

硕 士 研 究 生 读 书 报 告



题目 以太坊智能合约

作者姓名 李江浩

作者学号 21651082

指导教师 李启雷

学科专业 移动互联网与游戏开发

所在学院 软件学院

提交日期 二○一七年一月

Contracts In Ethereum

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: Li Qilei

By

Li Jianghao

Zhejiang University ,P.R. China

2017

摘要

随着网络时代的高速发展，区块链作为一种数字货币的底层技术，越来越多的人开始关注这种技术。本文重点探讨了智能合约在区块链中的使用。主要从智能合约的简介，区块链的介绍，区块链与智能合约的关系，以及如何进行以太坊合约参数编码等问题进行了阐述。

**关键词**：区块链，智能合约，以太坊

Abstract

With the rapid development to the Internet,blockchain is the underlying technology as a kind of digital currency and more and more people are paying attention to this technology.This paper mainly discusses the intelligent contract based on blockchain, including the introduction of intelligent contracts and blockchain, the relationship between contracts and blockchain, and the coding function of Ethereum Contract.

**Keywords：**blockchain, contracts, Ethereum Contract

1引言

随着计算机软件项目的规模越来越大，软件工程的理念已深入人心，人们也越来越认识到软件质量的重要性，在这种情况下，软件需求作为软件工程的一个阶段，就显得尤为重要了。如果软件开发人员与客户不进行充分的交流与沟通，没有就产品的功能性需求和非功能性需求达成共识，就匆匆开始着手编写代码，其结果很可能是既满足不了用户的需求，又浪费了人力物力。

为了避免以上问题的出现，我们必须重视需求分析，并且以需求工程的角度分两个方面软件需求开发、软件需求管理来具体探讨。

**2智能合约与区块链概念**

智能合约这个术语至少可以追溯到1995年，是由多产的跨领域法律学者NickSzabo提出来的。他在发表在自己的网站的几篇文章中提到了智能合约的理念。他的定义如下：“一个智能合约是一套以数字形式定义的promises，包括合约参与方可以在上面执行这些承诺的协议。”如果对定义中的信息进一步地探讨，将明确一下一些概念：

承诺

一套承诺指的是合约参与方同意的（经常是相互的）权利和义务。这些承诺定义了合约的本质和目的。以一个销售合约为典型例子。卖家承诺发送货物，买家承诺支付合理的货款。

数字形式

数字形式意味着合约不得不写入计算机可读的代码中。这是必须的，因为只要参与方达成协定，智能合约建立的权利和义务，是由一台计算机或者计算机网络执行的。

协议

协议是技术实现（technicalimplementation），在这个基础上，合约承诺被实现，或者合约承诺实现被记录下来。选择哪个协议取决于许多因素，最重要的因素是在合约履行期间，被交易资产的本质。再次以销售合约为例。假设，参与方同意货款以比特币支付。选择的协议很明显将会是比特币协议，在此协议上，智能合约被实施。因此，合约必须要用到的“数字形式”就是比特币脚本语言。比特币脚本语言是一种非图灵完备的、命令式的、基于栈的编程语言，类似于Forth[1]。

区块链技术起源于2008年中本聪《比特币：一种点对点电子现金系统》。比特币的概念就此诞生，比特币的最大特点是去中心化。使用比特币人们无需借助银行就能完成支付，最近几个月来金融巨头们突然开始关注比特币背后的技术，并且把这种技术用在了非货币领域，比如股票交易、选举投票等等。这种技术就叫做区块链，一种实时记录全部交易的去中心化公开数据库，在区块链上进行支付时全网计算机共同查询区块链数据，共同验证这笔支付交易是否有效。确认支付后将写入区块链并产生一条不可篡改记录。区块链是一种几乎不可能被更改的分布式数据库，“分布式”不仅体现为数据的分布式存储，也体现为数据的分布式记录（即由系统参与者共同维护）。这里以以太坊智能合约为例再说明一下

以太坊智能合约基本概念

公钥加密系统。Alice有一把公钥和一把私钥。她可以用她的私钥创建数字签名，而Bob可以用她的公钥来验证这个签名确实是用Alice的私钥创建的，也就是说，确实是Alice的签名。当你创建一个以太坊或者比特币钱包的时候，那长长的0xdf...5f地址实质上是个公钥，对应的私钥保存某处。类似于Coinbase的在线钱包可以帮你保管私钥，你也可以自己保管。如果你弄丢了存有资金的钱包的私钥，你就等于永远失去了那笔资金，因此你最好对私钥做好备份

