小蚁智能合约 2.0 白皮书

该文档正在编辑中

该文档正在编辑中，我们会尽快完成，你可以在 Github wiki (https://github.com/AntShares/AntShares/wiki/) 上查看其它文档，或者来我们的 小蚁官方网站 (http://www.antshares.org) 逛逛。

小蚁是一个开源的社区项目，如果你感兴趣，你也可以通过 pull request 的方式来贡献开发文档，开发文档的项目地址为 github.com/AntShares/docs (https://github.com/AntShares/docs) ，感谢您的付出。

一、前言

智能合约是一套以数字形式定义的承诺，包括合约参与方可以在上面执行这些承诺的协议。区块链技术给我们带来了一个去中心化的，不可篡改的，高可靠性的系统，在这种环境下，智能合约才大有用武之地。智能合约是区块链最重要的特性之一，也是区块链能够被称为颠覆性技术的主要原因。它正使我们的社会结构发生日新月异的变革。

二、特点

2.1 确定性

如果一个程序在不同的计算机、或者在同一台计算机上的不同时刻多次运行，对于相同的输入能够保证产生相同的输出，则称该程序的行为是确定性的，反之则称该程序的行为是非确定性的。 区块链是一个通过多方存储、多方计算的方式来实现数据不可篡改、计算结果可信的分布式系统，智能合约会在区块链网络的多个节点中运行。如果一个智能合约是非确定性的，那么不同节点运行的结果就可能不一致，从而导致共识无法达成，网络陷入停滞。因此，在设计智能合约系统的时候，需要排除一切可能导致非确定性的因素。

2.1.1 时间

获取系统时间是一个很常用的系统函数，可能会被大量应用在一些具有时效性合约程序中。但是获取时间是一个非确定性的系统函数，不同的节点调用返回的结果也会不一致，在分布式系统中很难获得统一的精确时间。因此，小蚁智能合约只提供了基于区块时间戳的系统调用，可以将整个区块链看成一个时间戳服务器，并取得任意一个区块被构造时的时间戳。由于小蚁平均每 15 秒产出一个区块，那么合约运行时的时间大约等于最新的区块时间加上 15 秒左右。

2.1.2 随机数

很多智能合约程序会用到随机数的功能，比如对赌合约和一些小游戏等，但是随机函数是一个典型的非确定性函数，每次调用都会得到不同的结果。在分布式系统中，解决这个问题的办法有很多：可以让全体节点采用相同的随机种子，这样整个随机函数的返回序列都是确定性的，但是这种方法会提前暴露整个随机结果，使得这种随机数的实用价值大大降低；另一种方法是让全体节点进行通信以协作的方式来生成随机数，它可以采用一些密码学技巧来产生公平的随机数，但缺点是性能非常差，需要额外的通信开销；还有一种方法是引入一个中心化的随机数提供者，由它来提供随机数并保证一致性，但是这种方法的缺点是显而易见的，用户必须无条件地信任这个提供者。 运行在小蚁上的智能合约有两种方式来获取随机数：第一是每个区块在被构造时，共识节点都会对一个随机数达成共识并填充到区块的 Nonce 字段中，合约程序可以读取到任意区块的 Nonce 字段；第二是合约程序可以利用区块的散列值作为随机数的生成手段，由于区块的散列值具有一定的随机性，这种方式可以得到一个较弱的随机数。

2.1.3 数据源

如果一个程序在运行时获取数据，而数据源提供的是非确定性的数据，那么该程序也可能会变成非确定性的程序。例如，通过搜索引擎来获取某个关键词的前 10条搜索结果——搜索引擎针对不同的 IP 地址来源可能会返回不同的排序结果。 小蚁向智能合约提供了两种确定性的数据源： 1) 区块链账本 合约程序可以通过交互服务来访问到整个区块链上的所有数据，包括完整的区块和交易，以及它们的每一个字段。区块上的数据都具有确定性和一致性，所以可以安全地被智能合约访问。 2) 合约存储空间 部署在小蚁上的每一个应用合约都有一块仅可由该合约本身来存取的存储空间，小蚁的共识机制确保了每一个节点上的存储状态都是一致的。 对于需要访问链外数据的情况，小蚁没有提供直接的方式，需要通过交易来将链外数据发送到链内，从而转化成以上两种类型的数据源，才能被智能合约所访问。

