Corda：分布式账本

由恒生研究院旗下区块链技术社区 51chain 真诚翻译

原著：麦克. 赫恩

2016 年 11 月 29 日

0.5 版本

摘要

一种具有节点间最小信任机制的无中心数据库技术，允许创建一个全球的分布式账本。这样的账本在金融、贸易、供应链跟踪和更多领域都有许多应用场合。我们提交的 Corda ,是一种去中心的全球数据库技术 ,它详细描述了如何实现一个去中心的应用开发平台的目标。在《Corda:介绍》一文中 ,我们阐述了它的高层描述 ,并提供详细的技术讨论。

注：本文介绍了 Corda 的设计意图。现在此时的参考实现并没有完成所描述的一切。

目录

因为排版需要，本页故意留白

1. 介绍

在许多行业 ,为了保持组织机构间特定数据库的彼此同步 ,需要付出大量的努力。在金融机构，通过对账、结算、清算来保证不同机构数据库之间的业务数据同步和解决不一致性，需要花费大量的工作量，例如这是一个银行实际整个工作中的主要部分。

为什么不直接使用一个共享的关系数据库呢？当然，只使用现有技术，肯定会解决许多问题，但它也会带来比答案更多的问题：

- 谁来运行这个数据库？我们将在哪里找到足够的供应天使（投资人）去拥有它？
- 它将在放在哪些国家？有什么措施可阻止这些国家滥用这些大量敏感信息？
- 如果它被黑客攻击该怎么办？
- 真的可以扩展关系数据库以适应整个金融系统吗？
- 如果金融系统需要停用以便去维护，会发生什么？
- 会有什么样噩梦般的 IT 官僚来守护数据库结构的更改？
- 您将如何管理访问控制？我们可以想象出许多其他问题。去中心的数据库可试图回答这些问题。

本文中我们对比去中心数据库和分布式数据库。分布式数据库，就像一个 BigTable， 通过将数据分布在多台计算机上以实现规模化的大型数据集和数据量。但它假定所有计算机都是由一个单一组织运行 ,组成数据库的节点互相信任，相互之间不做违规或泄漏数据的事。

而一个去中心的数据库系统，如基于比特币的系统，它们的节点之间为较弱的信任假设，会积极地检查互相的工作。这样的数据库更具有安全的交易特性和可用性，并且被广泛接受。

Corda 是一种具有以下新特点的去中心数据库平台：

- 新交易类型可以使用 JVM 字节码定义。

- 交易可以在不同的节点上并行执行，无需任何节点知道对方的交易。
- 节点部署在经认证的对等网络中，节点间的所有通信都是直接的。
- 它没有使用区块，通过可插拔的公证人模块来解决交易冲突。一个单一的 Corda 网络可能包含多个公证人，它们利用各种不同的算法提供担保。因此 Corda 不依赖于任何特定的共识算法。
- 数据基于"need-to-know"进行共享。节点之间提供一个交易的依赖关系图，数据仅根据需要才发送到另一个节点，但不进行所有交易的全局广播。
- 字节码-字节码传送机制，支持进行复杂的、多步交易的建模 – 流程建模。块代码可转换为带检查点的异步状态机，在发送和接收消息时将检查码写入节点的后台数据库。在一个节点上可以会有百万计的流同时活跃，并且它们可持续几天、跨节点重启甚至升级。

流程进度信息公开给节点管理员和用户，并可以与其他人及其他节点进行交互。一个流程库提供开发者复用通用流类型如公证、隶属广播等。

- 数据模型允许将任意对象图存储在分类帐本中。这些图叫做"状态"，它是数据的原子单位。
- 节点由关系数据库和放在账本上的数据支持，由于对象图定义中的插口可保留用于连接键，这些数据可以使用 SQL 语句查询，如同私有表关联查询一样。
- 平台提供了丰富的类型系统，如日期、货币、法律实体和金融实体，具体如现金、发行、交易等等。
- 状态可声明一个关系映射，并可通过 SQL 进行查询。
- 它从一开始就考虑与现有系统的整合。网络可以支持从其他数据库系统快速批量地导入数据，而无需在网络上加载。分类帐本上的事件是通过嵌入式 JMS 兼容的消息代理来暴露。
- 状态可以声明预定事件。例如若不及时偿还，债券状态可能会自动转换为"违约"状态。

Corda 遵从重用现有尽可能成熟的软件系统和基础设施的原则，并提供和比特币和以太坊（Ethereum）的全面比较。

2. 概述

Corda 是一个编写“CorDapps”的平台，它使用新功能将应用程序扩展到全局数据库上。这些应用程序定义了新的数据类型、新的节点间协议流和确定允许变更的“智能合约”。

什么是智能合约？这取决于我们正在谈论的计算模型。在去中心数据库应用中有两种相互竞争的计算模型：虚拟计算机模型和 UTXO 模型。以太坊使用虚拟计算机模型，它将数据库建模为一个全局计算机的内存对象图，由区块链决定单个线程的执行。比特币则使用 UTXO 模型，数据库由一系列不可篡改的数据行组成（这些行以哈希值为索引键）。交易定义了可追加新行的输出和消费现有行的输入。术语“智能合约”在每个模型中有不同含义。不同方法中关于权衡和术语的深入讨论，可参见 Corda 的介绍论文。

我们用的是 UTXO 模型，因此我们的交易结构就类似比特币的交易结构：它们有输入、输出和签名。和比特币不同的是：Corda 的数据库行可存放任意数据，而不仅仅是一个数值字段。因为交易消费和产生的数据不一定是一组键/值对，我们不用数据行而用状态来进行说明。和比特币相似的是，Corda 对象图用字节码程序关联，它必须接受交易是有效的。和比特币不同的是，程序必须使交易同时满足输入和输出状态。发行业务可以在不消费任何现有状态的情况下向数据库追加新状态，和比特币不同的是，这些交易不是特别的，可以由任何人随时创建。

与比特币和以太坊相比，Corda 不使用区块链来进行记录交易，即不使用矿工或工作证明机制。相反，每个状态可以指定一个公证人，公证人是一种服务，它保证只有当所有输入状态为未消费时，它才会签署一个事务。一个交易不允许同时多个公证人来消费状态，因此也就根本不需要公证人之间的两阶段提交。如果状态组合跨了多个公证人，则可用一个特殊的事务类型将它们首先移动到一个单一公证人处，详见第 7 章。

Corda 的交易格式还有各种其他功能，将在后面的章节中描述。

3. 端对端网络

3.1 网络概述

一个 Corda 网络包含下列组件：

- 节点，用基于 TLS 安全传输协议的 AMQP/1.0 进行通信。一般节点使用一个关系型数据库存储数据
- 许可服务，一个许可服务用于自动进行配置 TLS 证书
- 一个网络地图服务，用于在网络上发布网络上各节点信息
- 一个或多个公证服务，一个公证服务本身可能会分布在多个节点上
- 0 个或者多个 oracle 服务(神谕服务)。Oracle 服务是一个公认的服务，陈述一个事实而且该事实被认为是真时签署交易（完成交易）。oracle 服务还可以选择提供事实。这揭示了只要是完整并确定的事实则账本就能连接到现实世界，用于现实世界的应用。（举例说明： 在某一时刻的行情或外汇牌价通过 Oracle 服务提供并签名，数据可以做为交易数据结构的一部分）

系统还提供了一个基于内存实现的消息子系统，该子系统可以模拟节点之间的通信延迟和可视化节点间的通信。该功能在已下场景中很有用：用于调试和测试场景或者用于教学目的。我们将会在后续章节讲解 oracle 和公证服务。

3.2 身份认证和许可服务

和 Bitcoin 以及 Ethernum 不同的是，Corda 被设计为半私有网络，只有获得根证书颁发机构签署的认证身份才获准进入该网络。这种设想存在于网络各处，流程 API 提供的消息基于身份认证路由和分发到底层节点进行自动处理。该网络任何时候都不会进行全域广播方式传递消息。

这里的"身份"不是说必须是法律合法或者真实的身份。而是一个唯一的标识，例如一个 Email 地址具有全球唯一性的标识它归根到底来自 DNS 层次结构的顶层，所以 Corda 网络也可以使用任意自选的用户名。许可服务可以实现它喜欢的任

何策略，只要其签名身份具有全局唯一性。 如果有一个有效的 IP 混淆系统类似 Tor（著名开源网络匿名通信软件）就能让 Corda 网络完全匿名。

虽然简单的字符串标识可以满足一些网络的安全需求，金融行业一般还需要客户提供一定层级的校验信息，而且不同的合法机构、分支或者办公点可能共享同一个铭牌。Corda 重用了 PKIX（公钥基础架构组织）定义的基础架构，连接公钥用于身份识别，这样名字事实上符合 X.500(一系列计算机网络标准涵盖了电子目录服务，简单来说 X.500 基本上是用来查询有关人员的信息（如邮政地址、电话号码、电子邮件地址等）的一种服务。)规定的名称标准。当单个字符串就满足要求时"通用名"字段可以单独使用，类似 web PKI(web 公钥基础架构是一整套角色，规则以及创建、管理、分发、使用、存储、调用数字证书的过程，还有管理公钥加密) 在更复杂的部署附加的结构时，X.500 在用于区分相同名字的不同实体时很有用。例如 :有至少五个不同的公司都叫美国银行而且在更早的过去有超过 40 家独立的银行使用这个名字。

更复杂的关于以一些时变属性作为证据的身份认证概念不会在系统的这一层进行处理：基本身份认证总是使用 X.500 名称。注意，尽管消息总能被识别，交易自身可能仍然包含匿名公共密钥。

3.3 网络地图

每个网络都需要有一个网络地图服务，服务本身被包含在多个合作的节点组成中。这类似于 Tor 的目录授权的概念。网络动图通过发布节点的 IP 地址让网络内每个节点都能通过这些节点提供的身份认证服务相互访问到。当一个网络接入发生时，网络映射服务节点会检查接入节点是否在网络地图内。

网络地图可从基础 IP 地址抽象出更有用的商业概念，比如身份认证和服务。网络上的每一个参与者(我们称之为当事人 party)会在网络地图上发布一个或者多个 IP 地址，有的还会带上等效的域名。等效的域名在调试的时候会比较有用，但是这不是必须的。用户接口和 APIs 总是基于身份认证工作的，这和 Bitcoin 的地址概念（散列的公钥）不同，面向用户的一些应用一般基于字符串自动补全或者自动搜索用户 ID 而不是用二维码标识。

每个在网络里的节点启动的时候做得第一件事就是申请注册到网络地图，这样就可以订阅网络地图的动态变化状况。基于负载均衡和网络可视化目的节点可以有选择性地公告离自己最近的所在城市。

地图是一种可以被缓存并分发到全网络的一种文档。因此地图不用要求高可用性：如果地图服务处于离线状态，新接入节点就没法注册到该服务，已接入的节点也没法分发相关状态变化出去，不过，其他的会一切照常进行。

3.4 消息分发

网络结构上和 Email 架构类似。一般网络里的节点可以正常运转，但是有时候还是会因为崩溃、连接断开或者系统运维而暂时离线。消息会保存到硬盘并且重试发送，直到收到远端节点的消息应答。当远端节点发来希望消息已经被可靠地存储或得到完全处理的确认消息时本节点会把存储的消息分发出去。节点间的连接会视情况来决定是否建立或者断开：一般不假定连接是常连方式。理想状态下的网络应该是所有节点之间存在高质量的连接。但是 Corda 认识到这种设想和普通网络的置上并不现实。因此，一个节点的消息路由组件可以分离到防火墙外部署。不管是在防火墙外还是防火墙内的 DMZ 区，都是为了保证网络中节点之间的完全联通。这种情况下一个节点即使被分成多个非全双工连接的子服务也能作为第一类节点接入网络，另外，这种节点可能会有多个 IP。

本参考功能实现使用 Apache Artemis 嵌入式消息服务的消息代理来实现日志，负载均衡，流控制，高可用性集群，消息流内存过载处理等等有用的属性。网络使用了一种实现了二进制消息标准的 AMQP/1.0 消息协议并整合了 TLS 安全信息传递和端认证加密协议。

3.5 序列化，会话，去重和签名

所有信息均使用压缩二进制格式编码。每个消息的 AMQP 消息头里均包含一个用作唯一性键值的 UUID 值，这样意外重复发送的信息将被忽略。

消息还可以有一个相关的组织成 64bit 的会话 ID。注意，这不同于会话的 AMQP 概念。此会话可以长活并可以跨节点重启以及网络中断重连。会话因为组消息而存在，组消息是流的组成部分，下边将会详述。

节点成功处理的消息会生成一个已签名的确认消息我们称之为“收据 Receipt”。注意，这和存在 AMQP 级别的未签名确认信息不同，未签名确认只是简单标志出有消息成功从链路上下载了。收据(Receipt)可以在在处理消息后过一段时间再生成，可以在对消息进行确认的情况下分批生成收据以分摊签名开销，收据用内容散列值作为 ID。

使用收据的目的是节点和交易对手方有争议需要调解时提供一个无法抵赖的证据证明。Corda 不会试着去支持可抵赖的消息。

4. 流架构

4.1 概述

通常，分布式电子账需要一个复杂多方协议。比特币支付通道协议中，涉及两个当事人的资金，可以把资金放入多签名实体里，在消费时，与交易对手方共享使用该实体的交易，使用额外一个交易用于一方或者另一方失败时可以正常终止。这种协议通常涉及：可靠的私人信息传递，校验点信息硬盘存取，交易签名，p2p 网络交互，给用户的进度报告，维护一个复杂状态机用于记录下属状态：超时或者各种出错情况，以及可能的和内部系统的另外部分交互等。所有这些都是相关的。比特币支付通道的实现 java 库差不多有 9000 行代码，其中只有很少部分涉及了密码学内容。

另外一个例子，核心比特币协议只允许您将交易追加到帐本上（分类账）。传输一些其他的有用信息(比如：文本消息，退款地址，身份信息等等)是不支持的，因此这些附加信息需要使用其他的手段来处理----典型的方法是把这些需要传输的元数据封装在原始的账本交易信息数据里以及负责向接收者广播这种内嵌源数据信息的交易。比如比特币的 BIP70 支付协议就是这样的。

在 Corda 的交易中数据不是进行全球广播的，而是只有交易相关方需要看到交易数据时才会将数据传输给相关方。再举个更简单的用例：发送头寸时涉及到的交易对手方和相关第三方比如公证人之间发生的多步协商场景。未放入账本的一些额外信息被视为是必不可少的而不是可有可无。因此不像传统的区块连系统主要用全球广播方式传输信息，Corda 使用小型多方子协议我们称之为流的方式传输信息。