点对点网络。就像BitTorrent,以太坊分布式网络中的所有节点都地位平等，没有中心服务器。

区块链。区块链就像是一个全球唯一的帐簿，或者说是数据库，记录了网络中所有交易历史。（前文已提过）

以太坊虚拟机(EVM)。它让你能在以太坊上写出更强大的程序（比特币上也可以写脚本程序）。它有时也用来指以太坊区块链，负责执行智能合约以及一切。

节点。你可以运行节点，通过它读写以太坊区块链，也即使用以太坊虚拟机。完全节点需要下载整个区块链。轻节点仍在开发中。

矿工。挖矿，也就是处理区块链上的区块的节点。

工作量证明。矿工们总是在竞争解决一些数学问题。第一个解出答案的(算出下一个区块)将获得以太币作为奖励。然后所有节点都更新自己的区块链。所有想要算出下一个区块的矿工都有与其他节点保持同步，并且维护同一个区块链的动力，因此整个网络总是能达成共识。

以太币：缩写ETH。一种你可以购买和使用的真正的数字货币。

Gas：在以太坊上执行程序以及保存数据都要消耗一定量的以太币，Gas是以太币转换而成。这个机制用来保证效率。

DApp：以太坊社区把基于智能合约的应用称为去中心化的应用程序(Decentralized App)。DApp的目标是(或者应该是)让你的智能合约有一个友好的界面，外加一些额外的东西，例如IPFS（可以存储和读取数据的去中心化网络，不是出自以太坊团队但有类似的精神)[2]。DApp可以跑在一台能与以太坊节点交互的中心化服务器上，也可以跑在任意一个以太坊平等节点上。(花一分钟思考一下：与一般的网站不同，DApp不能跑在普通的服务器上。他们需要提交交易到区块链并且从区块链而不是中心化数据库读取重要数据。相对于典型的用户登录系统，用户有可能被表示成一个钱包地址而其它用户数据保存在本地。许多事情都会与目前的web应用有不同架构。

**3智能合约与区块链关系**

智能合约一定要在区块链技术上实现吗？答案是否定的。以信用卡为例，就是信用卡的自动还款服务，我们可以把这个理解为一种智能合约。在具体的时间（信用卡还款日），党还款条件呗满足，计算机机系统就会自动完成这笔交易。然而这些服务仍是运行在传统的计算机系统之上，而这些系统并没有利用区块链技术。

既然传统计算机技术就可以实现自动交易的合约，那为什么这些应用没有广泛应用呢？那是因为还有一个问题没有解决，就是信任问题。在现实世界中，合约是真实存在的，签印之后人们才认定它有效；在计算机世界中，合约是记录在代码里的。因此数字化的合约，极有可能会有被篡改之类的到的风险，抑或是被黑客攻击的技术风险。回到之前的自动还款例子，如果银行的官方网站提供这项服务，大家也许可以接受，因为很多人相信银行，但是同样的服务，换成由淘宝新开的店铺提供，或是其他网站，从技术上来说，实现并不困难，但是很少有人会选择这种服务。因为人们不回相信提供服务的人，而且一旦出现纠纷，举证也是一件非常困难的事情，最重要的证据都保存在对方的计算机系统中，只是修改一个数据，“事实就可能改变”。

而区块链的出现，可以很好的解决这个问题。这要从区块链技术的几个基础特性讲起，区块链技术为我们带来了一个去中心化的，不可篡改的，高可靠性的系统，首先是不可篡改，这样就不需要担心合约的内容会被更改；其次是高可靠性，我们不用担心系统在条件满足时不执行合约；然后就是去中心和给我们带来的全网备份，完备的记录完全可以支持事后的审计。这样的系统，我们不需要去相信和我们签订合约的对方，只需要相信区块链系统会把剩下的事完成就可以了。这就是区块链技术带来的革命性变化——去信任。正是在去信任的环境下，智能合约才大有用武之地。

**4以太坊编写智能合约的DApp所需的基本工具**

编写和部署智能合约并不要求你运行一个以太坊节点。下面有列出基于浏览器的*IDE*和*API*。但如果是为了学习的话，还是应该运行一个以太坊节点，以便理解其中的基本组件，何况运行节点也不难。

运行以太坊节点可用的客户端

以太坊有许多不同语言的客户端实现（即多种与以太坊网络交互的方法），包括C++,Go, Python, Java, Haskell等等。为什么需要这么多实现？不同的实现能满足不同的需求（例如Haskell实现的目标是可以被数学验证），能使以太坊更加安全，能丰富整个生态系统。其他时候还会使用一个叫testrpc的工具,它使用了Python客户端[pyethereum](https://github.com/ethereum/pyethereum)。C++的客户端现在仍然在用其中的ethminer组件和geth配合挖矿，因此这些不同的组件是可以一起工作的。

关于挖矿：挖矿很有趣，有点像精心照料你的室内盆栽，同时又是一种了解整个系统的方法。虽然以太币现在的价格可能连电费都补不齐，但以后谁知道呢。人们正在创造许多酷酷的DApp,可能会让以太坊越来越流行。