2.1.4 合约调用

小蚁的智能合约具有相互调用的能力，但不能递归调用。递归可以在合约内部实现，但不能跨越当前合约的边界。此外合约之间的调用关系必须是静态的：即无法在运行时指定调用的目标。这样做使得程序的行为在运行前就可以被完全确定，其调用关系也可在运行前完全确定，以此为依据可以对多个合约进行动态分区，从而实现并行化执行的能力。

2.2 高性能

智能合约的执行环境会对合约的性能起到非常重要的作用。当我们分析执行环境的性能时，有两个指标是非常关键的：第一是指令的执行速度，第二是执行环境本身的启动速度。对于智能合约而言，执行环境的启动速度往往要比指令的执行速度更为重要。智能合约中较多是一些甚少涉及 IO 操作的逻辑判断指令，这些指令的执行速度很容易得到优化。而智能合约每次被调用，都必须启动一个新的虚拟机/容器。因此执行环境本身的启动速度（启动一个虚拟机/容器）对智能合约系统的性能影响更大。小蚁采用了轻量级的 AVM（Antshares Virtual Machine）作为其智能合约的执行环境，它的启动速度非常快，占用资源也很小，适合像智能合约这样短小的程序。通过 JIT（即时编译器）技术对热点智能合约进行静态编译和缓存可以显著提升虚拟机的执行效率。

2.3 扩展性

2.3.1 高并发与动态分区

当谈及一个系统的扩展性时，会涉及到两个方面：垂直扩展和水平扩展。垂直扩展是指对处理流程进行优化使得系统能够充分利用现有设备的能力，这种方式很容易就会碰上天花板，一个串行系统的处理能力取决于单台设备的硬件极限。当我们需要对系统进行扩展时，如果有办法将串行系统改造成并行系统，那么理论上我们只需要增加设备的数量，就可以获得近乎无限的扩展性，这种方式就是横向扩展。我们在考虑对区块链系统进行扩展时，是否有无限扩展的可能？换言之，区块链能否并行地对业务进行处理？ 区块链是一个分布式的大账本，里面记录了各式各样的状态数据，同时也记录了这些状态如何变化的规则，智能合约正是用来记录这些规则的载体。区块链能否并行地对业务进行处理，就取决于多个智能合约能否并发执行——即合约的执行是否是顺序无关的。从根本上来说，如果合约之间不会相互影响，或者说如

果合约不会同时对相同的状态数据进行修改，那么它们的执行就是顺序无关的，可以并发执行，否则就只能串行执行，无法横向扩展。 基于上面的分析，我们可以很容易设计出一个具备"无限扩展"能力的智能合约系统。只需要简单地规定： 1) 一个智能合约只能修改属于该合约自己的状态记录； 2) 同一个事务批次（区块）中，一个合约只能被运行一次； 这样一来，所有的智能合约之间都是顺序无关可以平行处理了。但是，如果"一个智能合约只能修改属于该合约自己的状态记录"，就意味着合约间无法相互调用，每个合约都是一个孤岛；如果"一个区块中，一个合约只能被运行一次"，就意味着用智能合约发行的某种数字资产在一个区块里只能处理一笔交易。这显然和"智能"二字的设计初衷大相径庭。毕竟合约间的相互调用，同一区块中多次调用同一个合约，都是我们想要的设计目标。 好在，小蚁智能合约之间的调用关系是静态的，无法在运行时指定调用的目标。这样做使得程序的行为在运行前就可以被完全确定，其调用关系也可在运行前完全确定。我们要求每一个合约都显式地申明自身可能会调用哪些合约，从而使运行环境能够在运行合约程序之前，先计算出完整的调用树，并根据这个调用树来对合约的执行进行分区，让可能会对相同状态记录进行修改的合约在同一个分区中串行执行，而分区与分区之间可以并发执行。

2.3.2 低耦合

耦合是指两个或两个以上的实体相互依赖于对方的一个量度。小蚁智能合约的系统采用低耦合的设计，合约程序在一个虚拟机（小蚁虚拟机）中执行，并通过交互服务层与外部通信。因此，对智能合约功能的绝大部分升级，都可以通过增加交互服务层的 API 来实现。

三、合约类型

3.1 验证合约

验证合约的作用是对合约账户中资产的所有权进行验证，当一笔交易要对某个账户中的资产进行转账时，需要执行该账户所对应的验证合约。验证合约可以接受一组参数（通常是数字签名或者其它判断条件），并在验证后返回一个布尔值，用于向系统表明本次验证是否成功。 用户可以事先将验证合约部署到链上，也可以在转账的过程中直接在交易中公布合约内容。如果事先将合约部署到链上，那么该合约不但可以用作验证合约，还可以被其它的合约调用。可以通过调用"自毁"的系统函数来从链上删除已经部署好的合约，删除后就无法再被使用。

3.2 函数合约

函数合约用于提供一些公共的或者常用的功能，以供其它合约来调用。它使得智能合约的代码可以被重用，从而使开发者可以编写出更复杂的业务逻辑。函数合约必须事先部署到链上才能被调用，并通过"自毁"的系统函数来从链上删除。

3.3 应用合约

应用合约向用户提供了一组相关的功能，并且可以保存自身运行的状态数据，还能够在不同的应用间相互调用。可以通过调用"自毁"的系统函数来从链上删除已经部署好的合约，删除后就无法再被使用，对应的存储区也会被删除。

四、虚拟机

4.1 虚拟硬件

小蚁虚拟机提供了虚拟硬件层来对智能合约的执行提供支持，它们包括： 1) CPU CPU 负责读取并按顺序执行合约中的指令，根据指令的功能进行流程控制、算数运算、逻辑运算等。未来可以对 CPU 的功能进行扩展，引入 JIT（即时编译器）的功能，从而提高指令的执行效率。 2) 调用栈 调用栈用于保存每一次函数调用时程序执行的上下文信息，以便在函数执行完毕并返回后，能够继续在当前的上下文中执行。 3) 计算栈 小蚁虚拟机中所有的运行时数据都存放在计算栈中，当执行不同的指令时，会对计算栈中的数据元素进行相应的操作。例如执行加法指令时，会将参与相加的两个操作数从计算栈中弹出，并将相加后的运算结果压入栈顶。函数调用的参数也必须按从右到左的顺序压入计算栈，函数执行完毕后可以从栈顶取回函数返回值。 4) 备用栈 当需要对计算栈中的元素进行调度或重新排列时，可以临时将计算栈中的元素存入备用栈，并在将来取回。

4.2 指令集

小蚁虚拟机提供了一套简单而实用的指令集，用于构造智能合约程序。按功能划分，主要包含以下几类： 1) 常数指令 2) 流程控制指令 3) 栈操作指令 4) 字符串指令 5) 逻辑运算指令 6) 算数运算指令 7) 密码学指令 8) 数据操作指令 值得注意的是，小蚁虚拟机的指令集中内建提供了一系列的密码学指令，如ECDSA、SHA 等算法，以优化智能合约中用到密码学算法时的执行效率。此外，数据操作指令直接对数组及复杂数据结构提供支持。

4.3 交互服务层

智能合约运行所在的虚拟机是一个沙箱环境，当合约需要访问沙箱外的数据或者将运行时数据持久化保存时，就需要用到交互服务层，它为虚拟机中运行的智能合约提供了对外沟通的媒介。小蚁在交互服务层中，向智能合约程序公开了一系列的系统函数和服务，合约可以像调用普通函数一样访问它们，所有系统函数都是可并发的，所以无需担心扩展性问题。

4.4 调试功能

通常，智能合约的开发过程是非常困难的，因为没有良好的调试和测试方法。小蚁在虚拟机层面提供了程序调试功能的支持，可以对合约代码设置断点，还可以单步、单过程执行等。得益于虚拟机与区块链之间的低耦合设计，可以很方便地将小蚁虚拟机与各种 IDE 直接集成，从而提供一个与最终生产环境一致的测试环境。

五、高级语言

5.1 C#, VB.Net, F#

开发者可以直接使用几乎任何他们擅长的高级语言来进行小蚁智能合约的开发工作。第一批被支持的语言是 C#、VB.Net、F#等，小蚁提供了这些语言的编译器和插件，用于将高级语言编译成小蚁虚拟机所支持的指令集。由于编译器会针对 MSIL（微软中间语言）来进行编译，所以理论上任何.Net 中的语言或者可被转译成 MSIL 的语言都可以直接被直接支持。 这些语言的开发者人数众多，且具有非常强大的集成开发环境，开发者可以在Visual Studio 中进行代码的编写、生成、测试、调试等一系列的开发工作，同时还可以利用小蚁提供的智能合约开发模板来进行快速入门。

5.2 其它语言

后续，小蚁将依据难易程度来分批次开发各种其它高级语言的编译器，可能会有这些语言： 1) Java 2) C, C++, GO 3) Python, JavaScript 未来会陆陆续续增加新的高级语言支持，使得 90%的开发者无需学习新的语言即可参与到小蚁智能合约的开发中来，甚至可将现有业务系统中的代码直接移植到区块链上。

六、服务

6.1 区块链账本

小蚁智能合约可以在运行时通过交互服务提供的系统函数，获取小蚁区块链的完整账本数据，包括完整的区块和交易，以及它们的每一个字段。具体可以查询到这些数据： 1) 区块链的高度； 2) 区块头、区块； 3) 交易； 4) 交易的类型、属性、输入、输出等； 透过这些数据，可以开发出一些有意思的应用，比如自动权益分红、基于智能合约的工作量证明等。

6.2 数字资产

通过交互服务提供的数字资产接口，智能合约不但可以查询到小蚁区块链上各种数字资产的属性和统计信息，还可以在运行时创建新的数字资产。由智能合约所创建的数字资产，可以在合约外部进行发行、转账、交易等操作，它们与小蚁上的原生资产一样，可以用任何与小蚁兼容的钱包软件来管理。具体的接口主要有这些： 1) 资产属性查询； 2) 资产统计数据查询； 3) 资产生命周期管理：创建、修改、销毁等； 4) 资产管理：多国语言的名称、总量变更、精度变更、管理员变更等；

6.3 持久化

每一个被部署到小蚁区块链上的智能合约程序，都会拥有一个只有该合约本身可以读写的存储区。智能合约对自己存储区中的数据拥有完全的操作权限：可以读取、写入、修改、删除。数据以键值对的形式来存放，并提供这些接口： 1) 遍历存储区中的所有记录； 2) 根据指定键返回特定记录； 3) 根据指定键来修改或写入新的记录； 4) 根据指定键来删除记录； 一般，合约只能对自己存储区中的数据进行读写，但有一个例外：当发生合约相互调用的时候，被调用合约可以通过跨域请求来访问调用者的存储区。一共有三种存储区域可以被请求： 1) 当前区域 2) 直接调用者的存储区域 3) 根调用者的存储区域 跨域请求使得小蚁智能合约可以实现更加丰富的库功能，这些库可以为调用者提供高度封装的数据管理能力。

七、费用

7.1 部署费用

小蚁区块链的分布式结构提供了高冗余的存储能力，对这种能力的使用不是无偿的。在小蚁区块链上部署智能合约需要支付一笔费用，这笔费用目前固定为500ANC，由系统收取并成为系统收益。未来可能会根据系统运行的实际情况来调整价格。部署在区块链上的智能合约可以被多次使用，直到该合约被部署者销毁。

7.2 执行费用

小蚁区块链为智能合约提供了一个可信的执行环境，而合约的执行也需要消耗各个节点的计算资源，因此用户需要为智能合约的执行支付费用。费用的数量由每一次执行所消耗的计算资源决定，计价单位也是 ANC，由小蚁区块链的共识节点收取。如果执行过程中由于费用不足导致智能合约执行失败的，已被消耗的费用不会退回，这可以防止恶意攻击对全网算力的消耗。

八、应用场景

8.1 超导交易

区块链上的数字资产天生具有流动性的需求，通常点对点的交易无法提供足够高的流动性，因此需要由交易所来为用户提供交易服务。数字资产的交易所一般会分为两类：第一类是中心化交易所，用户需要将数字资产充值到交易所，然后在网站上挂单进行撮合交易；第二类是去中心化交易所，它的交易系统内置在区块链系统中，由系统提供撮合服务。 中心化交易所能够提供非常高的性能和多样化的服务，但需要有较强的信用保证，否则将会出现道德风险，比如挪用用户资金、跑路等。相对的，去中心化交易所虽然没有道德风险，但是用户体验较差，性能上也有较大的瓶颈。有没有办法将两者的优点结合起来，提供一个两全其美的方法？ 超导交易就可以做到这一点，它是这样一种机制：用户无需充值，就可以直接使用自己在区块链上的资产来进行交易，交易的结算在链上完成，但撮合的过程在链外进行，由一个中心化的交易所来提供撮合服务。由于撮合在链外，它的效率可以像中心化交易所一样高，但资产还在用户的控制之下，交易所可以根据用户的交易意愿来进行撮合服务，却无法挪用资金或者跑路，没有道德风险。 目前，小蚁社区中已经出现一些利用小蚁智能合约来实现区块链超导交易的项目，例如蓝鲸淘。

8.2 智能基金

8.3 跨链互操作

可以预见，在未来较长的一段时间内，世界上将会有若干公有链以及成千上万的联盟链和私有链共存。这些独立的区块链系统都是价值与信息的孤岛，彼此之间无法互联互通。通过跨链互操作机制，可以将无数个孤立的区块链连接起来，使得不同的区块链上的价值可以相互交换，形成真正的价值互联网。 小蚁智能合约为跨链互操作的实现提供支持，不但可以实现跨链资产交换，还可以运行跨链分布式事务，在不同区块链上运行智能合约，并保证它们的一致性。

8.4 预言机

一部预言机可以视为是与一个预言者（oracle）相连接的图灵机。所谓预言者的概念，是一个可以回答特定问题集合的一个实体。在区块链中，预言机为智能合约打开了外部世界的大门，使得智能合约可以将真实世界的信息作为合约执行的条件。 小蚁智能合约没有提供直接访问外部数据的能力，比如访问互联网上的资源等，因为这样做会引入非确定性，使得各个节点对合约执行的结果出现不一致。 在小蚁区块链中实现预言机，需要由可信的第三方将外部数据通过交易的形式发送到区块链上，使得这些信息成为账本数据的一部分，从而消除非确定性。 前文中提到的可信第三方，可以是一个由合约缔造者双方共同信任的人或机构，也可以是一个去中心化的由经济激励来保证的数据提供程序，可以用小蚁智能合约来实现这样一个预言机。

8.5 以太坊系列 DAPP

比特币开创了区块链及电子现金的时代，以太坊则开创了智能合约的时代。以太坊作为区块链智能合约的先驱，为智能合约系统的设计思想、经济模型、技术实现等方面均做出了巨大的贡献。同时，以太坊平台上也已经存在了大量的 DAPP（分布式应用程序），这些 DAPP 的功能各异，有对赌协议、数字资产、电子黄金、游戏平台、医疗保险、婚恋平台，覆盖各行各业。所有这些 DAPP，理论上都可以很方便地移植到小蚁的平台上，成为小蚁的应用。

九、探索

9.1 版本控制

智能合约需要部署到区块链上，才能够被分布式执行。合约代码一旦部署，就无法再修改，因为程序的行为需要在运行前就被用户所确定。如果合约代码可以被任意修改或升级，势必会带来一些道德风险：例如原本合约的行为是将资产转移到用户指定的账户，但却在有一天被"升级"为所有资产被转移到合约开发者的账户中去。由于无法在机制上防止这种类型的风险，目前小蚁智能合约一旦部署就无法再升级了，只能通知用户将资产转移到新的合约中，然后销毁旧合约。未来需要一种机制，能够在保证安全的情况下，让智能合约可以动态升级。

9.2 IPFS 资源访问

目前，小蚁系统上的智能合约只能访问自己存储区中的数据，未来可能会需要一种合约间共享的公共存储区，这个存储区通过哈希表的方式来检索文件，并分布式地存储在小蚁的各个节点上。也许可以通过 DHT（Distributed Hash Table，分布式哈希表）或者 IPFS（InterPlanetary File System）来实现。