流框架提供了一个编程模型，该模型看上去就好像给开发人员提供了一种能力:启动成千上万个长生存期线程，可支持存活的节点重启甚至升级。我们提供了 APIs 用于发送和接收对象图给（从）网络上的标识，嵌入子流以及向订阅报告进度。这样业务逻辑就能在更高的层级上进行表示，将保证其可靠和有效的细节抽象出去。后续描述的组件用于完成这些功能:

即时状态机编译器。用阻塞模式编写的代码一般没法终止或者显性重启。第一次调用流的调用方法时，会发生类重写到实现为可恢复状态机模式的字节码到字节码的转换（应该指的是类序列化和反序列化）。这些状态机有时候被称为纤维或者协程，转换引擎 Corda uses(Quasar)能在栈的任意深度飞速重写代码。因此，开发者可能把他们的逻辑分拆成多个方法和类，使用循环以及一般把他们的程序搭建成好像是在单个阻塞方式线程里执行一样。只有很少几件事他们不该做的：等待/休眠，直接访问网络 APIs 或执行一些可能阻塞框架外其他任务的任务。

透明检查点。当一个流程希望等待接收来自其他方的或者用户输入的信息时，底层栈的数据帧会挂起在堆上，然后使用对象系列化框架序列化信息数据到节点的底层关系型数据库里去。写入的对象头上会附加小的表结构定义前缀，用于考虑对象布局可能的交叉变更，虽然栈布局跨变化可移植性会留待将来实现。流程会根据需求进行恢复或者暂停，这意味着同时会有很多流程在内存处于激活状态是可行的。检查点根据本地存储变化数据和网络消息确认进行原子级处理。

身份到 IP 地址的映射。流程是基于身份 ID 编写的。当流程是激活状态，框架负责根据给定的身份 ID 将消息路由到给定的正确 IP 地址并进行随后的处理。同时能够处理多主机时的负载均衡。

子流程库。流程可以调用子流程，我们还提供了一个子流程库用于自动执行一般任务比如：公证交易或原子化互换资产所有权。

进度报告。流程可以提供进行到哪一步的进度跟踪。可以使用人性化的进度步骤说明标签和其他类似进度条的标签数据。进度跟踪器是分层的而且相关步骤可以有子跟踪器给子流来调用。

流程急救中心。流程可以被暂停，如果流程抛出异常或者请求人明确介入。当一个流程处于停止状态时，会出现在流程中心，节点管理员可以选择终止流程或者提供其他解决方案。这种请求人工介入的处理方案在另外一方不能确定为什么你要联系他时是很有用的，比如：发送一个支付的具体原因并不清楚时，或者用于支付的资产不被认为可接收时。

流程使用反向 DNS 表示法命名，而且有些由基本协议定义。注意框架并没有要求实现原始协议，这仅仅是为了协助开发而给出的功能。

4.2 数据可见性和依赖解决

当一个交易作为流程的一部分呈现给节点时可能需要进行校验。例如只有你确信我拥有这些资金并且准备用来支付给您的时候，发送一条简单的信息："我正在支付给你 1000 英镑"，才有意义。检查交易有效性"由"ResolveTransactions"流程负责。该流程在交易事务图上执行广度优先搜索，下载任何缺失交易到本地存储中并对这些交易事务进行校验，至多一直检索到发行交易。如果交易事务的任何传递性依赖是无效的那么我们认为该交易无效。

一个节点在请求其他节点接受一笔交易时需要提供该交易完整的依赖图谱。这样去哪里检索交易相关数据就很清晰。因为交易总是在流程内进行通信，而且内嵌在解析流，必要的依赖关系会被自动从对各端抽取和校验。交易会围绕网络传播开来，因此不需要使用分布式哈希表。

这种方案有几个结论。一是高流动性资产比如现金的流转交易可能最终变成很长一条交易链上的一部分。一张图的末端解析行为可以涉及到多个往返，因此需要一些时间来完全完成。Corda 网络到底能多快完成发送支付是较难描述的: 这很大程度上取决于使用的频繁程度以及节点之间的距离。节点可以根据对交易是否将被提取的预估来提前预推交易以优化速度。这些工作将在未来实现。

虽然本系统比创建固化数据分区简单而且比全域广播提供了更好的隐私保护，在没有额外的隐私度量时，到底谁能看到交易数据也比较难确定。这种不确定性可以通过几个因子来减低。

小型子图交易。分类账的一些不涉及广泛流通资产的情形。例如：两个机构希望同时看到对一笔特定合同状态同步情况，并且支付使用帐外支付。，可以使用仅涉及对手方的交易。关于表内帐 vs 表外帐现金的讨论将在后续章节讨论进行。

交易隐私技术。Corda 支持各种交易数据隐私保护技术。如：随机化公钥以保证难以将交易链接到对应的身份识别上。Tear-offs 模式允许交易的一小部分暴露出

来供交易各方校验认证，(详见 5.7 章节 Oracles and Tear-offs)。在将来的版本中，系统安全硬件和（或）零知识证明可以用于让一方在不接触到底层的数据的情况下验证交易的有效性。

状态重发行。在一些状态表示特定发行人背书的资产的场景，并且发行人是可信任的执行原子操作，即使分类账不强制要进行原子操作 。这时状态可以简单从分类账上标注为"退出 Exit"然后重新发行（补发 re-issue）。因为退出交易和补发交易之间没有联系，这缩短了链。在实践中大多数高流动性资产的发行人（如银行）已经被授信执行高敏感事务,比有签名数据结构的可靠发行伙伴更加可靠。因此这种方式（状态补发）不会成为一个问题。

5. 数据模型

5.1 交易结构

状态是 Corda 中信息的原子单位。状态不会改变：要么是流通（“未被花费”）状态，要么是不再有效的被消费（“已被花费”）状态。交易会消费 0 个或多个状态（输入），并创造 0 个或多个新状态（输出）。由于状态不能在创造它的交易之外存在，所以状态的被消费与否，可以通过创造它的交易的标识符以及它在交易输出列表中的索引来鉴别。

交易由下列组件构成：

输入引用	指向交易消费的状态的（hash，输出索引）对。
输出状态	每个状态自己为新状态、为定义了它所允许的转换功能的合约、并最终为状态指定了公证人。
附件	交易指定了一个经排序的 zip 文件的 hash 值列表。每个 zip 文件会未交易包含代码、数据、证书或者辅助文档。合约代码在检查交易的有效性时有权限使用附件的内容。
指令	一个输入状态允许有多个输出状态。例如，一种资产可以被发行、被转移给账本上的新的所有者，或者在被所有者赎回之后从账本上退出、不再需要被追踪。一条指令本质上是传递给合约的一个参数，指定了比从被校验状态可获得的更多的所需信息（比如来自谕示服务的数据）。每条指令有一个关联的公钥列表。与状态类似，指令都是对象图。
签名	交易所需签名的集合等价于所有指令的公钥的并集。
类型	交易可以是普通类型交易，也可以是变更公证人的交易。针对每种交易类型的验证规则不同。
时间戳	如果被提供，那么一个时间戳定义了该笔交易可被认为已发生的时间范围。下文会对此进行更详细的讨论。
摘要	关于交易具体行为的文本摘要，由交易相关的智能合约进行检查。该域对安全签名设备（参见§10）十分有用。

由于签名被添加在交易的末尾，而交易是由用于签名的 hash 来识别的，所以签名的延展性不会成为一个问题。绝不会需要用 hash 来识别包括签名信息在内的交易。签名可以以并行的方式被生成和检查，它们也不会直接暴露给合约代码。实际上，合约会检查指令指定的公钥集合是否恰当，因为只有当每一条指令列出的每一个公钥都有一个相匹配的签名时，交易才会是有效的。公钥的结构是不透明的。这样一来，算法的灵活性就得到了保留：新的签名算法在部署时不需要调整智能合约本身的代码。

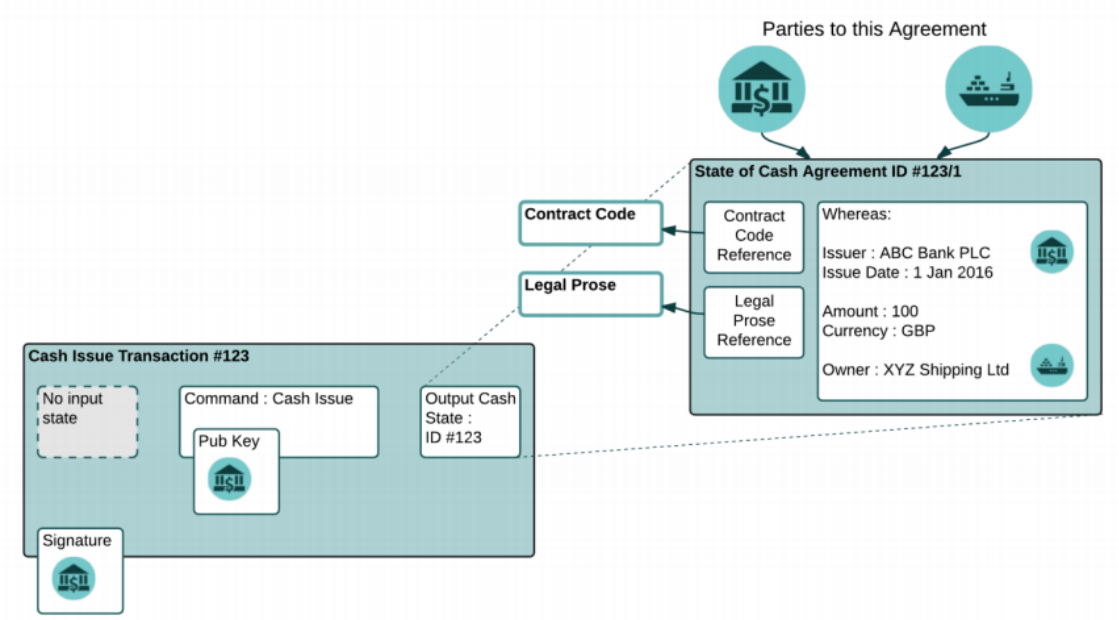

图 1：一个现金发行交易的例子

例子。在上图中，我们可以看到一个现金发行交易的例子。交易（左下）包含了 0 个输入，和一个输出，即新发行的现金状态。现金状态（右上扩展显示）包含了一些重要信息：1）被发行的现金的细节——总量、货币、发行方、所有者等等，2）合约代码，其 verify()函数负责对该发行交易和未来消费该状态的交易进行校验，3）一个包含了重要法律条文的文件的 hash，该文件为这个状态及其合约代码的行为提供了基本法律监管环境。

该交易还包含了一条指令，指明了该交易的目的是发行现金。指令还指定了一个公钥。现金状态的校验函数负责检查指令指定的公钥属于交易的参与方，这些参与方需要提供自己的签名使得该交易有效。在这个例子中，则意味着

verify()函数必须检查确认指令指定了一个与现金状态的发行者相对应的公钥。Corda 框架负责检查交易已经被所有指令列出的公钥所签名。这样一来，verify()函数只需要确保所有需要签名的参与方都已经被指令所指定，而框架则负责确保交易已经被指令列出的所有参与方签名。

5.2 复合密钥

术语"公钥"在上面的描述中实际上指的是一种*复合密钥* 。复合密钥是一种树，其树叶是附带了算法标识符的常规密码学公钥。树中的节点同时指定了它每个子节点的权重和它必须达到的加权阈值。一个签名集合的有效性可以通过这样的方式确认：从底往上行经这棵树，对其中所有具有有效签名的密钥的权重求和，并与阈值相比较。通过使用权重和阈值，可以编码多种多样的情况，包括使用 AND 和 OR 的布尔表达式。

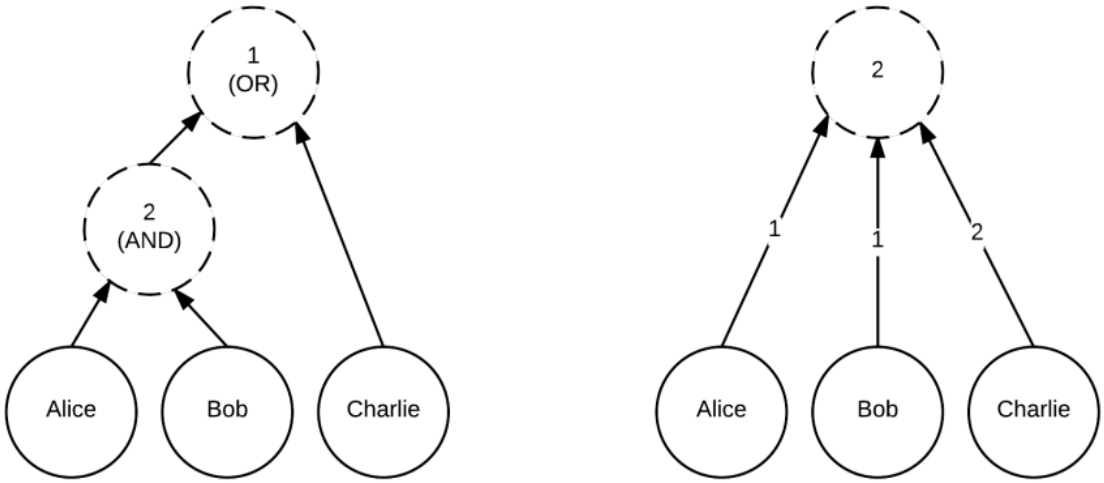

图 2：复合密钥举例

复合密钥可用在多种场景。例如，资产可以在一个 2 取 2 复合密钥的控制之下：一个密钥属于一个用户，另一个密钥属于一个独立的风险分析系统。当交易显得可疑，比如在一个很短的时间窗口内转移了太多价值时，风险分析系统将拒绝对交易签名。另一个例子涉及到将合作结构编码到密钥中，允许 CFO 可以独自签名一笔大额交易，但其下属却需要共同签署完成。复合密钥对于公证处也十分有用。一个分布式公证处的每个参与者由树的一片叶表示，特定的阈

值设定可以使得在部分参与者离线或拒绝签名的情况下，整个团体的签名仍然有效。

虽然在文献中已有可以精确地产生复合密钥和签名的阈值签名方案，但为了允许使用不同算法来混合密钥，我们选择了一种低空间效率的显式形式。这样一来，在逐步淘汰旧算法和采用新算法的过程当中，就不必要求团体中的所有参与者同时进行升级。

