和geth的通讯在8545端口上（geth中默认的RPC端口），你可以在geth命令中加入–rpcport 这个参数来改变这个端口值。Ethminer 会自动找到geth的端口。注意你需要使用–rpccorsdomain localhost设置跨域头信息 。你也可以使用-F http://127.0.0.1:3301 来设置 ethminer 的端口。如果你想要在你的电脑上设置多个矿工，那你必须要为每个矿工设置不同的端口。如果你是在测试网络或私有网络上挖矿，那推荐你使用cpu挖矿即可。

如果默认的ethminer不工作，那可以试图指定OpenCL驱动：–opencl-device X ,X是{0,1,2,...},当使用了-M (基准问题测试) 运行ethminer ，那么你应该可以看到下面类似信息：

```
Benchmarking on platform: { "platform": "NVIDIA CUDA", "device": "GeForce GTX 750 Ti",
"version": "OpenCL 1.1 CUDA" }


Benchmarking on platform: { "platform": "Apple", "device": "Intel(R) Xeon(R) CPU E5-1620
v2 @ 3.70GHz", "version": "OpenCL 1.2 " }
```

geth开启debug：

```
geth  --rpccorsdomain "localhost" --verbosity 6 2>> geth.log
```

调试矿工

```
make -DCMAKE_BUILD_TYPE=Debug -DETHASHCL=1 -DGUI=0
gdb --args ethminer -G -M
```

GPU挖矿模式中hashrate 不可用。使用ethminer, miner.hashrate来查看hashrate总是返回0。

在ethminer 中使用 eth

在单个GPU上挖矿

使用 eth 在单一个GPU上挖矿,唯一要做的事情就是在运行eth的时候，使用下面的参数：

```
eth -v 1 -a 0xcadb3223d4eebcaa7b40ec5722967ced01cfc8f2 --client-name "OPTIONALNAMEHERE"
-x 50 -m on -G
```

- -v 1 设置日志信息为1，不要为太多的信息打扰
- -a 你的钱包地址 设置coinbase的地址，挖矿的奖励给这设置的账号。这个参数非常重要，确保不要填写错误，不然将接收不到以太币。
- –client-name “OPTIONAL” 设置一个可选的客户端名称，用于在网络上的识别
- -x 50 要求一个比较多的同伴数量。用于在一开始寻找同伴。
- -m on 开启真正执行
- -G 设置GPU挖矿

当客户端在运行的时候，你可以用 geth attach 命令或[ethconsole](https://github.com/ethereum/ethereum-console) 与之交互。

在一个有多个GPU系统上挖矿

使用多个GPU和eth挖矿与使用多个GPU和geth挖矿是非常相似的。确保你的eth节点正确配置了coinbase地址：

```
eth -v 1 -a 0xcadb3223d4eebcaa7b40ec5722967ced01cfc8f2 --client-name "OPTIONALNAMEHERE"
-x 50 -j
```

这里增加了一个-j参数，用来启动JSON-RPC服务，eth用它和ethminer实例通信。另外，我们删除了挖矿相关的参数，ethminer 将为我们做挖矿相关的事情。每一个GPU执行一个不同的ethminer 实例：

```
ethminer --no-precompute -G --opencl-device X
```

其中X是openCL驱动的索引，用{0,1,2,...}来替代X。你可以执行ethminer –list-devices命令来获取openCL驱动列表。

下面是一个输出示例：

```
[0] GeForce GTX 770
    CL_DEVICE_TYPE: GPU
    CL_DEVICE_GLOBAL_MEM_SIZE: 4286345216
    CL_DEVICE_MAX_MEM_ALLOC_SIZE: 1071586304
    CL_DEVICE_MAX_WORK_GROUP_SIZE: 1024
```

基准测试

使用ethminer 加参数 -G -M 可以对一个GPU进行基准测试。

如果你有多个GPU，然后你想对每个GPU单独测试，那可以加上 –opencl-device X 选项，其中X是GPU的索引号。

```
ethminer -G -M --opencl-device X
```

使用ethminer ——list-devices 列出的数字代替X。

下载geth windows二进制文件，开始挖矿，下载地址：
https://build.ethereum.org/builds/Windows%20Go%20master%20branch/

解压，然后再命令行里定位到解压目录，执行 geth –rpc

执行命令后，以太坊区块链就会下载数据，请设置好防火墙，以防止同步被防火墙阻止。

- 下载download and install ethminer，C++的挖矿软件，并安装
- 打开另一个命令窗口，定位到你的安装目录
- 确保geth已经同步完区块链。如果已经不再同步了，你可以在命令行里输入 ethminer -G 开始挖矿。

在这个过程中可能会出现一些问题，你可以按Ctrl+C中止挖矿。如果错误显示“Insufficient Memory”，说明你的计算机内存不足。

矿池挖矿

在矿池里挖矿，可以知道自己的预期收入是多少。但矿池一般会收取0-5%的手续费。矿池把挖到的以太币根据各矿工捐献的挖矿比例来进行分配。

需要注意的是，奖励的分配权在矿池手里，有些黑矿池，小的第三方矿池可能会偷走你的收益！

2017年2月6日 / 以太坊-区块链 / 留下评论

以太坊网络 2017-02-05

以太坊网络数据统计

EthStats.net 是一个展示以太网网络的实时的统计数据的仪表盘。这仪表盘上包含了许多重要的数据，如当前区块，交易，gas价格等。这页面上展示的节点只是实际网络中的节点的一部分。任何人都可以添加他们自己的节点到这个仪表盘上。

EtherNodes.com 这网站统计了当前和历史上的有关节点的数量，包括了主网和测试网络。

公有链，私有链，联盟链

现在大多数的以太坊项目都运行在以太坊公有链上，以太坊公有链可以为这些项目提供大量的用户，网络节点，货币和市场。但是，也有一些项目更加适合运行在私有链或团体链（一群值得信赖的伙伴）上。比如，一些垂直领域里面的公司，如银行正在试图使用以太坊建立他们自己的私有区块链平台。

以下是它们在权限方便的区别：

公有链：公有链是指一个世界上任何人都可以参与的区块链。用户可以查看，可以发送交易，也可以参与保持数据一致性的运算等。

私有链：完全的私有链是指写权限被一个人或一个组织控制的链。私有链的读权限是可以公开的或者是有限度的在一定范围公开的。比如数据库的管理，公司内部的管理等。

联盟链：联盟链是指，数据一致性的运算被预先设定好的几个节点共同控制的链。比如，有15家银行组成了一个财团链，在这个链上的每一个节点的每一次的操作都需要10个节点的共同签名才能被验证。这区快链上的读权限可能是公开的，也有可能是部分公开的。

如何链接

geth会不断尝试连接网络上的其他的节点，直到它找到同伴为止。geth使用一个发现协议（discovery protocol）来寻找自己的同伴。节点会互相发送广播来找出网络中的其他节点。

查看链接和节点ID

在交互控制台里可以查看有多少个伙伴链接着自己，net模块有2个属性可以显示同伴的数量和你是否是一个正在监听的节点。

```
> net.listening
true

> net.peerCount
4
```

想要得到同伴的更多的信息，如IP地址和端口号,支持的协议的话，可以使用admin对象的peers()方法。

```
> admin.peers
[{
      ID:
```

```
'a4de274d3a159e10c2c9a68c326511236381b84c9ec52e72ad732eb0b2b1a2277938f78593cdbe734e6002b
f23114d434a085d260514ab336d4acdc312db671b',
      Name: 'Geth/v0.9.14/linux/go1.4.2',
      Caps: 'eth/60',
      RemoteAddress: '5.9.150.40:30301',
      LocalAddress: '192.168.0.28:39219'
 }, {
      ID:
'a979fb575495b8d6db44f750317d0f4622bf4c2aa3365d6af7c284339968eef29b69ad0dce72a4d8db5ebb4
968de0e3bec910127f134779fbcb0cb6d3331163c',
      Name: 'Geth/v0.9.15/linux/go1.4.2',
      Caps: 'eth/60',
      RemoteAddress: '52.16.188.185:30303',
      LocalAddress: '192.168.0.28:50995'
 }, {
      ID:
'f6ba1f1d9241d48138136ccf5baa6c2c8b008435a1c2bd009ca52fb8edbbc991eba36376beaee9d45f16d5d
cbf2ed0bc23006c505d57ffcf70921bd94aa7a172',
      Name: 'pyethapp_dd52/v0.9.13/linux2/py2.7.9',
      Caps: 'eth/60, p2p/3',
      RemoteAddress: '144.76.62.101:30303',
      LocalAddress: '192.168.0.28:40454'
 }, {
  ID:
'f4642fa65af50cfdea8fa7414a5def7bb7991478b768e296f5e4a54e8b995de102e0ceae2e826f293c481b5
325f89be6d207b003382e18a8ecba66fbaf6416c0',
  Name: '++eth/Zeppelin/Rascal/v0.9.14/Release/Darwin/clang/int',
  Caps: 'eth/60, shh/2',
  RemoteAddress: '129.16.191.64:30303',
  LocalAddress: '192.168.0.28:39705'
 } ]
```

通过geth检查端口和找到你的加密uri

```
> admin.nodeInfo
{
  Name: 'Geth/v0.9.14/darwin/go1.4.2',
  NodeUrl:
'enode://3414c01c19aa75a34f2dbd2f8d0898dc79d6b219ad77f8155abf1a287ce2ba60f14998a3a98c0cf
14915eabfdacf914a92b27a01769de18fa2d049dbf4c17694@[::]:30303',
  NodeID:
'3414c01c19aa75a34f2dbd2f8d0898dc79d6b219ad77f8155abf1a287ce2ba60f14998a3a98c0cf14915eab
fdacf914a92b27a01769de18fa2d049dbf4c17694',
  IP: '::',
  DiscPort: 30303,
  TCPPort: 30303,
  Td: '2044952618444',
  ListenAddr: '[::]:30303'
}
```

更快的下载区块链

以太坊客户端一旦运行，就会自动去下载区块链数据。下载的速度跟客户端的设置，同伴的数量和链接的速度有关。

使用GETH

–fast：

当你使用geth客户端的时候，如果是第一次同步，以前从来没有同步过，那么你可以使用 –fast 参数，来减少同步数据的时间。

–cache=1024

根据你计算机的内存大小来调整该参数大小，该参数默认大小是16M，可以改成246M,512M，或1G,2G。

–jitvm

开启JIT VM。

整个使用案例如：

```
geth --fast --cache=1024 --jitvm console
```

导出和导入区块链数据

如果你有一个区块链同步完整的以太坊节点，那么你可以导出这节点的区块链数据，然后导入到您的一个新的以太坊节点。你可以命令geth export filename导出数据，然后使用geth import filename导入数据到新的节点。

静态节点

以太坊支持静态节点，即你的节点始终会去链接的那个节点，一旦断开，你的节点又会去重新连接那个节点。你可以在<datadir>/static-nodes.json中配置你的静态节点

```
[

"enode://f4642fa65af50cfdea8fa7414a5def7bb7991478b768e296f5e4a54e8b995de102e0ceae2e826f2
93c481b5325f89be6d207b003382e18a8ecba66fbaf6416c0@33.4.2.1:30303",
"enode://pubkey@ip:port"
]
```

你也可以在运行时，在JavaScript控制台使用 admin.addPeer() 来添加静态节点：

```
>
admin.addPeer("enode://f4642fa65af50cfdea8fa7414a5def7bb7991478b768e296f5e4a54e8b995de10
2e0ceae2e826f293c481b5325f89be6d207b003382e18a8ecba66fbaf6416c0@33.4.2.1:30303")
```

链接可能会遇到的问题

有时候你可能无法链接，可能的原因有：

1. 计算机本地时间和网络时间没同步。
2. 链接被防火墙阻止了，你可以关闭防火墙，或在geth中使用addPeer()手动添加伙伴节点。

当你在测试一个节点，或在试验一个有固定节点组成的网络的时候，你可以会需要关闭发现协议，geth中加入–nodiscover参数即可。

测试网络

Morden 测试网络

Morden 是一个公开的以太坊测试网络。

使用

eth (C++ client) 支持0.9.93及以上版本的。运行客户端的时候需加入 –morden参数

PyEthApp (Python client) v1.0.5之后支持morden

GETH (GO CLIENT)

详情

所有的参数跟主网是一样的，除了一下几点：

1. 网络名字：Morden
2. 网络ID:2
3. json(如下)
4. 初始账号的随机数值（IAN）是2^20
    1. 所有账号的状态树都有一个大于IAN的随机数
    2. 当一个账号被插入到状态树时，它的随机数值等于IAN
5. Genesis（创世）通用块hash:
   0cd786a2425d16f152c658316c423e6ce1181e15c3295826d7c9904cba9ce303
6. Genesis（创世）通用状态root:

f3f4696bbf3b3b07775128eb7a3763279a394e382130f27c21e70233e04946a9

获得测试网络的以太币

有2个方法可以获得测试网络的以太币

1. 使用你的cpu/gpu挖矿
2. 使用 Ethereum wei faucet

建立本地私有测试网络。

ETH (C++ CLIENT)

使用–genesis 和 –config 可以连接或者创建一个新的网络

–genesis 和 –config 可以同时使用。在这种区块下，由–config 提供的创世区块的描述会被 –genesis 选项覆盖掉。

GETH (GO CLIENT)

你可以在你的私有测试网络上预先设置或挖你的以太币，这比在Morden 公共测试网络上有效的多。

在一个私有链中，你需要指定以下东西：

- 自定义genesis文件
- 自定义数据目录
- 自定义网络ID
- （推荐）关闭节点发现协议

genesis 文件

这genesis（创世）区块是区块链的起点，是区块链的第一块区块，0号区块，唯一一个没有前任的区块。这个协议确保了没有其他节点会和你的节点的区块链版本一致，除非它们的创世区块和你的一模一样。通过这种方法，你就可以创建任意多的私有区块链。

CustomGenesis.json

```
{
    "nonce": "0x0000000000000042",     "timestamp": "0x0",
    "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData": "0x0",     "gasLimit": "0x8000000",     "difficulty": "0x400",
    "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase": "0x3333333333333333333333333333333333333333",     "alloc": {     }
}
```

保存，然后在启动geth的时候添加如下参数

–genesis /path/to/CustomGenesis.json

私有链的命令行参数

–nodiscover

添加这个参数，确保没有人能发现你的节点。不然的话，可能有人无意中会链接到你的区块链。

–maxpeers 0

使用maxpeers 0,如果你不希望其他人连接到您的测试链。您也可以调整这个数,如果你知道有多少同伴会连接你的节点。

–rpc

在你的节点上激活RPC接口。这参数在geth中默认启用。

```
--rpcapi "db,eth,net,web3"
```

这个命令描述哪些接口可以通过RPC来访问，默认情况下，geth开启的是web3接口。

–rpcport “8080”

将端口号设置成8000以上的任何一个你网络中打开的端口。默认是8080。

–rpccorsdomain http://chriseth.github.io/browser-solidity/

设置可以连接到你的节点的url地址，以执行RPC客户端的任务。最好不要使用通配符 * ,这样将允许任何url都可以链接到你的RPC实例。

–datadir “/home/TestChain1”

私有链的数据目录，确保与公共以太坊链的数据目录区分开来。

–port “30303”

这是“网络监听的端口”,您可以用它手动的和你的同伴相连。

–identity “TestnetMainNode”

为你的节点设置一个ID。用于和你们的一系列同伴进行区分。

运行**geth**

依照上面所述，配置好之后，运行：

```
geth --identity "MyNodeName" --rpc --rpcport "8080" --rpccorsdomain "*" --datadir
"C:\chains\TestChain1" --port "30303" --nodiscover --rpcapi "db,eth,net,web3" --
networkid 1999 init /path/to/CustomGenesis.json
```

以上参数更改成你本地配置的。每次，当你想要使用你自定义的私有链时，你都需要先使用geth来运行以上的命令来启动它。

下面这命令将初始化你的创世区块：

```
geth --identity "MyNodeName" --rpc --rpcport "8080" --rpccorsdomain "*" --datadir
"C:\chains\TestChain1" --port "30303" --nodiscover --rpcapi "db,eth,net,web3" --
networkid 1999 console
```

如果你已经有了一个正在运行的geth节点，你可以附加另一个geth实例到你的节点上：

```
geth attach
```

现在，你可以在你自己的私有链上创建一个账号，并且把它设置成你的etherbase （接收挖矿奖励的账号），在JavaScript命令台里输入：

```
personal.newAccount("password")
```

注意：把密码改成你自己的。

然后，把这个账号设置成etherbase

```
miner.setEtherbase(personal.listAccounts[0])
```

如果设置成功，将会打印出 true 。

最后，你就可以在你的私有链上挖矿了：

```
miner.start()
```

预先分配以太币到你的账号。

在CustomGenesis.json文件中把difficulty 设置成“0x400”，能让你快速的在你的私有链上挖到矿，你可以每分钟挖到数百个以太币，足够你测试使用。如果你还是要对自己的账号预先设定以太币数量，那么你需要按如下操作：

1. 创建好你的私有链后，创建一个新的账号
2. 复制你刚创建的账号地址
3. 添加下面的命令到你的Custom_Genesis.json文件里面

```
"alloc":
{
        "<your account address e.g. 0x1fb891f92eb557f4d688463d0d7c560552263b5a>":
        { "balance": "200000000000000000" }
}
```

注意：用你的地址替换上面的 0x1fb891f92eb557f4d688463d0d7c560552263b5a。

保存该文件，然后重新运行你的私有链控制台命令行。当数据加载完毕后，关闭控制台。

我们要 把一个地址赋给变量 primary，并且检查它的余额。

运行命令geth account list，找到刚刚分配了金额的账号 ：

```
> geth account list
Account #0: {d1ade25ccd3d550a7eb532ac759cac7be09c2719}
Account #1: {da65665fc30803cb1fb7e6d86691e20b1826dee0}
Account #2: {e470b1a7d2c9c5c6f03bbaa8fa20db6d404a0c32}
Account #3: {f4dd5c3794f1fd0cdc0327a83aa472609c806e99}
```

另外，你也可以使用 geth console （和先前第一次运行geth的时候，保持一样的参数），当出现确认提示了，输入：

```
> eth.accounts
```

这将返回你拥有的账户的数组。

```
> primary = eth.accounts[0]
```

注意：把 0 替换成你的账户的索引。

输入以下命令：

```
> primary = eth.accounts[0]
```

这应该返回7.5，表明在你的账号里，你有这么多的以太币。我们之所以在CustomGenesis.json文件放一个很大的数据，是因为CustomGenesis.json中balance字段是以 wei 为单位，而wei是以太币中最小的单位！

2017年2月5日 / 以太坊-区块链 / 留下评论

以太坊账号管理 2017-02-03更新

账号管理

账号在以太坊中扮演着核心的角色。以太坊共有两种账号类型：外部账号(EOA)和合约账号。在这里我们先重点关注外部账号，简称账号。合约账号简称合约，合约账户在合约章节中将详细介绍。外部账户和合约账户都是账户的通用概念，都是状态对象。外部账户的余额就是外部账户的一个状态对象，合约账户的状态除了有余额还有合约存储。所有账户的状态都是以太坊网络的状态，以太坊网络的状态随着每一个区块的更新而变化。用户通过交易和以太坊区块链进行交互，在这个过程中，账户起着至关重要，必不可少的作用。

如果限制以太坊只有外部账号，并且限制它们只能交易，那么我们就是只是做了一个山寨币，而且只能交易以太币（ether）。

账号代表了使用者的一个对外的身份，用户使用公钥去签名一个交易，然后以太坊虚拟机就可以安全的 校验这交易发起者的身份。

秘钥文件

每一个账号都有一对密钥，一个私钥和一个公钥。账号和地址是一一对应的。账号被来自公钥的最后20个字节的地址索引着的。每一个私钥/地址对都被编码进一个秘钥文件。秘钥文件是一个json格式的文本文件，可以用任何的文本工具打开和编辑它。秘钥文件的重要组成部分—你的-账号的私钥，是被你在创建账号时输入的密码加密保护的。秘钥文件存储在你的以太坊客户端的keystore子目录中。确保定期备份你的key文件！！

创建一个秘钥等同于创建一个账号！

1. 你不需要告诉别人你创建了一个账号
2. 你不需要和区块链进行同步
3. 你不需要运行一个客户端
4. 你甚至不需要联网

当然你的新帐户不会有以太币。但它只属于你，没有你的秘钥文件和密码，其他人甚至都不能访问它。

注意，当你把一个来自其他节点的秘钥文件添加进你自己的节点的时候，账户的顺序有可能发生变化，所以在编写代码或脚本的时候，不要使用账户的索引顺序来指定一个账号.

