[学习区块链？github上的那些区块链项目 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/90400262)

Youtube相关教程网址：

[Solidity入门上手 - Solidity 智能合约开发环境全讲解（最佳实践） - YouTube](https://www.youtube.com/watch?v=opAzBhEPJP0)

[千锋教育区块链 09 以太坊开发基础 MetaMask - YouTube](https://www.youtube.com/watch?v=BAsA22Geqac&list=PLwDQt7s1o9J4l1LQ3x2vZ54TPhIh6UvMr&index=10)

----------------------智能合约入门部分：------------------------------

关于metamask和remix的应用：

Remix是编写智能合约的语言，可以向智能合约中去充值以太币

写完之后首先进行编译，编译之后选择环境发布

退出后想找到之前的智能合约可以先编译源程序然后输入智能合约地址即可

已经编写的智能合约地址：

0x7ACd0F1c7C1dEc5F032A4c50b5e09B3c61a7f120

一个水龙头智能合约

我的metamask账号地址：

0xD80afEeC3B1B630d7aD6913E620639334d273091

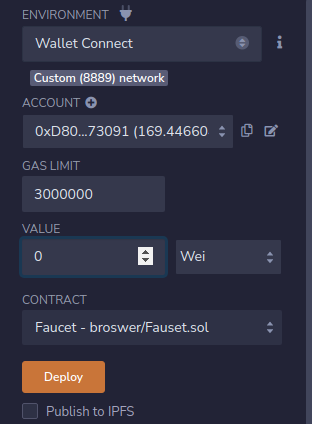
注意执行发布时需要选择正确的环境

google账号：

[vvullnetdukagjini870@gmail.com](mailto:vvullnetdukagjini870@gmail.com)

密码：

Wbf20011015



给智能合约转钱IMG_256

----------------------智能合约入门部分：------------------------------

----------------------以太坊客户端部分：------------------------------

以太坊客户端：

不同的以太坊客户端符合参考标准和标准化通信协议，则可以进行相互操作。黄皮书是以太坊的官方行为规范。

Go-ethereum是最火的http://github.com/ethereum/go-ethereum

以太坊的全节点：整个主链的一个副本，存储并维护链上的数据并随时验证新区块的合法性，做开发不需要再本地部署全节点。全节点优点：以太坊的灵活性和抗审查能力、权威的验证所有交易、可以离线查询、；缺点：需要巨大的硬件和带宽资源，必须及时维护

远程客户端：metamask

轻节点：不保存历史数据 只保存当前的状态，可以对块和交易进行验证

公共测试网络：testnet节点需要同步和存储的数据更少，可以在几小时内同步，只需要测试以太，没有办法涵盖主网的特性，例如gas price

本地的私有链：磁盘上几乎没有数据，也不同步别的数据，是一个干净的环境；无需获取以太，可以任意分配，也可以随时挖矿，没有其他用户，也没有其他合约。必须部署所有内容，依赖项和合约库必须自己部署

启动geth得到同时可以选择同步区块（全节点） geth --datadir ./data

启动节点模式：geth --datadir .--syncmode fast 只下载不验证

Geth --testnet --datadir .--syncmode fast 同步测试网络区块

其实[(99条消息) Ubuntu 服务器：从零开始搭建一个区块链应用\_chao3150的博客-CSDN博客\_服务器安装cnpm](https://blog.csdn.net/chaoge_dgqb/article/details/113400260)这篇文档里的内容就是教我们自己搭建一个本地的私有链。

----------------------以太坊客户端部分：------------------------------

进入geth控制台

输入web3查看web3对象 admin对象 eth是核心对象（accounts、blocks都在eth）链上数据相关 personal对象txpool miner

查看基本信息

eth.accounts账户信息为0

eth.getBalance(“addr”)查看账户的钱 在创世区块分配的账号，虽然account为0

web3.fromWei(eth.getBalance("093f59f1d91017d30d8c2caa78feb5beb0d2cfaf"),'ether') 从wei转到ether 其实只有18个以太

eth.blockNumber 查看区块高度

personal.newAccount() 新建一个账号 密码123456

地址：0x634f2cb16c61e0df5e2bdbc2454933da2f8830c6

eth.accounts再次查看账户可以看到新账户 keystore文件里就有新的账户了

eth.getBalance(eth.accounts[0]) 查看账户余额

personal.unlockAccount(eth.accounts[0]) 解锁账户

Miner挖矿获得以太币

eth.coinbase

miner.start(1)

如果挖矿失败：miner.setEtherbase(eth.accounts[0])

查看eth.hashrate是否大于0

miner.stop()

eth.sendTransaction({from:eth.accounts[0],to:"093f59f1d91017d30d8c2caa78feb5beb0d2cfaf",value:web3.toWei(7)}) 进行转账