5.3 时间戳

交易时间戳指定了一个[start, end]时间窗口，可以断定交易的发生时间是在这个窗口之中。时间戳以窗口形式表示的原因是，在分布式系统中并不存在确切的时间点，而只有大量的没有共时性的时钟。这不仅是受到物理法则的影响，还由于共享交易的本性——尤其是如果对交易的签名需要多人授权的话，构造联合交易的过程可能会持续几小时或几天。

值得注意的是，交易时间戳的目的，是为了满足智能合约代码的逻辑强制性，而向合约代码传达交易在时间轴上的位置。虽然同样的时间戳可能还会被用于其它目的，比如监管报告或者用户界面上的事件排序，然而并没有要求像那样的方式使用时间戳，并且尽管会与其他参与者观察到的时间不能精确匹配，使用本地观察到的时间戳有时候是更好的选择。或者，如果需要时间轴上一个精确的点并且这个点必须被多个参与者认同，那么可以约定使用时间窗口的中间点。尽管这样不会精确地对应某个事件（如键击或者口头协议），这一方法仍然会有用。

时间戳窗口可以是开放的，用于传达某个交易的发生早于一个特定时间或晚于一个特定时间，但具体早或者晚多久并不重要。这样的用法类似于比特币交易的 **nLockTime** 域，该域指定了一个*在……之后发生* 的约束。

时间戳由公证服务执行检查。由于公证服务的参与者们本身也没有精确同步的时钟，所以一笔在给定时间窗口的边界提交的交易在被提交的瞬间是否被认为有效也是不可预料的。然而，从其它观察者的角度而言，公证处的签名是决定

性的：如果一笔交易拥有公证处的签名，则该交易就被假定已在给定的时间内发生。

基准时钟。为了在交易处于单个参与者的完全控制下时可以使用相对较窄的时间窗口，公证处被期望与美国海军天文台的原子钟进行同步。该原子钟的精确馈送可以从 GPS 卫星获得。注意，Corda 所使用的 Java 时间轴是以 UTC 时间表示，闰秒被包含在一天的最后 1000 秒中，因此每一天都准确包含 86400 秒。需要投入特别的关注以确保 GPS 中闰秒计数器的变化被正确处理，使其可以与 Java 时间保持同步。在设置交易的时间窗口时，必须留心处理用户与公证服务之间、公证服务内部消息传递的网络传播的延时。

5.4 附件与合约字节码

交易可以拥有若干数量的*附件*，并通过文件 hash 来识别附件。先前未出现过的新附件的保存和传送独立于交易数据，并且只能通过标准的解析流获取。

附件是一系列 zip 文件，且不能被合约代码单独引用。zip 包中的文件被一起折叠在单个逻辑文件系统中，重复的文件只在第一次被提到时解析。这一做法并非巧合，这正是 Java 类路径所使用的机制。

Corda 中的智能合约使用由《The Java Virtual Machine Specification SE 8 Edition》规定的 JVM 字节码来定义，一些小的差别会在后续章节描述。一个合约只是简单的一个实现了 **Contract** 接口的类，而 Contract 接口转而暴露单个叫 **verify** 的函数。verify 函数被传入一个交易，如果该交易被认为无效则抛出一个异常；否则函数返回，不带任何结果。被使用的 verify 函数的集合是每个状态指定的合约的并集（被表示为约束，参见§5.9）。Corda 规范中内嵌了 Java 规范，可以使得开发者编写多种不同语言的代码、使用经过良好开发的工具链，并复用已由 Java 和其它 JVM 兼容语言编写的代码。

Java 标准制定了一个全面的类型系统用于表示通用业务数据：时间和日历处理通过 **JSR310** 规范的一个实现提供，十进制计算既可以使用可移植浮点算法也可以使用提供的 bignum 库，等等。这些库经过商业 Java 社区多年的精心设计，将功能基于这一资源具有显著意义。

合约字节码也可以定义自己的状态 – 可以是任意对象图。由于 JVM 类并不是一种方便从非 JVM 平台进行协作的形式，所以可使用的类型被做了限制，并且提供了一个标准化的二进制编码方案。状态可以用一个小的标准化注释集合来给它的属性添加标签。这些将有利于控制状态被序列化成 JSON 和 XML（分别使用 **JSR367** 和 **JSR222**）的方式，有利于表达静态验证约束（**JSR349**），有利于控制状态被插入到关系型数据库的方式（**JSR338**）。这个特性在后续讨论。

附件也可能会包含提供给合约代码的数据文件。这些文件和字节码文件可以在同一个 zip 包中，也可以在另一个必须提供给被验证交易的 zip 包中。这类数据文件的例子可能包括货币类型定义、时区数据和公共假期日历。任何公共信息都可能以这样的方式被引用。附件是特意为那些在账本上会被许多参与者反复使用的数据而设的。数据文件被合约代码通过 API 获取，这些 API 与获取类路径上的文件时所用的 API 相同。平台对附件可包含的数据的种类和大小做了强制约束，以避免人为地在全局账本上放置不合适的文件（视频、PPT 等等）。

注意，是由交易的创建者选择被附加的文件。因此，状态对自己乐意接受的数据设置限制是一种典型做法。附件*提供*数据，但不对数据做*验证*，所以当存在有人会通过提供恶意数据来获取经济利益的风险时，必须有一个约束机制能够防止这样的事情发生。这根植在状态自己内部编码的合约约束里：一个状态不能仅仅只指定一个实现了 **Contract** 接口的类，还应当对提供给它的 zip/jar 文件设置约束。而这约束反过来又可以被用于确保合约对数据可靠性进行检查——或直接检查数据的 hash，或要求数据被可信赖的第三方签名。

5.5 硬分叉、规范与争议解决

不同的分布式账本系统通常在底层政治理念和技术选择上有所不同。以太坊项目最初承诺是可以实现"代码即律法"的"不可停止的应用"。在一个重要的智能合约被黑客攻击之后，由于缺少这段程序意图做什么的非代码形式的说明书，出现了关于发生的事件到底能不能被描述成黑客攻击的争论。分歧最终导致了社区内部的分裂。

因为 Corda 合约都是简单的 zip 文件，所以它很容易就能包含描述合约实际意图的 PDF 或其它格式的文档。并没有要求必须使用这个机制，也没有要求这些文档具有法律效力。尽管如此，在金融应用案例中，如果发生了分歧，那么把他们包含的法律意义上的合同比包含的软件实现更为重要。

编写一个不可升级的合约在技术上是可能的。如果这种合约管理一种只存在于账本上的资产，比如加密货币，那么这可以提供一种近似的"代码即律法"。我们把关于这个理念所蕴含的智慧的讨论留给政治学者和 reddit。

平台日志 在 Corda 中没有和区块链的"硬分叉"直接等价的机制，所以放弃问题交易链或欺诈交易链的唯一方法是在带外就抛弃一个完整的交易子图达成一致意见。既然不存在一个全局的可见性，这个一致的达成就不需要包括网络上的所有参与者：只需要包括那些可能已经接收并处理相关交易的参与者。缺少全局可见性的另一方面后果是没有单个点准确记录了谁见过哪笔交易。确定那些必须就抛弃一个子图达成一致意见的实体的集合，就意味着需要关联节点的活动日志。Corda 节点用日志记录了充分的信息，可以确保这样的关联可以实现。平台定义了一个任何人可用的流来协助这个过程。还提供了一个能生成"调查请求"并发送到一个种子节点的工具。流通知节点管理员，要求一个决策，并且充足的信息被传递到这个节点，用于尝试说服管理员进行参与（如一个签署的法庭指令）。如果管理员通过节点浏览器接受了这个请求，则交易链中后续的跳转被返回。这个工具以这样的方式半自动地抓取网络，找到所有会被提议的回滚操作所影响的参与者。平台不参与认定什么类型的交易回滚是正当的，在定位必须同意的参与方之外，只对实现回滚操作提供最小的支持。

一旦涉及到的参与者被确认，至少有两种策略可以修改账本。一种是使用简单修正数据库的交易扩展交易链，使其符合预期的现实。为了使这个方法成为可能，编写的智能合约必须在提交的签名达到充分的阈值时能够于正常业务逻辑之外被任意修改。这个策略简单，在状态包含的参与方数量较少且都没有在账本上遗留有害信息的动机时最为有意义。对于由盗窃或诈骗产生的资产状态，其包含的参与者会反抗所有以上述方法进行修补的尝试，因为他们可以在账本出错后、恢复到实际状态前的这段时间差里从现实世界获取利益。针对这种情况，需要使用一种更复杂的方法，即除去不合作参与者之外的所有参与者都同

意将相关状态标记为不再被消费/已被花费。这本质上是一种受限形式的数据库回滚。

5.6 身份查找

在所有受区块链启发的系统中，都有在想要知道跟自己做交易的是谁和不让别人知道这一点之间的紧张状态。一个标准的技术手段是在共享数据中使用随机化的公钥，并私下保存公钥与身份信息之间的映射。比如，为每一次接收付款都生成一组新密钥被认为是良好的习惯。这一技术利用了这样一个事实：校验账本的完整性并不需要准确地知道是谁参与了交易，而只需要知道他们都遵循已经达成一致的系统规则。

比特币和以太坊这样的平台拥有相对专门的机制来连接身份与密钥。通常，用户有义务使用从网站、店铺招牌等等收集的信息在钱包软件中手动给公钥添加标注。由于这些机制是专有的并且很枯燥，所以很多用户并不会去操心，这使得理解钱之后到底去了哪里变得困难。这还使得安全签名设备和风险分析引擎的部署变得复杂。比特币的 **BIP 70** 指定了一种使用连接到网络 PKI 的 **X.509** 证书来对"支付请求"签名的方式，这个方式提供了一种在密码学上安全的并且标准化的方法来获知你在跟谁进行交易。这个系统中的身份与在网络 PKI 中使用的相同：一个域名，电子邮件地址或者 EV（扩展验证）组织名。

Corda 更进一步地采用了这个理念。状态可以定义 **Party** 类型的域，该域封装了一个身份信息和一个公钥。当一个状态从交易中的原始形式被反序列化时，**Party** 对象的身份信息域为空，只有公钥（复合密钥）被提供。如果一个交易与连接短时公钥和长期身份密钥的 X.509 证书链共同被反序列化，那么身份信息域就被设置。这样一来，同一种数据表示方法就可以同时用在匿名场景，如校验交易的依赖性，和需要识别身份的场景，如与一个对手方直接交易。交易流包含的子流用于传输所使用的密钥的证书，这些证书之后被保存到数据库当中。而交易解析流并不传输这类数据，而是将交易保存在匿名保管链上。

确定性密钥派生 Corda 允许但不强制要求使用如 **BIP 32** 这样的确定性密钥派生方案。之所以基础架构不假定公钥间存在某种数学关系，是因为一些密码学

方案与这样的系统并不兼容。因此我们采取了使用 X.509 证书把短时公钥连接到长期密钥的有效方案。

5.7 谕示和数据剥离

有时候，在某种程度上向对手方透露交易的一小部分信息以允许他们检查签名并对交易签名，是很方便的。一个典型用例是*谕示(Oracle)*。谕示(Oracle)被定义为一个可信任的网络服务，对一个包含有关于账本外部世界陈述的交易进行签名，前提是该陈述必须是真实的。

下面是一些谕示可能会检查的声明的例子：

- 在一个特定时间，一支股票的价格为 X。
- 在一个特定时间，一个约定的利率为 Y。
- 一个特定组织是否已宣布破产
- 特定时间特定地点的天气状况

值得一问的是为什么一个智能合约不能简单地自己从网络服务中获取这些信息：为什么我们对谕示坚持这样的理念。这一做法的原因是，账本上的所有计算都必须是确定性的。每个人都必须能够检查一笔交易的有效性，并在任何时间（包括未来数年后）和任何种类的计算机上得到相同的结果。如果一个智能合约能够做读取系统时间或获取任意网页之类的事情，那么就有可能出现在某些计算机判定交易有效的同时，另外的计算机却判定交易无效（例如远程服务器掉线）的情况。要解决这个问题就意味着验证交易时所需要的所有数据都必须在账本上，而这反过来又意味着我们必须接受一些特定观察者的观点。否则，就会出现针对发生了什么事情的争论。

一种实现谕示的方法是让其对一个小的数据结构进行签名，然后该数据结构被嵌入到交易的某个地方（一个状态或指令当中）。但我们采取的是一种不同的方式：谕示对整个交易进行签名，不需要让谕示看到的数据则在交易发送之前被剥离。这一方式的完成，是依靠将交易构造成一棵 Merkle 树，被用于签名的 hash 则是树的根。通过将所需的数据元素和将它们连接到根 hash 的 Merkle 分支一起提交给对手方（如下图所示），对手方就可以在只能看到交易的部分数

据的情况下对整个交易进行签名。此外，让对手方确信某些第三方已经对交易签名，也是很简单的。通常提供给一个谕示的是包含有数据的指令或状态的 Merkle 分支和包含时间戳域的 Merkle 分支，仅此而已。作为结果的签名包含有标志位，用于指示结构的哪些部分被提交用作签名，避免单个签名涉及的内容超过预期。