创建一个账号

注意：一定要记住你的密码和备份你的秘钥文件！！！因为发送交易，甚至发送以太币都是必须要同时使用到密码和秘钥文件的。所以切记切记备份好你的秘钥文件和密码，把它们备份到一个绝对安全的地方。丢失了秘钥文件或密码，那你账户中的所有的以太币也就全部都丢失了。没有密码是绝对无法访问你的账户的，并且以太坊没有“忘记密码？找回密码”这一功能选项。

使用GETH ACCOUNT NEW

如果你安装了以太坊命令控制台钱包（geth ），那创建账号只需要在命令行里执行 geth account new 命令 即可。

请注意执行geth account,您不需要运行钱包客户端或同步区块链数据。

```
$ geth account new

  Your new account is locked with a password. Please give a password. Do not forget this
password.
  Passphrase:
  Repeat Passphrase:
  Address: {168bc315a2ee09042d83d7c5811b533620531f67}
```

无交互的创建账号的方式如下，使用 --password 标志，并且提供一个纯文本的密码文件给它当参数。这密码文件由原始的密码和最后的一个换行符组成。

```
$ geth --password /path/to/password account new
```

注意：无交互的创建账号的方式只适合于在测试网络或一个你绝对信任的安全环境中使用。将你的密码保存在文件中，或者暴露在其他任何的地方都是非常危险的。如果你以这种方式来创建账号，那最好能确保这文件除了你本人之外，其他人是不可读的，甚至是不可见的。在Mac/Linux 系统中，你可以使用下面的方式创建和保护你的密码文件：

```
touch /path/to/password
chmod 600 /path/to/password
cat > /path/to/password
>I type my pass
```

打开你本地的keystore文件夹，就可以看到秘钥文件所对对应着的你的所有的账号，或者，也可以使用geth account的子命令 list:

```
$ geth account list

account #0: {a94f5374fce5edbc8e2a8697c15331677e6ebf0b}
account #1: {c385233b188811c9f355d4caec14df86d6248235}
account #2: {7f444580bfef4b9bc7e14eb7fb2a029336b07c9d}
```


秘钥文件的文件名格式是 UTC--<created_at UTC ISO8601>-<address 十六进制>.keystore文件夹中的秘钥文件的展示顺序是按照账户的创建时间来的，而命令行中的展示顺序是按照字母排序来的。

使用 geth console

为了能使用geth命令，创建新的帐户,我们必须首先以命令控制台模式开启geth (或者您可以使用 geth attach 附加一个控制台到已经运行钱包客户端):

```
> geth console 2>> file_to_log_output
instance: Geth/v1.4.0-unstable/linux/go1.5.1
coinbase: coinbase: [object Object]
at block: 865174 (Mon, 18 Jan 2016 02:58:53 GMT)
datadir: /home/USERNAME/.ethereum
```

这控制台允许你与本地节点通过命令来交互。例如,尝试使用命令来列出你的账户:

```
> eth.accounts

{
code: -32000,
message: "no keys in store"
}
```

这表明你还没有账户。您也可以通过控制台来创建一个帐户:

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0xb2f69ddf70297958e582a0cc98bce43294f1007d"
```

我们刚刚创建的第一个帐户。如果我们再次尝试列出我们的账户,就可以看到我们的新帐户了:

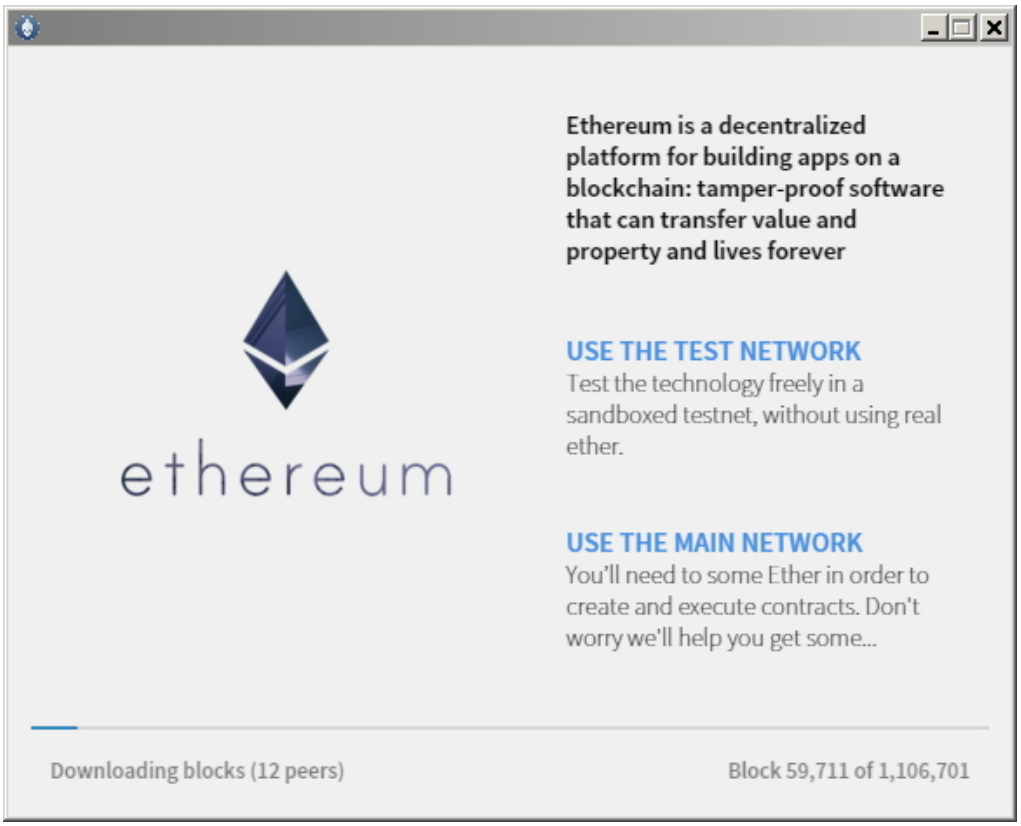
```
> eth.accounts
["0xb2f69ddf70297958e582a0cc98bce43294f1007d"]
```

使用图形化钱包（Mist）工具

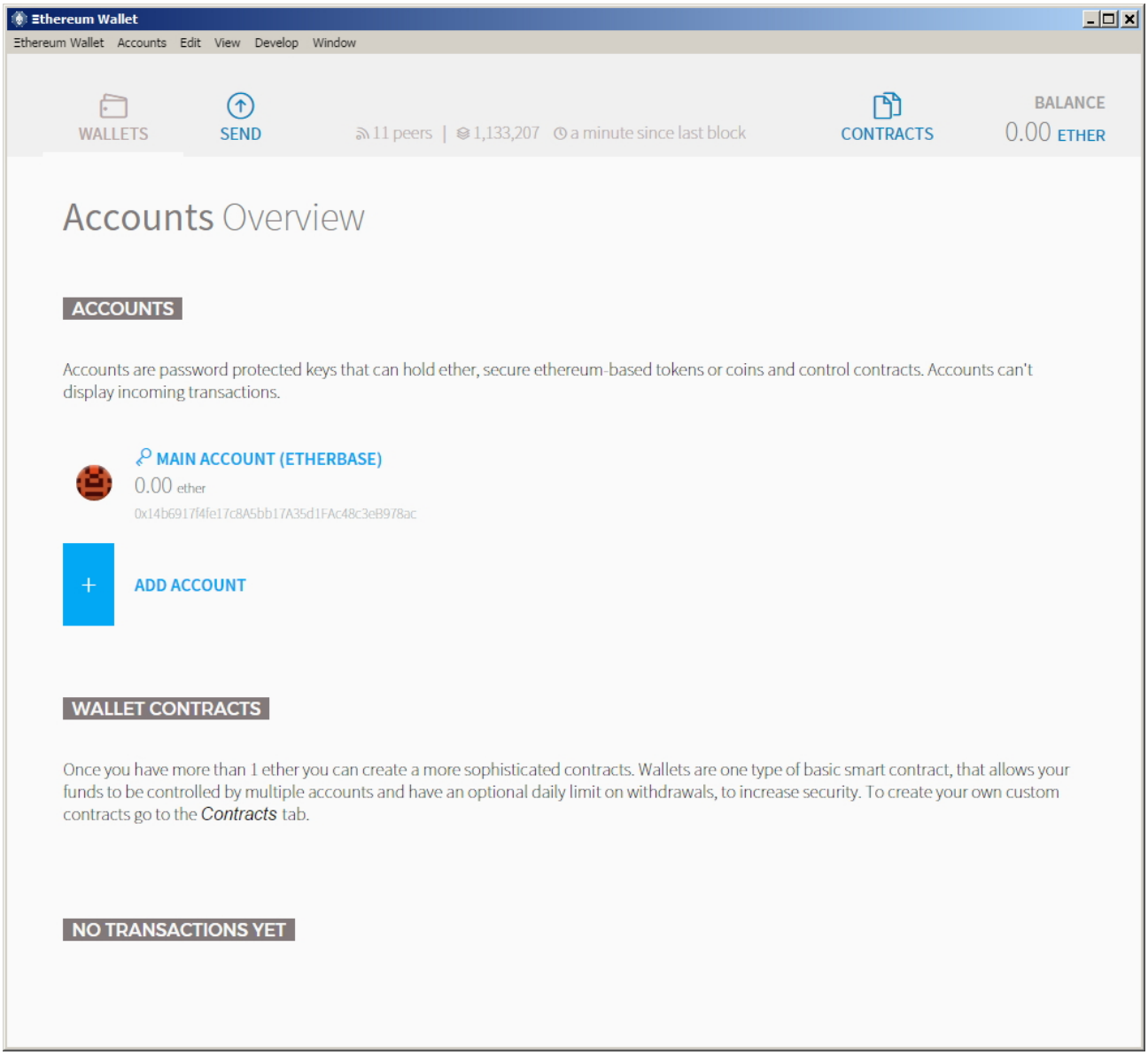
如果你不喜欢敲命令，那么你可以使用“官方”提供的图形化客户端工具：Mist 以太坊钱包。目前， Linux, Mac OS X, 和 Windows 系统都已经有相关版本提供下载了。

使用Mist创建一个账号是非常简单的，事实上，在你安装mist的时候就已经创建了第一个账号。

1. 下载适合你操作系统的最新版本的钱包，解压，并运行可执行文件。



2. 等待区块链完全同步,然后按照屏幕上的指示,创建你的第一个帐户。
3. 当你第一次启动mist，您将看到您在安装过程中创建的帐户。默认情况下它将被命名为主要帐户（ MAIN ACCOUNT ）。



4. 添加一个账户很容易,只需在主屏幕上点击添加帐户（ADD ACCOUNT），然后输入密码。

创建一个多重签名的钱包

这Mist 以太坊钱包可以使用多重签名来保护你账户中的余额。使用多重签名的好处是，当要从账户中提取较大额度的金额时，需要多个账户的共同认证才能成功提取。在创建一个多重签名的钱包前，你首先需要创建多个账户。

在mist中创建账号很容易，只需要在 Accounts 选项卡，点击 'Add Account'。创建至少2个账号。如果有条件的话，第二个账号可以在你运行mist的另一台电脑上创建（理论上这样做更加的安全）。你只需要用到你的第二个账号的公钥，第二个账号也是你的存款的地址。你的主账号将用来创建多重签名钱包合约，所以多重签名钱包必须和你的主账号在同一台电脑上。

现在，你已经对你的账号进行了设置。为安全起见，你需要备份它们！如果没有备份，那一旦你电脑崩溃了，你将失去你账号中的所有金额。在顶部菜单点击‘Backup’。选择‘keystore’ 文件夹，选择 ‘copy’，打开你想放置备份文件的文件夹，然后右键黏贴。你可以复制这文件到你的另一个电脑或U盘或云盘等。

你现在应该向你的主账号添加不少于0.02的以太币（用于创建多重签名钱包的账号），这是创建多重签名钱包合约的交易费用。另外还需要至少1以太币，因为当前mist需要有足够的 ‘gas’来确保多重签名钱包合约能够正确的执行交易。所以一开始总共需要1.02的以太币。

现在我们已经准备好创建多重签名钱包。在 ‘Wallet Contracts’下面，选择 ‘Add Wallet Contract’。给它取个名称,选择主帐户所有者,并选择 ‘Multisignature Wallet Contract’。你会看到:

“这是一个被X个用户共同控制的账号。你可以每天成交X个以太币。每日的任何交易都需要X个用户的确认。”(“This is a joint account controlled by X owners. You can send up to X ether per day. Any transaction over that daily limit requires the confirmation of X owners.”)

设置多少金额到多重签名钱包里，多少额度一个用户每日可以提取，需要多少个用户批准。

现在可以添加你账号的地址了，先确保所有的设置都是正确的，然后点击底部的 ‘Create’ 按钮。输入你的密码发送交易。在你的 ‘Wallet Contracts’ 页面，你应该可以看到你的新的钱包合约，并且显示 创建中（creating）.

当钱包创建完成后,您应该在屏幕上看到你的合约地址。选中整个地址,复制/粘贴到一个新的文本文件,并将文本文件保存到桌面命名为“Ethereum-Wallet-Address.txt”,或你想要的任何名字。

你需要像备份秘钥文件一样备份‘Ethereum-Wallet-Address.txt’。然后你就可以用这个多重签名钱包了。

如果你用备份来恢复，只要把你备份的文件复制到‘keystore’文件夹即可。如果你是一台全新的电脑，以前从来没在这电脑上装过mist，那么你可能需要手动创建‘keystore’文件夹。至于恢复多重签名钱包，用 ‘Import Wallet’替代我们之前创建它时所用的 ‘Multisignature Wallet Contract’ 即可。

排错

1. mist不同步，一个有效的解决办法是，同步你电脑的硬件时钟和NTP服务器完全一致，然后重启。
2. “错误密码”提示。如果你确定输入的是正确的密码，那么你可以重启下当前版本的mist偶有会有这问题。

使用ETH

geth钱包中任何涉及到操作密钥管理的部门都用eth相关的命令。

```
> eth account list  // 列出钱包中所有有效的账号.
> eth account new   // 创建一个新账号，并且把它添加到钱包里.
> eth account update [<uuid>|<address> , ... ]  // 对给定的账号重新解码和编码.
> eth account import [<uuid>|<file>|<secret-hex>] // 导入账号到钱包里.
```

下面是“wallet”相关：

> eth wallet import <file> //导入一个先前的钱包

也可以通过综合控制台来访问密钥管理(使用内置的控制台或geth附加):

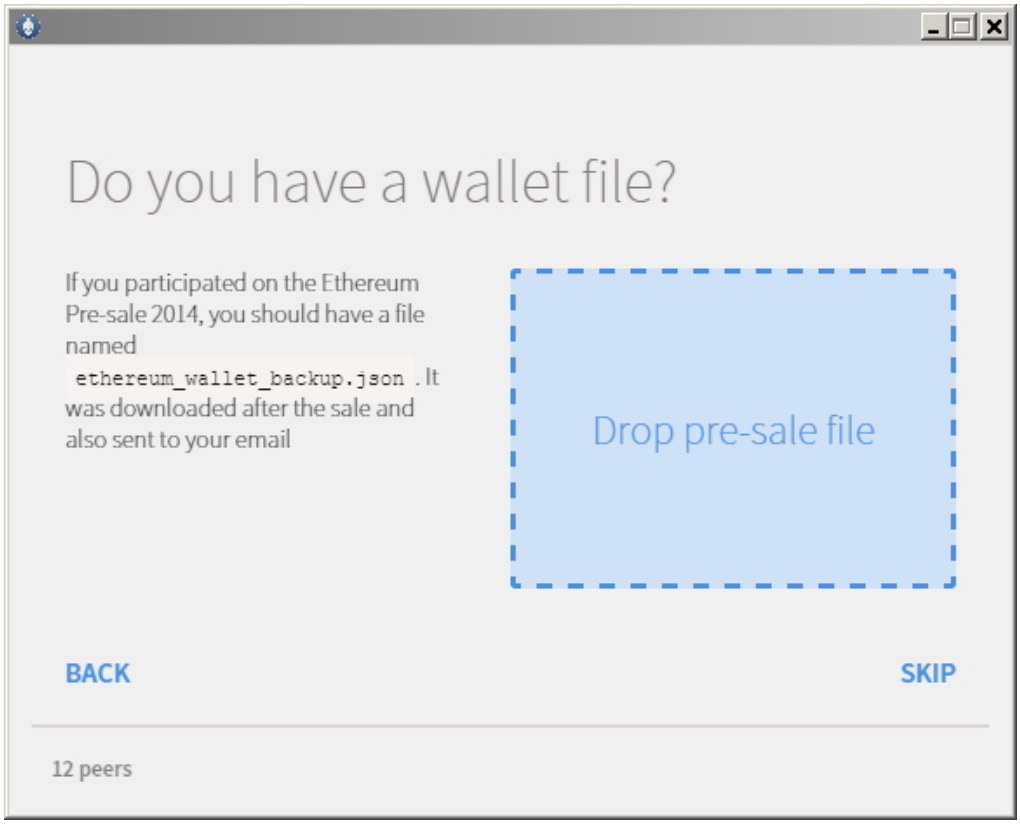
```
> web3.personal
{
      listAccounts: [],
      getListAccounts: function(callback),
      lockAccount: function(),
      newAccount: function(),
      unlockAccount: function()
}
```

导入之前的钱包

使用**mist**

使用图形化界面mist导入钱包是很容易的。事实上，当你安装mist的时候，你就会被询问是否要导入你之前的钱包。

把你想要导入的钱包的json文件拖拽到图中指定的区域，然后输入你的密码就可以了。



如果你在安装过程中没有导入钱包，那么你还可以在顶部菜单点击 'Accounts'子菜单 'Import Pre-sale Accounts'的方式来导入钱包。

使用 **geth**

如果你独立安装了geth,你可以在终端中通过执行下面的命令来导入钱包：

```
geth wallet import /path/to/my/presale-wallet.json
```

如果没有意外，输入密码即可导入成功。

更新一个账号

你可以升级你的密钥文件至最新的密钥文件格式或升级您的密钥文件的密码。

使用**geth**

你可以使用'account'的子命令'update'来更新一个你的账号，这个命令需要传入账号地址或索引位置。账号的索引由创建账号的时间决定的。

```
geth account update b0047c606f3af7392e073ed13253f8f4710b08b6
```

或

```
geth account update 2
```

比如：

```
$ geth account update a94f5374fce5edbc8e2a8697c15331677e6ebf0b

Unlocking account a94f5374fce5edbc8e2a8697c15331677e6ebf0b | Attempt 1/3
Passphrase:
0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b
account 'a94f5374fce5edbc8e2a8697c15331677e6ebf0b' unlocked.
Please give a new password. Do not forget this password.
Passphrase:
Repeat Passphrase:
0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b
```

账号会使用最新版的加密格式保存，你会被提示输入你的密码来解锁账号，然后保存更新后的文件。这个命令同时也可以用来把一个账号从被弃用的加密格式迁移到最新的加密格式，或者修改一个账号的密码。

备份还原账号

手动备份还原

如果你想发送一个交易，那么你必须要有一个秘钥文件。秘钥文件存储在以太坊节点数据目录的子目录'keystore'中，不同的平台默认数据目录是不一样的：

- Windows: `C:\Users\username\%appdata%\Roaming\Ethereum\keystore`
- Linux: `~/.ethereum/keystore`
- Mac: `~/Library/Ethereum/keystore`

备份秘钥文件(账户),可备份keystore目录下的单独的密钥文件，或干脆备份整个keystore目录。还原的时候，把这些文件还原到原先的目录即可。

导入一个未加密的私钥

可以使用geth来导入一个未加密的私钥

geth account import /path/to/<keyfile>

这个命令从纯文本中导入一个未加密的密钥，并创建一个新帐户和打印对应的地址。其中会要求你输入密码，牢记住密码，以便将来解锁账号。账号会被加密格式保护。

下面是一个指定了数据目录的列子。如果不使用 --datadir 标签，这新的账号将会被创建在默认的数据目录，比如，这秘钥文件将会被放在数据目录的子目录 keystore 中。

```
$ geth --datadir /someOtherEthDataDir  account import ./key.prv
The new account will be encrypted with a passphrase.
Please enter a passphrase now.
Passphrase:
Repeat Passphrase:
Address: {7f444580bfef4b9bc7e14eb7fb2a029336b07c9d}
```

2017年2月3日 / 以太坊-区块链 / 留下评论

以太坊 JSON RPC 接口

- JSON-RPC 终端

默认的JSON-RPC 终端:

C++	http://localhost:8545
Go	http://localhost:8545
Parity	http://localhost:8545
Py	http://localhost:4000

反馈
建议

Go

使用标签 `--rpc` ，开启 HTTP JSON-RPC

```
geth --rpc
```

修改默认的端口 (8545) 和监听地址 (localhost):

```
geth --rpc --rpcaddr <ip> --rpcport <portnumber>
```

如果使用浏览器来访问这RPC, 需要设置跨域站点请求:

```
geth --rpc --rpccorsdomain "http://localhost:3000"
```

JSON RPC 接口也可以在 geth console 控制台，使用 `admin.startRPC(addr, port)` 命令开启.

C++

可以使用 `eth` 程序加 `-j` 选项启动JSON RPC 接口:

```
./eth -j
```

指定 JSON-RPC 端口(默认是 8545):

```
./eth -j --json-rpc-port 8079
```

Python

默认开启： `127.0.0.1:4000`

可以通过一个配置选项改变监听的端口和IP地址.