交互式控制台。客户端运行起来后，你就可以同步区块链，建立钱包，收发以太币了。使用geth的一种方式是通过JavaScriptconsole,类似你在chrome浏览器里面按F12出来的那个，只不过是跑在终端里。此外还可以使用类似cURL的命令通过[JSONRPC](https://github.com/ethereum/wiki/wiki/JSON-RPC)来与客户端交互。DApp开发的流程并非这次报告的重点，因此这块就不多说了。但是我们应该记住这些命令行工具是调试，配置节点，以及使用钱包的利器。

在测试网络运行节点。如果你在正式网络运行geth客户端，下载整个区块链与网络同步会需要相当时间。（你可以通过比较节点日志中打印的最后一个块号和[stats.ethdev.com](http://stats.ethdev.com/)上列出的最新块来确定是否已经同步。)另一个问题是在正式网络上跑智能合约需要实实在在的以太币。在测试网络上运行节点的话就没有这个问题。此时也不需要同步整个区块链，创建一个自己的私有链就勾了，对于开发来说更省时间。

testrpc.用geth可以创建一个测试网络，另一种更快的创建测试网络的方法是使用testrpc.Testrpc可以在启动时帮你创建一堆存有资金的测试账户。它的运行速度也更快因此更适合开发和测试。你可以从testrpc起步，然后随着合约慢慢成型，转移到geth创建的测试网络上-启动方法很简单，只需要指定一个network id：geth--networkid"12345"。

接下来我们来谈谈可用的编程语言，之后就可以开始真正的编程了。写智能合约用的编程语言用Solidity就好。要写智能合约有好几种语言可选：有点类似Javascript的Solidity,文件扩展名是.sol.和Python接近的Serpent,文件名以.se结尾。还有类似Lisp的LLL。Serpent曾经流行过一段时间，但现在最流行而且最稳定的要算是Solidity了，因此用Solidity就好。

solc编译器。用Solidity写好智能合约之后，需要用solc来编译。它是一个来自C++客户端实现的组件（又一次，不同的实现产生互补），注意：以太坊正处于积极的开发中，有时候新的版本之间会有不同步。确认你使用的是最新的dev版本，或者稳定版本。如果遇到问题可以去以太坊项目对应的Gitter聊天室或者[forums.ethereum.org](http://forums.ethereum.org/)上问问其他人在用什么版本。

web3.jsAPI.当Solidity合约编译好并且发送到网络上之后，你可以使用以太坊的[web3.jsJavaScriptAPI](https://github.com/ethereum/wiki/wiki/JavaScript-API)来调用它，构建能与之交互的web应用。

**5以太坊智能合约之ABI参数编码**

在进行参数编码之前，一定要说明一下SHA-3。SHA-3之前名为Keccak算法，是一个加密杂凑算法，SHA-3并不是要取代SHA-2，因为SHA-2目前并没有出现明显的弱点。由于对MD5出现成功的破解，以及对SHA-0和SHA-1出现理论上破解的方法，NIST感觉需要一个与之前算法不同的，可替换的加密杂凑算法，也就是现在的SHA-3。

智能合约参数编码包括2部分：FunctionSelector+Argument Encoding。

对函数进行Keccakhash，获取到的hash取前4bytes（左边，高位在大端顺序中）作为function selector。例如function baz(uint32x,booly)，对这个函数，"baz(uint32,bool)"用ASCII编码的byte[]，然后Keccakhash，再取前4bytes，结果为0xcdcd77c0。

前4bytes是function selector，从第5个byte开始，跟着的是参数编码。参数的编码规则也适用于返回值的编码和事件参数的编码，而没有开头的4bytes指明函数。基本参数类型：uint<M>，int<M>，address，bool，fixed<M>x<N>，ufixed<M>x<N>，fixed, ufixed，bytes<M>，function，<type>[M]，bytes，string，<type>[]，其中bytes，string，任何类型的数组的参数类型称为动态类型，其他参数类型为静态类型，2大类型的编码方式有所不同。之后列举一些例子来说明常用的参数编码。假设有这样的2个函数function baz(uint32x,booly)returns(bool r)和functionsam(bytesname,boolz,uint[]data){}，并且调用function baz(uint32x,booly)参数69,true。以下是编码过程：

• 0xcdcd77c0:"baz(uint32,bool)"的ASCIIbyte[]的keccakhash的前4bytes

• 0x0000000000000000000000000000000000000000000000000000000000000045:第一个参数，uint3269填充到32bytes

• 0x0000000000000000000000000000000000000000000000000000000000000001:第二个参数，bool true填充到32bytes

最后总的合成以下编码0xcdcd77c000000000000000000000000000000000000000000000000000000000000000450000000000000000000000000000000000000000000000000000000000000001