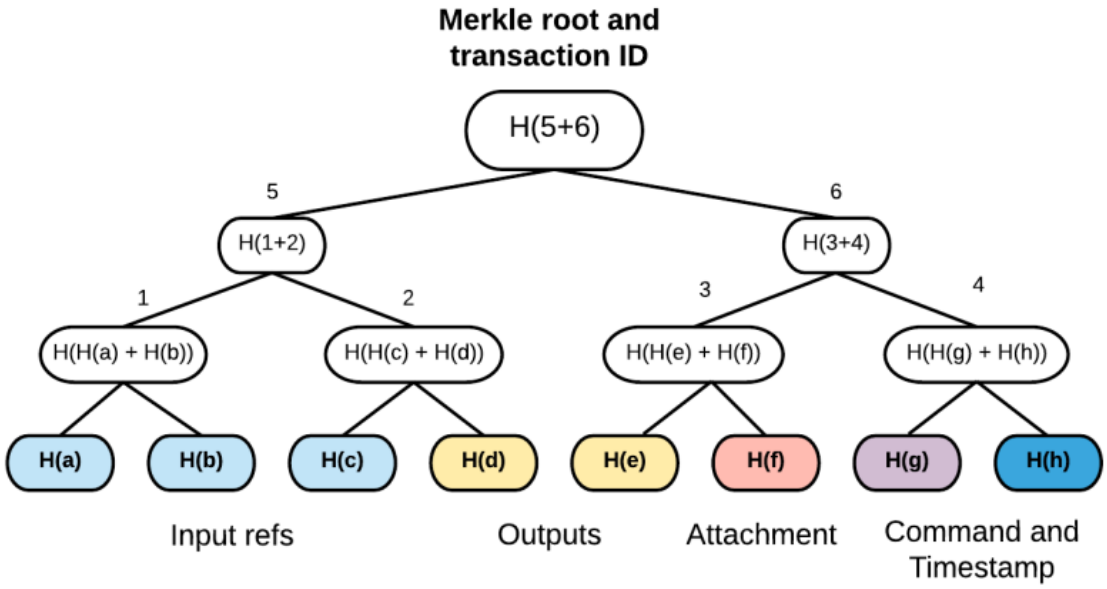

图 3：如何计算交易的身份标识 hash

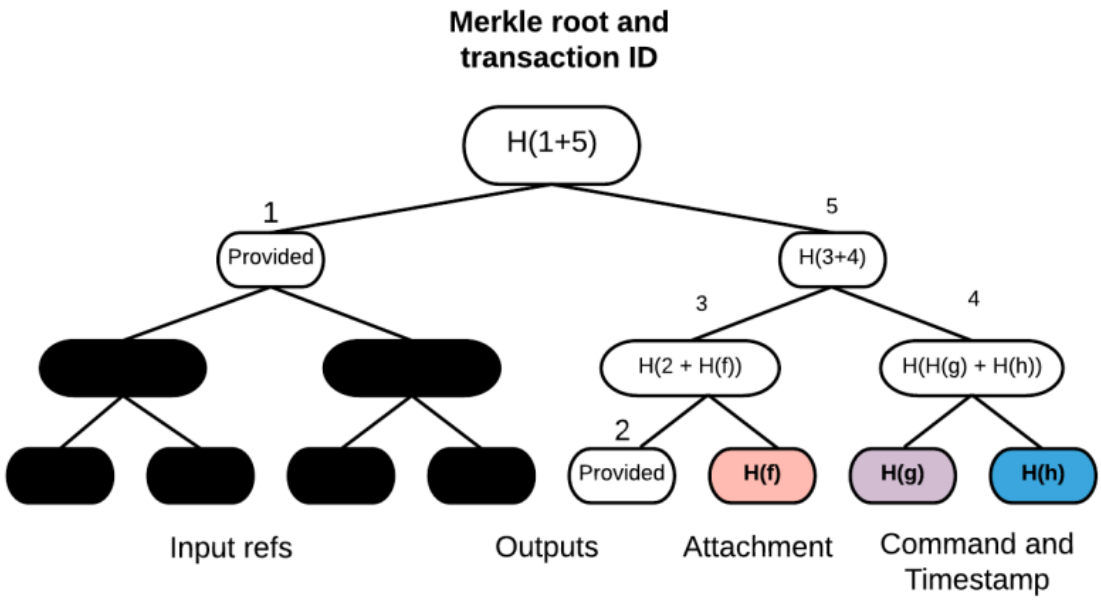

图 4：一条 Merkle 分支的结构

采用这种更加间接的方法的原因有两个。一是维持单个检查签名的代码路径。通过确保在一个交易中只有一个地方能够找到签名，可以很容易地实现算法的灵活性和并行/批量校验。如果一个签名可以出现在交易数据结构的任意地点，并且检验完全由合约代码自己控制（比如在比特币中），那么想要最大化签名检查的效率就变得更加困难。由于签名检查通常是区块链系统中速度最慢的部分之一，保留这样的性能就相当值得。

另一个原因是为了给谕示提供一个业务模型。如果谕示只是对陈述签名而没有其它内容，那么当只存在少数潜在的签名且确定这些陈述的真实性很昂贵时，经营谕示就会相当艰难。人们将会分享经过签名的陈述并在很多不同的交易中复用，这就意味着发行最初签名的成本会非常高，甚至高到无法运作。而由于是对特定交易做签名而非特定陈述，用户就无法自己共享签名（因为签名涉及的是一个在定义上一次性使用的结构），一个通过服务收费的谕示就可以将确认陈述真实性的成本分摊到很多用户身上。

5.8 留置处理

交易中的每个状态都会指定一个合约（布尔函数），该合约在被调用时以整个交易作为输入。为了使交易能够被视作有效，所有的合约都必须接受该交易。有时候我们会想要组合多个不同合约的行为。例如考虑一个"锁定期"的概念——状态的一个约束，防止状态在一个特定时间之前被修改（如卖出）。这是可以应用到很多种资产上的一段常规逻辑。在这种逻辑可以实现到一个库中并能够被所有想从中获益的合约调用，这要求所有的合约作者深思熟虑并将这功能包含进来。如果我们能够指示锁定期的逻辑在监管锁状态的合约的一起运行，那么就更好了。

考虑一份在到达某时间之前被期望冻结的资产。留置处理允许一个状态指定另一个必须在任何消费自己的交易中出现的状态。举个例子，一个时间锁合约能够定义一个包含锁的失效时间的状态，和一个简单的将这个失效时间与交易时间戳比较的合约。前述资产的状态可以被包含进一个不改变资产所有权但是在输出中包含一个时间锁状态的自消费交易当中。现在如果这个资产状态被使用，那么时间锁状态也必须被使用，这就会触发时间锁合约的执行。

被留置的状态只能指向一个留置状态，但这个被指向的状态自己可以再指向另一个留置状态，依此类推就会形成一条其中每一个状态都必须被满足的留置处理链。

一个留置状态必须在被留置状态的同一个交易中出现，因为状态仅通过索引值来进行相互间的引用。

5.9 合约约束

将状态和定义它们的合约绑定在一起的最容易的方式是通过 hash。这对于很简单并且稳定的程序是可以起作用的，但是更复杂的合约可能会需要升级。在这种情况下，状态引用合约的更可取的方式是通过签名者的身份标识。因为合约被保存在 zip 文件中，并且 Java 档案（JAR）文件也只是一个内部包含了额外文件的 zip 包，所以可以使用标准的 JAR 签名基础架构来识别合约代码的来源。简单的约束例如"任何叫这个名字的且被这些密钥签名的合约"这样的约束提供了一些升级上的灵活性，代价是会更多地被暴露给恶意合约开发者。要求进行组合签名有助于减少恶意开发者或黑客发布有害的合约版本的风险，代价是使发布新的版本更加困难。状态的创造者还可能会指定自己所希望的第三方来审查合约代码。不管选择了那些折衷方法，框架都能容纳它们。

一个合约约束可能会使用§5.2 所描述的复合密钥。标准 JAR 签名协议允许多个来自不同私钥的签名，因此可以满足使用复合密钥的需要。允许使用的签名算法有 **SHA256withRSA** 和 **SHA256withECDSA**。注意，用于代码签名的密码学算法并不总是与用作交易签名的算法相同，因为对于代码签名，我们发注意力放在基础架构的复用上。

5.10 事件调度

状态类可能会请求流在一个给定的时间启动。当一个状态被仓库认为是相关的、实现中的 CorDapp 被安装且被管理员加入了白名单（比如在配置文件中）时，节点就会对时间推移做出反应，开始新的与其它节点、人员或者内部系统的交互。金融合约通常内含有一个时间概念，所以这一特性对很多种类的状态

转换都很有用，比如，期权合约的过期，默认时间的管理，利率掉期的重定等等。

为了请求被调度的事件，一个状态可以实现 **SchedulableState** 接口，然后从 **nextScheduledActivity** 函数返回一个请求。这个状态在被提交给仓库时会被查询，调度程序将会保证相关流在正确的时间被启动。

6. 通用金融结构

6.1 资产

一个账本若不能记录资产的所有者，则不是很有用。我们定义了一组类，它模型资产的行为，并提供一些平台合约，以确保现金和债务的互操作性。

我们定义一个"可拥有状态（OwnableState）"的概念，它通过接口实现，任意状态(state)都遵从此接口。可拥有状态需要一个所有者字段，它是一个组合键(见 5.2 节)，被库（vault,见第 8 章）用于通用代码，以维护可拥有状态。

我们从"可拥有状态"可以派生一个可互换资产的概念来表示可度量的资产，它们在单一账本的对象图中的数量单位相似。具体而言，如英镑钞票是一种可互换的资产： 不论用一张 10 英镑钞票或两张 5 英镑钞票来表示 10 英镑，它们的总价值都是一样的。其他种类的替代资产可以是布伦特原油 （桶）(但并不是全世界所有种类的原油可互换 ,因为石油有不同等级)、洁净水(公升)、香蕉(公斤)、股票（股数）等等。

当现金在数字账本中体现时，会出现新的复杂情况：为了符合国家法定货币的要求，货币账本只记录那些有责任、有可能赎回为某种其他形式（如实物货币，或通过一些其他总账系统电汇等）的实体信息。这意味着两个都是价值 1000 英镑的账本条目，但可能不能完全可互换，因为所有条目都真实表示对发行人的请求权， 如果发行人不是中央银行 ，他可能会破产。即使假定违约永不发生，代表在哪里可赎回的资产数据必须通过存管链进行跟踪，以便可以从资产账本中"退出"，并索取对应的实物所有权。

Corda 的类型系统支持可编码这种复杂性。金额类型（Amount<T>）定义为整数类型的代币。这种类型不支持小数，因此当用来表示货币数量时，必须用分为单位，并使用分的数量来表示其他类型（如元）。代币可表示为任意类型，一种通用的代币类型是"发行的 Issued<T>"，它表示代币已被一方发行出来。它封装了资产 ,说明它由谁发行 ,及用一个旨在帮助发行人跟踪的不透明的参考字段(不可被平台解析)，如： 一个帐户号码、在存储中寻找资产的位置地址等。

6.2 债务

在金融市场中，通常用借据来支付，而不是现金（注意本节的现金指央行的存款余额）。当交易机构间对于对方有一定程度信任时，会经常这么做，以尽量减少手头的现金量 :你知道如果你支付给一个交易对手 ,很快其他交易也会支付给你，结果就能算出你欠其它机构的"净支出"，无论是双边或多边交易下。净额清算是一个过程，一系列总负债被替换为一种经济上相同的结果，可削减它们的合格头寸债务。该方法在概念上与贸易压缩相似，它通过两方或更多方的贸易来进行简化替换，最后的输出结果是需要实际转账的金额。

Corda 用债务合同模型化了一个净额债务，它是可互换资产（FungibleAsset）的子类。债务有生命周期，在资本账本中能用容器表示，可用于结算。合同不仅允许交易和用于替代性债务，也支持双边或多边净额清算。

重要的是需注意净额计算会变得非常复杂 ,金融业中的公司会在净额算法的质量上竞争。债务合同提供了计算简单的双边净额算法，并验证双边、多边净额算法的正确性。对于非常大的、复杂的多边净额算法，预计机构将使用现有的净额算法实现。

净额结算一般在市场关闭时进行。这是因为它是很难在交易头寸变化的同时，还计算净额并结算。这个问题和（程序）运行期间的垃圾收集器相似，压缩堆空间需要运行程序停止，以便堆内容可以被重写。如果一组交易机构希望实现"市场关闭"的检查表，他们就会遇到 5.8 节的障碍， 它会防止债务在特定时间内发生变化，而时间由公证人的时钟确定。

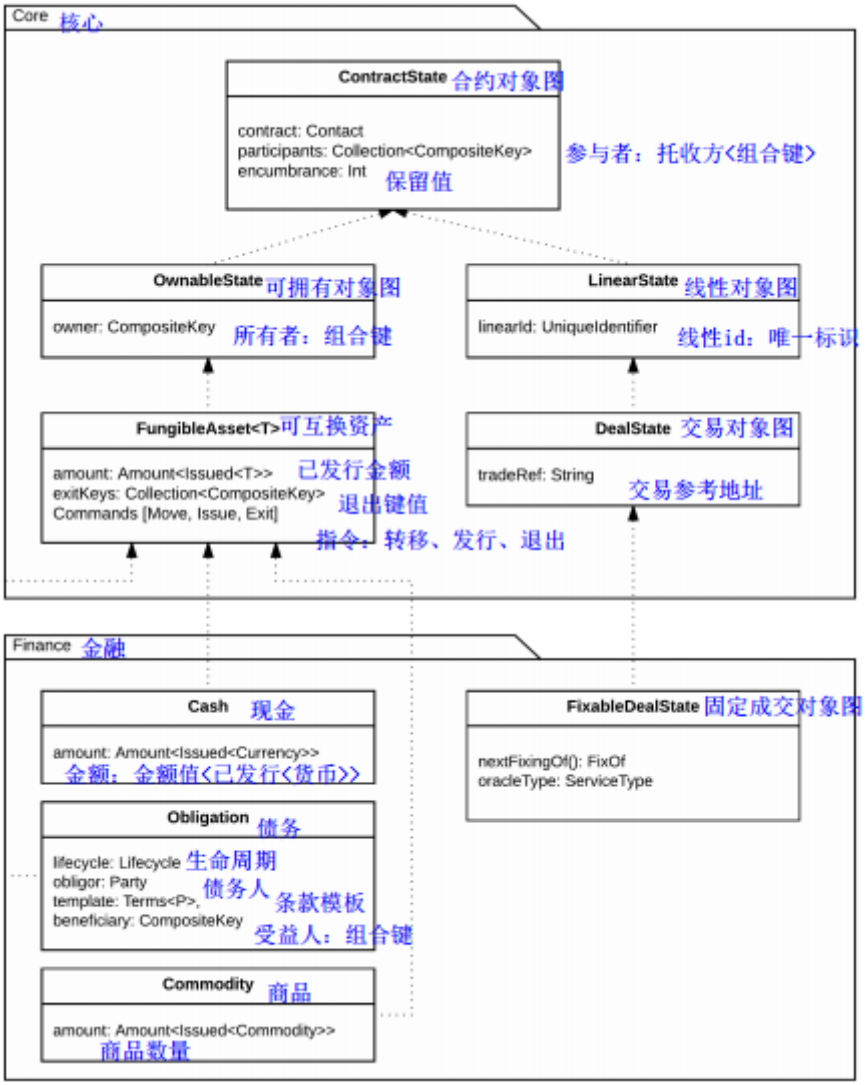

图 5：显示不同状态类型关系的类层次机构图

6.3 市场基础设施

交易是经济的活力源泉。分布式账本需要提供一个充满活力的平台，在此平台上交易可能随时发生。但是，去中心化的网络使得它很难在此之上建立竞争市场基础设施，尤其是高流动性资产，如证券。通常市场提供的功能，如低延迟订单簿、集成的合规性监管、价格行情订阅和其他，都受益于一个中央汇点。