```
pyethapp -c jsonrpc.listen_port=4002 -c jsonrpc.listen_host=127.0.0.2 run
```

JSON-RPC 支持

	cpp-ethereum	go-ethereum	py-ethereum
JSON-RPC 1.0	✓		
JSON-RPC 2.0	✓	✓	✓
Batch requests	✓	✓	✓
HTTP	✓	✓	✓
IPC	✓	✓	
WS		✓	

十六进制值的编码

当前有2个关键的数据类型会通过JSON传递: 无格式的字节数组和数据。传递这些参数的时候，都需要使用十六进制编码 ,格式有所不同:

当加密数据(integers, numbers): 使用前缀”0x”, 最简单的列子 (0应该使用 “0x0” 来表示). 比如:

- 0x41 (十进制中是65)
- 0x400 (十进制中是1024)
- 错误: 0x (应该至少是一个数字- 0是 “0x0”)
- 错误: 0x0400 (0不能放在第一位)
- 错误: ff (必须以前缀 0x开始)

当编码无格式的数据(字节数组, 账户地址, 哈希, 字节码数组)时: 同样要使用前缀 “0x”,2个十六进制的数字代表一个字节 .比如:

- 0x41 (长度 1, “A”)
- 0x004200 (长度 3, “\0B\0”)
- 0x (长度 0, “”)
- 错误: 0xf0f0f (必须是偶数位个数字)
- 错误: 004200 (必须使用前缀 0x)

目前 cpp-ethereum 和 go-ethereum 都有http形式的 JSON-RPC 接口 . 从版本 1.4 起，go-ethereum 开始支持 websocket

默认的区块参数

下面的方法有一个额外的默认区块参数:

- eth_getBalance
- eth_getCode
- eth_getTransactionCount
- eth_getStorageAt
- eth_call

默认区块参数有下面几个可选项:

- HEX String – 十六进制字符串，一个整数形的区块编号
- 字符串 "earliest" 最早的区块
- 字符串 "latest" – 最后一个被挖到的区块
- 字符串 "pending" – 区块或交易处于待定(pending) 状态

JSON-RPC methods

- web3_clientVersion
- web3_sha3
- net_version
- net_peerCount
- net_listening
- eth_protocolVersion
- eth_syncing
- eth_coinbase
- eth_mining
- eth_hashrate
- eth_gasPrice
- eth_accounts
- eth_blockNumber
- eth_getBalance
- eth_getStorageAt
- eth_getTransactionCount

- eth_getBlockTransactionCountByHash
- eth_getBlockTransactionCountByNumber
- eth_getUncleCountByBlockHash
- eth_getUncleCountByBlockNumber
- eth_getCode
- eth_sign
- eth_sendTransaction
- eth_sendRawTransaction
- eth_call
- eth_estimateGas
- eth_getBlockByHash
- eth_getBlockByNumber
- eth_getTransactionByHash
- eth_getTransactionByBlockHashAndIndex
- eth_getTransactionByBlockNumberAndIndex
- eth_getTransactionReceipt
- eth_getUncleByBlockHashAndIndex
- eth_getUncleByBlockNumberAndIndex
- eth_getCompilers
- eth_compileLLL
- eth_compileSolidity
- eth_compileSerpent
- eth_newFilter
- eth_newBlockFilter
- eth_newPendingTransactionFilter
- eth_uninstallFilter
- eth_getFilterChanges
- eth_getFilterLogs
- eth_getLogs
- eth_getWork
- eth_submitWork
- eth_submitHashrate
- db_putString
- db_getString
- db_putHex
- db_getHex
- shh_post
- shh_version
- shh_newIdentity
- shh_hasIdentity

- shh_newGroup
- shh_addToGroup
- shh_newFilter
- shh_uninstallFilter
- shh_getFilterChanges
- shh_getMessages

JSON RPC API 参考手册

WEB3_CLIENTVERSION

返回当前客户端版本.

Parameters

none

Returns

String – 这当前客户端的版本

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"web3_clientVersion","params":
[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc":"2.0",
  "result": "Mist/v0.9.3/darwin/go1.4.1"
}
```

WEB3_SHA3

返回给定参数的Keccak-256 (*不是标准的*SHA3-256) 值.

Parameters

1. DATA – 一个要被转化成一个SHA3哈希的数据

```
params: [
  '0x68656c6c6f20776f726c64'
]
```

Returns

DATA .一个SHA3值

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"web3_sha3","params":
["0x68656c6c6f20776f726c64"],"id":64}'

// Result
{
  "id":64,
  "jsonrpc": "2.0",
  "result": "0x47173285a8d7341e5e972fc677286384f802f8ef42a5ec5f03bbfa254cb01fad"
}
```

NET_VERSION

返回这当前网络协议的版本号.

Parameters

none

Returns

String 当前网络协议的版本号

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_version","params":[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc": "2.0",
  "result": "59"
}
```

NET_LISTENING

返回 true 如果客户端正在监听网络链接.

Parameters

none

Returns

Boolean – 监听时 true , 不然 false .

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_listening","params":[],"id":67}'
```

```
// Result
{
  "id":67,
  "jsonrpc":"2.0",
  "result":true
}
```

NET_PEERCOUNT

返回链接到当前客户端的伙伴数量.

Parameters

none

Returns

QUANTITY – 链接着的伙伴数量.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_peerCount","params":[],"id":74}'

// Result
{
  "id":74,
  "jsonrpc": "2.0",
  "result": "0x2" // 2
}
```

ETH_PROTOCOLVERSION

返回当前以太坊协议的版本.

Parameters

none

Returns

String – 当前以太坊协议的版本

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_protocolVersion","params":
[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc": "2.0",
  "result": "54"
}
```

ETH_SYNCING

返回一个关于同步状态的数据对象 或 false.

Parameters

none

Returns

Object|Boolean,一个同步的状态数据的对象，或 FALSE,当不在同步的时候:

- startingBlock: QUANTITY – The block at which the import started (will only be reset, after the sync reached his head)

- currentBlock : QUANTITY – 当前区块，和eth_blockNumber一样
- highestBlock : QUANTITY – The estimated highest block

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_syncing","params":[],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": {
    startingBlock: '0x384',
    currentBlock: '0x386',
    highestBlock: '0x454'
  }
}
// Or when not syncing
{
  "id":1,
  "jsonrpc": "2.0",
  "result": false
}
```

ETH_COINBASE

返回当前客户的coinbass 地址.

Parameters

none

Returns

DATA , 20 字节 – 这当前的coinbase 地址.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_coinbase","params":[],"id":64}'

// Result
{
  "id":64,
  "jsonrpc": "2.0",
  "result": "0x407d73d8a49eeb85d32cf465507dd71d507100c1"
}
```

ETH_MINING

返回 true 如果客户端正在挖矿.

Parameters

none

Returns

Boolean – 如果客户端正在挖矿返回 true ， 不然 false .

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_mining","params":[],"id":71}'

// Result
{
  "id":71,
  "jsonrpc": "2.0",
  "result": true
}
```

ETH_HASHRATE

返回节点挖矿的速率，每秒哈希的数量.

Parameters

none

Returns

QUANTITY – 每秒哈希的数量.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_hashrate","params":[],"id":71}'

// Result
{
  "id":71,
  "jsonrpc": "2.0",
  "result": "0x38a"
}
```

ETH_GASPRICE

返回一个gas的价格，单位 wei.

Parameters

none

Returns

QUANTITY – 整数，当前gas的价格，单位 wei.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_gasPrice","params":[],"id":73}'

// Result
{
  "id":73,
  "jsonrpc": "2.0",
  "result": "0x09184e72a000" // 10000000000000
}
```

ETH_ACCOUNTS

返回当前客户所有者的一个地址列表.

Parameters

none

Returns

数组.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_accounts","params":[],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": ["0x407d73d8a49eeb85d32cf465507dd71d507100c1"]
}
```

ETH_BLOCKNUMBER

返回最近的区块的编号.

Parameters

none

Returns

QUANTITY – 整数，当前区块的编号.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_blockNumber","params":[],"id":83}'

// Result
{
  "id":83,
  "jsonrpc": "2.0",
  "result": "0x4b7" // 1207
}
```

ETH_GETBALANCE

返回给定地址的账户余额.

Parameters

1. DATA, 20 个字节 – 要获取余额的地址.
2. QUANTITY|TAG – 整数，区块的编号,或字符串 "latest", "earliest" 或 "pending",查看 默认区块参数

```
params: [
   '0x407d73d8a49eeb85d32cf465507dd71d507100c1',
```

```
    'latest'
]
```

Returns

QUANTITY – 整数，余额值，单位wei.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getBalance","params":
["0x407d73d8a49eeb85d32cf465507dd71d507100c1", "latest"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x0234c8a3397aab58" // 158972490234375000
}
```

ETH_GETSTORAGEAT

返回指定地址上的一个存储位置上的值.

Parameters

1. DATA , 20 字节 – 这存储的地址.
2. QUANTITY – 整数，存储中位置.
3. QUANTITY|TAG – 整数，区块编号，或字符串 "latest", "earliest"或 "pending",查看更多默认区块参数。

Returns

DATA – 在存储位置上的值

Example

Calculating the correct position depends on the storage to retrieve. Consider the following contract deployed on `0x295a70b2de5e3953354a6a8344e616ed314d7251` from address `0x391694e7e0b0cce554cb130d723a9d27458f9298`.

```
contract Storage {
    uint pos0;
    mapping(address => uint) pos1;

    function Storage() {
        pos0 = 1234;
        pos1[msg.sender] = 5678;
    }
}
```

Retrieving the value of pos0 is straight forward:

```
curl -X POST --data '{"jsonrpc":"2.0", "method": "eth_getStorageAt", "params":
["0x295a70b2de5e3953354a6a8344e616ed314d7251", "0x0", "latest"], "id": 1}'
localhost:8545

{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000000000
0000000004d2"}
```

Retrieving an element of the map is harder. The position of an element in the map is calculated with:

```
keccack(LeftPad32(key, 0), LeftPad32(map position, 0))
```

This means to retrieve the storage on pos1["0x391694e7e0b0cce554cb130d723a9d27458f9298"] we need to calculate the position with:

```
keccak(decodeHex("000000000000000000000000391694e7e0b0cce554cb130d723a9d27458f9298" +
"0000000000000000000000000000000000000000000000000000000000000001"))
```


The geth console which comes with the web3 library can be used to make the calculation:

```
> var key = "0000000000000000000000000391694e7e0b0cce554cb130d723a9d27458f9298" +
"0000000000000000000000000000000000000000000000000000000000000001"
undefined
> web3.sha3(key, {"encoding": "hex"})
"0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9"
```

Now to fetch the storage:

```
curl -X POST --data '{"jsonrpc":"2.0", "method": "eth_getStorageAt", "params":
["0x295a70b2de5e3953354a6a8344e616ed314d7251",
"0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9", "latest"], "id":
1}' localhost:8545

{"jsonrpc":"2.0","id":1,"result":"0x00000000000000000000000000000000000000000000000000
00000000162e"}
```

ETH_GETTRANSACTIONCOUNT

返回在地址上发出的交易数量.

Parameters

1. DATA , 20 字节 – 地址.
2. QUANTITY|TAG – 整数，区块编号，或字符串 "latest" , "earliest" 或 "pending" , 查看 默认区块参数

```
params: [
   '0x407d73d8a49eeb85d32cf465507dd71d507100c1',
   'latest' // state at the latest block
]
```

Returns

QUANTITY – 从这个地址发出的交易数量.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getTransactionCount","params":
["0x407d73d8a49eeb85d32cf465507dd71d507100c1","latest"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

ETH_GETBLOCKTRANSACTIONCOUNTBYHASH

根据给定的区块哈希，返回这区块中交易的数量.

Parameters

1. DATA , 32 字节 – 一个区块的哈希值

```
params: [
   '0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238'
]
```

Returns

QUANTITY – 整数，在这个区块中的交易数量.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getBlockTransactionCountByHash","params":
["0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id":1}'
```

```
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xb" // 11
}
```

ETH_GETBLOCKTRANSACTIONCOUNTBYNUMBER

根据区块的编号，返回这个区块中的交易数量.

Parameters

1. QUANTITY|TAG – 整数，一个区块编号，或字符串 "earliest","latest" 或 "pending",查看 默认区块参数.

```
params: [
   '0xe8', // 232
]
```

Returns

QUANTITY – 整数，这个区块中的交易数量.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getBlockTransactionCountByNumber","params":
["0xe8"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
```

```
  "result": "0xa" // 10
}
```

ETH_GETUNCLECOUNTBYBLOCKHASH

根据一个给定的区块哈希，返回这个区块中的叔叔的数量.

Parameters

1. DATA , 32 字节 – 一个区块的哈希

```
params: [
   '0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238'
]
```

Returns

QUANTITY – 整数，在这个区块里的叔叔的数量.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getUncleCountByBlockHash","params":
["0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

ETH_GETUNCLECOUNTBYBLOCKNUMBER

根据给定的区块编号，返回这区块里面的叔叔数量.

Parameters

1. QUANTITY – 整数，区块编号，或字符串 “latest”, “earliest” 或 “pending”, 查看 默认去快参数

```
params: [
   '0xe8', // 232
]
```

Returns

QUANTITY – 整数，这个区块中叔叔的数量.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getUncleCountByBlockNumber","params":["0xe8"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

ETH_GETCODE

返回指定地址上的代码.

Parameters

1. DATA , 20 个字节 – 地址
2. QUANTITY|TAG – 整数，区块编号,或字符串 "latest", "earliest" 或 "pending",查看 默认区块参数

```
params: [
   '0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b',
   '0x2'  // 2
]
```

Returns

DATA – 给定地址上的代码.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getCode","params":
["0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b", "0x2"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060078
202905091905056"
}
```

ETH_SIGN

使用一个给定的地址来签名数据.

注意 不能用锁定的地址来签名.

Parameters

1. DATA , 20 个字节 – 地址
2. DATA , 32 个字节 – 要签名的数据的sha3 哈希

Returns

DATA : 签名

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_sign","params":
["0x8a3106a3e50576d4b6794a0e74d3bb5f8c9acaab",
"0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0xbd685c98ec39490f50d15c67ba2a8e9b5b1d6d7601fca80b295e7d717446bd8b7127ea4871e996cdc8cae
7690408b4e800f60ddac49d2ad34180e68f1da0aaf001"
}
```

ETH_SENDTRANSACTION

创建新的消息调用交易或一个合约, 如果这个数据字段包含代码.

Parameters

1. Object – 这交易对象
   - from : DATA , 20 个字节 – 发送交易的地址.
   - to : DATA , 20 个字节 – (可选，当创建新合约) 这交易指向的地址.
   - gas : QUANTITY – (可选, 默认: 90000) 整数，提供给交易执行 gas. 将返回未使用的 gas.
   - gasPrice : QUANTITY – (可选, 默认: 待定) 整数，每次支付gas时的gas价格
   - value : QUANTITY – (可选) 整数，这个交易发送的值
   - data : DATA – 一个编译的合约代码或这回调函数的哈希值和编码的参数. 查看更多 Ethereum Contract ABI

- nonce : QUANTITY – (可选) 整数.可以使用相同的nonce来覆盖你自己的待定的交易 .

```
params: [{
  "from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
  "to": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
  "gas": "0x76c0", // 30400,
  "gasPrice": "0x9184e72a000", // 10000000000000
  "value": "0x9184e72a", // 2441406250
  "data":
"0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675"
}]
```

Returns

DATA , 32 个字节 – 这交易哈希, 或0，如果尚不可用.

当你创建了一个合约，在这个交易被挖出后，可以使用 eth_getTransactionReceipt 获取合约地址.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_sendTransaction","params":[{see
above}],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
}
```

ETH_SENDRAWTRANSACTION

创建新的消息调用交易或一个为了签名交易的合约.

Parameters

1. DATA , 这签名了的交易数据.

```
params:
["0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675"]
```

Returns

DATA , 32 字节 – 这交易哈希值，或0，如果交易尚不可用.

当你创建了一个合约，在这个交易被挖出后，可以使用 eth_getTransactionReceipt 获取合约地址.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_sendRawTransaction","params":[{see
above}],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
}
```

ETH_CALL

Executes a new message call immediately without creating a transaction on the block chain.

Parameters

1. Object – The transaction call object
   - from : DATA , 20 Bytes – (optional) The address the transaction is sent from.
   - to : DATA , 20 Bytes – The address the transaction is directed to.
   - gas : QUANTITY – (optional) Integer of the gas provided for the transaction execution. eth_call consumes zero gas, but this parameter may be needed by some executions.

- `gasPrice`: `QUANTITY` – (optional) Integer of the gasPrice used for each paid gas
- `value`: `QUANTITY` – (optional) Integer of the value send with this transaction
- `data`: `DATA` – (optional) Hash of the method signature and encoded parameters. For details see Ethereum Contract ABI

2. `QUANTITY|TAG` – integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the default block parameter

Returns

`DATA` – the return value of executed contract.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_call","params":[{see
above}],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x"
}
```

ETH_ESTIMATEGAS

Makes a call or transaction, which won't be added to the blockchain and returns the used gas, which can be used for estimating the used gas.

Parameters

See eth_call parameters, expect that all properties are optional.

Returns

`QUANTITY` – the amount of gas used.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_estimateGas","params":[{see
above}],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x5208" // 21000
}
```

ETH_GETBLOCKBYHASH

Returns information about a block by hash.

Parameters

1. `DATA`, 32 Bytes – Hash of a block.
2. `Boolean` – If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

```
params: [
   '0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331',
   true
]
```

Returns

`Object` – A block object, or `null` when no block was found:

- `number`: `QUANTITY` – the block number. `null` when its pending block.
- `hash`: `DATA`, 32 Bytes – hash of the block. `null` when its pending block.

- parentHash : DATA , 32 Bytes – hash of the parent block.
- nonce : DATA , 8 Bytes – hash of the generated proof-of-work. null when its pending block.
- sha3Uncles : DATA , 32 Bytes – SHA3 of the uncles data in the block.
- logsBloom : DATA , 256 Bytes – the bloom filter for the logs of the block. null when its pending block.
- transactionsRoot : DATA , 32 Bytes – the root of the transaction trie of the block.
- stateRoot : DATA , 32 Bytes – the root of the final state trie of the block.
- receiptsRoot : DATA , 32 Bytes – the root of the receipts trie of the block.
- miner : DATA , 20 Bytes – the address of the beneficiary to whom the mining rewards were given.
- difficulty : QUANTITY – integer of the difficulty for this block.
- totalDifficulty : QUANTITY – integer of the total difficulty of the chain until this block.
- extraData : DATA – the “extra data” field of this block.
- size : QUANTITY – integer the size of this block in bytes.
- gasLimit : QUANTITY – the maximum gas allowed in this block.
- gasUsed : QUANTITY – the total used gas by all transactions in this block.
- timestamp : QUANTITY – the unix timestamp for when the block was collated.
- transactions : Array – Array of transaction objects, or 32 Bytes transaction hashes depending on the last given parameter.
- uncles : Array – Array of uncle hashes.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getBlockByHash","params":
["0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331", true],"id":1}'

// Result
{
"id":1,
"jsonrpc":"2.0",
"result": {
    "number": "0x1b4", // 436
    "hash": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331",
    "parentHash": "0x9646252be9520f6e71339a8df9c55e4d7619deeb018d2a3f2d21fc165dde5eb5",
    "nonce": "0xe04d296d2460cfb8472af2c5fd05b5a214109c25688d3704aed5484f9a7792f2",
    "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
    "logsBloom": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331",
    "transactionsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "stateRoot": "0xd5855eb08b3387c0af375e9cdb6acfc05eb8f519e419b874b6ff2ffda7ed1dff",
    "miner": "0x4e65fda2159562a496f9f3522f89122a3088497a",
    "difficulty": "0x027f07", // 163591
    "totalDifficulty":  "0x027f07", // 163591
    "extraData": "0x0000000000000000000000000000000000000000000000000000000000000000",
```

```
    "size":  "0x027f07", // 163591
    "gasLimit": "0x9f759", // 653145
    "gasUsed": "0x9f759", // 653145
    "timestamp": "0x54e34e8e" // 1424182926
    "transactions": [{...},{ ... }]
    "uncles": ["0x1606e5...", "0xd5145a9..."]
  }
}
```

ETH_GETBLOCKBYNUMBER

Returns information about a block by block number.

Parameters

1. `QUANTITY|TAG` – integer of a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the default block parameter.
2. `Boolean` – If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

```
params: [
   '0x1b4', // 436
   true
]
```

Returns

See eth_getBlockByHash

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getBlockByNumber","params":["0x1b4",
true],"id":1}'
```

Result see eth_getBlockByHash

ETH_GETTRANSACTIONBYHASH

Returns the information about a transaction requested by transaction hash.

Parameters

1. `DATA`, 32 Bytes – hash of a transaction

```
params: [
   "0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"
]
```

Returns

`Object` – A transaction object, or `null` when no transaction was found:

- `hash`: `DATA`, 32 Bytes – hash of the transaction.
- `nonce`: `QUANTITY` – the number of transactions made by the sender prior to this one.
- `blockHash`: `DATA`, 32 Bytes – hash of the block where this transaction was in. `null` when its pending.
- `blockNumber`: `QUANTITY` – block number where this transaction was in. `null` when its pending.
- `transactionIndex`: `QUANTITY` – integer of the transactions index position in the block. `null` when its pending.
- `from`: `DATA`, 20 Bytes – address of the sender.
- `to`: `DATA`, 20 Bytes – address of the receiver. `null` when its a contract creation transaction.
- `value`: `QUANTITY` – value transferred in Wei.
- `gasPrice`: `QUANTITY` – gas price provided by the sender in Wei.
- `gas`: `QUANTITY` – gas provided by the sender.
- `input`: `DATA` – the data send along with the transaction.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getTransactionByHash","params":
["0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id":1}'

// Result
{
"id":1,
"jsonrpc":"2.0",
"result": {
    "hash":"0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b",
    "nonce":"0x",
    "blockHash": "0xbeab0aa2411b7ab17f30a99d3cb9c6ef2fc5426d6ad6fd9e2a26a6aed1d1055b",
    "blockNumber": "0x15df", // 5599
    "transactionIndex":  "0x1", // 1
    "from":"0x407d73d8a49eeb85d32cf465507dd71d507100c1",
    "to":"0x85h43d8a49eeb85d32cf465507dd71d507100c1",
    "value":"0x7f110" // 520464
    "gas": "0x7f110" // 520464
    "gasPrice":"0x09184e72a000",
    "input":"0x603880600c6000396000f300603880600c6000396000f3603880600c6000396000f360",
  }
}
```

ETH_GETTRANSACTIONBYBLOCKHASHANDINDEX

Returns information about a transaction by block hash and transaction index position.

Parameters

1. `DATA`, 32 Bytes – hash of a block.
2. `QUANTITY` – integer of the transaction index position.

```
params: [
   '0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331',
   '0x0' // 0
]
```

反馈
建议

Returns

See eth_getBlockByHash

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getTransactionByBlockHashAndIndex","params":
[0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b, "0x0"],"id":1}'
```

Result see eth_getTransactionByHash

ETH_GETTRANSACTIONBYBLOCKNUMBERANDINDEX

Returns information about a transaction by block number and transaction index position.

Parameters

1. `QUANTITY|TAG` – a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the default block parameter.
2. `QUANTITY` – the transaction index position.

```
params: [
   '0x29c', // 668
   '0x0' // 0
]
```

Returns

See eth_gettransactionbyhash

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getTransactionByBlockNumberAndIndex","params":["0x29c",
"0x0"],"id":1}'
```

Result see eth_getTransactionByHash

ETH_GETTRANSACTIONRECEIPT

Returns the receipt of a transaction by transaction hash.

Note That the receipt is not available for pending transactions.

Parameters

1. `DATA`, 32 Bytes – hash of a transaction

```
params: [
   '0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238'
]
```

Returns

`Object` – A transaction receipt object, or `null` when no receipt was found:

- `transactionHash`: `DATA`, 32 Bytes – hash of the transaction.
- `transactionIndex`: `QUANTITY` – integer of the transactions index position in the block.
- `blockHash`: `DATA`, 32 Bytes – hash of the block where this transaction was in.
- `blockNumber`: `QUANTITY` – block number where this transaction was in.
- `cumulativeGasUsed`: `QUANTITY` – The total amount of gas used when this transaction was executed in the block.
- `gasUsed`: `QUANTITY` – The amount of gas used by this specific transaction alone.
- `contractAddress`: `DATA`, 20 Bytes – The contract address created, if the transaction was a contract creation, otherwise `null`.

- `logs`: `Array` – Array of log objects, which this transaction generated.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getTransactionReceipt","params":
["0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id":1}'

// Result
{
"id":1,
"jsonrpc":"2.0",
"result": {
     transactionHash:
'0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238',
     transactionIndex:  '0x1', // 1
     blockNumber: '0xb', // 11
     blockHash: '0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b',
     cumulativeGasUsed: '0x33bc', // 13244
     gasUsed: '0x4dc', // 1244
     contractAddress: '0xb60e8dd61c5d32be8058bb8eb970870f07233155' // or null, if none
was created
     logs: [{
         // logs as returned by getFilterLogs, etc.
     }, ...]
  }
}
```

ETH_GETUNCLEBYBLOCKHASHANDINDEX

Returns information about a uncle of a block by hash and uncle index position.

Parameters

1. `DATA`, 32 Bytes – hash a block.
2. `QUANTITY` – the uncle's index position.

```
params: [
   '0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b',
```

```
    '0x0' // 0
]
```

Returns

See eth_getBlockByHash

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getUncleByBlockHashAndIndex","params":
["0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b", "0x0"],"id":1}'
```

Result see eth_getBlockByHash

Note: An uncle doesn't contain individual transactions.

ETH_GETUNCLEBYBLOCKNUMBERANDINDEX

Returns information about a uncle of a block by number and uncle index position.

Parameters

1. `QUANTITY|TAG` – a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the default block parameter.
2. `QUANTITY` – the uncle's index position.

```
params: [
   '0x29c', // 668
   '0x0' // 0
]
```

反馈
建议

Returns

See eth_getBlockByHash

Note: An uncle doesn't contain individual transactions.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getUncleByBlockNumberAndIndex","params":["0x29c",
"0x0"],"id":1}'
```

Result see eth_getBlockByHash

ETH_GETCOMPILERS

Returns a list of available compilers in the client.

Parameters

none

Returns

`Array` – Array of available compilers.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getCompilers","params":[],"id":1}'

// Result
{
  "id":1,
```

```
  "jsonrpc": "2.0",
  "result": ["solidity", "lll", "serpent"]
}
```

ETH_COMPILESOLIDITY

Returns compiled solidity code.

Parameters

1. `String` – The source code.

```
params: [
   "contract test { function multiply(uint a) returns(uint d) {   return a * 7;   } }",
]
```

Returns

`DATA` – The compiled source code.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_compileSolidity","params":["contract
test { function multiply(uint a) returns(uint d) {   return a * 7;   } }"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": {
      "code":
"0x605880600c6000396000f3006000357c010000000000000000000000000000000000000000000000000000
0000090048063c6888fa114602e57005b603d60048035906020015060047565b8060005260206000f35b60006
00782029050605356 5b91905056",
      "info": {
        "source": "contract test {\n   function multiply(uint a) constant returns(uint
```

```
d) {\n       return a * 7;\n   }\n}\n",
        "language": "Solidity",
        "languageVersion": "0",
        "compilerVersion": "0.9.19",
        "abiDefinition": [
          {
            "constant": true,
            "inputs": [
              {
                "name": "a",
                "type": "uint256"
              }
            ],
            "name": "multiply",
            "outputs": [
              {
                "name": "d",
                "type": "uint256"
              }
            ],
            "type": "function"
          }
        ],
        "userDoc": {
          "methods": {}
        },
        "developerDoc": {
          "methods": {}
        }
      }

}
```

ETH_COMPILELLL

Returns compiled LLL code.

Parameters

1. `String` – The source code.

```
params: [
   "(returnlll (suicide (caller)))",
]
```

Returns

`DATA` – The compiled source code.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_compileLLL","params":["(returnlll
(suicide (caller)))"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b60216004356
02b565b8060005260206000f35b600081600702905091905056" // the compiled source code
}
```

ETH_COMPILESERPENT

Returns compiled serpent code.

Parameters

1. `String` – The source code.

```
params: [
   "/* some serpent */",
]
```

Returns

`DATA` – The compiled source code.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_compileSerpent","params":["/* some
serpent */"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b60216004356
02b565b8060005260206000f35b600081600702905091905056" // the compiled source code
}
```

ETH_NEWFILTER

Creates a filter object, based on filter options, to notify when the state changes (logs). To check if the state has changed, call eth_getFilterChanges.

A note on specifying topic filters:

Topics are order-dependent. A transaction with a log with topics [A, B] will be matched by the following topic filters:

- `[]` "anything"
- `[A]` "A in first position (and anything after)"
- `[null, B]` "anything in first position AND B in second position (and anything after)"
- `[A, B]` "A in first position AND B in second position (and anything after)"
- `[[A, B], [A, B]]` "(A OR B) in first position AND (A OR B) in second position (and anything after)"

Parameters

1. `Object` – The filter options:

- `fromBlock`: `QUANTITY|TAG` – (optional, default: `"latest"`) Integer block number, or `"latest"` for the last mined block or `"pending"`, `"earliest"` for not yet mined transactions.
- `toBlock`: `QUANTITY|TAG` – (optional, default: `"latest"`) Integer block number, or `"latest"` for the last mined block or `"pending"`, `"earliest"` for not yet mined transactions.
- `address`: `DATA|Array`, 20 Bytes – (optional) Contract address or a list of addresses from which logs should originate.
- `topics`: `Array of DATA`, – (optional) Array of 32 Bytes `DATA` topics. Topics are order-dependent. Each topic can also be an array of DATA with “or” options.

```
params: [{
  "fromBlock": "0x1",
  "toBlock": "0x2",
  "address": "0x8888f1f195afa192cfee860698584c030f4c9db1",
  "topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b", null,
[0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b,
0x000000000000000000000000aff3454fce5edbc8cca8697c15331677e6ebccc]]
}]
```

Returns

`QUANTITY` – A filter id.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_newFilter","params":[{"topics":
["0x12341234"]}],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

反馈
建议

ETH_NEWBLOCKFILTER

Creates a filter in the node, to notify when a new block arrives. To check if the state has changed, call eth_getFilterChanges.

Parameters

None

Returns

`QUANTITY` – A filter id.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_newBlockFilter","params":
[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":  "2.0",
  "result": "0x1" // 1
}
```

ETH_NEWPENDINGTRANSACTIONFILTER

Creates a filter in the node, to notify when new pending transactions arrive. To check if the state has changed, call eth_getFilterChanges.

Parameters

None

Returns

`QUANTITY` – A filter id.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_newPendingTransactionFilter","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":  "2.0",
  "result": "0x1" // 1
}
```

ETH_UNINSTALLFILTER

Uninstalls a filter with given id. Should always be called when watch is no longer needed. Additonally Filters timeout when they aren't requested with eth_getFilterChanges for a period of time.

Parameters

1. `QUANTITY` – The filter id.

```
params: [
  "0xb" // 11
]
```

Returns

`Boolean` – `true` if the filter was successfully uninstalled, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_uninstallFilter","params":
["0xb"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": true
}
```

ETH_GETFILTERCHANGES

Polling method for a filter, which returns an array of logs which occurred since last poll.

Parameters

1. `QUANTITY` – the filter id.

```
params: [
  "0x16" // 22
]
```

Returns

`Array` – Array of log objects, or an empty array if nothing has changed since last poll.

- For filters created with `eth_newBlockFilter` the return are block hashes (`DATA` , 32 Bytes), e.g. `["0x3454645634534..."]` .
- For filters created with `eth_newPendingTransactionFilter` the return are transaction hashes (`DATA` , 32 Bytes), e.g. `["0x6345343454645..."]` .
- For filters created with `eth_newFilter` logs are objects with following params:
  - `removed` : `TAG` – `true` when the log was removed, due to a chain reorganization. `false` if its a valid log.
  - `logIndex` : `QUANTITY` – integer of the log index position in the block. `null` when its pending log.

- `transactionIndex`: `QUANTITY` – integer of the transactions index position log was created from. `null` when its pending log.
- `transactionHash`: `DATA`, 32 Bytes – hash of the transactions this log was created from. `null` when its pending log.
- `blockHash`: `DATA`, 32 Bytes – hash of the block where this log was in. `null` when its pending. `null` when its pending log.
- `blockNumber`: `QUANTITY` – the block number where this log was in. `null` when its pending. `null` when its pending log.
- `address`: `DATA`, 20 Bytes – address from which this log originated.
- `data`: `DATA` – contains one or more 32 Bytes non-indexed arguments of the log.
- `topics`: `Array of DATA` – Array of 0 to 4 32 Bytes `DATA` of indexed log arguments. (In *solidity*: The first topic is the *hash* of the signature of the event (e.g. `Deposit(address,bytes32,uint256)`), except you declared the event with the `anonymous` specifier.)

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getFilterChanges","params":
["0x16"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": [{
    "logIndex": "0x1", // 1
    "blockNumber":"0x1b4" // 436
    "blockHash": "0x8216c5785ac562ff41e2dcfdf5785ac562ff41e2dcfdf829c5a142f1fccd7d",
    "transactionHash":
"0xdf829c5a142f1fccd7d8216c5785ac562ff41e2dcfdf5785ac562ff41e2dcf",
    "transactionIndex": "0x0", // 0
    "address": "0x16c5785ac562ff41e2dcfdf829c5a142f1fccd7d",
    "data":"0x000000000000000000000000000000000000000000000000000000000000000",
    "topics": ["0x59ebeb90bc63057b6515673c3ecf9438e5058bca0f92585014eced636878c9a5"]
    },{
      ...
    }]
}
```


ETH_GETFILTERLOGS

Returns an array of all logs matching filter with given id.

Parameters

1. QUANTITY – The filter id.

```
params: [
  "0x16" // 22
]
```

Returns

See eth_getFilterChanges

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getFilterLogs","params":
["0x16"],"id":74}'
```

Result see eth_getFilterChanges

ETH_GETLOGS

Returns an array of all logs matching a given filter object.

Parameters

1. Object – the filter object, see eth_newFilter parameters.

```
params: [{
  "topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b"]
```

```
}]
```

Returns

See eth_getFilterChanges

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getLogs","params":[{"topics":
["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b"]}],"id":74}'
```

Result see eth_getFilterChanges

ETH_GETWORK

Returns the hash of the current block, the seedHash, and the boundary condition to be met ("target").

Parameters

none

Returns

`Array` – Array with the following properties:

1. `DATA` , 32 Bytes – current block header pow-hash
2. `DATA` , 32 Bytes – the seed hash used for the DAG.
3. `DATA` , 32 Bytes – the boundary condition ("target"), 2^256 / difficulty.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getWork","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": [
      "0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
      "0x5EED0000000000000000000000000005EED0000000000000000000000000000",
      "0xd1ff1c01710000000000000000000000d1ff1c0171000000000000000000000000"
    ]
}
```

ETH_SUBMITWORK

Used for submitting a proof-of-work solution.

Parameters

1. `DATA`, 8 Bytes – The nonce found (64 bits)
2. `DATA`, 32 Bytes – The header's pow-hash (256 bits)
3. `DATA`, 32 Bytes – The mix digest (256 bits)

```
params: [
  "0x0000000000000001",
  "0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
  "0xD1FE5700000000000000000000000000D1FE5700000000000000000000000000"
]
```

Returns

`Boolean` – returns `true` if the provided solution is valid, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0", "method":"eth_submitWork", "params":
["0x0000000000000001",
"0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
"0xD1GE5700000000000000000000000000D1GE5700000000000000000000000000"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

ETH_SUBMITHASHRATE

Used for submitting mining hashrate.

Parameters

1. `Hashrate` , a hexadecimal string representation (32 bytes) of the hash rate
2. `ID` , String – A random hexadecimal(32 bytes) ID identifying the client

```
params: [
  "0x0000000000000000000000000000000000000000000000000000000000500000",
  "0x59daa26581d0acd1fce254fb7e85952f4c09d0915afd33d3886cd914bc7d283c"
]
```

Returns

`Boolean` – returns `true` if submitting went through succesfully and `false` otherwise.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0", "method":"eth_submitHashrate", "params":
```

```
["0x0000000000000000000000000000000000000000000000000000000000500000",
"0x59daa26581d0acd1fce254fb7e85952f4c09d0915afd33d3886cd914bc7d283c"],"id":73}'

// Result
{
  "id":73,
  "jsonrpc":"2.0",
  "result": true
}
```

DB_PUTSTRING

Stores a string in the local database.

Note this function is deprecated and will be removed in the future.

Parameters

1. `String` – Database name.
2. `String` – Key name.
3. `String` – String to store.

```
params: [
  "testDB",
  "myKey",
  "myString"
]
```

Returns

`Boolean` – returns `true` if the value was stored, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"db_putString","params":
["testDB","myKey","myString"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

DB_GETSTRING

Returns string from the local database.

Note this function is deprecated and will be removed in the future.

Parameters

1. `String` – Database name.
2. `String` – Key name.

```
params: [
  "testDB",
  "myKey",
]
```

Returns

`String` – The previously stored string.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"db_getString","params":
["testDB","myKey"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": "myString"
}
```

DB_PUTHEX

Stores binary data in the local database.

Note this function is deprecated and will be removed in the future.

Parameters

1. `String` – Database name.
2. `String` – Key name.
3. `DATA` – The data to store.

```
params: [
  "testDB",
  "myKey",
  "0x68656c6c6f20776f726c64"
]
```

Returns

`Boolean` – returns `true` if the value was stored, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"db_putHex","params":
["testDB","myKey","0x68656c6c6f20776f726c64"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

DB_GETHEX

Returns binary data from the local database.

Note this function is deprecated and will be removed in the future.

Parameters

1. `String` – Database name.
2. `String` – Key name.

```
params: [
  "testDB",
  "myKey",
]
```

Returns

`DATA` – The previously stored data.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"db_getHex","params":
["testDB","myKey"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": "0x68656c6c6f20776f726c64"
}
```

SHH_VERSION

Returns the current whisper protocol version.

Parameters

none

Returns

`String` – The current whisper protocol version

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_version","params":[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc": "2.0",
  "result": "2"
}
```

SHH_POST

Sends a whisper message.

Parameters

1. `Object` – The whisper post object:
   - `from`: `DATA`, 60 Bytes – (optional) The identity of the sender.
   - `to`: `DATA`, 60 Bytes – (optional) The identity of the receiver. When present whisper will encrypt the message so that only the receiver can decrypt it.
   - `topics`: `Array of DATA` – Array of `DATA` topics, for the receiver to identify messages.
   - `payload`: `DATA` – The payload of the message.
   - `priority`: `QUANTITY` – The integer of the priority in a rang from ... (?).
   - `ttl`: `QUANTITY` – integer of the time to live in seconds.

```
params: [{
  from:
"0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d44
2460f2998cd41858798ddfd4d661997d3940272b717b1",
  to:
"0x3e245533f97284d442460f2998cd41858798ddf04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9f
a872f50cb214e08ebf61a0d4d661997d3940272b717b1",
  topics: ["0x776869737065722d636861742d636c69656e74",
"0x4d5a695276454c39425154466b61693532"],
  payload: "0x7b2274797065223a226d6",
  priority: "0x64",
  ttl: "0x64",
}]
```

Returns

`Boolean` – returns `true` if the message was send, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_post","params":
[{"from":"0xc931d93e97ab07fe42d923478ba2465f2..","topics":
["0x68656c6c6f20776f726c64"],"payload":"0x68656c6c6f20776f726c64","ttl":0x64,"priority":
```

```
0x64}],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

SHH_NEWIDENTITY

Creates new whisper identity in the client.

Parameters

none

Returns

`DATA`, 60 Bytes – the address of the new identiy.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_newIdentity","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0xc931d93e97ab07fe42d923478ba2465f283f440fd6cabea4dd7a2c807108f651b7135d1d6ca9007d5b68a
a497e4619ac10aa3b27726e1863c1fd9b570d99bbaf"
}
```


SHH_HASIDENTITY

Checks if the client hold the private keys for a given identity.

Parameters

1. `DATA` , 60 Bytes – The identity address to check.

```
params: [

"0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d44
2460f2998cd41858798ddfd4d661997d3940272b717b1"
]
```

Returns

`Boolean` – returns `true` if the client holds the privatekey for that identity, otherwise `false` .

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_hasIdentity","params":
["0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d4
42460f2998cd41858798ddfd4d661997d3940272b717b1"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": true
}
```

SHH_NEWGROUP

(?)

Parameters

none

Returns

DATA , 60 Bytes – the address of the new group. (?)

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_newIdentity","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0xc65f283f440fd6cabea4dd7a2c807108f651b7135d1d6ca90931d93e97ab07fe42d923478ba2407d5b68a
a497e4619ac10aa3b27726e1863c1fd9b570d99bbaf"
}
```

SHH_ADDTOGROUP

(?)

Parameters

1. DATA , 60 Bytes – The identity address to add to a group (?).

```
params: [

"0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d44
2460f2998cd41858798ddfd4d661997d3940272b717b1"
]
```


Returns

`Boolean` – returns `true` if the identity was successfully added to the group, otherwise `false` (?).

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_hasIdentity","params":
["0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d4
42460f2998cd41858798ddfd4d661997d3940272b717b1"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": true
}
```

SHH_NEWFILTER

Creates filter to notify, when client receives whisper message matching the filter options.

Parameters

1. `Object` – The filter options:
   - `to`: `DATA`, 60 Bytes – (optional) Identity of the receiver. *When present it will try to decrypt any incoming message if the client holds the private key to this identity.*
   - `topics`: `Array of DATA` – Array of `DATA` topics which the incoming message's topics should match. You can use the following combinations:
     - `[A, B] = A && B`
     - `[A, [B, C]] = A && (B || C)`
     - `[null, A, B] = ANYTHING && A && B` `null` works as a wildcard

```
params: [{
   "topics": ['0x12341234bf4b564f'],
   "to":
"0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d44
2460f2998cd41858798ddfd4d661997d3940272b717b1"
}]
```

Returns

`QUANTITY` – The newly created filter.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_newFilter","params":[{"topics":
['0x12341234bf4b564f'],"to": "0x2341234bf4b2341234bf4b564f..."}],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": "0x7" // 7
}
```

SHH_UNINSTALLFILTER

Uninstalls a filter with given id. Should always be called when watch is no longer needed. Additonally Filters timeout when they aren't requested with shh_getFilterChanges for a period of time.

Parameters

1. `QUANTITY` – The filter id.

```
params: [
  "0x7" // 7
]
```

Returns

`Boolean` – `true` if the filter was successfully uninstalled, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_uninstallFilter","params":
["0x7"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

SHH_GETFILTERCHANGES

Polling method for whisper filters. Returns new messages since the last call of this method.

Note calling the shh_getMessages method, will reset the buffer for this method, so that you won't receive duplicate messages.

Parameters

1. `QUANTITY` – The filter id.

```
params: [
  "0x7" // 7
]
```

Returns

`Array` – Array of messages received since last poll:

- `hash`: `DATA`, 32 Bytes (?) – The hash of the message.
- `from`: `DATA`, 60 Bytes – The sender of the message, if a sender was specified.
- `to`: `DATA`, 60 Bytes – The receiver of the message, if a receiver was specified.
- `expiry`: `QUANTITY` – Integer of the time in seconds when this message should expire (?).

- `ttl` : `QUANTITY` – Integer of the time the message should float in the system in seconds (?).
- `sent` : `QUANTITY` – Integer of the unix timestamp when the message was sent.
- `topics` : `Array of DATA` – Array of `DATA` topics the message contained.
- `payload` : `DATA` – The payload of the message.
- `workProved` : `QUANTITY` – Integer of the work this message required before it was send (?).

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_getFilterChanges","params":
["0x7"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": [{
    "hash": "0x33eb2da77bf3527e28f8bf493650b1879b08c4f2a362beae4ba2f71bafcd91f9",
    "from": "0x3ec052fc33..",
    "to": "0x87gdf76g8d7fgdfg...",
    "expiry": "0x54caa50a", // 1422566666
    "sent": "0x54ca9ea2", // 1422565026
    "ttl": "0x64" // 100
    "topics": ["0x6578616d"],
    "payload": "0x7b2274797065223a226d657373616765222c2263686...",
    "workProved": "0x0"
    }]
}
```

SHH_GETMESSAGES

Get all messages matching a filter. Unlike `shh_getFilterChanges` this returns all messages.

Parameters

1. `QUANTITY` – The filter id.

```
params: [
  "0x7" // 7
```

```
]
```

Returns

See shh_getFilterChanges

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_getMessages","params":
["0x7"],"id":73}'
```

Result see shh_getFilterChanges

2017年1月20日 / 以太坊-区块链 / 留下评论

以太坊命令行选项

```
$ geth help
名字:
   geth - go语言班的命令行接口

   Copyright 2013-2016 The go-ethereum Authors

使用方法:
   geth [选项] 命令 [命令选项] [参数...]

版本:
   1.5.5-非稳定版

命令:
   init       引导并初始化一个新的创世区块
   import     导入一个区块链文件
   export     导出区块链到一个文件
   upgradedb  升级区块链数据库
   removedb   删除区块链和状态数据库
   dump       从存储中备份一个指定区块
   monitor    监控和可视化节点指标
```

```
   account     管理账户
   wallet      管理以太坊预售的钱包
   console     开启一个JavaScript环境的交互窗口
   attach      开启一个JavaScript环境的交互窗口 (链接一个节点)
   js          执行指定的js文件
   makedag     Generate ethash DAG (for testing)
   version     打印版本号
   license     显示许可信息
   help, h     显示命令的列表，或一个命令的帮助文档

以太坊选项:
  --datadir "/home/tron/.ethereum"  数据库和key文件的存储目录
  --keystore                        key文件的目录 (默认 = 在datadir内)
  --networkid value                 网络ID(整数, 0=Olympic (废弃), 1=Frontier, 2=Morden
(废弃), 3=Ropsten) (默认: 1)
  --olympic                         Olympic network: pre-configured pre-release test
network
  --testnet                         Ropsten network: 预先配置的测试网络
  --dev                             开发模式: 预先配置的私网和一些debug标识
  --identity value                  自定义节点名字
  --fast                            开启快速同步
  --light                           开启轻型客户端模式
  --lightserv value                 允许服务LES（轻型以太坊协议）请求的最大时间的百分比
(0-90) (默认: 0)
  --lightpeers value                LES的客户端的伙伴的最大数量(默认: 20)
  --lightkdf                        Reduce key-derivation RAM & CPU usage at some
expense of KDF strength

性能调优的选项:
  --cache value            内存缓存(最小16MB) (默认: 128)
  --trie-cache-gens value  内存中保留的trie节点数的数量 (默认: 120)


账户选项:
  --unlock value    要解锁的账户列表，用逗号分隔
  --password value  密码文件

API 和 控制台选项:
  --rpc                    开启HTTP-RPC 服务
  --rpcaddr value          HTTP-RPC 服务的监听地址(默认: "localhost")
  --rpcport value          HTTP-RPC 服务的监听端口(默认: 8545)
  --rpcapi value           通过 HTTP-RPC 提供的API(默认: "eth,net,web3")
  --ws                     开启 WS-RPC 服务
  --wsaddr value           WS-RPC 服务地址(默认: "localhost")
  --wsport value           WS-RPC 服务端口 (默认: 8546)
  --wsapi value            WS-RPC 提供的API(默认: "eth,net,web3")
  --wsorigins value        接受websocket请求的源地址
  --ipcdisable             关闭 IPC-RPC 服务
  --ipcapi value           IPC-RPC 提供的API(默认:
"admin,debug,eth,miner,net,personal,shh,txpool,web3")
  --ipcpath "geth.ipc"   datadir目录中的IPC socket/pipe 文件名(显示的路径)
  --rpccorsdomain value  逗号分隔的接受跨域请求的域名的列表(浏览器强制的)
  --jspath loadScript    加载JavaScript文件的跟路径 (默认: ".")
```

```
  --exec value           执行JavaScript语句(只在 console/attach 中有效)
  --preload value        逗号分隔的预加载到控制台的JavaScript文件的列表

网络选项:
  --bootnodes value     Comma separated enode URLs for P2P discovery bootstrap
  --port value          网络监听端口(默认: 30303)
  --maxpeers value      伙伴的最大数量(网络不可用时是 0) (默认: 25)
  --maxpendpeers value  Maximum number of pending connection attempts (defaults used if
set to 0) (default: 0)
  --nat value           NAT 端口映射结构(any|none|upnp|pmp|extip:<IP>) (默认: "any")
  --nodiscover          关闭伙伴发现机制 (手动添加伙伴)
  --v5disc              Enables the experimental RLPx V5 (Topic Discovery) mechanism
  --nodekey value       P2P 节点 key 文件
  --nodekeyhex value    P2P 十六进制的节点key (测试用)

挖矿选项:
  --mine                开启挖矿
  --minerthreads value    CPU线程数(默认: 8)
  --autodag             开启自动 DAG 再生
  --etherbase value       成功挖到区块的地址 (默认 = 第一个被创建的账户) (默认: "0")
  --targetgaslimit value  Target gas limit sets the artificial target gas floor for the
blocks to mine (default: "4712388")
  --gasprice value        Minimal gas price to accept for mining a transactions
(default: "20000000000")
  --extradata value       矿工设置的区块的额外数据 (默认=客户端版本)

GAS PRICE ORACLE OPTIONS:
  --gpomin value       建议的gas价格的下限(默认: "20000000000")
  --gpomax value       建议的gas价格的上限(默认: "500000000000")
  --gpofull value      Full block threshold for gas price calculation (%) (default: 80)
  --gpobasedown value  Suggested gas price base step down ratio (1/1000) (default: 10)
  --gpobaseup value    Suggested gas price base step up ratio (1/1000) (default: 100)
  --gpobasecf value    Suggested gas price base correction factor (%) (default: 110)

虚拟机选项:
  --jitvm           开启JIT VM
  --forcejit        强制JIT VM 优先
  --jitcache value  Amount of cached JIT VM programs (默认: 64)

日志和调试选项:
  --ethstats value          报告一个 ethstats 服务的URL(节点名字:secret@host:port)
  --metrics                 开启指标收集和报告
  --fakepow                 关闭工作量证明的验证
  --verbosity value         日志级别: 0=silent, 1=error, 2=warn, 3=info, 4=core,
5=debug, 6=detail (默认: 3)
  --vmodule value           Per-module verbosity: comma-separated list of <pattern>=
<level> (e.g. eth/*=6,p2p=5)
  --backtrace value         在一个指定的日志语句上请求一个堆栈树 (e.g. "block.go:271")
(默认: :0)
  --pprof                   开始pprof HTTP 服务
  --pprofaddr value         pprof HTTP 服务监听地址(默认: "127.0.0.1")
  --pprofport value         pprof HTTP 服务监听端口(默认: 6060)
  --memprofilerate value    Turn on memory profiling with the given rate (default:
```

```
524288)
  --blockprofilerate value  使用给定的速率开启区块分析(默认: 0)
  --cpuprofile value        把CPU配置信息写入到指定的文件
  --trace value             把执行的堆栈写入到一个给定的文件

实验中的选项:
  --shh       开启 Whisper（即时聊天协议）
  --natspec   开启 NatSpec 确认通知

其他选项:
  --solc value          Solidity 编译器 (默认: "solc")
  --netrestrict value   Restricts network communication to the given IP networks (CIDR
masks)
  --help, -h            显示帮助
```

2017年1月14日 / 以太坊-区块链 / 留下评论

web3.js 参考手册

- web3
  - version
    - api
    - node/getNode
    - network/getNetwork
    - ethereum/getEthereum
    - whisper/getWhisper
  - isConnected()
  - setProvider(provider)
  - currentProvider
  - reset()
  - sha3(string, options)
  - toHex(stringOrNumber)
  - toAscii(hexString)
  - fromAscii(textString, [padding])
  - toDecimal(hexString)
  - fromDecimal(number)
  - fromWei(numberStringOrBigNumber, unit)
  - toWei(numberStringOrBigNumber, unit)
  - toBigNumber(numberOrHexString)
  - isAddress(hexString)

- net
  - listening/getListening
  - peerCount/getPeerCount
- eth
  - defaultAccount
  - defaultBlock
  - syncing/getSyncing
  - isSyncing
  - coinbase/getCoinbase
  - hashrate/getHashrate
  - gasPrice/getGasPrice
  - accounts/getAccounts
  - mining/getMining
  - blockNumber/getBlockNumber
  - register(hexString) (Not implemented yet)
  - unRegister(hexString) (Not implemented yet)
  - getBalance(address)
  - getStorageAt(address, position)
  - getCode(address)
  - getBlock(hash/number)
  - getBlockTransactionCount(hash/number)
  - getUncle(hash/number)
  - getBlockUncleCount(hash/number)
  - getTransaction(hash)
  - getTransactionFromBlock(hashOrNumber, indexNumber)
  - getTransactionReceipt(hash)
  - getTransactionCount(address)
  - sendTransaction(object)
  - sendRawTransaction(object)
  - sign(object)
  - call(object)
  - estimateGas(object)
  - filter(array (, options))
    - watch(callback)
    - stopWatching(callback)
    - get()
  - contract(abiArray)
  - contract.myMethod()
  - contract.myEvent()
  - contract.allEvents()

- getCompilers()
- compile.lll(string)
- compile.solidity(string)
- compile.serpent(string)
- namereg
- sendIBANTransaction
- iban
  - fromAddress
  - fromBban
  - createIndirect
  - isValid
  - isDirect
  - isIndirect
  - checksum
  - institution
  - client
  - address
  - toString
- db
  - putString(name, key, value)
  - getString(- var rXArray = stringToNumberArray(form.rX.value); name, key)
- putHex(name, key, value)
- getHex(name, key)
- shh
  - post(postObject)
  - newIdentity()
  - hasIdentity(hexString)
  - newGroup(_id, _who)
  - addToGroup(_id, _who)
  - filter(object/string)
    - watch(callback)
    - stopWatching(callback)
    - get(callback)

用法

WEB3

这 web3 对象提供了所有的方法.

Example

```
var Web3 = require('web3');
// 使用 HTTP provider创建一个web3实例.
// 注意 在mist浏览器中 web3 对象可能已经存在了, 所以在实例化前，最好先检查一下它是否已存在。
var web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
```

WEB3.VERSION.API

```
web3.version.api
```

Returns

String – 以太坊JS的API版本.

Example

```
var version = web3.version.api;
console.log(version); // "0.2.0"
```

WEB3.VERSION.NODE

```
web3.version.node
// or async
web3.version.getNode(callback(error, result){ ... })
```

Returns

String – 客户端/节点 版本号.

Example

```
var version = web3.version.node;
console.log(version); // "Mist/v0.9.3/darwin/go1.4.1"
```

WEB3.VERSION.NETWORK

```
web3.version.network
// or async
web3.version.getNetwork(callback(error, result){ ... })
```

Returns

String – 网络协议版本号.

Example

```
var version = web3.version.network;
console.log(version); // 54
```

WEB3.VERSION.ETHEREUM

```
web3.version.ethereum
// or async
web3.version.getEthereum(callback(error, result){ ... })
```

Returns

String – 以太坊协议版本号.

Example

```
var version = web3.version.ethereum;
console.log(version); // 60
```

WEB3.VERSION.WHISPER

```
web3.version.whisper
// or async
web3.version.getWhisper(callback(error, result){ ... })
```

Returns

String – 通信协议版本号.

Example

```
var version = web3.version.whisper;
console.log(version); // 20
```

WEB3.ISCONNECTED

反馈
建议

```
web3.isConnected()
```

判断是否连接着一个已存在的节点

Parameters

none

Returns

Boolean

Example

```
if(!web3.isConnected()) {

   // 显示一个信息，告诉用户去开启一个节点

} else {

   // 实现业务
}
```

WEB3.SETPROVIDER

```
web3.setProvider(provider)
```

设置提供者.

Parameters

none

Returns

undefined

Example

```
web3.setProvider(new web3.providers.HttpProvider('http://localhost:8545')); //cpp/AZ的默认端口8080, go/mist的默认端口8545
```

WEB3.CURRENTPROVIDER

```
web3.currentProvider
```

得到当前设置的提供者.

Returns

Object – 设置的提供者 or null;

Example

```
// 检查是否是mist等. 已经设置了一个提供者
if(!web3.currentProvider)
    web3.setProvider(new web3.providers.HttpProvider("http://localhost:8545"));
```

WEB3.RESET

```
web3.reset(keepIsSyncing)
```


重置web3的状态.除了管理者外，重置一切.卸载所有的过滤器..

Parameters

1. Boolean – 如果 true 它会卸载所有的过滤器, 但是会保持 web3.eth.isSyncing() 的投票

Returns

undefined

Example

```
web3.reset();
```

WEB3.SHA3

```
web3.sha3(string, options)
```

Parameters

1. String – 要被hash的字符串， 使用 Keccak-256 SHA3 算法
2. Object – 把编码设置成 hex .如果hash后的字符串使用hex编码. 那么头部的 0x 这2个字符串会被自动忽略.

Returns

String – 经过Keccak-256 SHA3 加密后的数据.

Example

```
var hash = web3.sha3("Some string to be hashed");
console.log(hash); //
```

```
"0xed973b234cf2238052c9ac87072c71bcf33abc1bbd721018e0cca448ef79b379"

var hashOfHash = web3.sha3(hash, {encoding: 'hex'});
console.log(hashOfHash); //
"0x85dd39c91a64167ba20732b228251e67caed1462d4bcf036af88dc6856d0fdcc"
```

WEB3.TOHEX

```
web3.toHex(mixed);
```

把任何一个值转成十六进制的.

Parameters

1. String|Number|Object|Array|BigNumber – 需要被转换成十六进制的值.如果是一个对象或数组，那么会先使用 JSON.stringify 把它变成json字符串.如果是个大数字，则返回这大数字的十六进制的值.

Returns

String – 十六位进制的字符串.

Example

```
var str = web3.toHex({test: 'test'});
console.log(str); // '0x7b2274657374223a2274657374227d'
```

WEB3.TOASCII

```
web3.toAscii(hexString);
```

把一个十六位进制的字符串转换成ASCII 字符串.

Parameters

1. String – 十六位进制的字符串

Returns

String – 一个ASCII字符串

Example

```
var str =
web3.toAscii("0x657468657265756d0000000000000000000000000000000000000000000000000000000000000000");
console.log(str); // "ethereum"
```

WEB3.FROMASCII

```
web3.fromAscii(string [, padding]);
```

把ASCII 字符串转成十六进制字符串.

Parameters

1. String – 一个ASCII 字符串.
2. Number – 返回的十六位进制字符串的长度.

Returns

String – 十六位进制字符串.

Example

```
var str = web3.fromAscii('ethereum');
console.log(str); // "0x657468657265756d"

var str2 = web3.fromAscii('ethereum', 32);
console.log(str2); //
"0x657468657265756d000000000000000000000000000000000000000000000000"
```

WEB3.TODECIMAL

```
web3.toDecimal(hexString);
```

把十六位进制的字符串转成和数字.

Parameters

1. String – 要被转成数字的十六位进制字符串.

Returns

Number – 数字

Example

```
var number = web3.toDecimal('0x15');
console.log(number); // 21
```

WEB3.FROMDECIMAL

```
web3.fromDecimal(number);
```

把一个数字或数字字符串改成十六进制显示形式的.

Parameters

1. Number|String – 一个要被转换成十六进制字符串的数字.

Returns

String – 十六进制字符串.

Example

```
var value = web3.fromDecimal('21');
console.log(value); // "0x15"
```

WEB3.FROMWEI

```
web3.fromWei(number, unit)
```

把一个单位是wei的数据转换成下面列出的以太坊单位:

- kwei / ada
- mwei / babbage
- gwei / shannon
- szabo
- finney
- ether
- kether / grand / einstein

- mether
- gether
- tether

Parameters

1. Number|String|BigNumber – 一个数字或一个大数字实例
2. String – 以太币单位字符串

Returns

String|BigNumber – 一个数字字符串,或一个大数字实例,取决于传入 number 的参数.

Example

```
var value = web3.fromWei('21000000000000', 'finney');
console.log(value); // "0.021"
```

WEB3.TOWEI

```
web3.toWei(number, unit)
```

把单位转换成 wei. unit可用的参数有:

- kwei / ada
- mwei / babbage
- gwei / shannon
- szabo
- finney
- ether
- kether / grand / einstein
- mether

- gether
- tether

Parameters

1. Number|String|BigNumber – 一个数字或大数字实例.
2. String – 以上列出的单位.

Returns

String|BigNumber – 一个数字字符串,或一个大数字实例,取决于传入 number 的参数.

Example

```
var value = web3.toWei('1', 'ether');
console.log(value); // "1000000000000000000"
```

WEB3.TOBIGNUMBER

```
web3.toBigNumber(numberOrHexString);
```

把数字转换成大数字.

Parameters

1. Number|String – 一个数字，数字字符串，或十六位进制的数字.

Returns

BigNumber – 一个大数字实例

Example

```
var value = web3.toBigNumber('200000000000000000000001');
console.log(value); // 大数字的实例
console.log(value.toNumber()); // 2.0000000000000002e+23
console.log(value.toString(10)); // '200000000000000000000001'
```

web3.net

WEB3.NET.LISTENING

```
web3.net.listening
// or async
web3.net.getListening(callback(error, result){ ... })
```

这属性是只读的，获取该节点的网络监听状态.

Returns

Boolean – true 如果客户端正在监听网络链接。不然返回 false.

Example

```
var listening = web3.net.listening;
console.log(listening); // true of false
```

WEB3.NET.PEERCOUNT

```
web3.net.peerCount
// or async
```

```
web3.net.getPeerCount(callback(error, result){ ... })
```

这属性是只读的，返回链接着的伙伴的数量.

Returns

Number – 当前链接到这客户端的伙伴的数量.

Example

```
var peerCount = web3.net.peerCount;
console.log(peerCount); // 4
```

web3.eth

包含以太坊区块链相关的方式.

Example

```
var eth = web3.eth;
```

WEB3.ETH.DEFAULTACCOUNT

```
web3.eth.defaultAccount
```

以下方法中会使用到的默认地址(可选 你可以指定 `from` 属性来覆盖这默认值):

- web3.eth.sendTransaction()

- web3.eth.call()

Values

String , 20 Bytes – 你拥有的任何地址.

默认是 undefined .

Returns

String , 20 Bytes – 当前设置的默认地址.

Example

```
var defaultAccount = web3.eth.defaultAccount;
console.log(defaultAccount); // ''

// set the default account
web3.eth.defaultAccount = '0x8888f1f195afa192cfee860698584c030f4c9db1';
```

WEB3.ETH.DEFAULTBLOCK

```
web3.eth.defaultBlock
```

默认的区块，用于以下几个函数 (可选 你可以传入一个区块链参数来覆盖这参数):

- web3.eth.getBalance()
- web3.eth.getCode()
- web3.eth.getTransactionCount()
- web3.eth.getStorageAt()
- web3.eth.call()
- contract.myMethod.call()
- contract.myMethod.estimateGas()

Values

默认的区块参数值可以是以下的任一个值:

- Number – 一个区块的数字
- String – "earliest",创世区块
- String – "latest",最后的区块(区块链的最前面的一个区块)
- String – "pending",当前开采的区块(包含进行中的交易)

默认是 latest

Returns

Number|String – 这默认区块的数字或字符串

Example

```
var defaultBlock = web3.eth.defaultBlock;
console.log(defaultBlock); // 'latest'

// 设置默认区块
web3.eth.defaultBlock = 231;
```

WEB3.ETH.SYNCING

```
web3.eth.syncing
// 或异步
web3.eth.getSyncing(callback(error, result){ ... })
```

这属性是只读的,当节点正在同步的时候，返回一个同步对象，不然返回 false.

Returns

Object|Boolean – 当节点正在同步的时候，返回一个如下的同步对象，不然 false:

- startingBlock: Number – 同步开始处的区块的数字.
- currentBlock: Number – 当前节点已经同步到区块的数字.
- highestBlock: Number – 预估的要同步到的区块的数字.

Example

```
var sync = web3.eth.syncing;
console.log(sync);
/*
{
   startingBlock: 300,
   currentBlock: 312,
   highestBlock: 512
}
*/
```

WEB3.ETH.ISSYNCING

```
web3.eth.isSyncing(callback);
```

当一个同步开始，更新和停止的时候，方便函数调用 callback .

Returns

Object -一个正在同步的对象和下面的方法:

- syncing.addCallback() :添加另一个回调函数,当节点开始或停止同步的时候会被调用.
- syncing.stopWatching() : 停止同步的回调.

回调返回的值Callback return value

- Boolean – 当同步开始，回调函数会返回 true ，当同步结束，返回 false .
- Object – 当同步进行的时候，会返回正在同步的对象:
  - startingBlock: Number -同步开始处的区块的数字.

- currentBlock : Number – 当前已经同步到的区块的 数字.
- highestBlock : Number – 预期要同步到的区块数字.

Example

```
web3.eth.isSyncing(function(error, sync){
    if(!error) {
        // 停止所有应用的活动
        if(sync === true) {
           // 我们使用 `true`, 所以它会停止所有的过滤器，但不会停止 web3.eth.syncing 投
票
           web3.reset(true);

        // 如果正在同步，则输出正在同步的信息
        } else if(sync) {
           console.log(sync.currentBlock);

        // 执行应用的其他操作
        } else {
            // 运行你的应用的初始化函数
        }
    }
});
```

WEB3.ETH.COINBASE

```
web3.eth.coinbase
// 或 异步
web3.eth.getCoinbase(callback(error, result){ ... })
```

这属性是只读的,返回获得挖矿奖励的 coinbase 地址.

Returns

String – 这客户的coinbase 地址.

Example

```
var coinbase = web3.eth.coinbase;
console.log(coinbase); // "0x407d73d8a49eeb85d32cf465507dd71d507100c1"
```

WEB3.ETH.MINING

```
web3.eth.mining
//或 异步
web3.eth.getMining(callback(error, result){ ... })
```

这属性是只读的，返回客户端节点是否在挖矿.

Returns

Boolean – true 如果客户端是在挖矿,不然 false.

Example

```
var mining = web3.eth.mining;
console.log(mining); // true or false
```

WEB3.ETH.HASHRATE

```
web3.eth.hashrate
// 或异步
web3.eth.getHashrate(callback(error, result){ ... })
```

这属性是只读的，返回节点正在以多少的哈希速率挖矿.

Returns

Number – 每秒的哈希的数值.

Example

```
var hashrate = web3.eth.hashrate;
console.log(hashrate); // 493736
```

WEB3.ETH.GASPRICE

```
web3.eth.gasPrice
// or async
web3.eth.getGasPrice(callback(error, result){ ... })
```

这属性是只读的，返回当前是gas价格.

Returns

BigNumber – 返回当前gas的价格，是一个单位是 wei的大数字实例

Example

```
var gasPrice = web3.eth.gasPrice;
console.log(gasPrice.toString(10)); // "10000000000000"
```

WEB3.ETH.ACCOUNTS

```
web3.eth.accounts
// or async
web3.eth.getAccounts(callback(error, result){ ... })
```

这属性是只读的，返回当前节点控制的账户列表.

Returns

Array – 被当前客户端控制的地址的数组.

Example

```
var accounts = web3.eth.accounts;
console.log(accounts); // ["0x407d73d8a49eeb85d32cf465507dd71d507100c1"]
```

WEB3.ETH.BLOCKNUMBER

```
web3.eth.blockNumber
// 或异步
web3.eth.getBlockNumber(callback(error, result){ ... })
```

这属性是只读的，返回当前区块的数字.

Returns

Number – 当前区块的数字

Example

```
var number = web3.eth.blockNumber;
console.log(number); // 2744
```

WEB3.ETH.REGISTER

```
web3.eth.register(addressHexString [, callback])
```

(已不推荐)把给定的地址纳入到 web3.eth.accounts .这可以让一个没有私钥的账户和一个有私钥的账户发生关联 (e.g., 合约钱包).

Parameters

1. String – 要注册的地址
2. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

?

Example

```
web3.eth.register("0x407d73d8a49eeb85d32cf465507dd71d507100ca")
```

WEB3.ETH.UNREGISTER

```
web3.eth.unRegister(addressHexString [, callback])
```

(已不推荐) 注销一个给定的地址.

Parameters

1. String – 要注销的地址
2. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

?

Example

```
web3.eth.unregister("0x407d73d8a49eeb85d32cf465507dd71d507100ca")
```

WEB3.ETH.GETBALANCE

```
web3.eth.getBalance(addressHexString [, defaultBlock] [, callback])
```

得到一个在给定区块上的地址的余额.

Parameters

1. String – 要获取余额的地址.
2. Number|String – (可选)如果传入这个参数，就不会使用由web3.eth.defaultBlock设置的默认区块 .
3. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

String – 给定地址的当前余额，一个单位是wei的大数字的实例.

Example

```
var balance = web3.eth.getBalance("0x407d73d8a49eeb85d32cf465507dd71d507100c1");
console.log(balance); // 大数字的实例
console.log(balance.toString(10)); // '1000000000000'
console.log(balance.toNumber()); // 1000000000000
```

WEB3.ETH.GETSTORAGEAT

```
web3.eth.getStorageAt(addressHexString, position [, defaultBlock] [, callback])
```

获取存储在一个地址上的指定位置上值.

Parameters

1. String – 地址
2. Number – 索引的位置.
3. Number|String – (可选) 如果传了这参数，则就不会使用由web3.eth.defaultBlock设置的默认区块 .
4. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

String – 存储在指定位置上的值.

Example

```
var state = web3.eth.getStorageAt("0x407d73d8a49eeb85d32cf465507dd71d507100c1", 0);
console.log(state); // "0x03"
```


WEB3.ETH.GETCODE

```
web3.eth.getCode(addressHexString [, defaultBlock] [, callback])
```

获取指定地址上的代码数据.

Parameters

1. String – 要获取代码数据的地址.
2. Number|String – (可选) 如果传了这参数，则就不会使用由web3.eth.defaultBlock设置的默认区块 .
3. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

String – 在给定的地址 addressHexString 上的代码数据。

Example

```
var code = web3.eth.getCode("0xd5677cf67b5aa051bb40496e68ad359eb97cfbf8");
console.log(code); //
"0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060078
202905091905056"
```

WEB3.ETH.GETBLOCK

```
web3.eth.getBlock(blockHashOrBlockNumber [, returnTransactionObjects] [, callback])
```

返回一个使用区块数字或区块哈希来匹配到的区块.

Parameters

1. String|Number – 区块是数字或哈希. 或定义在default block parameter中的字符串 "earliest", "latest" 或 "pending" .
2. Boolean – (可选, 默认 false) 如果 true , 返回一个区块对象，会包含所有的交易, 如果 false 只包含交易的哈希值.
3. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

Object – 区块对象:

- number : Number -区块的数字. null 当区块还没确定.
- hash : String , 32 位- 区块的哈希. null 当区块还没确定.
- parentHash : String , 32 位- 父块的哈希.
- nonce : String , 8 位- 由工作量证明生成的哈希. null 当区块还没确定.
- sha3Uncles : String , 32 位- 区块中叔叔数据的SHA3.
- logsBloom : String , 256 位- 区块的bloom 过滤的日志的. null 当区块还没确定.
- transactionsRoot : String , 32 位-区块的交易trie（树形结构）的根.
- stateRoot : String , 32 位- 区块的trie的根的最终状态.
- miner : String , 20 位- 挖矿奖励的受益人的地址.
- difficulty : BigNumber -整数，区块的难度值.
- totalDifficulty : BigNumber – 整数，区块链到这个区块的整个难度值.
- extraData : String -区块的"extra data" 字段.
- size : Number – 整数，区块的字节长度.
- gasLimit : Number -这区块中 gas的上限.
- gasUsed : Number – 这区块中，所有的交易消耗的总的gas.
- timestamp : Number – 区块被整理出时的unix时间戳.
- transactions : Array – 交易对象的数组,或32位的交易哈希，取决于最后的一个传入的参数.
- uncles : Array – 数组，叔叔的哈希.

Example

```
var info = web3.eth.getBlock(3150);
console.log(info);
/*
{
  "number": 3,
  "hash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "parentHash": "0x2302e1c0b972d00932deb5dab9eb2982f570597d9d42504c05d9c2147eaf9c88",
```

```
  "nonce": "0xfb6e1a62d119228b",
  "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
  "logsBloom":
"0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000",
  "transactionsRoot":
"0x3a1b03875115b79539e5bd33fb00d8f7b7cd61929d5a3c574f507b8acf415bee",
  "stateRoot": "0xf1133199d44695dfa8fd1bcfe424d82854b5cebef75bddd7e40ea94cda515bcb",
  "miner": "0x8888f1f195afa192cfee860698584c030f4c9db1",
  "difficulty": BigNumber,
  "totalDifficulty": BigNumber,
  "size": 616,
  "extraData": "0x",
  "gasLimit": 3141592,
  "gasUsed": 21662,
  "timestamp": 1429287689,
  "transactions": [
    "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b"
  ],
  "uncles": []
}
*/
```

WEB3.ETH.GETBLOCKTRANSACTIONCOUNT

```
web3.eth.getBlockTransactionCount(hashStringOrBlockNumber [, callback])
```

返回给定区块中交易的数量.

Parameters

1. `String|Number` – 区块是数字或哈希. 或定义在default block parameter中的字符串 "earliest", "latest" 或 "pending" .
2. `Function` – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

反馈
建议

Returns

Number – 在给定的区块中的交易的数量.

Example

```
var number =
web3.eth.getBlockTransactionCount("0x407d73d8a49eeb85d32cf465507dd71d507100c1");
console.log(number); // 1
```

WEB3.ETH.GETUNCLE

```
web3.eth.getUncle(blockHashStringOrNumber, uncleNumber [, returnTransactionObjects] [,
callback])
```

返回一个区块的uncle(叔叔？)，通过一个给定的叔叔的索引位置.

Parameters

1. String|Number – 区块的数字或哈希.或定义在default block parameter中的字符串 "earliest", "latest" 或 "pending" .
2. Number – 叔叔的索引位置.
3. Boolean – (默认,默认 false) 如果 true ,返回区块对象，会包含所有的交易数据,如果f false 只包含交易的哈希.
4. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

Object – 返回的uncle（叔叔）对象. 查看 web3.eth.getBlock().

注意: 一个uncle不包含个人的交易.

Example

```
var uncle = web3.eth.getUncle(500, 0);
console.log(uncle); // 看web3.eth.getBlock
```

web3.eth.getTransaction

```
web3.eth.getTransaction(transactionHash [, callback])
```

返回与给定交易哈希值匹配的一个交易.

Parameters

1. String – 交易哈希.
2. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

Object – 一个交易对象

- hash: String, 32 位- 交易的哈希.
- nonce: Number – 在此之前，由发起者发起的交易的数字.
- blockHash: String, 32 位- 这交易所在的区块的哈希值. null 当它处于待定状态.
- blockNumber: Number – 当前交易所在的区块的数字. null 当它处于待定状态.
- transactionIndex: Number -整数，区块中这交易的索引位置. null 当它处于待定状态.
- from: String, 20位- 发起者的地址.
- to: String, 20位- 接收者的地址. null 如果它是合约创建的交易.
- value: BigNumber – 交易的值，单位Wei.
- gasPrice: BigNumber -发起者提供的 gas 价格，单位 Wei.
- gas: Number – 发起者提供的 gas.
- input: String – 和交易一起被发送的数据.

Example

```
var blockNumber = 668;
var indexOfTransaction = 0

var transaction = web3.eth.getTransaction(blockNumber, indexOfTransaction);
console.log(transaction);
/*
{
  "hash": "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
  "nonce": 2,
  "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "blockNumber": 3,
  "transactionIndex": 0,
  "from": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",
  "to": "0x6295ee1b4f6dd65047762f924ecd367c17eabf8f",
  "value": BigNumber,
  "gas": 314159,
  "gasPrice": BigNumber,
  "input": "0x57cb2fc4"
}
*/
```

WEB3.ETH.GETTRANSACTIONFROMBLOCK

```
getTransactionFromBlock(hashStringOrNumber, indexNumber [, callback])
```

基于一个区块的哈希或数字，和交易的索引位置来返回一个交易.

Parameters

1. String – 一个区块的数字或哈希.或定义在default block parameter中的字符串 "earliest", "latest" 或 "pending" .
2. Number – 这交易的索引位置.
3. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

Object – 一个交易对象,查看 web3.eth.getTransaction:

Example

```
var transaction = web3.eth.getTransactionFromBlock('0x4534534534', 2);
console.log(transaction); // 查看 web3.eth.getTransaction
```

WEB3.ETH.GETTRANSACTIONRECEIPT

```
web3.eth.getTransactionReceipt(hashString [, callback])
```

通过交易的哈希值，返回这交易的 收入.

注意 待交易的时候，收入是不可用的.

Parameters

1. String – 这交易哈希.
2. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

Object – 一个交易收入的对象，或 null 没有收入时:

- blockHash: String, 32 位- 交易所在的区块的哈希值.
- blockNumber: Number -这交易所在的区块的区块数.
- transactionHash: String, 32 位- 交易的哈希.
- transactionIndex: Number – 整数，这交易在区块里面的索引序号.
- from: String, 20 位- 发起者的地址.
- to: String, 20 位- 接收者的地址. null 如果是合约创建的交易.
- cumulativeGasUsed: Number – 在区块中，执行这个交易所花去的gas总和.
- gasUsed: Number – 这指定的交易本身消耗的gas 总和.
- contractAddress: String – 20 位- 合约地址, 如果这交易是由一个合约创建的,不然是 null .
- logs: Array – 日志对象的数组,由交易生产的.

Example

```
var receipt =
web3.eth.getTransactionReceipt('0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6
c445836d8b');
console.log(receipt);
{
  "transactionHash":
"0x9fc76417374aa880d4449a1f7f31ec597f00b1f6f3dd2d66f4c9c6c445836d8b",
  "transactionIndex": 0,
  "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a2c4e133e34b46",
  "blockNumber": 3,
  "contractAddress": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b",
  "cumulativeGasUsed": 314159,
  "gasUsed": 30234,
  "logs": [{
        // logs as returned by getFilterLogs, etc.
    }, ...]
}
```

WEB3.ETH.GETTRANSACTIONCOUNT

```
web3.eth.getTransactionCount(addressHexString [, defaultBlock] [, callback])
```

传入一个地址，获取从该地址发起交易的数量.

Parameters

1. String – 要获取交易数量的地址.
2. Number|String – (可选) 如果你传了这个参数，那么就不会使用由web3.eth.defaultBlock设置的默认区块.
3. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

Number – 由给定地址发起的交易数量.

Example

```
var number = web3.eth.getTransactionCount("0x407d73d8a49eeb85d32cf465507dd71d507100c1");
console.log(number); // 1
```

WEB3.ETH.SENDTRANSACTION

```
web3.eth.sendTransaction(transactionObject [, callback])
```

向网络发送一个交易.

Parameters

1. Object – 要被发送的 交易对象:
   - from: String – 发起的账户地址.使用 web3.eth.defaultAccount 属性, 如果没有特别指定的话.
   - to: String – (可选) 消息的目的地地址,如果是合约创建的交易，则留 undefined.
   - value: Number|String|BigNumber – (可选) 这交易的值，单位是 Wei,如果是合约创建的交易也有可能是捐赠.
   - gas: Number|String|BigNumber – (可选, 默认: 90000) 这交易使用的gas的总数 (未使用的gas会被退还).
   - gasPrice: Number|String|BigNumber – (可选, 默认: 待定) 这次交易的gas的价格，单位wei.
   - data: String – (可选)一个 合约的编码的代码.
   - nonce: Number – (可选)，整数. 使用相同的nonce可以覆盖你自己待定的交易.
2. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

String – 32位的交易哈希.

如果这交易是由一个合约创建的，那么使用 web3.eth.getTransactionReceipt() 去获得合约的地址，当这交易已经被开采后.

Example

```
// 编译好的solidity语言的源代码
var code =
"603d80600c6000396000f3007c01000000000000000000000000000000000000000000000000000000006000350463c6888fa18114602d57005b6007600435028060005260206000f3";

web3.eth.sendTransaction({data: code}, function(err, address) {
  if (!err)
    console.log(address); // "0x7f9fade1c0d57a7af66ab4ead7c2eb7b11a91385"
});
```

WEB3.ETH.SENDRAWTRANSACTION

```
web3.eth.sendRawTransaction(signedTransactionData [, callback])
```

发送一个已经签名的交易.例如，可以下面的方式签名:
https://github.com/SilentCicero/ethereumjs-accounts

Parameters

1. String – 以十六进制格式签名的交易数据
2. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

String – 32字节的交易哈希.

如果这交易是一个合约创建的，那使用 web3.eth.getTransactionReceipt() 获取合约的地址, 当这交易被开采后.

Example

```
var Tx = require('ethereumjs-tx');
var privateKey = new
Buffer('e331b6d69882b4cb4ea581d88e0b604039a3de5967688d3dcffdd2270c0fd109', 'hex')

var rawTx = {
  nonce: '0x00',
  gasPrice: '0x09184e72a000',
  gasLimit: '0x2710',
  to: '0x0000000000000000000000000000000000000000',
  value: '0x00',
  data: '0x7f7465737432000000000000000000000000000000000000000000000000000000000000600057'
}

var tx = new Tx(rawTx);
tx.sign(privateKey);

var serializedTx = tx.serialize();

//console.log(serializedTx.toString('hex'));
//0xf889808609184e72a00082271094000000000000000000000000000000000000000080a47f74657374320000000000000000000000000000000000000000000000000000000000600057
23625641fc956967b81d12e23709cead01446075a01ce999b56a8a88504be365442ea61239198e23d1fce7d00fcfc5cd3b44b7215f

web3.eth.sendRawTransaction(serializedTx.toString('hex'), function(err, hash) {
  if (!err)
    console.log(hash); //
"0x7f9fade1c0d57a7af66ab4ead79fade1c0d57a7af66ab4ead7c2c2eb7b11a91385"
});
```

WEB3.ETH.SIGN

```
web3.eth.sign(address, dataToSign, [, callback])
```

使用一个特定的账户签名数据.这账户需要处于解锁状态.

Parameters

1. String – 用于签名的地址.
2. String – 要被签名的数据.

3. Function -(可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

String -签名后的数据.

十六进制前缀后面的值, 相当于ECDSA 算法签名后的值，如下:

```
 r = signature[0:64]
s = signature[64:128]
v = signature[128:130]
```

Note that if you are using `ecrecover`, `v` will be either `"00"` or `"01"`. As a result, in order to use this value, you will have to parse it to an integer and then add `27`. This will result in either a `27` or a `28`.

Example

```
var result = web3.eth.sign("0x135a7de83802408321b74c322f8558db1679ac20",
    "0x9dd2c369a187b4e6b9c402f030e50743e619301ea62aa4c0737d4ef7e10a3d49"); // 第二个参数是 web3.sha3("xyz")
console.log(result); //
"0x30755ed65396facf86c53e6217c52b4daebe72aa4941d89635409de4c9c7f9466d4e9aaec7977f05e923889b33c0d0dd27d7226b6e6f56ce737465c5cfd04be400"
```

WEB3.ETH.CALL

```
web3.eth.call(callObject [, defaultBlock] [, callback])
```

执行一个消息调用交易, 直接在节点的虚拟机里执行, 不会被开采到区块链中.

Parameters

1. Object – 一个交易对象，详情见 web3.eth.sendTransaction, 不同之处在于 from 属性是可选的.
2. Number|String – (可选) 如果你传了这个参数，那么就不会使用由web3.eth.defaultBlock设置的默认区块.
3. Function – (可选) 如果传入一个回调函数，则HTTP请求就会变成异步的.

Returns

String – 返回调用的数据.

Example

```
var result = web3.eth.call({
    to: "0xc4abd0339eb8d57087278718986382264244252f",
    data: "0xc6888fa1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000003"
});
console.log(result); //
"0x0000000000000000000000000000000000000000000000000000000000000015"
```

WEB3.ETH.ESTIMATEGAS

```
web3.eth.estimateGas(callObject [, callback])
```

执行一个消息调用或交易, 直接在节点虚拟机内执行, 不会被开采进区块链中，返回消耗的gas总和.

Parameters

查看web3.eth.sendTransaction, 除了所有的属性都是可选的.

Returns

Number – 这模拟的调用/交易的gas消耗.

Example

```
var result = web3.eth.estimateGas({
    to: "0xc4abd0339eb8d57087278718986382264244252f",
    data: "0xc6888fa1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000003"
});
console.log(result); //
"0x0000000000000000000000000000000000000000000000000000000000000015"
```

WEB3.ETH.FILTER

```
// 可以是 'latest' 或 'pending'
var filter = web3.eth.filter(filterString);
// 或 对象是日志过滤选项
var filter = web3.eth.filter(options);

// 监听变化
filter.watch(function(error, result){
  if (!error)
    console.log(result);
});

// 再添加一个可以立刻开始的监听, 传入一个回调函数:
web3.eth.filter(options, function(error, result){
  if (!error)
    console.log(result);
});
```

Parameters

1. String|Object – 字符串 "latest" 或 "pending" 用来监听最后一个区块，或正在挖的那个区块y.或一个过滤选项的对象，如下：
   - fromBlock: Number|String -可选 区块的数字 (latest 最后一个被挖到的区块， pending 当前正在挖的区块).默认 latest .
   - toBlock: Number|String – 区块的数字 (latest 最后一个被挖到的区块， pending 当前正在挖的区块).默认 latest .
   - address: String – 20个字节，合约地址，或产生日志的地址.
   - topics: Array of Strings – 32个字节的主题数据组成的数组

Returns

Object – 一个包含下面方法的过滤对象:

- filter.get(callback)：返回所有匹配到的日志.
- filter.watch(callback)：Watches for state changes that fit the filter and calls the callback. See this note for details.
- filter.stopWatching()：停止监听和卸载这节点终端.

Watch callback return value

- String -当参数使用的是 "latest" ,返回这区块链的哈希.
- String – 当参数使用的是 "pending" ,返回一个最近的待处理的交易的哈希值.
- Object – 返回一个如下的日志对象:
  - logIndex：Number – 整数，区块中的日志索引的位置. null 没有日志的话.
  - transactionIndex：Number – 整数，创建这日志的交易的索引位置. null 没有日志的话.
  - transactionHash：String,32 字节- 创建这日志的交易的哈希值. null 没有日志的话.
  - blockHash：String,32 字节- 日志所在的区块的哈希. null 没有日志的话.
  - blockNumber：Number – 这日志所在的区块的数字. null 没有日志的话.
  - address：String,32 字节- 创建这日志的地址.
  - data：String – 包含一个或多个32字节的无索引的日志的参数.
  - topics：Array of Strings – 日志参数组成的数组

注意 事件的过滤返回值查看 合约事件

Example

```
var filter = web3.eth.filter('pending');

filter.watch(function (error, log) {
  console.log(log); //  {"address":"0x0000000000000000000000000000000000000000",
"data":"0x0000000000000000000000000000000000000000000000000000000000000000", ...}
});

// 获取所有过去的日志.
var myResults = filter.get(function(error, logs){ ... });

...
```

```
// 停止 和卸载过滤器
filter.stopWatching();
```

WEB3.ETH.CONTRACT

```
web3.eth.contract(abiArray)
```

为 一个solidity 合约创建一个合约对象

Parameters

1. Array – 函数的描述和合约事件组成的 ABI（应用程序二进制接口） 数组

Returns

Object – 一个合约对象:

```
var MyContract = web3.eth.contract(abiArray);

// 通过地址实例化
var contractInstance = MyContract.at([address]);

// 发布新合约
var contractInstance = MyContract.new([contructorParam1] [, contructorParam2], {data:
'0x12345...', from: myAccount, gas: 1000000});

// 获取数据，手动发布合约
var contractData = MyContract.new.getData([contructorParam1] [, contructorParam2],
{data: '0x12345...'});
// contractData = '0x1234564321345600000000023434234'
```

你可以在一个地址上初始化一个已经存在的合约，或使用编译的字节码来发布合约:

```
// 从一个现有的地址上实例化：
var myContractInstance = MyContract.at(myContractAddress);
```

```
// 或 发布一个新合约:

// 使用 Solidity 文件发布合约:
...
const fs = require("fs");
const solc = require('solc')

let source = fs.readFileSync('nameContract.sol', 'utf8');
let compiledContract = solc.compile(source, 1);
let abi = compiledContract.contracts['nameContract'].interface;
let bytecode = compiledContract.contracts['nameContract'].bytecode;
let gasEstimate = web3.eth.estimateGas({data: bytecode});
let MyContract = web3.eth.contract(JSON.parse(abi));

var myContractReturned = MyContract.new(param1, param2, {
   from:mySenderAddress,
   data:bytecode,
   gas:gasEstimate}, function(err, myContract){
    if(!err) {
       // NOTE: The callback will fire twice!
       // Once the contract has the transactionHash property set and once its deployed
on an address.

       // e.g. check tx hash on the first call (transaction send)
       if(!myContract.address) {
           console.log(myContract.transactionHash) // The hash of the transaction, which
deploys the contract

       // check address on the second call (contract deployed)
       } else {
           console.log(myContract.address) // the contract address
       }

       // Note that the returned "myContractReturned" === "myContract",
       // so the returned "myContractReturned" object will also get the address set.
    }
  });

// Deploy contract syncronous: The address will be added as soon as the contract is
mined.
// Additionally you can watch the transaction by using the "transactionHash" property
var myContractInstance = MyContract.new(param1, param2, {data: myContractCode, gas:
300000, from: mySenderAddress});
myContractInstance.transactionHash // The hash of the transaction, which created the
contract
myContractInstance.address // undefined at start, but will be auto-filled later
```


Example

```
// contract abi
var abi = [{
     name: 'myConstantMethod',
     type: 'function',
     constant: true,
     inputs: [{ name: 'a', type: 'string' }],
     outputs: [{name: 'd', type: 'string' }]
}, {
     name: 'myStateChangingMethod',
     type: 'function',
     constant: false,
     inputs: [{ name: 'a', type: 'string' }, { name: 'b', type: 'int' }],
     outputs: []
}, {
     name: 'myEvent',
     type: 'event',
     inputs: [{name: 'a', type: 'int', indexed: true},{name: 'b', type: 'bool', indexed:
false}]
}];

// creation of contract object
var MyContract = web3.eth.contract(abi);

// initiate contract for an address
var myContractInstance = MyContract.at('0xc4abd0339eb8d57087278718986382264244252f');

// call constant function
var result = myContractInstance.myConstantMethod('myParam');
console.log(result) // '0x25434534534'

// send a transaction to a function
myContractInstance.myStateChangingMethod('someParam1', 23, {value: 200, gas: 2000});

// short hand style
web3.eth.contract(abi).at(address).myAwesomeMethod(...);

// create filter
var filter = myContractInstance.myEvent({a: 5}, function (error, result) {
  if (!error)
    console.log(result);
    /*
    {
        address: '0x8718986382264244252fc4abd0339eb8d5708727',
        topics: "0x1234567890123456789012345678901234",
"0x0000000000000000000000000000000000000000000000000000000000000005",
        data: "0x0000000000000000000000000000000000000000000000000000000000000001",
        ...
    }
    */
});
```

合约方法

```
// Automatically determines the use of call or sendTransaction based on the method type
myContractInstance.myMethod(param1 [, param2, ...] [, transactionObject] [,
defaultBlock] [, callback]);

// 显示的调用这个方法
myContractInstance.myMethod.call(param1 [, param2, ...] [, transactionObject] [,
defaultBlock] [, callback]);

// 显示的发送一个交易到这个方法
myContractInstance.myMethod.sendTransaction(param1 [, param2, ...] [, transactionObject]
[, callback]);

// 获取调用的数据, 你可以使用其他方法调用这合约
var myCallData = myContractInstance.myMethod.getData(param1 [, param2, ...]);
// myCallData = '0x45ff3ff6000000000004545345345345..'
```

Parameters

- `String|Number` – (optional) Zero or more parameters of the function. If passing in a string, it must be formatted as a hex number, e.g. “oxdeadbeef”.
- `Object` – (optional) The (previous) last parameter can be a transaction object, see web3.eth.sendTransaction parameter 1 for more. **Note**: `data` and `to` properties will not be taken into account.
- `Number|String` – (可选) 如果你传了这个参数，那么就不会使用由web3.eth.defaultBlock设置的默认区块 .
- `Function` – (optional) If you pass a callback as the last parameter the HTTP request is made asynchronous. See this note for details.

Returns

`String` – If its a call the result data, if its a send transaction a created contract address, or the transaction hash, see web3.eth.sendTransaction for details.

Example

```
// creation of contract object
var MyContract = web3.eth.contract(abi);

// initiate contract for an address
var myContractInstance = MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

var result = myContractInstance.myConstantMethod('myParam');
console.log(result) // '0x25434534534'

myContractInstance.myStateChangingMethod('someParam1', 23, {value: 200, gas: 2000},
function(err, result){ ... });
```

CONTRACT EVENTS

```
var event = myContractInstance.MyEvent({valueA: 23} [, additionalFilterObject])

// watch for changes
event.watch(function(error, result){
  if (!error)
    console.log(result);
});

// Or pass a callback to start watching immediately
var event = myContractInstance.MyEvent([{valueA: 23}] [, additionalFilterObject] ,
function(error, result){
  if (!error)
    console.log(result);
});
```

You can use events like filters and they have the same methods, but you pass different objects to create the event filter.

Parameters

1. `Object` – Indexed return values you want to filter the logs by, e.g. `{'valueA': 1, 'valueB': [myFirstAddress, mySecondAddress]}`. By default all filter values are set to `null`. It means, that they will match any event of given type sent from this contract.
2. `Object` – Additional filter options, see filters parameter 1 for more. By default filterObject has field 'address' set to address of the contract. Also first topic is the signature of event.

3. `Function` – (optional) If you pass a callback as the last parameter it will immediately start watching and you don't need to call `myEvent.watch(function(){})`. See this note for details.

Callback return

`Object` – An event object as follows:

- `address`: `String`, 32 Bytes – address from which this log originated.
- `args`: `Object` – The arguments coming from the event.
- `blockHash`: `String`, 32 Bytes – hash of the block where this log was in. `null` when its pending.
- `blockNumber`: `Number` – the block number where this log was in. `null` when its pending.
- `logIndex`: `Number` – integer of the log index position in the block.
- `event`: `String` – The event name.
- `removed`: `bool` – indicate if the transaction this event was created from was removed from the blockchain (due to orphaned block) or never get to it (due to rejected transaction).
- `transactionIndex`: `Number` – integer of the transactions index position log was created from.
- `transactionHash`: `String`, 32 Bytes – hash of the transactions this log was created from.

Example

```
var MyContract = web3.eth.contract(abi);
var myContractInstance = MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

// watch for an event with {some: 'args'}
var myEvent = myContractInstance.MyEvent({some: 'args'}, {fromBlock: 0, toBlock:
'latest'});
myEvent.watch(function(error, result){
   ...
});

// would get all past logs again.
var myResults = myEvent.get(function(error, logs){ ... });

...

// would stop and uninstall the filter
myEvent.stopWatching();
```

CONTRACT ALLEVENTS

```
var events = myContractInstance.allEvents([additionalFilterObject]);

// watch for changes
events.watch(function(error, event){
  if (!error)
    console.log(event);
});

// Or pass a callback to start watching immediately
var events = myContractInstance.allEvents([additionalFilterObject,] function(error, log)
{
  if (!error)
    console.log(log);
});
```

Will call the callback for all events which are created by this contract.

Parameters

1. `Object` – Additional filter options, see filters parameter 1 for more. By default filterObject has field 'address' set to address of the contract. This method sets the topic to the signature of event, and does not support additional topics.
2. `Function` – (optional) If you pass a callback as the last parameter it will immediately start watching and you don't need to call `myEvent.watch(function(){})`. See this note for details.

Callback return

`Object` – See Contract Events for more.

Example

```
var MyContract = web3.eth.contract(abi);
var myContractInstance = MyContract.at('0x78e97bcc5b5dd9ed228fed7a4887c0d7287344a9');

// watch for an event with {some: 'args'}
var events = myContractInstance.allEvents({fromBlock: 0, toBlock: 'latest'});
events.watch(function(error, result){
   ...
```

```
});

// would get all past logs again.
events.get(function(error, logs){ ... });

...

// would stop and uninstall the filter
myEvent.stopWatching();
```

WEB3.ETH.GETCOMPILERS

```
web3.eth.getCompilers([callback])
```

Gets a list of available compilers.

Parameters

1. `Function` – (optional) If you pass a callback the HTTP request is made asynchronous. See this note for details.

Returns

`Array` – An array of strings of available compilers.

Example

```
var number = web3.eth.getCompilers();
console.log(number); // ["lll", "solidity", "serpent"]
```

WEB3.ETH.COMPILE.SOLIDITY

```
web3.eth.compile.solidity(sourceString [, callback])
```

Compiles solidity source code.

Parameters

1. `String` – The solidity source code.
2. `Function` – (optional) If you pass a callback the HTTP request is made asynchronous. See this note for details.

Returns

`Object` – Contract and compiler info.

Example

```
var source = "" +
    "contract test {\n" +
    "   function multiply(uint a) returns(uint d) {\n" +
    "       return a * 7;\n" +
    "   }\n" +
    "}\n";
var compiled = web3.eth.compile.solidity(source);
console.log(compiled);
// {
  "test": {
    "code":
"0x605280600c6000396000f3006000357c010000000000000000000000000000000000000000000000000000000
0000090048063c6888fa114602e57005b6037600436041565b8060005260206000f35b60006007820290506
04d565b91905056",
    "info": {
      "source": "contract test {\n\tfunction multiply(uint a) returns(uint d)
{\n\t\treturn a * 7;\n\t}\n}\n",
      "language": "Solidity",
      "languageVersion": "0",
      "compilerVersion": "0.8.2",
      "abiDefinition": [
        {
          "constant": false,
          "inputs": [
            {
```

```
            "name": "a",
            "type": "uint256"
          }
        ],
        "name": "multiply",
        "outputs": [
          {
            "name": "d",
            "type": "uint256"
          }
        ],
        "type": "function"
      }
    ],
    "userDoc": {
      "methods": {}
    },
    "developerDoc": {
      "methods": {}
    }
  }
}
```

WEB3.ETH.COMPILE.LLL

```
web3. eth.compile.lll(sourceString [, callback])
```

Compiles LLL source code.

Parameters

1. `String` – The LLL source code.
2. `Function` – (optional) If you pass a callback the HTTP request is made asynchronous. See this note for details.

Returns

`String` – The compiled LLL code as HEX string.

Example

```
var source = "...";

var code = web3.eth.compile.lll(source);
console.log(code); //
"0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b60216004356
02b565b8060005260206000f35b600081600702905091905056"
```

WEB3.ETH.COMPILE.SERPENT

```
web3.eth.compile.serpent(sourceString [, callback])
```

Compiles serpent source code.

Parameters

1. `String` – The serpent source code.
2. `Function` – (optional) If you pass a callback the HTTP request is made asynchronous. See this note for details.

Returns

`String` – The compiled serpent code as HEX string.

```
var source = "...";

var code = web3.eth.compile.serpent(source);
console.log(code); //
"0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b60216004356
02b565b8060005260206000f35b600081600702905091905056"
```

WEB3.ETH.NAMEREG

```
web3.eth.namereg
```

Returns GlobalRegistrar object.

Usage

see namereg example.

web3.db

WEB3.DB.PUTSTRING

```
web3.db.putString(db, key, value)
```

This method should be called, when we want to store a string in the local leveldb database.

Parameters

1. `String` – The database to store to.
2. `String` – The name of the store.
3. `String` – The string value to store.

Returns

`Boolean` – `true` if successfull, otherwise `false`.

Example

param is db name, second is the key, and third is the string value.

```
web3.db.putString('testDB', 'key', 'myString') // true
```

WEB3.DB.GETSTRING

```
web3.db.getString(db, key)
```

This method should be called, when we want to get string from the local leveldb database.

Parameters

1. `String` – The database string name to retrieve from.
2. `String` – The name of the store.

Returns

`String` – The stored value.

Example

param is db name and second is the key of string value.

```
var value = web3.db.getString('testDB', 'key');
console.log(value); // "myString"
```

WEB3.DB.PUTHEX

```
web3.db.putHex(db, key, value)
```

This method should be called, when we want to store binary data in HEX form in the local leveldb database.

Parameters

1. `String` – The database to store to.
2. `String` – The name of the store.
3. `String` – The HEX string to store.

Returns

`Boolean` – `true` if successfull, otherwise `false`.

Example

```
web3.db.putHex('testDB', 'key', '0x4f554b443'); // true
```

WEB3.DB.GETHEX

```
web3.db.getHex(db, key)
```

This method should be called, when we want to get a binary data in HEX form from the local leveldb database.

Parameters

1. `String` – The database to store to.
2. `String` – The name of the store.

Returns

`String` – The stored HEX value.

Example

param is db name and second is the key of value.

```
var value = web3.db.getHex('testDB', 'key');
console.log(value); // "0x4f554b443"
```

web3.shh

Whisper Overview

Example

```
var shh = web3.shh;
```

WEB3.SHH.POST

web3.shh.post(object [, callback])

This method should be called, when we want to post whisper message to the network.

Parameters

1. `Object` – The post object:
   - `from`: `String`, 60 Bytes HEX – (optional) The identity of the sender.
   - `to`: `String`, 60 Bytes HEX – (optional) The identity of the receiver. When present whisper will encrypt the message so that only the receiver can decrypt it.
   - `topics`: `Array of Strings` – Array of topics `Strings`, for the receiver to identify messages.

- `payload`: `String|Number|Object` – The payload of the message. Will be autoconverted to a HEX string before.
- `priority`: `Number` – The integer of the priority in a rang from ... (?).
- `ttl`: `Number` – integer of the time to live in seconds.

2. `Function` – (optional) If you pass a callback the HTTP request is made asynchronous. See this note for details.

Returns

`Boolean` – returns `true` if the message was send, otherwise `false`.

Example

```
var identity = web3.shh.newIdentity();
var topic = 'example';
var payload = 'hello whisper world!';

var message = {
  from: identity,
  topics: [topic],
  payload: payload,
  ttl: 100,
  workToProve: 100 // or priority TODO
};

web3.shh.post(message);
```

WEB3.SHH.NEWIDENTITY

```
web3.shh.newIdentity([callback])
```

Should be called to create new identity.

Parameters

1. `Function` – (optional) If you pass a callback the HTTP request is made asynchronous. See this note for details.

Returns

`String` – A new identity HEX string.

Example

```
var identity = web3.shh.newIdentity();
console.log(identity); //
"0xc931d93e97ab07fe42d923478ba2465f283f440fd6cabea4dd7a2c807108f651b7135d1d6ca9007d5b68a
a497e4619ac10aa3b27726e1863c1fd9b570d99bbaf"
```

WEB3.SHH.HASIDENTITY

```
web3.shh.hasIdentity(identity, [callback])
```

Should be called, if we want to check if user has given identity.

Parameters

1. `String` – The identity to check.
2. `Function` – (optional) If you pass a callback the HTTP request is made asynchronous. See this note for details.

Returns

`Boolean` – returns `true` if the identity exists, otherwise `false`.

Example

```
var identity = web3.shh.newIdentity();
var result = web3.shh.hasIdentity(identity);
console.log(result); // true

var result2 = web3.shh.hasIdentity(identity + "0");
console.log(result2); // false
```

WEB3.SHH.NEWGROUP

Example

```
// TODO: not implemented yet
```

WEB3.SHH.ADDTOGROUP

Example

```
// TODO: not implemented yet
```

WEB3.SHH.FILTER

```
var filter = web3.shh.filter(options)

// 注册监听
filter.watch(function(error, result){
  if (!error)
    console.log(result);
});
```

监听进来的whisper（类似：微信，一个即时通讯工具） 消息.

Parameters

1. `Object` – 这过滤器选项:
   - `topics`: `Array of Strings` – Filters messages by this topic(s). You can use the following combinations:
     - `['topic1', 'topic2'] == 'topic1' && 'topic2'`
     - `['topic1', ['topic2', 'topic3']] == 'topic1' && ('topic2' || 'topic3')`
     - `[null, 'topic1', 'topic2'] == ANYTHING && 'topic1' && 'topic2'` -> `null` works as a wildcard
   - `to`: Filter by identity of receiver of the message. If provided and the node has this identity, it will decrypt incoming encrypted messages.
2. `Function` – (optional) If you pass a callback the HTTP request is made asynchronous. See this note for details.

Callback return

`Object` – The incoming message:

- `from`: `String`, 60 Bytes – The sender of the message, if a sender was specified.
- `to`: `String`, 60 Bytes – The receiver of the message, if a receiver was specified.
- `expiry`: `Number` – Integer of the time in seconds when this message should expire (?).
- `ttl`: `Number` – Integer of the time the message should float in the system in seconds (?).
- `sent`: `Number` – Integer of the unix timestamp when the message was sent.
- `topics`: `Array of String` – Array of `String` topics the message contained.
- `payload`: `String` – The payload of the message.
- `workProved`: `Number` – Integer of the work this message required before it was send (?).

WEB3.ETH.SENDIBANTRANSACTION

```
var txHash = web3.eth.sendIBANTransaction('0x00c5496aee77c1ba1f0854206a26dda82a81d6d8',
'XE81ETHXREGGAVOFYORK', 0x100);
```

反馈
建议

从用户账号中，发送IBAN(国际银行账号)交易到另一个IBAN地址.

Parameters

- `string` – 发送交易的地址
- `string` – IBAN address to which we want to send transaction
- `value` – value that we want to send in IBAN transaction

WEB3.ETH.IBAN

```
var i = new web3.eth.iban("XE81ETHXREGGAVOFYORK");
```

WEB3.ETH.IBAN.FROMADDRESS

```
var i = web3.eth.iban.fromAddress('0x00c5496aee77c1ba1f0854206a26dda82a81d6d8');
console.log(i.toString()); // 'XE73380073KYGTWWZN0F2WZ0R8PX5ZPPZS
```

WEB3.ETH.IBAN.FROMBBAN

```
var i = web3.eth.iban.fromBban('ETHXREGGAVOFYORK');
console.log(i.toString()); // "XE81ETHXREGGAVOFYORK"
```

WEB3.ETH.IBAN.CREATEINDIRECT

```
var i = web3.eth.iban.createIndirect({
  institution: "XREG",
  identifier: "GAVOFYORK"
```

```
});
console.log(i.toString()); // "XE81ETHXREGGAVOFYORK"
```

WEB3.ETH.IBAN.ISVALID

```
var valid = web3.eth.iban.isValid("XE81ETHXREGGAVOFYORK");
console.log(valid); // true

var valid2 = web3.eth.iban.isValid("XE82ETHXREGGAVOFYORK");
console.log(valid2); // false, cause checksum is incorrect

var i = new web3.eth.iban("XE81ETHXREGGAVOFYORK");
var valid3 = i.isValid();
console.log(valid3); // true
```

WEB3.ETH.IBAN.ISDIRECT

```
var i = new web3.eth.iban("XE81ETHXREGGAVOFYORK");
var direct = i.isDirect();
console.log(direct); // false
```

WEB3.ETH.IBAN.ISINDIRECT

```
var i = new web3.eth.iban("XE81ETHXREGGAVOFYORK");
var indirect = i.isIndirect();
console.log(indirect); // true
```

WEB3.ETH.IBAN.CHECKSUM

```
var i = new web3.eth.iban("XE81ETHXREGGAVOFYORK");
var checksum = i.checksum();
console.log(checksum); // "81"
```

WEB3.ETH.IBAN.INSTITUTION

```
var i = new web3.eth.iban("XE81ETHXREGGAVOFYORK");
var institution = i.institution();
console.log(institution); // 'XREG'
```

WEB3.ETH.IBAN.CLIENT

```
var i = new web3.eth.iban("XE81ETHXREGGAVOFYORK");
var client = i.client();
console.log(client); // 'GAVOFYORK'
```

WEB3.ETH.IBAN.ADDRESS

```
var i = new web3.eth.iban('XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS');
var address = i.address();
console.log(address); // '00c5496aee77c1ba1f0854206a26dda82a81d6d8'
```

WEB3.ETH.IBAN.TOSTRING

```
var i = new web3.eth.iban('XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS');
console.log(i.toString()); // 'XE7338O073KYGTWWZN0F2WZ0R8PX5ZPPZS'
```

web3.js

你可以使用由web3.js提供的web3对象，让你的Dapp运行在以太坊区块链上。它的底层是通过RPC调用来实现与本地节点的交互的。只要节点输出了一个RPC层，那么web3.js都成正常工作。

web3 包含了一个 eth 对象- web3.eth （专门为了和以太坊区块链进行交互用的），和 shh 对象 – web3.shh (负责和Whisper（类似微信）通信协议的交互)。以后，我们会介绍web3的其他的所有协议。

开始

- 添加web3 Adding web3
- 使用回调 Using Callbacks
- 批量请求 Batch requests
- 在web3中使用大数字 A note on big numbers in web3.js
- API索引 API Reference

Adding web3

你可以使用下面的方式把web3.js 文件添加到你的项目中。

- npm: `npm install web3`
- bower: `bower install web3`
- meteor: `meteor add ethereum:web3`
- vanilla: 链接 `dist./web3.min.js`

然后创建web3实例，设置一个提供者。创建之前，最好先判断下，是否已经创建了这web3实例：

```
if (typeof web3 !== 'undefined') {
  web3 = new Web3(web3.currentProvider);
} else {
  // 使用 Web3.providers 设置一个提供者
  web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"));
}
```

有了web3实例对象后，可以就使用web3提供的接口了。

使用回调

所有的HTTP请求默认都是同步的。如果你想使用一个异步请求，在大多数的方法里，你可以在参数的最后添加一个你自己的回调函数，所有的回调函数都使用下面这样的形式：

```
web3.eth.getBlock(48, function(error, result){
    if(!error)
        console.log(result)
    else
        console.error(error);
})
```

批量请求 Batch requests

Batch可以让请求排队并一次性处理它们。

注意：批量请求并不会让速度变快！在异步请求的情况下，让请求立刻执行反而会让速度更快。批量请求主要用于那些需要依次处理的请求。

```
var batch = web3.createBatch();
batch.add(web3.eth.getBalance.request('0x0000000000000000000000000000000000000000',
'latest', callback));
batch.add(web3.eth.contract(abi).at(address).balance.request(address, callback2));
batch.execute();
```

在web3中使用大数字

JavaScript语言不能很好的处理大的数字，如下：

```
"1010101003243253453464564564564564564"
// "10101010032432534534645645645645645645"
10101010032432534534645645645645645645
// 1.0101010032432535e+38
```

web3.js自动的添加了一个处理大数字的类，如：

```
var balance = new BigNumber('131242344353464564564574574567456');
// or var balance = web3.eth.getBalance(someAddress);

balance.plus(21).toString(10); // toString(10) converts it to a number string
// "131242344353464564564574574567477"
```

但如果有超过20位的小数位的话，那这个类也不能正常使用了。因此，我们在使用以太币的时候，最好使用它的最小单位wei。当需要展示给别人看的时候，可以再转换成其他单位。

```
var balance = new BigNumber('13124.2344353464564666664574555567456');

balance.plus(21).toString(10); // toString(10) 把数字转换成字符串，但是最多只支持20个小数位
// "13145.23443534645646666646" // 超过20个的小数位会被截掉
```

API索引 API Reference

请参考web3.js 参考手册

链接以太坊客户端

以太坊客户端通过 JSON-RPC 的方式输出了许多接口，在我们的应用内部可以使用这些接口来与以太坊客户端进行交互。然后，如果开发者直接使用JSON-RPC 的话，那可能就要负担很多工作量，如：

- 实现JSON-RPC
- 用于创建合约，和与合约进行交互的二进制数据的编码和解码
- 256位的数学类型
- 管理命令的支持–如：创建和管理地址，交易签名等。

不过，现在已经封装了很多类，来专门解决以上的这些问题，可以让开发者只要专注于他们自己的应用即可。

Library	Language	Project Page
web3.js	JavaScript	https://github.com/ethereum/web3.js
web3j	Java	https://github.com/web3j/web3j
Nethereum	C# .NET	https://github.com/Nethereum/Nethereum
ethereum-ruby	Ruby	https://github.com/DigixGlobal/ethereum-ruby

以上各个类的详细的使用方法将在他自己的章节详细描述。

http://ethdocs.org/en/latest/connecting-to-clients/index.html

2017年1月11日 / 以太坊-区块链 / 留下评论

使用Docker安装以太坊客户端

我们正在把最新的开发快照版发布到 docker hub。你可以用以下的方式来运行这些镜像：

准备

在运行镜像之前，你需要把镜像先拉到本地，和准备好数据目录。

```
# get the lastest version from dockerhub (redo for updates).
docker pull ethereum/client-cpp
# create mountable datadirs; blockchain/account data will be stored there
mkdir -p ~/.ethereum ~/.web3
```

这些步骤只需要做一次就行了，如果要升级镜像的话，那就再运行以下 docker pull ... 命令。

执行

最简单的运行方式

```
  docker run --rm -it -p 0.0.0.0:8545:8545 -p 0.0.0.0:30303:30303 -v
~/.ethereum:/.ethereum -v ~/.web3:/.web3 -e HOME=/ --user $(id -u):$(id -g)
ethereum/client-cpp
```

运行这命令，会把区块数据和账户数据写到在你主机上的 ~/.ethereum 和 ~/.web3/ 目录上。大多数情况下，这样就足够了。客户端的功能和在本地构建的客户端的功能是一模一样的。

如果你想通过外网也可以访问这rpc端口，那只要把 -p 127.0.0.1:8545:8545 替换成 -p 0.0.0.0:8545:8545 即可。

为了方便，你可以使用下面的内容来创建一个shell脚本 /usr/local/bin/docker-eth ：

```
#!/usr/bin/env sh
mkdir -p ~/.ethereum ~/.web3
if ! id -nG $(whoami)|grep -qw "docker"; then SUDO='sudo'; else SUDO=''; fi
$SUDO docker run --rm -it \
    -p 127.0.0.1:8545:8545 \
    -p 0.0.0.0:30303:30303 \
    -v ~/.ethereum:/.ethereum \
    -v ~/.web3:/.web3 \
    -e HOME=/ \
    --user $(id -u):$(id -g) \
    ethereum/client-cpp $@
```

然后让给予这脚本执行权限： chmod +x /usr/local/bin/docker-eth ，然后你可以执行下面的命令启动客户端了

```
docker-eth
```


http://ethdocs.org/en/latest/ethereum-clients/cpp-ethereum/building-from-source/windows.html

2017年1月11日 / 以太坊-区块链 / 留下评论

以太坊客户端 2017.1.11更新

选择一个客户端

以太坊客户端为啥会有多个？

在从项目的早期，为了适应不同系列的操作系统，就实现了多个的客户端。客户端的多样性对于以太坊生态系统来说是一个巨大的优势。然而，对于像我们这样的终端用户来说，可能就会感到迷惑，不知道到底应该选择哪一个客户端。因为以太坊没有一个传统的通用的像“Ethereum Installer”的客户端。

截止到2016年9月，主要实现了go语言版版的go-ethereum。和用Rust语言开发的Parity版

Client	Language	Developers	Latest release
go-ethereum	Go	Ethereum Foundation	go-ethereum-v1.4.18
Parity	Rust	Ethcore	Parity-v1.4.0
cpp-ethereum	C++	Ethereum Foundation	cpp-ethereum-v1.3.0
pyethapp	Python	Ethereum Foundation	pyethapp-v1.5.0
ethereumjs-lib	Javascript	Ethereum Foundation	ethereumjs-lib-v3.0.0
Ethereum(J)	Java	<ether.camp>	ethereumJ-v1.3.1
ruby-ethereum	Ruby	Jan Xie	ruby-ethereum-v0.9.6
ethereumH	Haskell	BlockApps	no Homestead release yet

台式机或笔记本应该安装哪种客户端

Mist / Ethereum 钱包 可以满足大多数人的需求，所以一般情况下，我们使用这客户端就可以了。

Mist 现在已经和 go-ethereum 和cpp-ethereum捆绑在一起了。当Mist 启动的时候，如果你没有正在运行命令行的以太坊客户端，那么它就会使用它绑定的客户端（默认是geth）自动的同步数据。如果你想让Mist链接

到你的在私有链中，那么只要在运行Mist之前，先开启你的节点就可以了，Mist 会先链接你的节点而不会自己去开启一个。

如果你想要挖矿，那仅安装Mist是不够的。关于挖矿的具体方法可以参考挖矿的章节。

手机上客户端

在以太坊项目刚开始的时候，并没有开发适合手机端的客户端。Go语言开发团队正在发布实验中的 iOS 和 Android类。但是还没有真正可用的手机端客户端。

手机端主要的障碍是轻型的客户端尚不够完善。要知道客户端需要同步的区块链数据可能达到几G，甚至十几G，而这对内存宝贵的手机来说，明显是不现实的。目前已经开发完成一个私有的分支，并且最先只支持Go语言版的。在未来数月，随着捐助资金的到位，C++版的也会被陆续开发出来。

cpp-ethereum

快速开始

window操作系统安装

安装包下载地址：
https://github.com/ethereum/webthree-umbrella/releases/download/v1.2.9/cpp-ethereum-windows-v1.2.9.exe

可以去https://github.com/ethereum/webthree-umbrella/releases网查看和下载最新的版本。

目前为止以太坊c++版本的客户端支持的window操作系统有Windows 7, Windows 8/8.1, Windows 10 and Windows Server 2012 R2。

目前以太坊客户端只支持64位操作系统!

如果在安装或使用过程中报**msvcr120.dll** 缺少 **msvcp120.dll**，则你需要安装Visual C++ Redistributable Packages for Visual Studio 2013

运行

没有任何特别配置的话，客户端将与公共的区块链同步数据。当然，它也可以创建或同步到私有链。

可以使用geth或以太坊控制台与你的节点进行交互。

以太坊命令控制台

以太坊控制台是一个连着eth/geth节点的node.js应用，用来访问web3.0的对象。

https://github.com/ethereum/ethereum-console

使用npm安装

> npm install -g ethereum-console

> ethconsole

2017年1月11日 / 以太坊-区块链 / 以太坊客户端 / 留下评论

Geth & Eth 以太坊网络的命令行工具

命令行工具是为区块链开发者准备的工具。使用命令行工具，可以把你的服务连接到正在区块链上运行着的应用或连接到你的私有区块链上。

客户端

客户端下载地址：https://geth.ethereum.org/downloads/

为了安全考虑，以太坊有三种独立实现的客户端。这些客户端的功能几乎是相同的。哪一个客户端适合你，取决于你使用的是哪个平台，哪种开发语言，和你打算用区块链网络来做什么。

如果你正在构建一个企业，需要最大限度的一直连接着以太坊网络，那么强烈推荐你至少使用一种以上的客户端，以确保安全可靠。

Geth

使用go语言开发的客户端叫做Geth。Geth 已经通过了安全审计，并且是将来终端用户使用的的**Mist** 浏览器的基础，所以如果你有web开发的经验，对构建dapp的前端感兴趣的话，你应该尝试使用Geth。

在苹果上安装

安装好 Homebrew，并且确保它升级到最新版：

```
brew update
brew upgrade
```

然后使用下面命令安装以太坊：

```
brew tap ethereum/ethereum
brew install ethereum
```

点击查看更多详细的介绍。

在WINDOWS上安装

下载最新稳定版的二进制安装包。下载zip包，解压得到geth.exe，打开命令行，然后输入：

```
chdir 解压目录
open geth.exe
```

点击查看更多详细的介绍。

在LINUX上安装

在Ubuntu上只要执行下面这些目录即可：

```
sudo apt-get install software-properties-common
sudo add-apt-repository -y ppa:ethereum/ethereum
```

```
sudo apt-get update
sudo apt-get install ethereum
```

点击查看更多详细的介绍。

Eth

使用C++语言开发的客户端叫**Eth**。如果你想为了增加安全性而并行运行2个客户端，或者你想要使用GPU挖矿，那么你适合使用这个C++ “Eth” 客户端。

在苹果上安装

安装好 Homebrew ，并且确保它升级到最新版：

```
brew update
brew upgrade
```

然后使用下面命令安装cpp-ethereum：

```
brew tap ethereum/ethereum
brew install cpp-ethereum
brew linkapps cpp-ethereum
```

点击查看更多详细的介绍。

在LINUX上安装

使用Apt-get命令，把下面的命令黏贴到你的终端即可：

```
apt-get install cpp-ethereum
```


点击查看更多详细的介绍。

在WINDOWS上安装

目前eth已经封装到mist 浏览器中了。点击下载最新版mist 浏览器，解压即可。

Python

使用**Python** 语言实现的客户端叫做Pyethapp。如果你想要理解以太坊是如何工作的，和如何扩展它。这个客户端的代码可能是最容易阅读的，并且它有一个可用于快速开发的强大的合约测试类。如果你是一个python开发者，想构建一个去中心化的应用，或者你对以太坊的学术研究很感兴趣，那么你应该选择这个客户端。

运行

Geth 和Eth 提供了多个接口：命令行的方式，JSON-RPC server 方式和交互式控制台方式。

在本例中，我们会使用交互式控制台的方式，一个JavaScript的环境，可能包含着你想要的所有的功能。根据你的客户端，黏贴下面的命令：

Geth指令

```
geth console
```

Eth指令

```
eth
```

然后，如果你使用是也是geth，则：

```
geth attach
```

不然使用下面的npm 命令，Eth 可能会消耗一点时间：

```
npm install -g ethereum-console
ethconsole
```

第一次使用这命令的时候，可能会看到一个许可，在你正式使用命令之前，你必须同意这个许可。

链接到一个私有测试网络

有的时候，如果你不想要链接到一个真正的公网的话，那么你可以创建一个属于你自己的私有网络。一般情况下，推荐在私有链上做技术开发。因为你的私有链中只有你一个用户，所以你有可能挖到该网的全部区块，验证所有的交易和执行所有的智能合约。在自己的私有链上，可以方便开发者掌握合约的每一个细节。

Geth:

```
geth --datadir ~/.ethereum_private init ~/dev/genesis.json

geth --fast --cache 512 --ipcpath ~/Library/Ethereum/geth.ipc --networkid 1234 --datadir ~/.ethereum_private  console
```

Eth:

```
eth --private 12345 --genesis-json ~/test/genesis.json --db-path ~/.ethereum_experiment
```

把12345 改成任意个随机数字，来作为你的网络ID。最好修改一下你的创世（genesis ）区块的内容，以防止有人使用真实的链意外的链接到了你的私有网络。

https://ethereum.org/cli

2017年1月10日 / 以太坊-区块链 / Geth & Eth、以太坊网络的命令行工具 / 留下评论

阿拉伯人 /