生成交易hash：0x08707bcc2b18b637c481e74d39d6cf3edef55bb5ebc59299b7fe727facab264f

发布交易后重启挖矿进行交易，交易完成

**课后需要尝试介绍开发者账户（使用--dev命令）**

dev模式下：有开发者账号（很牛逼，不需要自己挖矿，不需要解锁账户，非常的方便**）**

JSON-RPC：就是一个接口，作为一个http服务，端口为8545，默认情况下值接受localhost的链接，一个其他的控制台，没必要使用

----------------------以太坊客户端部分：------------------------------

----------------------以太坊账户部分：------------------------------

**比特币的做法：**

比特币在基于UTXO的结构中存储有关数据余额的数据，系统的状态就是UTXO的集合

每个交易花费若干个输入的UTXO，并创建若干个新的UTXO（输入--输出）输入必须有效且没有花费（矿工验证），对于一个交易，必须包含与有每个输入的所有者匹配的签名，总输入必须大于总输出

系统中用户的余额是用户具有私钥的UTXO的总值

**以太坊的做法：**

以太坊的状态：系统中的所有账户的列表

每个账户都包含了一个余额以及以太坊定义的特殊数据（代码（智能合约）和内部存储）

发送账户有足够余额则交易有效，发送账户先扣款，收款账户记入收入

如果接收账户有相关代码，则代码自动运行，可能更改内部存储，或者代码还可能向其他账户发送额外的消息

**优缺点分析**：

比特币的优点：更高程度的隐私但不适用于Dapp的复杂逻辑状态&&潜在的可扩展性，只需要有UTXO的那些人去维护Merkle树的所有权证明就够了

以太坊的优势：

节省大量空间：将UTXO合为一个账户，每个交易只需要一个输入、一个签名和一个输出

更好的替代性：UTXO让货币从来源分成了“可花费”、“不可花费”两类

更容易编码和理解

便于维护持久轻节点

**以太坊账户类型：**

外部账户（EOA）：由人来控制 用户账户 普通账户

有对应的以太币余额、可发送交易、由用户私钥控制、没有关联代码

合约账户（contract accounts）：账户内部是一个合约 内部账户

有对应的以太币余额

有关联代码

由代码控制

可通过交易或来自于其他合约的调用信息来触发代码执行

执行代码时可以操作自己的操作空间，也可以调用其他合约

比特币介绍：

[比特币是什么？ - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/133202649)

以太坊交易（Transaction）

签名的数据包，由EOA发送到另一个账户

消息的接收方地址

发送方签名

金额

数据（可选）

Start gas 就是gas limit上限

Gas price

消息（message）

合约可以向替他合约发送消息

消息是不会被序列化的虚拟对象，只存在于以太坊的虚拟环境（evm）

可以看做函数调用

消息发送方

消息接收方

金额

数据

Start gas

合约：（constract）

可以读写自己的内部存储（32字节kv数据库，有点像map）

可以向其他合约发送消息，依次触发执行

一旦合约执行结束，并且有它发送的消息触发所有的子执行结束，evm就会终止运行

合约的应用：

（一）

维护一个数据存储账本，存放对其他合约或外部世界有用的内容

最典型的例子是模拟货币的合约（代币）

（二）

转发合同，只有在某些条件被满足时才转账（钱包合约）

（三）

管理多个用户之间的持续合同或关系

----------------------以太坊账户部分：------------------------------

----------------------以太坊交易部分：------------------------------

以太坊的交易是唯一可以触发状态更改或导致合约在EVM中执行的事务

以太坊是一个全局单例状态机，交易是唯一可以改变其状态的东西

合约不是自己运行的，以太坊上的一切变化都起源于交易

**交易的数据结构**

Nonce：由发起人EOA发出的序列号，用于防止交易消息重播，一个标量值，等于从这个地址发送的交易数，不会存到账户状态，而是动态计算的；nonce强制任何地址的交易按顺序处理

Gas price：交易发起人愿意支付的gas单价

Start gas：最大gas量

To：目的以太坊地址

Value：要发送的以太数量

Data：可变长度二进制数据负载（payload）

V，r，s：EOA的ESDSA签名的三个组成部分

交易消息结构：RLP递归长度前缀

并发与nonce：允许操作并发，但强制执行单例状态

解决方案：1：用一台服务器为各个应用分配nonce，先来先服务。出现单点故障后，后续的交易都会失败并且阻塞

2：生成交易后不分配nonce，也不签名，而是放入一个队列等待。另起一个节点跟踪nonce并签名交易。同样会有单点故障的可能，但最终分配nonce仍然是逐个排队上电。

为了解决重放攻击引入nonce，但是却牺牲了并发性

Gas

由于交易或消息触发EVM运行时，每个指令都会在网络的每个节点上执行。对于每个执行的操作，都存在固定的成本，我们把这个成本用gas表示。

Gas：交易发起人需要为每项操作支付的成本的名称，用以太币从矿工那里购买gas

Gas与消耗的系统资源对应。gas指代硬件资源的消耗，是具有自然成本的。设计上与以太解耦，但是仍然与以太紧密相关。币的价格会有很大波动，与大家对以太坊的认可程度有关。gas与以太币的价格不挂钩。gas的价格实际由，由矿工决定，他们可以拒绝处理gas价格低于最高限额的交易。

Gas limit：给定了一个允许消耗gas的上限，相当于押金

Gas used：实际中运行消耗的gas

最终支付的：gasprice\*gasused是最终支付给矿工的手续费

在geth终端查看可能收的手续费：eth.estimateGas({from:eth.accounts[0],to:eth.accounts[0],value:10})

交易的接受者（to）：交易接收者，是一个20字节的以太坊地址。地址可以使EOA或者合约地址，以太坊没有进一步的验证，任何20字节都被认为是有效的。如果将交易发送到无效地址，将销毁发送的以太

Value和data：仅有value就是转账；仅有data一半是合约调用；调用合约且发送以太：同时又data和value，两个都没有就是在浪费gas

向EOA或合约传递data

当交易包含负载时，它很有可能是发送到合约地址的，但同时可以发送至EOA

发送给EOA时，数据负载的解释取决于钱包

如果发送给合约地址，EVM会解释为函数调用解码并调用

发送合约的数据有效负载时32字节的16进制序列化编码

----函数选择器

----函数参数

特殊交易--部署合约

合约创建交易被发送到特殊的地址，即零地址0X0仅用作目的地，具有特殊含义“创建合约”

----------------------以太坊交易部分：------------------------------

----------------------以太坊虚拟机：------------------------------

以太坊虚拟机就是智能合约的运行环境

每个网络节点都会运行EVM

EVM是沙盒封装的，而且是完全隔离的，EVM中运行的代码是无法访问网络、文件系统和其他进程的。每个节点是世界计算机的一部分

合约通常以高级语言（solidty）编写，通过EVM编译器编译为字节码，最终通过客户端上载部署到区块链网络中。

无论账户是否存储代码，两类账户对EVM来说处理方式完全一样

每个账户都有一个KV对形式的持久化存储。KV的长度都是256位。

EVM可以看做从一个账户发送到另一个账号的消息，它可以包含二进制数据和以太币

如果目标账户含有代码，代码会在EVM中执行，并以payload作为入参，这就是合约的调用

EVM执行交易时，gas将按照特定的规则主键消耗殆尽

合约被交易触发调用时，指令会在全网的每个节点上执行；这需要消耗算力成本，gas用来量化表示这个成本消耗

EVM的数据存储：

Storage：持久化的存储空间，可以理解为合约的数据库，永久保存合约状态变量，永久存储在区块链中，读写gas的开销非常大

Memory：内存 每一次消息调用，合约会获得一块干净的内存空间

Stack：栈 EVM不是基于寄存器的，而是基于栈的，所有的计算都在一个被称为栈的区域执行，免费试用内存，但是数量有限

EVM指令集：都是针对256位的字进行操作；具备常用的算数、位、逻辑和比较操作，也可以做到有条件、无条件跳转，合约可以访问当前块的相关属性，比如块高度和时间戳。

消息调用：合约可以通过消息调用的方式来调用其他合约或者发送以太币到非合约账户；合约可以决定在其内部的消息调用中，对于剩余的gas，应发送多少和保留多少

委托调用：目标地址的代码将在发起调用的合约的上下文中执行，在委托调用中msg.sender和msg.value的值不变，可以由此实现库，可复用的代码库可以放在一个合约的存储上，通过委托调用引入相应代码

合约的创建和自毁：

Create calls：合约可以创建其他合约

合约代码再链上移除：selfdestruct，合约账户上剩余的以太币会发送给指定的目标，然后其存储和代码从状态中移除

更改solc版本：[Ubuntu20.04+solc+solc-select安装/使用记录 - 简书 (jianshu.com)](https://www.jianshu.com/p/f472573e3048)

----------------------以太坊虚拟机：------------------------------

----------------------solidty：------------------------------

增加address数据类型

语言内嵌框架支持支付。提供payable等关键字

使用区块链进行数据存储

Solidty编写的智能合约先进性编译成字节码，编译过程中产生智能合约的二进制接口规范（ABI）

通过交易的方式将字节码部署到以太坊网络，每次成功部署都会产生一个新的智能合约账户

使用js编写的DAPP通常通过web3.js+ABI调用智能合约

编译器：remix+solcjs

Byte32定长 string:可能是无限长

在geth客户端中：给Byte32传信息需要传16进制字符，可以调用geth客户端的相应接口

Web3.toHex

Web3.toUtf8

事件以及事件的监听：

可以用web3监听事件

----------------------solidty：------------------------------