以太坊技术开发从入门到精通，干货篇。

目标读者：

专业的程序员；

想深入了解以太坊/区块链及其生态的读者；

如果你已经有一定的以太坊技术基础，只想研究一些落地项目，可以直接跳到后面的项目模块。

预备知识：

了解区块链的概念以及比特币的运行机制。如果不了解，可以先看看阿里云整理的区块链菜鸟入门系列（https://yq.aliyun.com/articles/60131?utm_content=m_41917）；

有基本的编程知识将是极大的加分项，了解系统/架构/数学等知识。

不知你是否和我一样，一开始被各种数学问题所迷惑，如拜占庭问题，双花问题等。我的建议是一开始先抛开这些问题，对区块链的原理有一个基本的全局的了解，然后再回过头来思考这些问题。

事实上，如果我们只想基于以太坊开发智能合约应用，也不必完全理解区块链的架构。

以太坊

以太坊和比特币一样，底层框架都是区块链协议，区块链本质上是一个应用了密码学技术的分布式数据库系统。在看了前面提及的阿里云整理的科普文章之后，为了能进一步了解以太坊，建议看一下以太坊白皮书：https://github.com/ethereum/wiki/blob/master/%5B%E4%B8%AD%E6%96%87%5D-%E4%BB%A5%E5%A4%AA%E5%9D%8A%E7%99%BD%E7%9A%AE%E4%B9%A6.md

智能合约

智能合约是一段运行在以太坊区块链系统之上的一段代码，合约根据事先制订的规则来自动转移数字资产。例如，一个人可能有一个存储合约，形式为“A可以每天最多提现X个币，B每天最多Y个，A和B一起可以随意提取，A可以停掉B的提现权”。

以太坊账户

以太坊中有两种账户：外部账户(EOA)和合约账户

外部账户具有以下特性：

1.有一个以太币余额；

2.可以发送交易（以太币转账或者激活合约代码）；

3.通过私钥控制；

4.没有相关联的代码。

合约账户拥有以下特性：

1.有一个以太币余额；

2.有相关联的代码；

3.代码执行是通过交易或者其他合约发送的call来激活；

4.当被执行时 -- 运行在随机复杂度 （图灵完备性）-- 只能操作其拥有的特定储存，例如可以拥有其永久state -- 可以call其他合约.

所有以太坊区块链上的行动都是由各账户发送的交易激活。每次一个合约账户收到一个交易，交易自带的参数都会成为代码的输入值运行。合约代码会被以太坊虚拟机（EVM）在每一个参与网络的节点上运行，以作为它们新区块的验证。

什么是Gas

智能合约由区块链网络中的每个完整节点重复执行，使得合约执行的消耗变得昂贵，所以这也促使大家将能在链下进行的运算都不放到区块链上进行。对于每个被执行的命令都会有一个特定的消耗，用单位gas计数。每个合约可以利用的命令都会有一个相应的gas值。gas值的存在避免智能合约进入死循环，你不能编写永不结束的程序，因为你用尽了gas，计算将被节点拒绝。

在以太坊中，每笔交易都被要求包括一个gas limit和一个交易愿为单位gas支付的费用。矿工可以有选择的打包这些交易并收取这些费用。在现实中，由于矿工会优先选择打包费用高的交易，所以用户所选择支付的交易费用多少会影响到该交易被打包所需等待的时长。

如果该交易由于计算，包括原始消息和一些触发的其他消息，需要使用的gas数量小于或等于所设置的gas limit，那么这个交易会被处理。

如果gas总消耗超过gas limit，那么所有的操作都会被复原，但交易是成立的并且交易费任会被矿工收取。区块链会显示这笔交易完成尝试，但因为没有提供足够的gas导致所有的合约命令都被复原（out-of-gas）。

所有交易里没有被使用的超量gas都会以以太币的形式打回给交易发起者。因为gas消耗一般只是一个大致估算，所以许多用户会超额支付gas来保证他们的交易会被接受。

去中心化应用DApp

DApp是一种“服务端”运行在区块链网络上的应用，类似于app运行在Android/iOS等设备上，DApp运行在以太坊网络上。以太坊在GitHub下有一个代码仓库dapp-bin（https://github.com/ethereum/dapp-bin），里面有一些文档和示例。使用前，你需要看看文件最近的状态，因为他们将很可能已经被淘汰。

DApp客户端

目前有四个可运行的，分别由C++，Go，Python和Java实现的几乎全兼容以太坊协议的客户端。C++

+和Go实现的客户端目前完全兼容。

1. go-ethereum

go-ethereum客户端通常被称为geth，是目前用户最多，使用最广泛的客户端。通过Geth客户端与以太坊网络进行连接和交互可以实现账户管理、合约部署、挖矿等众多有趣且实用的功能（https://ethereum.github.io/go-ethereum）。

2. pyethapp

Pyethapp是以python为基础的客户端，实现以太坊加密经济状态机。python实现旨在提供一个更容易删节和扩展的代码库。Pyethapp利用两个以太坊核心组成部分来实现客户端：

pyethereum ——核心库，以区块链、以太坊模拟机和挖矿为特征；

pydevp2p ——点对点网络库，以节点发现和运输多码复用和加密连接为特征。

Github: https://github.com/ethereum/pyethapp

维基百科: https://github.com/ethereum/pyethapp/wiki/Getting-Started

Gitter聊天: https://gitter.im/ethereum/pyethapp

3. Parity

Parity 声称是世界上最快速最轻便的客户端。它用Rust语言写成，可靠性、性能和代码清晰度都有所增强。Parity由Ethcore开发。Ethcore由以太坊基金会的几个会员创建。

网站: https://ethcore.io/parity.html

Github: https://github.com/ethcore/parity

Gitter聊天: https://gitter.im/ethcore/parity

DApp浏览器

一个DApp浏览器，正如它字面所表达的，用来让DApp客户端（常常使用JS与以

太坊的智能合约进行交互）的使用更加容易。

DApp浏览器的主要目的是：

提供到一个以太坊节点的连接（或者连接到一个本地节点或者远程节点），和一个方便的切换不同节点（甚至是不同的网络）。

提供一个帐户（或者一个钱包）来方便用户与DApp交互。

1. Mist

Mist（https://github.com/ethereum/mist）是以太坊官方的DApp浏览器。一个漂亮的界面来与以太坊节点交互，与智能合约发、收交易。

2. Status

Status（https://status.im/）是一个手机上可以使用的DApp浏览器。

3. MetaMask

MetaMask（https://metamask.io/）是一个Google浏览器扩展，把Chrome变成了一个DApp浏览器。它的核心特性是注入以太坊提供的js客户端库web3，到每一个界面，来让DApp连接到MetaMask提供的以太坊节点服务。不过这个Chrome扩展，可以允许你管理你的钱包，以及连接到不同的以太坊网络（译者注：包括本地的开发网络）。

4. Parity

Parity是一个以太坊客户端（也是一个全节点的实现），集成到了Web浏览器，并使之成为一个DApp浏览器。

以太坊代币

现在你应该知道我们可以通过写智能合约，并将状态存到区块链上了？那如果，在状态这块，我们存的是一个Map类型，键是地址，值是整数。然后我们将这些整数值叫做余额，谁的余额呢？它就是我们要说的代币（代币的数据结构就是这样简单，存的就是某个用户当前的余额）。

是的，所有你刚才听到的代币，只是一些数据，存储在一个哈希表里，通过api或

者所谓的协议，来进行增删改查。这是一个简单的基本合约（https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/BasicToken.sol）。

你可以看看ethereum的创建一个众筹合约的官方教程（https://www.ethereum.org/crowdsale）。你将会发现它仅仅是一个合约（Crowdsale）与另一个合约（MyToken）交互，和前面的基本合约类似。并没有什么神奇的地方。

人们使用代币来做各种各样的事情，阻拦大家如何使用的只有想像力。代币常常用来激励用户与某个协议进行交互，或者证明对某个资产的所有权，投票权等等。

以太坊的创始人Vitalik最近有一个关于代币发售模型，也是一篇不错的文章（https://vitalik.ca/general/2017/06/09/sales.html）。

与智能合约进行交互

你与智能合约的交互（也称做调用函数和读取状态）通过连接到某个以太坊节点，并执行操作码。当前有各种各样的以太坊客户端，可以方便进行开发。Geth和parity都提供了控制台或浏览器的方式来更好的与智能合约交互。

如果你想要一个程序的库用来与智能合约交互的接口，也有这样的客户端实现。对于JS语言，可以使用web3.js。以于go语言，可以使用在go-ethereum中的abigen的程序，提供了go包，用来与智能合约交互。

如果只是用来测试和开发，可以使用Ganache来运行一个本地节点（译者注：这个节点压短区块时间等，可以方便打整的开发与测试）。

当你部署了一个智能合约，你实际进行的操作是向地址0x0发送了一个交易，使用当前合约内容作为参数，一个以太坊交易详解（https://medium.com/@codetractio/inside-an-ethereum-transaction-fa94ffca912f）。

Truffle和Embark

一旦你开始写智能合约，你会重复做大量的操作，比如编译源码为字节码和abi，部署到网络，测试然后部署合约等等。你也许希望更关注于你想要实现的东西。

Truffle和Embark框架，标准化和自动化了这些琐碎的工作。它们提供了一个好的开发，部署，以及更为重要的，测试智能合约的体验。

你可以通过官方文档来开启使用Truffle的旅程。

我公众号之前也写了一篇使用truffle开发Dapp的文章，作为入门也是一个不错的选择：基于以太坊开发第一个去中心化应用——宠物商店。

Embark（https://github.com/embark-framework/embark）提供了类似的，帮助开发者组织工程的稍有些不同的工具。

当你一开始接触智能合约这块时，应该尽量不要使用框架。直到你明白了使用框架能带来的价值时，才应该开始使用，正如你不应该通过rails new来学习HTML语言一样。

ETHPM

分享是关心，所以ETHPM是一个去中心化的智能合约包管理资源库(https://www.ethpm.com/registry)。使用ETHPM，你可以关联或连接到某个著名的合约或库，减少代码重复，尽可能理想的为未来的开发提供好的基础。

这里的这个规范(https://github.com/ethereum/EIPs/issues/190)，详细的说明了相关的信息以及背景。Truffle和Embark均可与之集成，并创造一个愉快的开发体验。

以太坊网络

Mainnet-以太坊主网，通常是所有客户端的默认网络。

Ropsten - 以太坊使用工作量证明的主测试网络。这个网络，因为低的计算量，容易遭到DDOS攻击，分片，或者其它问题。垃圾邮件攻击后被暂时放弃，最近才恢复使用。链接：https://github.com/ethereum/ropsten

Kovan - parity客户端组成的测试网络，使用授权证明来提升对垃圾邮件攻击的抗扰度，并且持续4秒的阻塞时间。链接：https://github.com/kovan-testnet/proposal

Rinkeby - geth客户端组成的测试网络，使用集团共识，尽管计算量低，但是对恶意行为者更有弹性。链接：https://www.rinkeby.io/

你可以自己搭建你自己的测试网络，也许使用kubernetes（https://github.com/MaximilianMeister/kuberneteth）或者docker-compose（https://capgemini.

github.io/blockchain/ethereum-docker-compose），但也许你将很快就可以不需要花什么时间。

智能合约编程语言

Solidity

Solidity是第一批的描述智能合约的语言。当前是最流行的语言，因此也有最多的例子，文档，和教程。你应该学习这个，除非你有要学习其它的理由。

你可以使用基于浏览器的Remix IDE来进行快速验证。

下面是一个Solidity的合约：

pragma solidity ^0.4.11;contract BasicToken { mapping(address => uint256) balances; function transfer(address _to, uint256 _value) returns () { balances[msg.sender] = balances[msg.sender] - _value; balances[_to] = balances[_to] + _value; } function balanceOf(address _owner) constant returns (uint256 balance) { return balances[_owner]; }}

LLL

LLL是一门Lisp风格的底层编程语言，就像语言名称看到的这样。虽然以太坊官方并没有将它作为主要需要支持的语言，但它仍旧持续进行着更新，且与solidity在同一个资源库。

这是一个使用LLL语言写的一个ERC20代币的合约，链接：https://github.com/benjaminion/LLL_erc20/blob/1c659e890e2b30408555b9467a8dfd8988211b3b/erc20.lll

如果你正在学习，也许不是那么的容易习惯LLL语言的写法。

Serpent

Serpent是一个类Python的高级语言，最终也会被编译为EVM字节码。它主要被Augur团队使用。

但最近Zeppelin Solution团队发现其编译器有一个严重的bug，链接：https://blog.zeppelin.solutions/serpent-compileraudit-3095d1257929。在这个问题被修复之前都不建议继续使用。

如果你对Augur如何解决这些漏洞感兴趣，你可以阅读Zeppelin Solution的这篇文章。链接：https://blog.zeppelin.solutions/augur-rep-token-critical-vulnerability-disclosure-3d8bdffd79d2

Serpent的合约看起来如下：

def register(key, value): # Key not yet claimed if not self.storage[key]: self.storage[key] = value return(1) else: return(0) # Key already claimeddef ask(key): return(self.storage[key])智能合约的安全

一旦一个智能合约部署到了以太坊的网络上，它将是永不可变的，且将永久存在。如果你写了一个bug，你将不能下架这个有问题的版本，你只能在后续的版本中修复。

由于许多工程师开发的Ethereum和其他智能合同平台来自于Web开发，所以这个概念实在是太新，而且是疯狂的。

ConsenSys有一个非常棒的资源叫智能合约的最佳实践，你应该深入的理解一下。链接：https://github.com/ConsenSys/smart-contract-best-practices

一个Parity的钱包被黑的解释（https://blog.zeppelin.solutions/on-the-parity-wallet-multisig-hack-405a8c12e8f7）。

在你部署你的智能合约的时候，由于你管理的是真正的资金，你应该先开一个赏金计划（https://blog.zeppelin.solutions/setting-up-a-bug-bounty-smart-contract-with-openzeppelin-a0e56434ad0e），并尽量保证它完整的测试过。

Whisper

Whisper（https://github.com/ethereum/go-ethereum/wiki/Whisper-Overview）是一个集成进以太坊的消息系统。它允许DApp发布小量的信息来进行非实时的消息通信。

它使用shh协议。尽管它已经有段时间没有更新了，这是一个使用Whisper协议实现一个聊天客户端的例子。链接：https://github.com/ethereum/meteor-dapp-whisper-chat-client。

去中心自动化组织

这是一个组织（就像，一群人），其中，使用代码来保证最终的强制执行，而不是使用传统的法律文件。这群人使用智能合约来做常见组织做的所有的事情，比如在某件事上进行投票，比如决定是否对什么进行投资等等。

副作用是决策，管理，以及对什么进行投资的结果将会不可改变的存储在区块链上。

之前slock.it创建了标准的DAO框架来说明这个理念。这里（https://github.com/slockit/DAO/）有对DAO概念的总览，以及如何使用框架来实现一个自己的DAO（这个项目由于bug被黑客攻击了）。

Aragon

Aragon（https://aragon.one/）也正在应对挑战，设计一个根据智能合约逻辑运作的公司，重点是创建一个可以接受投资，处理会计，支付雇员，分配股权，正如我们现在知道的完成每天的公司的业务。他们也实现了漂亮的DApp客户端来让他们的协议使用起来更为简单。

查看这里Aragon核心合约（https://github.com/aragon/aragon-core/tree/master/contracts）来更多的理解它是如何做的。

存储

IPFS&FileCoin

IPFS（星际文件系统）是一个协议，用来分发文件。你可以认为它是一个基于bittorrent和git这样概念的一个文件系统，文件可以定位，且是不可变的。IPFS以IPLD数据模型存储信息，它非常有趣，提供了一些特别的特性，你可以通过下面的说明了解一些。

这是一个新的协议，它有一个http的网关和文件系统适配器，这让你可以通过http，挂载整个互联网文件系统到你本地的盘/ipfs。IPFS还提供了一个寻址服务IPNS（星际命名空间），它允许可变的状态（需要注意的是在IPFS里的所有东西都是不可变的）。你甚至可以使用DNS TXT记录来定位到你的IPNS客户端，允许你生成用户友好的链接来指向到对应的数据。

FileCoin是Protocol Lab为创建一个去中心化的基于IPFS的存储市场的努力结果，

也就是向整个网络提供存储资源的激励层。FileCoin的共识协议没有使用浪费资源的工作量证明，而是使用了Proff of Replication和Proof of SpaceTime来保证每片数据被复制某个特定的拷贝数量且存储某个特定的时间。

你应该读一下IPFS的白皮书，FileCoin的白皮书，以及IPLD的规范。

相关链接

IPFS：https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

FileCoin：https://filecoin.io/filecoin.pdf

IPLD：https://github.com/ipld/specs/tree/master/ipld

由于当前FileCoin还没有上线，你可以使用当前的IPFS存储网络来运行html/css/js，并把它作为一个类似orbit-db的数据库。

Swarm

Swarm是一个去中心化的存储网络，集成于以太坊生态系统，作为第一阵营的项目，看看这里关于IPFS与这个项目的比较和优劣。但本质上，基本上是一样的，除了它们有不同的哲学，并在底层使用稍微不同的协议。

链接：https://github.com/ethersphere/go-ethereum/wiki/IPFS-&-SWARM

项目

Augur

Augur是一个去中心化的预测市场，让大家对于某个现实世界的事件进行对赌。一方面，用户在某个具体可以发生的事件上投注，一旦结果成真，它们赢得的代币有真正的价值。为了实现这个，你需要实现一个去中心化的先知协议，来输入现实世界中的信息，它使用REP协议代币来进行经济激励。

Augur白皮书：http://www.augur.link/augur.pdf；

第一时间获得Augur的最新进展，可以关注其Medium账号：https://medium.com/@AugurProject；

你还可以看看Augur项目的合约代码：https://github.com/AugurProject/augur-core；

以及了解下Augur Master Plan：https://medium.com/@AugurProject/augur-master-plan-42dda65a3e3d

Gnosis

Gnosis与Augur有类似的理念，也是一个去中心化的预测市场。这是项目的白皮书：https://gnosis.pm/resources/default/pdf/gnosis-whitepaper-DEC2017.pdf

以及与Augur项目的对比：https://medium.com/@akhounov/hopefully-impartial-comparison-of-gnosis-and-augur-f743d11d6d37。

0xProject

0xProject创建了一个交换代币的协议，以及一个DApp来实现这个协议。开发者可以创建一个基于它们自己的分布式应用创建交易所（技术上叫中继层），而用户也不用信任这些app就可完成交易，结算在区块链上完成。0x协议旨在使用离线的第三方来广播交易和管理订单（可以创建/更新/删除订单，而不用直接向Ethereum发送缓慢/昂贵的交易），但最终会使用Ethereum进行结算。

它们实现了场外交易，一个DApp使用这个协议来在用户之间直接交换代币。你可以在github上查看他们的合约。

链接：https://github.com/0xProject/0x-monorepo/tree/development/packages/contracts

Swap

ConsenSys的Swap协议也是非常类似的，但更专注于p2p的直接交易（而不是基于订单表），这里有一个白皮书（https://swap.tech/pdfs/SwapWhitepaper.pdf），可以看看，这里有一个关于Swap协议的介绍（https://blog.airswap.io/introducing-swap-a-protocol-for-decentralized-peer-to-peer-trading-on-the-ethereum-blockchain-d4058f3179cf）。

Bancor

代币的流动性是相对来说在加密币的生态中是一个非常大的问题。在用户间的交易

需要满足买方和卖方两边的想法。

Bancor是一个协议，可以让你的代币:

1.可以根据订单自动给予价格；

2.可以通过持有其它的代币作为抵押器来即时创造流动性。

Open Zeppelin & zeppelinOS

Zeppelin Solutions是一个科技公司，在这个领域内正完成一些伟大，而且专业的事。它们实在做了太多事，太难一一说清了。

他们管理了Open Zeppelin，一系列经过审查的，最佳的智能合约实践，你可以继承并应用于你自己的DApp中。你可以查看他们的github资源来学习更多。你应该读一下里面的每一个合约。

他们坚持代码复用的理念，然后进一步创建了Zeppelin OS。你可以忽略OS，它不是传统意义上的操作系统的概念。zeppelinOS特性，工具和服务的集合，旨在提供稳固的开发人员体验，同时最大限度地提高智能合同安全性。

zeppelinOS中的其中一部分是“zeppelinOS Kernel”。其实他们不是传统意义上的核心，而且是一组库。它们是通过库模型实现的可升级的智能合约，可以在出现安全问题时独立的进行更新。因为你在合约内包含的代码更少，部署也将花费更少的gas，而开发者也减少了代码的重复。

zeppelinOS还有一些其他整齐的规划，比如调度程序（智能合约的异步执行，因为默认合同一般不会主动触发某个行为），市场级的协议和链下开发者体验工具。

社区

以太坊爱好者（http://ethfans.org/）是目前最好的以太坊中文技术社区，持续推广和普及以太坊的技术，帮助以太坊释放区块链和智能合约的潜力，并为开发者提供更好的平台和机会。

最后

区块链技术现在还在快速发展之中，显然，这篇文章将非常快的过时，所以如果某个协议，平台，技术，或团队，你非常喜欢，你可以告诉我，我考虑将他们加到文

章内。