Corda 数据模型允许分布式账本与现有市场、交易所的整合。一个账本上的资产卖单可以附带一个部分签署的交易。部分签名是一种允许在签署后，在控制方式下改变签名数据的签名。部分签名直接等同与比特币的 sighash 标记，它们使用

相同工作原理——签名包含描述该交易的部分元数据。通常包括所有的事务（信息），但使用这个元数据，它可能创建一个只涵盖一些输入和输出信息的签名，同时允许稍后添加更多信息。

此功能的目的是用于整合账本与市场、交易所的订单簿。考虑到在一个证券交易所中，一个买单可以提交一个部分签署的交易，它会标识一个现金输入对象图和一个输出对象图，以表示一定数量股票为买方所有。若本次交易无效时，因为现金不出现在输出列表中（未支付），因此也就没有股票的输入（获得）。一个卖单可以是部分签署的交易镜像组成，该交易有一个股票对象图作为输入和一个现金对象图作为输出。当两个订单在订单簿产生交叉时（可成交），交易所可将两个部分签名的交易合并起来，创建一个有效的交易，由公证人分发给买方和卖方。通过这种方式，交易和结算成为原子化，在市场参与者眼里，资产所有权在账本上同步更新了。请注意，在这个设计中，分布式账本本身不是一个市场，并没有处理分发或匹配订单。相反，它着重于交易前和交易后的生命周期管理。

中央对手方。在许多市场，存在一些中央基础设施如清算所（或称中央对手方、CCP）和中央证券存管机构（CSD）。它们提供治理、规则定义和执行、风险管理和共享数据和处理等服务。由于具有部分数据的可视性、灵活的交易验证逻辑和可插拔的公证设计等手段，Corda 将是未来特别适合 CCPs 和 CSDs 机构的分布式账本服务。

7. 公证人和共识

Corda 不安排时间信息进信息区块。这有时令人奇怪，因为它可以被描述为一个区块链系统或"激发的区块链"。相反，Corda 网络有一个或多个公证服务，它提供交易排序和时间戳服务 ,从而将其他系统抽象出的矿工角色转化为一个可插拔的组件。

公证人将由多个互相不信任的参与方组成，它们使用一个标准的一致性算法。公证人通过复合公钥（§5.2）进行标识和签名，公钥遵循国际分类账本的加密条件规范。注意，虽然它一般用 BFT 算法为公证服务，但没有要求必须这么做，在法规足以保证协议合规时，也可用高性能算法（如 Raft）。由于在统一的监管区域内（例如伦敦或纽约）可以多个公证人共存，一个单一网络中可以提供一个全球金融公证用于一般用途，和一个特定区域的 Raft 公证用于低延迟交易。

公证人接受提交给他们的交易后，或返回交易的签名，或返回拒绝错误，并指出双倍花费（"双花"）已经发生。对象图的选择公证签名可表明交易的终结。一旦所有其它必要的签名已聚集齐，应用开发人员可通过调用触发器终结交易流程。一旦终结流程返回交易成功，交易可以考虑提交到数据库。

7.1 与比特币的比较

比特币将时间线组织进区块链，每个块指向矿工选择建立的前一个块。块也含有粗略的时间戳信息。矿工可以选择尝试和扩展任何前一个块的块链，但鼓励建立在最近公布的块上 ,事实上系统中的其他节点只根据工作证明累积来承认一个块是否为链的一部分。新发行比特币的每个块包含一个奖励，一个不认可的块代表一种损失，一个认可的块则代表一份利润。

比特币使用工作量证明（POW）机制，是因为它的设计目标允许无限数量有身份的参与方加入和离开网络，同时使得它们很难执行 Sybil 攻击（攻击的一方产生多重身份来获得对网络的不正当影响）。对于一个点对点对等网络，志愿者们不能或不想承诺任何长期关系，并且还没做身份验证时，这是一个适当的设计。由于寻找工作证明的概率性质，用工作证明自然导致需要量子化的时间分块。然后这些分块必须根据相关性排序。建立在最近公布的工作证明上的激励机制，因为

在网络上计算证明需要耗时间，会导致紧张局势的现实。这意味着工作证明是以很慢的速度产生，很少有在同一时间计算出来。由于交易有可能以一个更高的速度产生，这意味着需要工作证明能强化支持多个交易，即需要更多的区块（而工作证明机制无法支持）。

一个 Corda 网络像电子邮件系统一样，在这个意义上，节点具有长期稳定的身份，使它们可以向其他人证明所有权。Sybil 攻击会通过网络进入过程受阻。这让我们抛弃有其多不良缺点的工作量证明（机制）：

- 对于一些简单任务，工作证明量机制的消耗过多，（它消耗的能量）可和我写作时整个城市的电力消耗相当。在人类需要更少能耗，而不是更多能耗时，这对生态是不可取的。
- 高能耗使采矿权集中于廉价或自由电力地区。这将导致在不可预知的地缘政治的复杂性，许多用户宁愿不去做（采矿）。
- 无身份标识的参与者方式，意味着所有交易必须广播到所有网络节点，因为没有可靠的方法来知道谁是矿工，这恶化了隐私。
- 该算法不提供终结（确定状态），只是一个概率的近似，这和现有的商业和法律的假设不适用。
- 在理论上可能出现大量矿工退出，甚至所有矿工退出，同时未违反任何协议承诺。

一旦工作量证明处理不再需要量子化的时间进信息块，因为不再需要慢速发布以冲突解决申请，并且因为参与方声称正确性的排序都是事先已知的，则普通签名已足够。

7.2 算法的灵活性

共识算法是一个热门的研究领域，新的算法经常被开发出来，以改进算法艺术。不像大多数分布式账本系统，Corda 没有紧密结合一个特定的算法。它不仅支持升级新开发的算法，也反映了一种事实，即可对不同情况和网络进行不同权衡。

举个简单例子，公证人在节点之间使用 Raft 算法，则在同一个城市中能提供非常良好的性能和低延迟，代价是任一当选为指挥者的节点会更多地暴露于恶意攻

击或错误中。在（公证人）成员们组成一个分布式公证服务的情况下，所有大的机构和监管机构为了他们自身利益交易的安全性，他们不希望测试和破坏自己的账本，这样来获得性能可能会有意义。在其他情况下，现有法律或信托关系不健全时，使用缓慢但（健壮的）拜占庭容错算法如 BFT-SMaRT 可能更好。另外也可使用硬件安全功能：如新加坡交易所 SGX®可能把非 BFT 算法转为更可信的方式，如使用远程认证和硬件保护。

在同一个网络能够支持多个公证人具有其他优势：

- 有可能淘汰那些不再想提供迁移对象图服务的公证人（即参与者组）。
- 通过将新公证人在线并行运行的方式，增加系统的可扩展性。只要能访问局部地方的账本（即对象图没在公证人间不断迁移），就允许有扩展性限制的共识算法或节点的硬件在一处故障时，仍能跳过故障继续工作。
- 在一些情况（但不是所有），数据传播的监管约束可以由有管辖权的具体公证人负责。这在两个司法管辖区对资产有互不相容的约束时，将不能很好地工作，因为资产可能经常在世界各地转移。但在特定区域内使用账本来跟踪交易对象图或其他行为，它可以有效工作的。
- 公证人可以基于他们的可用性和性能进行竞争。
- 用户可以在已验证和非验证公证人之间挑选。如下。
- 这些技术在应用时，有一些模型是可能的，如在“自公证”交易时，发行人发行资产会觉得方便，因为是属于他们发行的资产，这必然需要支持多个公证人在同一网络。这样的模型很可能是一个过渡对象图，尤其是因为这样的模型本质上在可支持的操作范围内是有限的。
- 单独的网络可以开始独立和以后合并在一起（见下文）。

7.3 验证的和未验证的公证人

验证公证的方式，解决并充分检查了要求他们（公证人）排解冲突的交易。因此，在网络退化至只有一个公证人并不使用任何隐私功能时，他们可获得对每个交易的充分可见性。非有效验证的公证人承担交易的合法性，它不要求交易数据或依赖关系超出对象图消费使用列表。这种公证人可能成为账本的“楔形（wedged）”，因为任何知道对象图的哈希和索引的公证人，就可以不需检查地销毁它。如果问

题的原因是偶然的，不正确的数据可以提交给一个未验证的公证人，说服它回滚提交，但如果错误是恶意的，那么由这样的公证人控制可能会造成永久损坏。

因此，用户可能在他们的隐私/安全范围内选择他们的首选点，这取决于每个对象图独立地期望数据如何被使用。当对象图不太可能长久或传播很远时，只有那些会知道他们的交易哈希（hash）信息的实体是值得信赖，用户可以选择把数据保存在这些公证人处。对于流动性资产，如果交易标识符发生泄漏，一个验证的公证人也应始终防止价值被破坏和盗窃。

7.4 融合网络

由于没有一种单一的区块链来合并两个独立的网络，如通过简单建立两个网络节点的双向连接，然后配置双方彼此信任的公证和认证机构。

这种能力似乎是没有意义的：一个分布式账本的目的不就是给每个人一个单一全球数据库吗？这是，但仍然需要达到这一最终对象图的实际途径。通常情况下，消费者认为机构作为一个单独的公司，实际上是许多持相互特许标牌的不同实体组成，它们之间达成协议，并相互与对方做内部交易。由于监管、税收的原因，及由于历史的融合或只通过纯粹纸面工作的爱好者，这种设置可以发生。大公司可能因此经历相同的同步问题，去中心的分类账本正是可解决分散但纯粹是在组织边界内的方案。在这种情况下，要解决的主要问题是没有恶意的行为，而是不同部门、不同的软件开发实践、（不同的）链接用户目录等。在应对与更广泛世界进行同步的大问题前，这样的组织可以从技术内部经验和清理自己的内部视图获得好处。

当合并网络时，双方必须相信对方的公证人从未签署“双花”。当把一个私有网络合并到全球账本时，它应该可以简单地依靠激励来提供这种保证：公司中没有一个处进行了“双花”。然而，如果需要更多的证据，一个独立的公证人可以运行在已启用审计日志的硬件安全模块上。公证人本身只会使用一个私有数据库，并在一台机器上运行，其日志导出到一个使用全球性网络的人处，以异步事后核查。

7.5 保证数据分布

在任何一个全球共识系统中，用户都面临着他们是否有数据库最新对象图的问题。区块链的程序员往往简化假设，因为没有一个正式的矿工位置地图，交易需要通过广播分发给矿工，如果他们来了最新数据，他们可以听广播流来获知。另外，也没有什么能阻止某些人私下提供一个矿工，该矿工作为一个交易的已知位置，他们同意不广播。该网络的其余部分第一次发现这个交易是当它广播一个包含了（此交易）的信息块。用于做"双花"欺诈的攻击被称为芬尼攻击，基于工作证明的系统依赖于激励措施以阻止这样的攻击：引用比特币白皮书的话："他应该发现它更有利可图的游戏规则。而不是破坏系统和损坏他自己的有效财富。" 在实践中，这种方法在大部分的时间似乎工作得很好，因为矿工通常不接受私自提交的交易。

在一个没有全球广播的系统中是非常不同的：公证人集群必须直接接受交易，也没有机制确保每个人都看到该交易正在发生。有时这并不重要：大多数交易对你来说都是不相关的，如果不得不下载他们只是浪费资源。但偶尔你也希望意识到，账本的对象图已经被其他人改变了。一个简单例子是一个期权合约，你希望期权在期满中止，除非对方已经行使了它。因此在行使期权不必需卖方签署是有利的，因为如果导致卖方损失钱时，卖方会拒绝。虽然卖方会发现，如果买方行使期权时，他们（卖方）试图到期中止，由于公证人通知他们，他们在有效期内的交易是一个"双花"，则最好立即找到（这些交易）。

实现这个目标的最显著方式，是给公证人有责任来确保所有感兴趣的各方了解交易。然而，这将需要公证人员知道谁是当事人，从而发生讨厌的隐私泄漏（问题）。它也会将给公证人额外的网络负载，因为它将经常发交易数据给参与方，而参与方可能已经有此信息，或可能根本就不在乎。在许多情况下，可能没有要求公证人作为一个受信任的第三方来实现数据分发的目的，博弈论假设或法律保证已足够强大，可以信任的同行可提供交易数据作为其定期工作流的一部分。

要解决这个问题，应用开发人员可以选择是否请求交易由公证人分发。这可通过简单地依靠标准的身份查找流完成（见§5.6）。当一个对象图被消费时，如果一个节点希望被公证人通知，它可以发送对象图机的随机密钥的证书链给公证人集群，

然后按照通常做法，将其存储在本地数据库中。一旦公证人集群确认了交易，关键身份标识会被查出来，发出顺利解决的交易副本。在正常操作中，公证人没有提供长期身份标识的随机密钥链接证书，因此就不知道谁参与了该操作（假设使用过程中，源 IP 地址屏蔽了，见第 15 章）

8. 金库

在任何基于区块链的系统中，大多数节点都有一个钱包，或者我们称之为“一个金库”。

该库包含从账本中提取的数据，该数据被认为与该节点的所有者相关，存储在一个可以很容易地查询和使用的表单中。它还包含私有密钥材料，这用于在库中给事务消费的对象图签名。像一个数字货币钱包，Corda 库了解如何创建交易，通过结合资产对象图和可能增加资产的变化以使价值平衡，从而把价值传递给他人。这个过程通常被称为“硬币选择”。硬币选择可以是一个复杂的过程。在 Corda 中，没有按每笔交易的网络费用，这费用是其他系统复杂性的重要来源。但交易必须遵守规则，以确保发行人和参考数据通过手手传递后保存为资产。

高级库的实现也可能是在后台执行分裂和合并对象图。目的是增加支持并行创建事务的数量。因为对一个交易签名可能涉及到人的干预（见§10），可能需要相当长的时间，因此能够创建多个并行交易是重要的。金库必须管理对象图的“软锁”，以防止多个事务同时使用相同的输出。违反软锁机制将导致由公证人产生双重花费，并被公证人拒绝。如果一个金库包含一个用户的全部现金余额于一个对象图中，则每次只能有一个单一交易被创建，这可能会征收不可接受的业务管理费用。通过自动创建自我发送的事务，可将大对象图拆分成多个较小的对象图，使并行创建的事务数可增加。此外，为了避免碰到交易规模的限制，可能需要将许多小对象图合并成一个更有价值的大对象图。最后，在某些情况下，金库可能会将资产对象图发送给发行人重新发行，从而修剪长的交易链和提高隐私。

金库还负责管理应用程序安装后，有关节点实施的计划事件（见§5.10）。

8.1 直接 SQL 访问

一个分布式账本最终只是一个共享的数据库，尽管它有一些独特的功能。因此，以下功能是非常期望的，以提高应用程序开发人员的生产力：

- 能够在账本中存储私有数据，它和半公共数据关联。
- 能够使用广泛理解的工具如 SQL 语句来查询总账数据。

- 能够使用整个应用程序的私有数据(如客户票据)和账本数据之间的连接。
- 能够在基础表上定义关联约束和触发器。
- 能够在特定时间点做查询，例如午夜时分。
- 复用行业标准和高度优化的数据库引擎。
- 和任何特定数据库引擎保持独立，且不隔离太多有用功能。

Corda 使用的 JVM 字节码语言（包括注解 annotation）的一个子集。金库使用在 JSR 338 规范定义的 java 持久化架构（JPA）中的注解功能。这些注解定义了一个类如何映射到一个关系表结构，包括哪些成员是主键、SQL 类型映射字段等等。当一个交易通过流提交给金库时，金库发现对象图认为是相关的（即包含一个由节点拥有的关键字），相关的 corda 应用作为一个插件已安装到该节点，对象图通过对象关系映射生成 SQL 更新和插入语句。请注意，当对象图被消费使用时，数据并没有被删除，但是一个连接可以用一个专用的元数据表来执行，以消除数据集上的已消费对象图。这可以让数据在某个时间点查询，使用外部工具将数据行收回到历史表。

节点带有一个开箱即用的嵌入式数据库引擎，也可以配置为指向一个单独的关系数据库。节点存储的不仅是对象图数据，而且还在数据库中存储了所有节点的工作数据，包括流程检查点。因此，一个节点的对象图和它参与的所有通信，都可以通过简单地备份数据库本身来支持。JPA 注解独立于任何特定数据库引擎或 SQL 方言，因此对象图不能使用任何专有的列类型或其他特征，但由于 ORM 只用于写路径，用户可自由连接后台数据库直接执行 SQL 查询，他们可任意选用他们喜好的数据库引擎功能。他们还可以创建自己的表和根据底层数据为最终用户应用创建合并视图，只要它们不施加任何约束以防止节点去同步数据库的账本实际内容。对象图是任意的对象图。虽然没有禁止一个对象图包含不同表的多个类，典型的关系表示不会是对象图表示的直接翻译。

对象图可从库中查询，用于 ORM 映射类，它往往会跳过账本的具体数据，与用户喜欢不透明的公共密钥无关，可以扩展单一字段像"金额<发行<货币> >"类型到多个数据库的列。

值得注意的是，虽然库只对 JPA 注解响应，对象图用其他方式被注解经常是有用的，例如定制其映射为 XML 或 JSON，或者实施验证约束。这些注解不会直接影响节点的行为，但在与周围软件的对象图工作时可能是有用的。

8.2 键值的随机化

区块链系统的标准保密技术，是随机的无关联的公共密钥代替现实用的身份验证。这些假名的所有者可能透露给对手使用一个简单的交互协议，如爱丽丝选取一个随机数（仅用一次的数字）并将其发送到鲍伯，由鲍伯用公钥对应的临时私钥签署，以证明他的所有权。

为每一个新的交易或资产转移快速产生新的密钥，结果许多私有密钥会被创建。这些密钥必须全部备份和安全保管，这在数据上规模时会成为一个重大的管理问题。解决这一问题的典型方法是通过确定性密钥导出的使用，就像通过比特币社区首创的 BIP 32"层次确定性钱包"那样。确定性密钥导出允许所有私有密钥材料来自一个单一的、小池的信息熵（如一个仔细保护和备份的 128 位随机数据）。更重要的是，全 BIP 32 技术采用的是与椭圆曲线组合来支持它，（corda 的）公钥可以不访问底层的私有密钥材料来导出。这允许设备提供新的公钥给那些没能与密钥签名的对手方，以使系统更好的安全性和运营效率。

Corda 对使用方的数字签名算法的数学特性不设置任何约束。但建议在可能的情况下使用分层确定型的密钥派生。

9. 领域专用语言

9.1 条款

在编写智能合约时，很多被渴求的特性和模式会反复出现。比如，可以预期所有生产级别的资产合约都会想拥有下列特性：

- 发行交易和退出交易
- 转移交易（所有权的再分配）
- 可互换性管理（参见§6）
- 支持升级合约到新版本

很多这些看似简单的特性都有隐含的边界情况。一个例子是需要防止创建包含 0 或者负数数量的资产状态。另一个例子是，为了资产互换的目的，要确保状态被汇总，避免不慎假设交易一次只会转移一种资产。Corda 标准库提供了一个叫*条款*的小型类库，实现了可复用的合约逻辑块，而不是期待合约开发者来重新实现这些低层次逻辑。合约编写者可以创建他们自己的条款，然后把合约条款的集合一起传递给一个翻译它们的库函数。

9.2 组合库

用于表示金融合约的领域专用语言是热门研究领域。其中一项开创性的工作是由 Peyton-Jones, Seward 和 Eber[PJSE2000]所做出的"组合合约"，使用一个 Haskell 组合器的小型库将金融合约模型化。然后这些模型可以被用于对潜在的交易进行估值。虽然区块链系统对"合约"一词的使用，在观念上和 PJSE 存在一些微小差别，但 PJSE 的基础概念也适用于我们的上下文。平台提供了一个实验性的基于 Kotlin 语言扩展特性建立的*通用合约*。为了避免语言上的混淆，通用合约指的是一个将代码/数据捆绑在一起的"部署措施"，而不是指一个合约。用这个语言表示的一个欧洲外汇看涨期权看上去像这样：

```
val european_fx_option = arrange {
    actions {
        acmeCorp may {
```

```
            "exercise" anytime {
                actions {
                    (acmeCorp or highStreetBank) may {
                        "execute".givenThat(after("2017-09-01")) {
                            highStreetBank.owes(acmeCorp, 1.M, EUR)
                            acmeCorp.owes(highStreetBank, 1200.K, USD)
                        }
                    }
                }
            }
        }
        highStreetBank may {
            "expire".givenThat(after("2017-09-01")) {
                zero
            }
        }
    }
}
```

程序员可以定义任意的“actions”，同时定义针对“actions”何时被调用的约束。**zero** 字段指示交易的终结。

可以看出，领域专用语言同时结合了*什么*被允许，和特定交易数据如*何时*、*多少*被允许。因此，核心模型中的代码和数据之间的区别被淡化了。基于早先的工作，不仅得以实现估值/现金流计算，还允许合约逻辑在数据库层面直接执行。

9.3 形式上可验证的语言

Corda 合约可以被升级。然而在说服大型网络上众多参与者接受新版本合约这一过程存在固有的协调问题的情况下，一个频繁被提及的需要是形式上可验证语言被利用来试着保证合约实现的正确性。

我们并不尝试自己处理这个问题。尽管如此，由于 Corda 专注于所有 JVM 字节码的确定性执行，所以以此为目标指令集的形式上可验证语言可用于对智能合约进行表示。一个良好的例子是 David Pearce 博士的 Whiley 语言，在编译阶段对程序的完整性证明进行检查。通过在工业标准级的平台上进行构建，我们从分布式系统世界外的计算机科学社区中获得了减少边界研究的能力。

9.4 投影式编辑

用于表示合约逻辑的自定义语言和类型系统可以自然地与*投影式编辑*相结合，源代码的编辑不是以文本形式，而是通过一个可感知结构的编辑器进行。这样的语言不仅可以由传统的、面向语法驱动文本的结构所组成，还可以包含图形、表格甚至它们的递归组合。由于很多金融合约频繁出现数据表格并天然地面向英语，一个用于构建智能合约逻辑的专用环境会深受用户感激。

10. 安全签名设备

10.1 背景

数字金融系统，尤其是区块链系统的一个共同特点是应用安全客户端硬件，用于保存私钥，并使用私钥进行签名操作。这与交易回滚的一种零容忍方法相结合之后就成了它们减少日常开销的方式之一：即试图保证交易授权过程的健壮性和安全性，并由此保证签名的可靠性。

很多银行已经向客户推广 CAP（芯片认证程序）读取器，允许使用智能卡的验证/响应协议登录到网上银行。用户需要输入正确的代码并手工将响应信息拷贝到计算机。这些设备便宜，但是只有小的不可靠的低分辨率屏幕，并且在 PC 上存在恶意软件时容易受到混淆攻击，例如，恶意软件会说服用户他们正在执行登录验证，而实际上他们正在授权向一个新帐户付款。这类设备的主要优点是签名密钥被保存在一个健壮且便宜的智能卡中，因此可以在不替换密钥的同时将设备替换掉。

这个领域中最顶尖的设备有 Ledger Blue 和 SatoshiLabs 的 TREZOR 等。这些设备由比特币社区开发，也是为比特币社区而开发。它们比 CAP 读取器更贵，配备更好的屏幕，并通过 USB 连接来免除打字输入。像 Ledger Blue 这样的高端设备还支持 NFC 和蓝牙功能。这些设备还在另一个重要方面与 CAP 读取器不同：它们不是对任意的小的验证数进行签名，实际上，它们理解它们所专注的网络的本地交易格式，并解析交易，提取其中的消息展示给用户，然后用户简单地以按键的方式确认执行屏幕所显示的动作。接着交易在内部被签名，再通过 USB/NFC/蓝牙连接被传送回 PC。

这样的设置意味着设备本身控制了资产，而不是通过设备给控制你所有资产的强大服务器授权。由于没有智能卡等价物，私钥可以可以以“钱包字”（从密钥内容派生的 12 个随机单词）的形式从设备中导出。因为椭圆曲线生成的私钥比较小（256 位），所以与金融行业通常使用的更大的 RSA 密钥相比就没那么冗长。

只由个人的、职员控制的设备持有签名密钥，组织的节点自己无法给交易签名，这样的做法有明确的好处：

- 如果节点被怀有恶意的入侵者或内部人员攻击，他们无法窃取资产、修改协议或任何需要人为批准的事情，因为他们获取不到签名密钥。从密钥管理的视角看，也不存在单点故障的问题。
- 谁签发了特定行为也更清楚——签名证明了是那台设备被用来签发了一个行为。没有后门或管理员工具可以代表别人来创建交易。
- 集成了指纹读取和其他生物识别认证功能的设备使员工间的设备分享/交换更加困难，从而能够进一步增强信任。智能手机或者平板电脑也可以被作为交易认证设备使用。

10.2 混淆攻击

任何想要把智能签名设备集成到分布式账本系统的人会面对的最大的问题是设备如何处理交易。对于比特币，由设备直接处理交易是很直率的做法，因为其交易格式小而简单（原理上而言——实际上，比特币协议的一个可修复的怪异之处使得这些设备必须如何运行的问题显著地复杂化了）。因此把一个比特币交易转变成屏幕上人类可理解的确认信息就很简单：

```
Confirm payment of 1.23 BTC to 1AbCd0123456......
```

由于不透明的付款地址无法预知，这条确认消息很容易受到混淆攻击的影响。一个足够聪明的病毒/攻击者能够将你期望付款的合法对手方的合法地址替换为他的一个地址，于是你就会把正确的数量的金额支付到错误的地方。同样的问题也会影响对 IBAN 和其他账户号进行校验的金融验证者：用户的 IBAN 来源可能是一封电邮或他们通过受侵害的机器浏览的一个网站。**BIP70** 协议被设计来解决这一攻击问题，其设计做法是允许提供一个证书链来把目标密钥与一个稳定的人类可理解的经校验的身份标识连接起来。

对于一个通用账本，我们面对的额外问题是交易可以有很多不同类型， 包括设备被制造出来之后才被创造的新交易类型。因此，在设备内创建一个简明的确认消息的做法将会演变成一个要求频繁做固件升级的变化无常的问题。而固件

升级又是所有安全硬件方案中的一个潜在缺点，所以理想的做法是最小化升级的次数。

10.3 交易摘要

为了解决这个问题，我们在交易格式（衔接输入、输出、指令、附件等等）中增加了一个顶层的摘要域。这个新的顶层域是一个字符串列表。智能合约有了一个新的职责，它们被要求生成一条描述交易在做什么的英文消息，然后检查这条消息存在于交易当中。平台确保不会出现意料之外的消息。该域是字符串列表而非单个字符串的原因是在高级应用场景中，一个交易可能会同时做好几件事情。

因为确认消息的计算工作现在被移到了智能合约当中，成为了交易的一部分，所以交易可以被发送到签名设备：设备需要做的全部，就是提取这个确认消息打印到屏幕上，并提供一个可用的是/否按键用于决定要不要进行签名。由于设备做的签名把确认消息也覆盖在内，并且消息由合约基于状态中机器可读的数据进行检查，所以我们就可以知道消息是正确的、合法的。

上述设计简单，但是存在一个问题，那就是大量设备所并不需要的数据也被发送到设备上。由于签名设备的内存大小通常有严格约束，交易的复杂度最终会不幸地受到用户签名设备上可用 RAM 的限制。为了解决这个问题，我们可以使用数据剥离机制（§5.7），只呈现摘要和把它们连接到根的 Merkle 分支。然后设备就可以在只看到文本摘要的情况下对整个交易内容进行签名。同时考虑到触发摘要检查的合约将会被状态所触发，于是这个签名也将会覆盖交易的机器可理解的版本。

注意，我们在这里假设合约本身是非恶意的。一个恶意用户可以构建一个会生成误导性消息的合约。用户想看到保管库中的状态并与之协作的话，需要将相伴的 CorDapp 作为一个插件加载到节点上并将其加入到白名单。永远不会出现场景会请求用户去签署一个包含未批准合约的交易，尽管节点会执行这样的合约，作为对交易依赖的校验的一部分。

10.4 身份置换

合约代码的运作只会用到公钥的不透明表示。由于在一条受监管链上的交易需要匿名化，所以合约想要在沙盒内部获取到身份信息是不可能的。因此，合约也无法生成一条完整的包含人类可理解的身份名称的消息，哪怕节点本身拥有这样的信息。

为了解决这个问题，交易连同连接匿名公钥和长期身份证书的 X.509 证书链一起，被提供给签名设备。对于包含了用户的交易，证书链应当一直可用（因为按照定义，交易本身知道交易的对手方都是谁）。设备可以校验这些证书链，建立到人类可读的名字索引的映射。被放置在交易内部的消息也许会包含由指令通过反斜杠语法获取的公钥的数字索引，并且设备在进行渲染之前必须执行消息置换操作。必须留意确保发布给网络参与者的 X.500 命名不包含故意迷惑用户的文字，比如，包含引用标记、局部指令、特殊符号等等的名字。这可以在网络许可层面做强制规定。

10.5 多语言支持

合约被期望生成交易的一个人类可读形式的版本。这在传统上应该是使用英语的。原理上，我们可以定义支持不同语言的消息的交易格式，如果合约也支持多语言，签名设备就可以挑选出正确的语言。尽管如此，必须注意保证用户看到的不同语言的消息被正确翻译，不会导致歧义和困惑，否则会出现可利用的混淆攻击方式。

11. 客户端 RPC 和应答采集

分布式账本的所有实际部署都会面临如何与现存的由周边工具和程序构成的生态系统相集成问题。理想情况是，与节点交互的程序将会被松散结合，被认证，在节点的短暂中断和重启时保持稳定，并且运行速度的差异（比如投产使用时会比实现时要快）能被透明处理。

为满足这些需要，Corda 提出了一种包含了几个不寻常特性的简单 RPC 机制。底层传输使用的是消息队列（AMQP）和返回包含 Rx 可观察量(订阅)的对象的方法，这些可观察量可能转而发出更多的可观察值量。

将数据结构的快照，连同一个会发出代表数据结构增量的对象的可观察值一起返回，是 RPC 的一般模式。客户端库具有降数据结构和其增量重新构建为一个可观察量集合，该集合的类型可以直接绑定到 JavaFX 用户界面。用这种方法，在保持全新的富客户端应用中渲染全局账本上的数据结构，就成了对开发者工作要求最少的简单明确的操作：以功能化的方式将数据块简单串起来是很高效的。在这些可观察量集合上做的如映射、过滤、排序等等响应式变换，使得通过函数式编程构建用户界面变得容易。

由于 RPC 传输时通过节点的消息队列代理实现，所以框架可以自动从节点/节点组件的重启、客户端 IP 地址改变以及类似的通讯中断中恢复。同理，需要长期存活并且幸免于重启、升级和移动的程序，可以请求把那些可观察量发送到一个持久化队列。背压和队列管理都是由代理提供的。额外的 RPC 处理性能可以通过给代理增加更多的 RPC 处理器而获得，代理会自动在它们之间做负载均衡。

可以问一句为什么 Corda 不使用典型的 REST+JSON 的方法来与节点通讯。原因如下：

- 相较于文本协议，我们更偏向于使用二进制协议，因为基于文本的协议更易于受到转义和其它缓冲区管理问题的影响，这些问题会导致安全事故。

- 消息队列代理提供了大量用于构建可靠应用的基础设施，而普通 HTTP 协议则没有，比如背压管理、负载均衡、队列浏览、速度差异管理等等。
- 基于 REST 的协议对回到客户端的执行结果流做了多项约定，而这些约定对我们的任务都是不理想的。

将实时数据结构直接连接到 UI 工具包的能力，也有助于避免 XSS 攻击、XSRF 攻击和类似的基于丢失缓冲区边界追踪的安全问题。

12. 数据分发组

交易数据的分发默认是由应用提供的流来定义（参见§4）。流指定了交易应该在何时被发送到哪个节点。这些目的地通常是基于状态的内容和可用的身份查询证书计算得出，因为金融数据的应用场景中通常包含了相关参与方的身份信息。尽管有时候，应当接收数据的参与方的集合并不能提前知道，并且还可能在交易被创建后发生变化。对于这些场景，数据的部分可见性并不合适，需要有另一种可选的机制。

一个数据分发组（DDG）通过生成一个密钥对和密钥对的自签名证书来创建。组在内部通过它们的公钥被识别，并可以在证书中给其一个字符串形式的名字，但是软件中没有任何地方假设这个名字是独一无二的：名字只是为了人类使用，可能会和其它独立的组相冲突。如果发生了冲突，用户界面通过在名字后面添加几个对公钥做 base58 编码的字符来消除歧义，像这样："我的大众化组名（a4T）"。不管怎样，由于组并不是全局可见的，冲突不太可能是常见情况或需要很多编码字母来解决冲突，而且有些组甚至可能并不是打算给人类使用的。

一旦一个组被创建，其它节点就可以通过使用一个邀请流被邀请加入到组中。成员关系既可以是只读的也可以是可读可写的。要添加一个只读节点，只需要发送证书，也就是公钥。要添加一个可读/写节点，需要发送证书和私钥。未来对该设计的一个精心改进可能是支持给每个成员一个单独的私钥，以使得可以追踪是谁添加了交易到组里，但这是留给未来的工作。两种情况下，节点都会在本地数据库记录下哪些它所邀请的节点接受了邀请。

当邀请被接收之后，目标节点以正常方式运行流的另一侧，即，如果配置了信任邀请节点则自动接受成员关系，或者为了由外部系统进行处理而向消息队列发送一条消息，或者踢给一个人类管理员请求批准。邀请节点加入已经是其成员的组的话会被拒绝。接受邀请的节点也会记录下是哪个节点发起的邀请。所以结果是会在保管库里保存邀请者和被邀请者之间的双向记录关系。最终，邀请流的邀请方一侧会推送组里存在的所有交易 ID 的列表，而被邀请一侧则对所

有交易做解析。最后结果是组内的所有交易都被发送给新的节点（包括所有依赖关系）。

注意，如果交易被添加到组里的速度大于等于新节点下载和校验的速度，那么这一最初的下载过程可能是无止尽的。因此，在想要在试图加入一个组，并为加入的行为引入"完成"的概念的时候，更好的做法是把加入行为视作发生在一个特定的时间点上，并把作为结果的洪水般的交易数据视作一个持续的流，而不是视作一次传统的交易数据文件下载。

当交易被发送到保管库，总是会经受一个相关联的测试，不论交易是否在组里（参见§8）。这个测试被扩展后也用于检查节点所在的所有组的签名。如果发现一组匹配，那么交易的状态就全都被认为是相关的。此外，保管库查找被它邀请加入这个组的节点和邀请它加入组的节点，把所有最近发送过这个交易给自己的节点移除，然后发起一个 **PropagateTransactionToGroup** 流给这些节点。这个流的另一侧检查是否已经知道这个交易，如果不知道，则请求获取这个交易，检查交易确实是被问题中的组签名，对交易进行解析，如果成功则发送交易给保管库。以这种方式，一个由任意组成员添加的交易会沿着成员关系树上下传播，知道所有成员都已经见过这个交易。传播是幂等的——如果保管库之前已经见过该交易，就不会再次处理。

我们的架构到目前为止有一些优点，和一个大的缺陷。优点是：

简单 核心数据模型不变。访问控制使用已有的工具处理，如签名、证书和流。

隐私性 在其它成员不知晓的情况下加入一个组是可能的。在非组成员不知晓的情况下创建组也是可能的。

可扩展性 组不需要在任何中央目录注册。一个在四个参与方间存在的组，只会对这四个参与方施加成本。

性能 组的创建可以跟你生成密钥对和邀请其它节点加入一样快。

职责 对于组里的每个成员，在协议下总是会有一个节点（邀请节点）有责任把新数据发送给它。不像 Kademlia 风格的分布式 hash 表，或比特币风格的全局广播，你永远不会处于在没有成员违反协议的情况下你却没收到数据的境地。也不会在哪个点

> 上，你需要选择随机的一些节点，礼貌地请求它们为你做什么事并希望他们会选择坚持。

它的大缺陷是脆弱。如果你有一棵成员关系树，然后一个节点掉线了一段时间，那么数据的传播就会割裂，备份在离线节点的父节点和子节点的外传队列上，直到离线节点回来。

为了对组做加强，我们可以增加一个新特性，成员关系广播。组内拥有写权限的成员可以选择签署一份成员关系公告并将其通过关系树传播。这些公告会被记录在组内每个节点的本地数据库中。节点在发送新增交易时可以选择将这些被公告的节点也包括在内。这就把成员关系树转换成了一张可能包含环的图，但是由于节点会忽略它们已接收过的交易/附件，所以无限的传播循环是不可能的。一个组偏向于隐私性还是可用性，会在定义它的证书中示意：如果偏向于可用性，就意味着成员必须总是公告自身信息（这将导致形成一个多边网络）。

网络图为网络定义了事件边界，即一个离线节点在被认为是永久消失之前所允许经过的时间跨度。一旦一个节点离线的时间超过了事件边界，所有邀请了它的节点都将它从它们的本地表格中移除。如果一个节点是被一个已消失节点邀请入组，并且没有其它节点公告对它而言可用的成员关系，那么这个节点必须向一个队列递交一条消息，并且/或者通知管理员，因为它现在已经被从组里驱逐了出去。

得出的布置结果可能看上去像是一个八卦网络。然而底层的成员关系树结构被保留了下来。因此当所有节点都在线（或足够节点在线）时，消息可以被保证传播到网络中的每一个节点。不会出现组的一部分与剩余部分分隔开而没有任何人感知到这个事实的情况，而这种情况在八卦网络中虽然不太可能但却还是会发生。也不会像是数据没有被完全复制的分布式 hash 表，所以我们也避免了出现数据被加到组里后，因为节点中断而不再可用的情况。当出现问题时，总是可以推断网络的行为的原因，并总是可以对职责进行分配。

注意，当成员被加入到组里之后是无法被移除的。我们可以提供一个移除公告，但它也只能被作为参考建议：没有办法阻止节点将这个公告忽略。也无法

枚举组内的成员，因为当你加入时并没有做成员关系广播的需要，也决不能强制建立这样的需求。

13. 确定性 JVM

所有节点在处理同一交易时始终对其有效与否达成一致的结论相当重要。交易类型是使用 JVM 字节码定义的，这就意味着字节码的执行必须是完全确定性的。开箱即用的标准 JVM 并不是完全确定性的，因此我们必须做一些修改以满足我们的需求。非确定性来源于一下几个方面：

- 外部输入源，如文件系统，网络，系统属性，时钟。
- 随机数产生器。
- 关于何时终止长时间运行的程序的不同决定。
- Object.hashCode()，典型的实现是返回一个指针地址或者给对象赋一个随机数值。这会表现为对 hash 映射和 hash 集合的不同的迭代顺序。
- 硬件浮点运算的差异。
- 多线程。
- 节点对 API 的不同实现。
- 垃圾回收器回调。

为了保证合约校验函数哪怕是在面对无限循环的时候也是完全纯粹的，我们构建了一个新的 JVM 沙箱类型。它利用了一个字节码静态分析与重写许可，和一个允许沙箱控制 hash 码生成行为的 JVM 小补丁。合约代码在第一次需要被执行时被重写，然后保存下来留待未来使用。

字节码的分析和重写会执行以下任务：

- 在执行昂贵的字节码之前插入对一个账户对象的调用。这一重写的目标是确定性地终止那些运行时间已经长得不可接受的，或者使用的内存总数不可接受的代码。昂贵的字节码包括方法调用、分配、反向跳转和抛出异常。
- 阻止异常处理器捕获 **Throwable**，**Error** 或 **ThreadDeath**。
- 调整常量池引用，改为将代码重链接到一个“影子”JDK。影子 JDK 在一个专用的沙箱程序包里面复制了正常 JDK 的一个子集，去掉了那些合约代码不应访问的功能，如文件 IO 或外部熵。

- 为每个方法设置 **strictfp** 标志，该标志要求 JVM 以一种硬件独立的方式做浮点运算。虽然我们预计浮点运算在绝大多数合约中都不会有重要用处（大整数和大十进制库都是可用的），但对那些需要进行浮点运算的人来说还是可用的。
- 除了特定用例外，禁止出现 **invokedynamic** 字节码，因为支持这个功能的库在历史上有安全问题，并且这个功能主要只是被脚本语言所需要。对 Java 代码自己使用的特定 lambda 表达式和字符串拼接元工厂的支持是允许的。
- 禁止本地方法。
- 禁止终结器。

使用的开销测算策略很简单：只是对已知的执行起来昂贵的字节码进行计数。方法的大小被限制，跳转也被计入到预算当中，所以这一策略被保证会最终被结束。尽管如此，手动构造出执行时间极长的字节码序列还是有可能的。成本测算是被设计用于确保无限循环会被终止，并且如果校验一个交易的开销庞大得超过预期（比如包含了复杂度随交易大小指数增长的算法），所有节点都对退出的时间达成精确一致。它并不意图作为一种对抗拒绝服务攻击的保护措施。如果一个节点向你发送设计用途为简单地浪费你 CPU 时间的交易，那么在缺少全局广播的情况下，简单地封堵该节点就能有效解决问题。

操作码的预算对于不同的操作码类型是分开的，所以并没有统一的开销模型。此外，测算本身的开销也高。一个更复杂的设计可以是通过只测算"记账区块"的入口点（即运行以方法返回或反向跳转的基本区块即运行）来提前对字节码开销做尽可能多地静态计算。因为只有抽象的开销是紧要的（这并不是一个分析器工具），且所作的限制可以确信是相对较高的，所以不需要去测算每一个基本区块。当

分支两边的目标都不包含反向跳转时，使用两边中的最大值就足够了。如果这种一次一类操作码的记账方式结果不充分，那么这种设计将会被调研。

更大的复杂性来自于约束内存使用的需要。沙箱对*被分配*的字节数而不是*被留存*的字节数强加了一个限额，以达到简化实现的目的。这一策略对那些会产生大量垃圾，但是堆大小的峰值相对较低的智能合约苛刻得没有必要。在实践中

为了把限额设定在一个有效的通用水平上，就可能需要一个与垃圾回收器集成的更复杂的策略。

对**Object.hashCode()**的控制表现为新的允许 JVM 的线程本地随机数产生器在执行开始前重设种子的新的 JNI 调用。种子从被校验的交易的 hash 得出。

最后，需要重点注意，被测算的不止是智能合约代码，而是所有可及的代码。尤其是"影子 JDK"也被提前测算并保存在磁盘上。

14. 可扩展性

区块链与受区块链启发的系统的可扩展性，已经是自中本聪 2008 年提出该技术之后经常性的讨论话题。我们做了很多会影响并保证可扩展性的选择和权衡。由于大多数最初预期的使用场景中不会涉及到高水平的业务量，所以参考实现并没有做大的优化。尽管如此，当需要时，这个架构也可以实现更高水平的可扩展性。

局部可见性。节点只会得到它以某种方式牵涉其中的交易，或者那些依赖于它所牵涉的交易的交易。这种松散连接的设计意味着节点完全可能从没见过交易图的大部分，因此节点也不需要对它做处理。这使得与其它分布式去中心化数据库系统进行直接的规模比较变得困难，因为他们总是用交易每秒的形式来测量性能。对于 Corda，写入操作是按需延迟复制，很难对整个网络报出一个每秒交易数。

分布式节点。一个 Corda 节点的中心是一个消息队列代理。节点的逻辑结构是一系列微服务，并且有潜力在未来运行于分开的机器上。举个例子，内嵌的关系型数据库可以被替换为运行在专用硬件上的外部数据库。虽然单个流无法被并行处理，但是一个沉重负载下的节点通常可以并行运行很多个流。由于流通过代理进入网络，通过普通的数据库连接获取本地的状态，更强大的流处理能力可以通过上线额外的流处理器而获得。这跟 RPC 处理的场景是同样的。

交易外部签名。Corda 交易的标识符，是以除了签名之外的交易内容计算得出的 Merkle 树的根。这有一个缺点是无法通过交易的标准标识符来区分它是被完整签名还是部分签名，但是也意味着签名可以很容易地被并行校验。Corda 智能合约被有意地与底层密码学技术相隔离，它们无法自己请求对签名进行检查：它们在签名校验发生*之后*运行，如果所需的签名缺失，它们根本不会被执行。这保证了就算在交易的智能合约代码无法被并行处理的情况下，单个交易的所有签名也可以被同时检查。（注意，不像其它系统，包含了同一合约的不同交易是*可以*被并行检查的。）

多公证处。在某些场景下，是可能通过上线额外的公证处集群来提高可扩展性的。注意，这只有在交易图具有潜在的可利用结构（如地理上的偏好）时才能

增加性能，因为一个纯粹随机的交易图最后会不断地跨公证处，将状态在公证处之间移动的额外交易会抵消掉性能收益。然而在现实交易中，交易图完全不是随机的，因此这个方法将能发挥作用。

资产再发行。在资产发行者既可靠又在线的场景中，他们可能退出并使用一个新的引用域把一个资产状态再发行回账本上。这有效缩短了资产的依赖图，提升了隐私和可扩展性，而代价则是失去了原子性（发行者可能由于无能或者恶意，将资产退出并不再发行）。

非验证公证处。在交易被公证之前校验其有效性，有可能是非 BFT 公证处的主要开销。在原始吞吐量比账本完整性更重要的场景中，有可能使用非验证公证处。参考§7.3。

Corda 网络的主要瓶颈预计是在公证处集群上，尤其是由相互不信任的节点组成的拜占庭容错（BFT）集群。BFT 集群很可能在一定程度上更慢，因为其底层协议罗嗦且对延迟敏感；还有部分原因是，使用 BFT 协议有益的主要情形，是不存在一个可用作解决欺诈和其它争议的共同法律体系。也就是说，当集群参与者分布在全世界各处时，光速就成了主要的限制因素。

Corda 节点的主要瓶颈预计会是流的检查点，因为这一处理包括走堆栈然后将状态的快照写出到稳定存储中。这两种操作都是计算密集型的。这似乎是意料之外的，其它平台的瓶颈通常是在签名的检查操作上。然而值得注意，其它平台之所以不在检查点操作上存在瓶颈的主要原因，是他们通常根本不提供任何种类的应用层健壮性服务，所以状态检查点的成本（它最终总要还的！）被计到开发者而不是平台头上。当流开发者知道网络通信是幂等的并因此可以重放时，他们就能在流需要被逐出到磁盘上的时候，以额外浪费的工作量为代价，从检查点处理中退出，以获得吞吐量。注意检查点和交易数据可以被保存在任何 NoSQL 数据库中（比如 Cassandra），代价是需要更复杂的备份策略。

归因于局部可见性，节点"及时"地检查交易图，而不是将其作为一个来自于其他参与者的稳定的公告流。这使得如何衡量 Corda 节点可扩展性的问题复杂化了。其它的区块链系统使用每单位时间内交易数的恒定速率来表示性能。然而我们的"单位时间"并不是均匀分布的：假如在提交一份有价值资产时，你需要

检查一个由更多交易组成的交易图，并且用户期望交易立刻生效，那么每秒检查 1000 笔交易的能力是不够好的。平台的未来版本可能会提供新特性，以允许开发者消除 Corda 交易检查的这一尖刻性质，比如当开发者知道他们马上就会需要请求数据时，就向一个节点预推送交易。

15. 隐私

私密性不是独立的功能，它在本文中许多其他方面都有描述，本节总结了各处描述的私密功能。Corda 利用多种技术来提高分布式账本系统中用户的私密性。

部分数据的可见性。事务不像其他系统那样在全局范围内广播。

事务分离。交易的结构像 Merkle 树，可以有个别子组件透露给参与方，它已获知 Merkle 树的根哈希值。此外，它们可以在不知道对方的情况下就签署交易。见 5.7 节。

密钥随机化。库生成和使用都有一个随机密钥，它和没有相应证书的身份标识不关联。

图修剪。涉及流动资产的大型交易图可以"修理"，它要求发行人使用一个新的参考字段重新发行资产到账本。这个操作不是原子化的，但能有效地把资产从旧版本链到新版本，这意味节点在验证期间不会尝试探索原始的依赖图。

Corda 已考虑未来整合额外的隐私技术。在所有潜在升级中，有三个特别值得一提。

安全硬件。虽然我们将数据传播的范围缩小到只需要查看数据的节点，在一个去中心的数据库中，"需要"还可以是一个直观的概念，数据往往只需要进行安全检查。我们已经成功地在安全飞地保护 JVM 下，使用英特尔 SGX ™试验了运行合约验证。安全硬件平台允许在一个不可调试、防篡改的环境中执行计算，对于在该环境中运行的软件，只能获得仅对该实例访问的加密密钥。对于在互联网上的软件第三方远程证明，它确实运行在安全状态。通过节点远程证明对方在一个地区运行智能合约的逻辑验证，这让一个事务在一个密钥下被传送到一个对等加密的节点成为可能。

安全硬件开启了潜在的单击隐私模型，这将极大地简化写智能合约的任务。然而，它仍然需要将敏感数据发送到对方，然后对方可能试图攻击硬件或利用侧通道，从加密的容器内提取商业信息。

混合网络。有些节点可能知道与它们无直接关系的交易，例如公证人或监管节点。即便使用了随机密钥，节点仍然可以获得有价值的身份信息，如通过简单地检查源 IP 地址或该节点发出的用于公证的认证证书。对这个问题的传统加密解决方案，是混合网络。最著名的混合网络是 Tor，但更合适的设计将是一个自我匿名转信站。在混合网络中，使用一小组随机选择的节点所拥有的密钥，消息像洋葱般的方式被反复加密。洋葱中的每一层包含下一"跳"的地址。

一旦消息传递到第一跳，它将解密以获知未来的加密层并转发过去。返回路径以类似的方式运行。向 Corda 协议添加混合网络将允许用户选择升级隐私，代价是较高的延迟和更多的失败网络节点暴露。

零知识证明。在去中心化数据库系统中，解决私密问题的圣杯是使用零知识证明去说服对方一个交易是有效的，而不向它们透露交易的内容。虽然这些技术还未实际用于通用智能合约的执行，但近年来我们已取得了巨大的进步，我们已经设计了数据模型假设，希望将有某天迁移到使用简洁的非交互式参数的零知识证明方式（zkSNARKs）。这些算法允许计算一个固定大小的数学证明，利用混合公共和私有输入来保证程序的正确执行。程序可以直接表示为一个低级多元多项式编码的代数约束系统，或通过简单的模拟 CPU(vnTinyRAM)执行，它本身作为一个大的预计算的约束集实现。由于程序共享了认同功能（即智能合约）与私有输入数据的组合，这足以验证正确性，只要证明的程序可以递归地核实其他证据，即输入事务的证明。BCTV zksnark 算法依赖于对 vntinyram 组成的操作码执行递归证明，所以这不是一个问题。与 Corda 最显著的整合需要紧密地编写通用智能合约（如现金）的汇编语言版本，这需要手写并和 JVM 版本保持一致。不明显但更强大的整合将涉及 vntinyram 后端及提前 JVM 字节码编译器，如 Graal，或者 Graal 的图直接编译为系统约束。根据最近的研究，SSA 形式直接编译器编译的约束和"可扩展的概率可验证明"是最好的集成，这是一个开放的研究问题。

16. 结论

我们介绍了 Corda 是一种为金融行业设计的去中心化数据库系统。它允许统一的数据集分散在相互之间不信任的多节点上、使用运行在 JVM 上的智能合同、提供访问控制和运行模式定义。它是一种新的基于延续的持久化框架，可协助开发人员在网络上协调数据流。它的身份管理系统能确保各方始终知道他们和谁交易。公证人确保分布式共识系统的算法灵活性。该系统运行不需要挖矿或区块链。

它为金融逻辑建模提供了标准类型系统。整个设计考虑了安全：它支持集成安全设备的交易授权签字、事务处理的安全飞地、表达复杂授权策略的组合密钥、基于缓存长度预定的二进制协议，以免整个系统出现常见的缓冲区管理漏洞。与成熟数据库引擎相似，用户可通过普通的 SQL 语句来查询分析与他们有关的账本数据，可以在熟悉的编程语言中轻松处理复杂的多方事务。

最后，该平台定义了标准方法来整合全球总帐与金融基础设施，如高性能市场和净额清算服务。

17. 致谢

作者想要感谢 Richard Gendal Brown、James Carlyle、Shams Asari、Rick Parker、Andras Slemmer、Ross Nicoll、Andrius Dagys、Matthew Nesbit、Jose Coll、Katarzyna Streich、Clinton Alexander、Patrick Kuo、Richard Green、Ian Grigg、Mark Oldfield 和 Roger Willis，感谢他们对这项设计的见解和贡献。我们也想要感谢 Sofus Mortesen 在通用合约分布式账本方面的贡献，还有很多的架构师和来自全球金融机构的主题专家，他们贡献了他们的知识、需求和思想。同时也要感谢为我们开发的底层框架、协议和组件的作者。

最后，我们要感谢 Satoshi Nakamoto，没有他将无法实现任何文中提到的内容。

参考文献

[1] Brown, Carlyle, Grigg, Hearn. Corda: An introduction. http://r3cev.com/s/corda-introductory-whitepaper-final.pdf, 2016.

[2] Fay Chang, Jerey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst., 26(2):4:1{4:26, June 2008.

[3] Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[4] Lindholm, Yellin, Bracha, & Buckley. The Java Virtual Machine Specification Java SE 8 Edition. https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf, 2015.

[5] Buterin et al. A Next-Generation Smart Contract and Decentral-ized Application Platform. https://github.com/ethereum/wiki/wiki/%5BEnglish%5D-White-Paper, 2014.

[6] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13, SSYM'04, pages 21{21, Berkeley, CA, USA, 2004. USENIX Association.

[7] OASIS. Advanced message queuing protocol (amqp) version 1.0, 2012.

[8] Mike Hearn. Bitcoin micropayment channels. https://bitcoinj.github.io/working-with-micropayments, 2014.

[9] Mike Hearn, Gavin Andresen. Bitcoin payment protocol. https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki, 2013.

[10] java.time.Instant documentation. https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html, 2014.

[11] PKWARE. Zip

le format. https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT, 1989.

[12] David Siegel. http://www.coindesk.com/understanding-dao-hack-journalists/, 2016.

[13] Pieter Wiulle. Hierarchical deterministic wallets. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, 2013.

[14] Stefan Thomas. Crypto-conditions. https://interledger.org/five-bells-condition/spec.html, 2016.

[15] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference, USENIX ATC'14, pages 305{320, Berkeley, CA, USA, 2014. USENIX Association.

[16] Christopher Malmo. http://motherboard.vice.com/read/bitcoin-is-unsustainable, 2015.

[17] Tim Swanson. http://tabbforum.com/opinions/settlement-risks-involving-public-blockchains, 2016.

[18] Alysson Bessani, Jo~ao Sousa, and Eduardo E. P. Alchieri. State machine replication for the masses with bft-smart. In Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN '14, pages 355{362, Washington, DC, USA, 2014. IEEE Computer Society.

[19] Hal Finney. Best practice for fast transaction acceptance - how high is the risk? https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384.

[20] Jsr 338: Java persistence api. http://download.oracle.com/otn-pub/jcp/persistence-2_1-fr-eval-spec/JavaPersistence.pdf?AuthParam=1478095024_77b7362fd5bd185ebf8d2cd2a071a14d, 2013.

[21] Jsr 349: Bean validation constraints. https://www.jcp.org/en/jsr/detail?id=349, 2013.

[22] Simon Peyton Jones, Jean-Marc Eber, and Julian Seward. Composing contracts: An adventure in

nancial engineering (functional pearl). SIGPLAN Not., 35(9):280-292, September 2000.

[23] David J. Pearce and Lindsay Groves. Designing a verifying compiler: Lessons learned from developing whiley. Science of Computer Programming, 113, Part 2:191-220, 2015. Formal Techniques for Safety-Critical Systems.

[24] Markus Voelter and Sascha Lisson. Supporting diverse notations in mps' projectional editor. In Proceedings of the 2nd International Workshop on The Globalization of Modeling Languages co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, GEMOC@Models 2014, Valencia, - Spain, September 28, 2014., pages 7-16, 2014.

[25] Bitcoin trezor device. https://bitcointrezor.com/, 2016.

[26] Reactivex. https://www.reactivex.io, 2016.

[27] C. Mitchell and Institution of Electrical Engineers. Trusted Computing. Computing and Networks Series. Institution of Engineering and Technology, 2005.

[28] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM, 24(2):84-90, February 1981.

[29] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In 23rd USENIX Security Symposium (USENIX Security 14), pages 781-796, San Diego, CA, August 2014. USENIX Association.

[30] Graal research compiler. http://openjdk.java.net/projects/graal/, 2016.

[31] Eli Ben-Sasson, Iddo Ben-Tov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear pcps. Cryptology ePrint Archive, Report 2016/646, 2016. http://eprint.iacr.org/2016/646.

名词翻译对照

state 状态

transaction 交易

flow 流

trading flow 交易流

resolution flow 解析流

object graph 对象图

Java classpath Java 类路径

deterministic key derivation 确定性密钥派生（§5.6）

oracle 谕示（§5.7）

encumbrance 留置（§5.8）

clause 条款（§9.1）

projectional editing 投影式编辑（§9.4）

challenge 验证（§10.1）

event horizon 事件边界（§12）

accounting block 记账区块（§13）

恒生区块链官方网站

http://www.51chain.net

区块链技术社区门户

http://bbs.51chain.net

让区块链开发变简单

微信扫描二维码·关注恒生技术之眼

掌握恒生区块链创新技术与应用动态