调用functionsam(bytesname,boolz,uint[]data)参数"dave",true,[1,2,3]

• 0xa5643bf2:"sam(bytes,bool,uint256[])"的ASCIIbyte[]的keccakhash的前4bytes

• 0x0000000000000000000000000000000000000000000000000000000000000060:第一个参数是动态类型，此处记录第一个参数实际数据开始的位置，距离开头0x60bytes偏移（functionselector4bytes不计入）

• 0x0000000000000000000000000000000000000000000000000000000000000001:第二个参数，bool true

• 0x00000000000000000000000000000000000000000000000000000000000000a0:第三个参数是动态类型，此处记录第三个参数实际数据开始的位置，距离开头0xa0bytes偏移（functionselector4bytes不计入）

• 0x0000000000000000000000000000000000000000000000000000000000000004:第一个参数的实际数据从这里开始，此处为byte array的长度，4

• 0x6461766500000000000000000000000000000000000000000000000000000000:第一个参数的内容，"dave"的utf8编码，右侧填充至32bytes

• 0x0000000000000000000000000000000000000000000000000000000000000003:第三个参数的实际数据从这里开始，此处为uint256array的长度，3

• 0x0000000000000000000000000000000000000000000000000000000000000001:第三个参数的第一个元素，uint2561填充到32bytes

• 0x0000000000000000000000000000000000000000000000000000000000000002:第三个参数的第二个元素，uint2562填充到32bytes

• 0x0000000000000000000000000000000000000000000000000000000000000003:第三个参数的第三个元素，uint2563填充到32bytes

最后总的合成以下编码[3]

0xa5643bf2

0000000000000000000000000000000000000000000000000000000000000060

0000000000000000000000000000000000000000000000000000000000000001

00000000000000000000000000000000000000000000000000000000000000a0

0000000000000000000000000000000000000000000000000000000000000004

6461766500000000000000000000000000000000000000000000000000000000

0000000000000000000000000000000000000000000000000000000000000003

0000000000000000000000000000000000000000000000000000000000000001

0000000000000000000000000000000000000000000000000000000000000002

0000000000000000000000000000000000000000000000000000000000000003

**6智能合约的应用**

在以太坊中有两类账户，它们共用同一个地址空间。

外部账户，该类账户被公钥-私钥对控制。合约账户，该类账户被存储在账户中的代码控制。外部账户的地址是由公钥决定的，合约账户的地址是在创建合约时确定的，账户以地址为索引，地址是由公钥衍生而来的，取公钥的最后20个字节。每对私钥／地址都编码在一个钥匙文件里，两类账户唯一区别是：合约账户存储了代码，外部账户则没有，每个账户都有一个key-value形式的持久化存储。key，value的长度都是256-bit。智能合约可以应用于一些交易之中，一笔交易是一条消息，从一个账户发送到另一个账户。交易可以包含二进制数据（payload）和以太币。如果目标账户包含代码，该代码和输入数据会被执行。如果账户目标账户是零账户（账户地址是0），交易将会创建一个新的合约。创建合约交易的payload被当作EVM字节码执行。执行的输入作为合约代码被永久存储。因此要在移动端结合区块链技术，使用智能合约来开发APP，和传统的APP开发不同，需要根据secp256k1算法随机生成一个32字节构成的私钥，交易hash值通过私钥进行加密，会获得64字节的数字签名。同时，可由私钥生成65字节数据，取低位的64字节作为相应的公钥，再由公钥生成当前账户的地址。通过github上secp256k1的开源代码，封装出libsecp256k1.a，调用intsecp256k1\_ecdsa\_sign\_compact(const secp256k1\_context\_t \*ctx, constunsignedchar \*msg32, unsignedchar \*sig64, const unsigned char \*seckey, secp256k1\_nonce\_function\_tnoncefp, const void \*ndata,int \*recid)接口生成签名，调用int secp256k1\_ec\_pubkey\_create(const secp256k1\_context\_t \*ctx, unsigned char \*pubkey,int \*pubkeylen,const unsigned char \*seckey,int compressed)接口生成公钥。

**7总结**

虽然智能合约在实践运用中存在一些问题，比如智能合约开发技术、监管标准不统一，智能合约风险管理及危机应对场景不完善，智能合约涉及的法律责任界定不明确等一系列问题，但是区块链智能合约技术以其独特的技术优势吸引了众多研究开发，特别是政府机构也开始重视区块链的发展，必将有一系列的政策来扶持发展，越来越多的行业将会应用智能合约加快行业的发展。

参考文献

[1]Florian Glatz. <https://medium.com/@heckerhut/whats-a-smart-contract-in-search-of-a-consensus-c268c830a8ad>

[2]jan. 以太坊智能合约编程之菜鸟教程 <http://ethfans.org/posts/101-noob-intro>

[3]"[EthereumContractABI](https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI)" https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI