滴雨科技区块链-架构篇

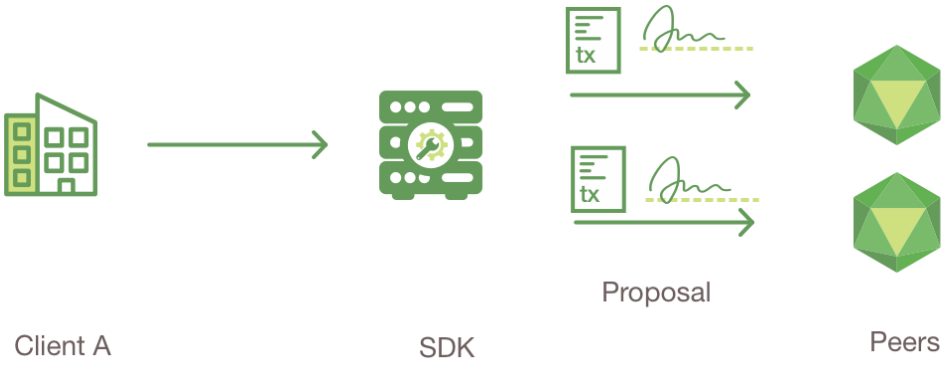
1 交易流程

本文档概述了标准资产交换过程中发生的交易机制。该方案包括买卖萝卜的两个客户 A 和 B。他们每个人在网络上都有一个对等方，通过它们发送交易并与分类账进行交互。

假设条件

此流程假定已建立并运行通道。应用程序用户已经在组织的证书颁发机构（CA）中注册并注册，并收到了必要的加密材料，用于对网络进行身份验证。

链码（包含一组代表萝卜市场初始状态的键值对）安装在对等点上，并在通道上实例化。链码包含逻辑，该逻辑定义了一组交易指令和萝卜的议定价格。背书政策也被定为这个 chaincode，指出双方 `peerA` 并 `peerB` 必须认可任何交易。

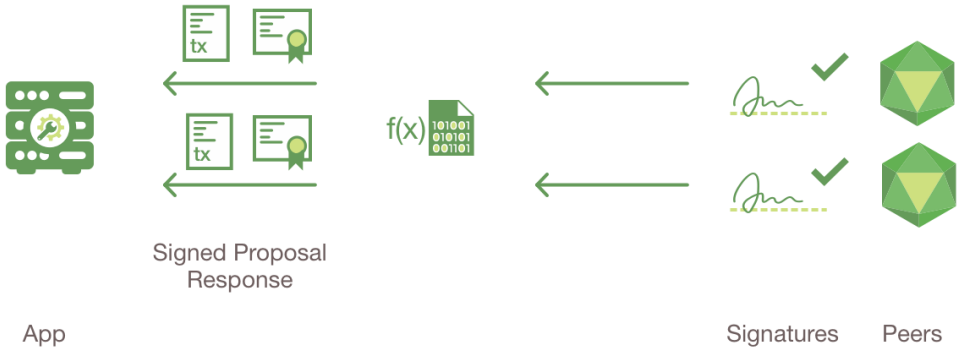


1.1 客户 A 发起交易

发生了什么？客户端 A 正在发送购买萝卜的请求。此请求的目标是 `peerA` 和 `peerB`，分别代表客户 A 和客户 B。认可政策规定，两个对等方都必须认可任何交易，因此请求转到 `peerA` 和 `peerB`。

接下来，构造交易建议。利用受支持的 SDK（节点，Java，Python）的应用程序利用可用的 API 之一来生成交易建议。该提议是为了读取和/或更新分类帐而调用具有某些输入参数的链码功能的请求。

SDK 充当填充程序，将交易建议打包为正确设计的格式（基于 gRPC 的协议缓冲区），并使用用户的加密凭据为该交易建议生成唯一的签名。



1.2 认可同行验证签名并执行交易

背书的对等方验证（1）交易建议书格式正确，（2）过去尚未提交过交易建议书（重放攻击保护），（3）签名有效（使用 MSP）和（4 ）提交者（在示例中为客户 A）被适当授权在该通道上执行提议的操作（即，每个背书对等方都确保提交者满足该通道的 *Writer* 政策）。认可对等方将交易建议输入作为所调用链码功能的参数。然后针对当前状态数据库执行链码，以产生包括响应值，读取集和写入集（即表示要创建或更新的资产的键/值对）的事务结果。此时，不会对分类帐进行任何更新。这些值的集合以及背书的对等方的签名作为“建议响应”传递回 SDK，该 SDK 解析供应用程序使用的有效负载。

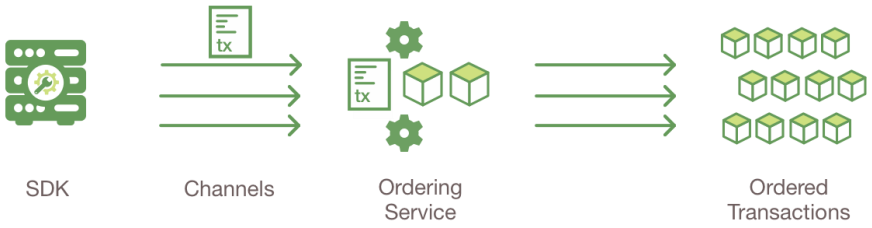
注意

MSP 是对等组件，允许对等组件验证来自客户端的交易请求并签署交易结果（背书）。写入策略是在频道创建时定义的，并确定哪些用户有权向该频道提交交易。有关会员资格的更多信息，请查看我们的 会员资格文档。

1.3 检查提案答复

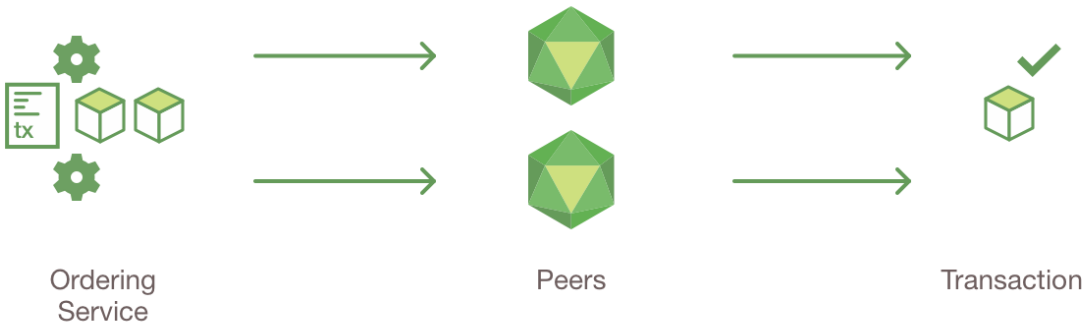
该应用程序验证背书的对等方签名，并比较提案响应以确定提案响应是否相同。如果链码仅用于查询分类帐，则应用程序将仅检查查询响应，并且通常不会将交易提交给排序服务。如果客户端应用程序打算将交易提交给排序服务以更新分类帐，则应用程序将确定提交之前是否已满足指定的背书策略（即 peerA 和 peerB 都背书了）。该体系

结构使得即使应用程序选择不检查响应或以其他方式转发未认可的交易，认可策略仍将由对等方强制执行，并在提交验证阶段得到维护。



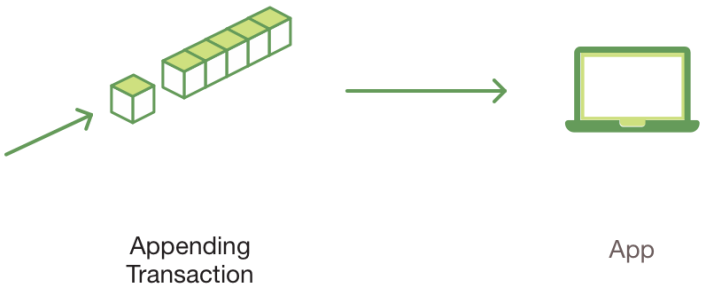
1.4 客户将背书组合成交易

该应用程序在“交易消息”中将交易建议和响应“广播”到排序服务。事务将包含读/写集，背书的对等方签名和通道 ID。排序服务不需要检查交易的全部内容即可执行其操作，它仅从网络中的所有渠道接收交易，按渠道按时间顺序对它们进行排序，并为每个渠道创建交易块。



1.5 交易已验证并提交

交易块被“交付”给渠道上的所有对等方。验证块中的事务以确保执行背书策略，并确保自读取集是由事务执行生成的以来，读取集变量的分类帐状态没有发生变化。区块中的交易被标记为有效或无效。



1.6 分类帐已更新

每个对等方将块附加到通道的链中，并且对于每个有效事务，将写集提交到当前状态数据库。每个对等方均会发出一个事件，以通知客户端应用程序交易（调用）已被不可变地附加到链上，并通知交易是经过验证还是无效。

注意

提交事务后，应用程序应侦听事务事件，例如，使用 `submitTransaction` 自动侦听事务事件的 API。如果不监听事务事件，您将不会知道您的事务是否实际上已被排序，验证并提交到分类账。

请参阅序列图以更好地了解事务流。

2 Hyperledger Fabric SDK

Hyperledger Fabric 打算为多种编程语言提供许多 SDK。交付的前两个是 Node.js 和 Java SDK。我们希望在后续版本中提供 Python，REST 和 Go SDK。

- Hyperledger Fabric Node SDK 文档。
- Hyperledger Fabric Java SDK 文档。

3 服务发现

3.1 为什么我们需要服务发现？

为了在对等方上执行链码，将交易提交给排序者，并就交易状态进行更新，应用程序连接到 SDK 公开的 API。

但是，SDK 需要大量信息，以便允许应用程序连接到相关的网络节点。除了通道上排序者和对等方的 CA 和 TLS 证书及其 IP 地址和端口号之外，它还必须了解相关的认可策略以及已安装链码的对等方（因此应用程序知道哪些对等方发送链码提案给）。

在 v1.2 之前，此信息是静态编码的。但是，此实现对网络更改（例如添加已安装相关链码的对等项或临时脱机的对等项）的动态响应不是动态的。静态配置还不允许应用程序对背书策略本身的更改做出反应（如新组织加入渠道时可能发生的情况）。

此外，客户端应用程序无法知道哪些对等方已更新分类帐，哪些未更新。结果，该应用程序可能向其账本数据与网络其余部分不同步的对等方提交提案，从而导致交易在提交时失效，从而浪费资源。

所述**发现服务**由具有对等体动态地计算所需要的信息，并在消耗性的方式呈现给 SDK 改进了这个过程。

3.2 服务发现在 Fabric 中的工作方式

该应用程序被引导，知道应用程序开发人员/管理员信任的一组对等端，以提供对发现查询的真实响应。客户端应用程序要使用的一个好的候选对等体是同一组织中的一个。请注意，为了使发现服务知道对等节点，必须为它们

`EXTERNAL_ENDPOINT` 定义一个。要查看如何执行此操作，请查看我们的服务发现 CLI 文档。

该应用程序向发现服务发出配置查询，并获取与网络的其余节点进行通信所需的所有静态信息。通过向对等方的发现服务发送后续查询，可以随时刷新此信息。

该服务在对等方（而不是应用程序）上运行，并使用 Gossip 通信层维护的网络元数据信息来找出哪些对等方处于联机状态。它还从对等方的状态数据库中获取信息，例如任何相关的认可策略。

通过服务发现，应用程序不再需要指定他们需要认可的对等方。SDK 可以简单地将查询发送给发现服务，询问给定通道和链码 ID 所需的对等节点。然后，发现服务将计算由两个对象组成的描述符：

1. **布局**：对等体组的列表以及应从每个组中选择的对等体数量。
2. **组到对等体映射**：从布局中的组到通道的对等体。实际上，每个组很可能是代表各个组织的同级物，但是由于服务 API 是通用的并且对组织无知，因此这仅仅是一个“组”。

以下是评估每个组织中有两个对等方的策略的描述符示例 。`AND(Org1, Org2)`

```
Layouts: [
     QuantitiesByGroup: {
       “Org1”: 1,
       “Org2”: 1,
     }
],
EndorsersByGroups: {
  “Org1”: [peer0.org1, peer1.org1],
  “Org2”: [peer0.org2, peer1.org2]
}
```

换句话说，背书策略要求来自 Org1 中一个对等方和 Org2 中一个对等方的签名。它提供了可以认可的组织（`peer0` 以及 `peer1` Org1 和 Org2）中可用对等方的名称。

然后，SDK 从列表中选择一个随机布局。在上面的示例中，背书策略为 Org1 `AND` Org2。如果不是 `OR` 策略，则 SDK 会随机选择 Org1 或 Org2，因为来自任一 Org 对等方的签名都将满足该策略。

SDK 选择布局后，它将根据客户端指定的条件从布局的对等方中进行选择（SDK 可以执行此操作，因为它可以访问分类帐高度等元数据）。例如，根据布局中每个组中对等点的数量，它可能更喜欢具有更高分类帐高度的对等点-或排除应用程序已发现其处于脱机状态的对等点。如果没有一个基于该标准的对等体是可取的，那么 SDK 将从最符合该标准的对等体中随机选择。

3.3 发现服务的功能

发现服务可以响应以下查询：

- **配置查询**：返回 `MSPConfig` 渠道中所有组织的以及渠道的排序者端点。
- **对等成员资格查询**：返回已加入频道的对**等成员**。
- **背书查询**：返回频道中给定链码的背书描述符。
- **本地对等成员资格查询**：返回响应查询的对等方的本地成员资格信息。默认情况下，客户端需要是对等方的管理员才能响应此查询。

3.4 特殊要求

当对等体在启用 TLS 的情况下运行时，客户端在连接到对等体时必须提供 TLS 证书。如果未将对等方配置为验证客户端证书（clientAuthRequired 为 false），则此 TLS 证书可以是自签名的。

4 频道

Hyperledger Fabric `channel` 是两个或多个特定网络成员之间进行通信的专用“子网”，目的是进行专用和机密交易。渠道由成员（组织），每个成员的锚点，共享账本，链码应用程序和排序服务节点定义。网络上的每个事务都在一个通道上执行，在该通道上必须对各方进行身份验证并授权在该通道上进行交易。加入通道的每个对等方都有成员资格服务提供者（MSP）赋予的自己的身份，该成员身份提供商会向其通道对等方和服务进行身份验证。

要创建新渠道，客户端 SDK 会调用配置系统链码并引用诸如和成员（组织）之类的属性。该请求为通道分类帐创建一个，该分类帐存储有关通道策略，成员和锚点对等点的配置信息。在将新成员添加到现有频道时，此创世块或（如果适用）与新成员共享一个较新的重新配置块。 `anchor peersgenesis block`

注意

有关配置事务的属性和原型结构的更多详细信息，请参见“ 通道配置（configtx）”部分。

选举用于信道上的每个部件确定与代表部件的排序服务，其对等体连通。如果没有识别出领导者，则可以使用算法来识别领导者。共识服务对交易进行排序，并以块的形式将其交付给每个主要对等方，然后使用协议将该块分配给其成员对等方，并跨整个通道。 `leading peergossip`

尽管任何一个锚点对等点都可以属于多个通道，因此可以维护多个分类帐，但是没有分类帐数据可以从一个通道传递到另一个通道。分类账的按渠道划分是由配置链码，身份会员服务和 Gossip 数据分发协议定义和实现的。数据的分发（包括有关交易，分类帐状态和通道成员资格的信息）仅限于通道上具有可验证成员身份的对等方。通过通道对等体和分类帐数据的这种隔离，允许需要私有和机密交易的网络成员与业务竞争者和其他受限制成员在同一区块链网络上共存。

5 CouchDB 作为状态数据库

5.1 状态数据库选项

状态数据库选项包括 LevelDB 和 CouchDB。LevelDB 是对等进程中嵌入的默认键值状态数据库。CouchDB 是可选的替代外部状态数据库。与 LevelDB 键值存储一样，CouchDB 可以存储以链码建模的任何二进制数据（CouchDB 附件功能内部用于非 JSON 二进制数据）。但作为 JSON 文档存储，当将链码值（例如资产）建模为 JSON 数据时，CouchDB 还可启用针对链码数据的丰富查询。

LevelDB 和 CouchDB 都支持核心链代码操作，例如获取和设置密钥（资产）以及基于密钥进行查询。可以按范围查询键，并且可以对组合键进行建模以启用针对多个参数的等效查询。例如，复合关键字 `owner,asset_id` 可以用于查询某个实体拥有的所有资产。这些基于密钥的查询可用于针对分类帐的只读查询，以及用于更新分类帐的事务。

如果您将资产建模为 JSON 并使用 CouchDB，则还可以使用 Chaincode 中的 CouchDB JSON 查询语言对链码数据

值执行复杂的丰富查询。这些类型的查询非常适合了解分类帐中的内容。这些类型的查询的建议响应通常对客户端应用程序有用，但通常不会作为事务提交给排序服务。实际上，对于富查询，不能保证结果集在链代码执行和提交时间之间是稳定的，因此，除非查询可以保证结果集在链代码执行时间和执行时间之间是稳定的，否则富查询不适用于更新事务。提交时间，或者可以处理后续事务中的潜在更改。例如，

CouchDB 作为对等体与单独的数据库进程一起运行，因此在设置，管理和操作方面还有其他注意事项。您可能会考虑从默认的嵌入式 LevelDB 开始，如果需要其他复杂的丰富查询，请移至 CouchDB。将链式代码资产数据建模为 JSON 是一个好习惯，这样您就可以选择将来执行复杂的丰富查询。

注意

CouchDB JSON 文档的密钥只能包含有效的 UTF-8 字符串，并且不能以下划线（“ _”）开头。无论您使用的是 CouchDB 还是 LevelDB，都应避免在键中使用 U + 0000（无字节）。

CouchDB 中的 JSON 文档不能使用以下值作为顶级字段名称。这些值保留供内部使用。

- `Any field beginning with an underscore, "_"`
- `~version`

5.2 从 Chaincode 使用 CouchDB

5.2.1 链码查询

大部分的 chaincode 垫片的 API 可与性 LevelDB 或 CouchDB 的状态数据库，可以使用例如 `GetState`，`PutState`，`GetStateByRange`，`GetStateByPartialCompositeKey`。此外，当您将 CouchDB 用作状态数据库并在链码中将资产建模为 JSON 时，您可以通过使用 `GetQueryResult` API 并传递 CouchDB 查询字符串来对状态数据库中的 JSON 执行丰富查询。查询字符串遵循 CouchDB JSON 查询语法。

该 marbles02 织物样品 演示使用 CouchDB 的查询从 chaincode。它包括一个 `queryMarblesByOwner()` 函数，该函数通过将所有者 ID 传递到链码中来演示参数化查询。然后，它使用 JSON 查询语法在状态数据中查询 docType 与“大理石”匹配的 JSON 文档和所有者 ID：

```
{"selector":{"docType":"marble","owner":<OWNER_ID>}}
```

5.2.1.1 CouchDB 分页

Fabric 支持针对丰富查询和基于范围的查询的查询结果分页。支持分页的 API 允许将页面大小和书签用于范围查询和丰富查询。为了支持有效的分页，必须使用 Fabric 分页 API。具体来说，`limit` 由于 Fabric 本身管理查询结果的分页并隐式设置传递给 CouchDB 的 pageSize 限制，因此在 CouchDB 查询中将不使用 CouchDB 关键字。

如果使用分页查询 API（`GetStateByRangeWithPagination()`，`GetStateByPartialCompositeKeyWithPagination()` 和 `GetQueryResultWithPagination()`）指定了 pageSize，则一组结果（由 pageSize 绑定）将与书签一起返回到链码。书签可以从链码返回给调用客户端，客户端可以在后续查询中使用书签来接收结果的下一个“页面”。

分页 API 仅用于只读事务，查询结果旨在支持客户端分页要求。对于需要读写的事务，请使用非分页的 chaincode 查询 API。在 chaincode 中，您可以将结果集迭代到所需的深度。

不论是否使用分页 API，所有链码查询均受绑定 `totalQueryLimit`（默认为 100000）`core.yaml`。这是链码将迭代并返回给客户端的最大结果数，以避免意外或恶意的长时间运行的查询。

注意

无论 chaincode 是否使用分页查询，对等方都将根据中的 `internalQueryLimit`（默认值为 1000）从批量查询

CouchDB `core.yaml`。此行为可确保在执行链代码时在对等方和 CouchDB 之间传递合理大小的结果集，并且对链代码和调用客户端透明。

一个例子使用分页被包括在使用的 CouchDB 教程。

5.2.1.2 CouchDB 索引

为了使 JSON 查询高效，CouchDB 中的索引是必需的，并且对于具有某种排序的任何 JSON 查询都是必需的。索引可以与链码一起打包在 `/META-INF/statedb/couchdb/indexes` 目录中。每个索引必须在其自己的文本文件中 `*.json` 定义，扩展名必须遵循 CouchDB 索引 JSON 语法，并以 JSON 格式格式化索引定义 。例如，为了支持上述大理石查询，在 `docType` 和 `owner` 字段上提供了样本索引：

```
{"index":{"fields":["docType","owner"]},"ddoc":"indexOwnerDoc",
"name":"indexOwner","type":"json"}
```

样本索引可以在这里找到。

链码 `META-INF/statedb/couchdb/indexes` 目录中的任何索引都将与链码打包在一起以进行部署。当将链码都安装在对等方上并在对等方的一个通道上实例化后，索引将自动部署到对等方的通道和链条特定的状态数据库（如果已配置为使用 CouchDB）。如果先安装链码，然后在通道上实例化链码，则索引将在链码**实例化**时部署。如果已经在通道上实例化了链码，并且您随后在对等节点上安装了链码，则索引将在链码**安装** 时进行部署。

部署后，链码查询将自动使用索引。CouchDB 可以根据查询中使用的字段自动确定要使用的索引。或者，在选择器查询中，可以使用 `use_index` 关键字指定索引。

在安装的链码的后续版本中可能存在相同的索引。要更改索引，请使用相同的索引名称，但要更改索引定义。安装/实例化后，索引定义将重新部署到对等方的状态数据库。

如果您已经有大量数据，并在以后安装链码，则安装时创建索引可能需要一些时间。同样，如果您已经有大量数据并实例化链码的后续版本，则创建索引可能需要一些时间。避免在这些时候调用查询状态数据库的链码函数，因为在初始化索引时链码查询可能会超时。在事务处理期间，当将块提交到分类账时，索引将自动刷新。

5.3 CouchDB 配置

通过将 `stateDatabase` 配置选项从 goleveldb 更改为 CouchDB，可以将 CouchDB 作为状态数据库启用。另外，`couchDBAddress` 需要配置为指向对等方要使用的 CouchDB。如果 CouchDB 配置了用户名和密码，则用户名和密码属性应使用管理员用户名和密码填充。本 `couchDBConfig` 节提供了其他选项，并在适当的位置进行了记录。重新启动对等后，对 *core.yaml 的*更改 将立即生效。

您还可以传入 docker 环境变量以覆盖 core.yaml 值，例如 `CORE_LEDGER_STATE_STATEDATABASE` 和 `CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS`。

以下是 `stateDatabase` 来自 *core.yaml* 的部分：

```
state:
  # stateDatabase - options are "goleveldb", "CouchDB"
  # goleveldb - default state database stored in goleveldb.
  # CouchDB - store state database in CouchDB
  stateDatabase: goleveldb
  # Limit on the number of records to return per query
  totalQueryLimit: 10000
  couchDBConfig:
     # It is recommended to run CouchDB on the same server as the peer, and
     # not map the CouchDB container port to a server port in docker-compose.
     # Otherwise proper security must be provided on the connection between
     # CouchDB client (on the peer) and server.
     couchDBAddress: couchdb:5984
     # This username must have read and write authority on CouchDB
     username:
     # The password is recommended to pass as an environment variable
     # during start up (e.g. LEDGER_COUCHDBCONFIG_PASSWORD).
     # If it is stored here, the file must be access control protected
```

```
    # to prevent unintended users from discovering the password.
    password:
    # Number of retries for CouchDB errors
    maxRetries: 3
    # Number of retries for CouchDB errors during peer startup
    maxRetriesOnStartup: 10
    # CouchDB request timeout (unit: duration, e.g. 20s)
    requestTimeout: 35s
    # Limit on the number of records per each CouchDB query
    # Note that chaincode queries are only bound by totalQueryLimit.
    # Internally the chaincode may execute multiple CouchDB queries,
    # each of size internalQueryLimit.
    internalQueryLimit: 1000
    # Limit on the number of records per CouchDB bulk update batch
    maxBatchUpdateSize: 1000
    # Warm indexes after every N blocks.
    # This option warms any indexes that have been
    # deployed to CouchDB after every N blocks.
    # A value of 1 will warm indexes after every block commit,
    # to ensure fast selector queries.
    # Increasing the value may improve write efficiency of peer and CouchDB,
    # but may degrade query response time.
    warmIndexesAfterNBlocks: 1
```

CouchDB 的托管与 Hyperledger 面料供应搬运工容器具有与设置在传递环境变量 CouchDB 的用户名和密码的能力 `COUCHDB_USER`，并 `COUCHDB_PASSWORD` 使用多克撰写脚本的环境变量。

对于 Fabric 随附的 docker 映像之外的 CouchDB 安装，必须编辑该安装的 local.ini 文件以设置管理员用户名和密码。

Docker compose 脚本仅在创建容器时设置用户名和密码。该 *local.ini* 文件必须进行编辑，如果用户名或密码创建容器后改变。

注意

在每次对等启动时都会读取 CouchDB 对等选项。

5.4 查询的良好做法

避免对查询使用链码，这将导致整个 CouchDB 数据库的扫描。全长数据库扫描将导致较长的响应时间，并且将降低网络的性能。您可以采取以下一些步骤来避免冗长的查询：

- 使用 JSON 查询时：
    - 确保在 chaincode 包中创建索引。
    - 避免查询操作符如 `$or`，`$in` 和 `$regex`，从而导致整个数据库扫描。
- 对于范围查询，组合键查询和 JSON 查询：
    - 使用分页支持（从 v1.3 开始），而不是一个较大的结果集。
- 如果要构建仪表板或收集汇总数据作为应用程序的一部分，则可以查询从链上复制数据的链外数据库。这将使您可以查询和分析针对您的需求进行优化的数据存储中的区块链数据，而不会降低网络性能或中断交易。为此，应用程序可以使用块或链码事件将事务数据写入链下数据库或分析引擎。对于接收到的每个块，块侦听器应用程序将循环访问块事务，并使用每个有效事务的键/值写入来构建数据存储 `rwset`。基于对等通道的事件服务提供可重播事件，以确保下游数据存储的完整性。

6 基于对等渠道的事件服务

6.1 总体概述

在 Fabric 的早期版本中，对等事件服务被称为事件中心。每当将新块添加到对等者的分类帐中时，该服务都会发送事件，而不管该块所涉及的通道如何，并且只有运行事件对等体的组织成员才能访问该事件（即，与该事件相关联的成员）。

从 v1.1 开始，有两个提供事件的新服务。这些服务使用完全不同的设计来按通道提供事件。这意味着事件的注册发生在通道级别而不是对等端，从而可以对访问对等端的数据进行精细控制。接收事件的请求是从对等组织外部的身份接受的（由通道配置定义）。这还提供了更高的可靠性，以及一种接收可能错过的事件的方式（无论是由于连接问题还是由于对等方正在加入已经运行的网络）。

6.2 可用服务

- `Deliver`

该服务将已提交到分类帐的整个块发送出去。如果链码设置了任何事件，则可以在 `ChaincodeActionPayload` 块的中找到这些事件。

- `DeliverFiltered`

此服务发送“已过滤”的块，这是有关已提交到分类账的块的最少信息集。它旨在用于对等方所有者希望外部客户主要接收有关其交易和那些交易状态的信息的网络中。如果链码设置了任何事件，则可以在 `FilteredChaincodeAction` 过滤后的块中找到这些事件。

注意

链码事件的有效载荷将不包含在已过滤的块中。

6.3 如何注册活动

通过将包含传递搜索信息消息的信封发送到对等方（包含所需的开始和停止位置，搜索行为）（直到准备就绪，否则阻塞，如果未准备就绪，则通过失败）来注册来自任一服务的事件。有辅助变量，`SeekOldest` 并且 `SeekNewest` 可以用于指示最早的（即第一）块或总帐的最新（即最后一个）块。要使服务无限期发送事件，该 `SeekInfo` 消息应包含的停止位置 `MAXINT64`。

注意

如果在对等方上启用了双向 TLS，则必须在信封的通道标题中设置 TLS 证书哈希。

默认情况下，这两种服务都使用“通道读取器”策略来确定是否授权请求客户端的事件。

6.4 传递响应消息概述

事件服务发送回 `DeliverResponse` 消息。

每条消息包含以下内容之一：

- status – HTTP 状态代码。如果发生任何故障，两个服务都将返回适当的故障代码；否则，它将在服务完成发送消息请求的所有信息后返回 。`200 - SUCCESSSeekInfo`
- 阻止–仅由`Deliver`服务返回。
- 过滤的块–仅由`DeliverFiltered`服务返回。

过滤的块包含：

- 频道 ID。
- 号（即块号）。
- 过滤交易的数组。
- 交易编号。
    - 类型（例如`ENDORSER_TRANSACTION`，`CONFIG`。
    - 交易验证码。
- 过滤的交易操作。
    - **过滤后的 chaincode 操作数组。**
        - 交易的链码事件（有效载荷无效）。

6.5 SDK 事件文档

有关使用事件服务的更多详细信息，请参阅 SDK 文档。

7 私人数据

注意

本主题假设您对私有数据文档中的概念性材料有所了解 。

7.1 私人数据收集定义

集合定义包含一个或多个集合，每个集合都有一个策略定义，其中列出了集合中的组织，以及用于在背书时控制私人数据分发的属性以及（可选）是否清除数据的属性。

从 Fabric v2.0 Alpha 引入的 Fabric 链码生命周期开始，集合定义是链码定义的一部分。集合由通道成员批准，然后在将链码定义提交给通道时进行部署。所有通道成员的收集文件都必须相同。如果使用对等 CLI 批准并提交链码定义，请使用该`--collections-config`标志指定集合定义文件的路径。如果您使用 Fabric SDK for Node.js，请访问如何安装和启动链码。要使用先前的生命周期过程来部署私有数据集合，请`--collections-config`在实例化 chaincode 时使用该标志。

集合定义由以下属性组成：

- `name`：集合名称。

- `policy`：私有数据收集分发策略定义了允许哪些组织的对等方保留使用 `Signature` 策略语法表示的收集数据，每个成员都包含在 `OR` 签名策略列表中。为了支持读/写事务，私有数据分发策略必须定义比链代码认可策略更广泛的组织集，因为对等方必须具有私有数据才能认可提议的事务。例如，在有十个组织的渠道中，私有数据收集分发策略中可能包含五个组织，但是认可策略可能要求其中三个组织都认可。
- `requiredPeerCount`：在每个背书人签署背书并将提案响应返回给客户之前，每个背书人对等人（跨授权组织）必须成功分发的最小对等人数目。要求传播作为背书的条件，将确保即使背书对等体变得不可用，私有数据也可以在网络中使用。如果 `requiredPeerCount` 是 `0`，它是指不分配**要求**，但可能有一些分布，如果 `maxPeerCount` 大于零。一个 `requiredPeerCount` 的 `0` 通常不建议这样做，因为如果背对端不可用，可能会导致网络中的私有数据丢失。通常，您需要在认可时至少分配一些私有数据，以确保私有数据在网络中多个对等点上的冗余。
- `maxPeerCount`：出于数据冗余的目的，每个认可对等方将尝试向其分发私有数据的其他对等方（在授权组织中）的最大数量。如果在认可时间和提交时间之间没有认可的对等方可用，则作为集合成员但在认可时尚未收到私有数据的其他对等方将能够从分发了私有数据的对等方中获取私有数据。如果将此值设置为 `0`，则不会在背书时分发私有数据，从而会在提交时在所有授权对等方上强制对背书对等方的私有数据。
- `blockToLive`：表示数据在块上应在专用数据库上保留的时间。数据将在指定数量的专用数据库上保留，此后将被清除，从而使该数据已从网络中删除，因此无法从链码中查询它，也无法将其提供给请求对等方。要无限期保留私有数据，即从不清除私有数据，请将 `blockToLive` 属性设置为 `0`。
- `memberOnlyRead`：值为， `true` 表示对等方自动强制仅允许属于集合成员组织之一的客户端读取私有数据。如果来自非成员组织的客户端尝试执行执行读私有数据的链码功能，则链码调用会因错误而终止。 `false` 如果要在单个 chaincode 函数中对更细粒度的访问控制进行编码，请使用的值 。

这是一个样本集合定义 JSON 文件，其中包含两个集合定义的数组：

```
[
 {
    "name": "collectionMarbles",
    "policy": "OR('Org1MSP.member', 'Org2MSP.member')",
    "requiredPeerCount": 0,
    "maxPeerCount": 3,
    "blockToLive":1000000,
    "memberOnlyRead": true
 },
 {
    "name": "collectionMarblePrivateDetails",
    "policy": "OR('Org1MSP.member')",
    "requiredPeerCount": 0,
    "maxPeerCount": 3,
    "blockToLive":3,
    "memberOnlyRead": true
 }
]
```

本示例使用 BYFN 示例网络中的组织， `Org1` 以及 `Org2`。 `collectionMarbles` 定义中的策略授权两个组织都使用私有数据。当链码数据需要对排序服务节点保持私有时，这是一种典型配置。但是， `collectionMarblePrivateDetails` 定义中的策略限制了对渠道中组织子集的访问（在这种情况下 `Org1`）。在实际情况下，渠道中将有许多组织，每个集合中有两个或多个组织在它们之间共享私有数据。

7.2 私人数据传播

由于私有数据不包括在提交给排序服务的交易中，因此也不包括在分发给通道中所有对等方的块中，因此认可对等方在将私有数据分发给授权的其他对等方方面起着重要作用。组织。这样可以确保通道集合中私有数据的可用

性，即使背书的对等方在背书后变得不可用。为了帮助进行这种传播，集合定义中的 `maxPeerCount` 和 `requiredPeerCount` 属性可控制背书时的传播程度。

如果背书的对等方不能成功地将私有数据至少传播到 `requiredPeerCount`，则会将错误返回给客户端。认可对等方将尝试将私有数据分发给不同组织的对等方，以确保每个授权组织都具有私有数据的副本。由于交易不是在链码执行时提交的，因此认可对等方和接收方对等方将私有数据的副本存储在 其区块链旁边的本地中，直到交易被提交为止。 `transient store`

当授权对等方在提交时在其临时数据存储区中没有私有数据的副本时（要么是因为它们不是背书方，要么是因为他们在背书时未通过分发接收到私有数据），他们将尝试拉取根据对等方配置 文件中的对等方属性，*在可配置的时间内配置*来自另一个授权对等方的私有数据。 `peer.gossip.pvtData.pullRetryThresholdcore.yaml`

注意

仅当请求的对等方是私有数据分发策略所定义的集合的成员时，被要求提供私有数据的对等方才返回私有数据。

使用时的注意事项 `pullRetryThreshold`：

- 如果发出请求的对等方能够检索内的私有数据 `pullRetryThreshold`，它将把事务提交到其分类帐（包括私有数据哈希），并将私有数据存储在其状态数据库中，该数据库与其他通道状态数据在逻辑上是分开的。
- 如果发出请求的对等方无法检索中的私有数据 `pullRetryThreshold`，它将把事务提交到其区块链（包括私有数据哈希），而没有私有数据。
- 如果对等方有权使用私有数据，但该对等方丢失了，那么该对等方将无法认可引用了丢失的私有数据的将来的交易-将检测到链式查询丢失的密钥（基于存在的密钥密钥在状态数据库中的哈希值），并且链码将收到错误。

因此，将 `requiredPeerCount` 和 `maxPeerCount` 属性设置得足够大以确保通道中私有数据的可用性非常重要。例如，如果每个认可对等方在事务提交之前变得不可用，则 `requiredPeerCount` 和 `maxPeerCount` 属性将确保私有数据在其他对等方上可用。

注意

为了使集合正常工作，正确配置跨组织 Gossip 很重要。请参阅我们关于 Gossip 数据分发协议的文档，尤其要注意“锚点”和“外部端点”的配置。

7.3 从 Chaincode 引用集合

一组填充 API 可用于设置和检索私有数据。

可以将相同的链码数据操作应用于通道状态数据和私有数据，但是在私有数据的情况下，将在链码 API 中与数据一起指定集合名称，例如 `PutPrivateData(collection,key,value)` 和 `GetPrivateData(collection,key)`。

单个链码可以引用多个集合。

7.3.1 如何在链码提案中传递私人数据

由于链码提案存储在区块链上，因此在链码提案的主要部分中不要包含私有数据也很重要。链码建议中的一个特殊字段称为 `transient` 字段，可用于传递来自客户端的私有数据（或链码将用于生成私有数据的数据），以在对等方上进行链码调用。链码可以 `transient` 通过调用 GetTransient（） API 来检索 字段。此 `transient` 字段将从渠道交易中排除。

7.3.2 保护私人数据内容

如果私有数据相对简单且可预测（例如交易金额），则未经授权访问私有数据集合的渠道成员可以尝试通过域空间的强力哈希猜测猜测私有数据的内容，以期找到与链上私有数据哈希的匹配。因此，可预测的私有数据应包括与私有数据密钥连接并包含在私有数据值中的随机“盐”，以使得无法通过蛮力实际找到匹配的哈希。可以在客户端生成随机“盐”（例如，通过对安全伪随机源进行采样），然后在调用链码时将其与私有数据一起传递到瞬态字段中。

7.3.3 私人数据的访问控制

在 1.3 版之前，仅对同级实施基于集合成员资格的对私有数据的访问控制。基于链码提议提交者组织的访问控制需要以链码逻辑进行编码。从 v1.4 开始，集合配置选项 `memberOnlyRead` 可以根据链码提议提交者的组织自动执行访问控制。有关集合配置定义以及如何设置它们的更多信息，请参考 本主题的“私有数据集合定义”部分。

注意

如果您想要更精细的访问控制，可以将其设置 `memberOnlyRead` 为 false。然后，您可以在链代码中应用自己的访问控制逻辑，例如，通过调用 GetCreator（）链代码 API 或使用客户端身份 链代码库。

7.3.4 查询私有数据

可以像普通通道数据一样使用垫片 API 查询私有数据收集：

- `GetPrivateDataByRange(collection, startKey, endKey string)`
- `GetPrivateDataByPartialCompositeKey(collection, objectType string, keys []string)`

对于 CouchDB 状态数据库，可以使用 shim API 传递 JSON 内容查询：

- `GetPrivateDataQueryResult(collection, query string)`

局限性：

- 根据上面的“私有数据分发”部分中的说明，调用执行范围或富 JSON 查询的链码的客户端应注意，如果他们查询的对等方缺少私有数据，则他们可能会收到结果集的子集。客户端可以查询多个对等方并比较结果，以确定对等方是否可能缺少某些结果集。
- 不支持在单个事务中执行范围或富 JSON 查询并更新数据的 Chaincode，因为无法在无法访问私有数据的对等项或缺少私有数据的对等项上验证查询结果他们有权使用。如果链码调用既查询又更新私有数据，则提案请求将返回错误。如果您的应用程序可以容忍链码执行和验证/提交时间之间的结果集更改，则可以调用一个链码函数来执行查询，然后调用第二个链码函数来进行更新。请注意，可以调用 GetPrivateData（）以检索单个密钥，并且可以与 PutPrivateData（）调用在同一事务中进行，因为所有对等方都可以基于哈希密钥版本来验证密钥读取。

7.3.5 对集合使用索引

注意

Fabric v2.0 Alpha 中引入的 Fabric 链码生命周期不支持将 chainDB 索引与链码一起使用。要使用先前的生命周期模型来部署带有私有数据集合的 bedDB 索引，请访问私有数据体系结构指南 v1.4 版本。

作为状态数据库的 CouchDB 主题描述了索引，该索引可通过 `META-INF/statedb/couchdb/indexes` 在安装链码时将索引打包在目录中而应用于通道的状态数据库以启用 JSON 内容查询。同样，通过将索引打包在 `META-INF/statedb/couchdb/collections/<collection_name>/indexes` 目录中，索引也可以应用于私有数据集合。此处提供示例索引。

7.4 使用私人数据时的注意事项

7.4.1 私人数据清除

可以定期从同级清除私有数据。有关更多详细信息，请参见 `blockToLive` 上面的集合定义属性。

此外，回想一下在提交之前，对等方将私有数据存储在本地临时数据存储中。事务提交时，该数据将自动清除。但是，如果从未将事务提交到通道并因此也从未提交过，则私有数据将保留在每个对等方的临时存储中。通过使用对 `peer.gossip.pvtData.transientstoreMaxBlockRetention` 等 `core.yaml` 文件中的对等属性，在可配置的数字被阻止之后，将从临时存储中清除此数据 。

7.4.2 更新集合定义

要更新集合定义或添加新集合，您可以将链码升级到新版本，并在链码升级事务中传递新的集合配置，例如，`--collections-config` 如果使用 CLI，则使用标志。如果在链码升级过程中指定了集合配置，则必须包括每个现有集合的定义。

升级链码时，您可以添加新的私有数据集合，并更新现有的私有数据集合，例如，将新成员添加到现有集合或更改集合定义属性之一。请注意，您不能更新集合名称或 blockToLive 属性，因为无论对等体的块高如何，都需要一致的 blockToLive。

当对等方提交包含链码升级事务的块时，集合更新将生效。请注意，集合不能删除，因为通道的区块链上可能有先前的私有数据哈希无法删除。

7.4.3 私人数据核对

从 v1.4 开始，添加到现有集合的组织的对等方将在加入集合之前自动获取已提交到集合的私有数据。

此私有数据“对帐”还适用于有权接收私有数据但尚未接收到它的对等方（例如，由于网络故障），通过跟踪在块提交时“丢失”的私有数据。

私人数据对帐会根据 core.yaml 中的 `peer.gossip.pvtData.reconciliationEnabled` 和 `peer.gossip.pvtData.reconcileSleepInterval` 属性定期进行。该对等方将定期尝试从预期具有该数据的其他收集成员对等方获取私有数据。

请注意，此私有数据协调功能仅适用于运行 Fabric v1.4 或更高版本的对等方。

8 读写集语义

本文档讨论有关读写集语义的当前实现的详细信息。

8.1 事务模拟和读写集

在进行交易模拟时，为交易 `endorser` 准备了一个读写集。该含有独特的密钥和他们所承诺的版本号（而不是值）在仿真过程的事务读取列表。该含有独特的密钥列表（尽管可能有重叠出现在读集中的键），交易写入和它们的新值。如果事务执行的更新是删除密钥，则为密钥设置一个删除标记（代替新值）。 `read setwrite set`

此外，如果事务为一个键多次写入一个值，则仅保留最后写入的值。同样，如果事务读取密钥的值，则即使在发出读取之前事务已更新密钥的值，也将返回处于提交状态的值。换句话说，不支持“读即写”语义。

如前所述，密钥的版本仅记录在读取集中。写入集仅包含唯一键的列表及其由事务设置的最新值。

可能有多种实现版本的方案。版本控制方案的最低要求是为给定密钥生成非重复标识符。例如，对版本使用单调递增的数字可以是这样一种方案。在当前实现中，我们使用基于区块链高度的版本控制方案，其中将提交事务的高度用作该事务修改的所有密钥的最新版本。在此方案中，事务的高度由元组表示（txNumber 是块内事务的高度）。与增量编号方案相比，此方案具有许多优点-主要是，它使其他组件（如 saidb，事务模拟和验证）能够做出有效的设计选择。

以下是通过假设交易的仿真准备的示例读写集的说明。为了简单起见，在图示中，我们使用增量数字表示版本。

```
<TxReadWriteSet>
  <NsReadWriteSet name="chaincode1">
    <read-set>
      <read key="K1", version="1">
      <read key="K2", version="1">
    </read-set>
    <write-set>
      <write key="K1", value="V1"
      <write key="K3", value="V2"
      <write key="K4", isDelete="true"
    </write-set>
  </NsReadWriteSet>
<TxReadWriteSet>
```

此外，如果事务在模拟过程中执行范围查询，则范围查询及其结果将添加为读写集 `query-info`。

8.2 使用读写集进行事务验证和更新世界状态

A `committer` 使用读写集的读集部分检查事务的有效性，并使用读写集的写集部分更新受影响密钥的版本和值。

在验证阶段， `valid` 如果事务的读取集中存在的每个密钥的版本与世界状态下的同一密钥的版本相匹配，则考虑事务-假设所有先前的 `valid` 事务（包括同一块中的先前的事务）已落实（*committed-state*）。如果读写集还包含一个或多个查询信息，则执行附加验证。

此额外的验证应确保在查询信息中捕获的结果的超范围（即范围的并集）中没有插入/删除/更新任何键。换句话说，如果我们在提交状态验证期间重新执行任何范围查询（在模拟过程中执行事务），则该范围查询应产生与模拟时事务观察到的结果相同的结果。此检查可确保如果事务在提交期间观察到幻像项，则该事务应标记为无效。请注意，此幻像保护仅限于范围查询(即 `GetStateByRange` 链码中的函数)，尚未针对其他查询实现(即， `GetQueryResult` 链代码中的功能)。其他查询存在幻象的风险，因此，除非应用程序可以保证结果集在仿真与验证/提交时间之间的

稳定性，否则，仅应将其用于未提交排序的只读事务中。

如果事务通过了有效性检查，则提交者将使用写集更新世界状态。在更新阶段，对于写集中存在的每个密钥，将同一密钥在世界状态下的值设置为写集中指定的值。此外，世界状态下的密钥的版本被改变以反映最新版本。

8.3 模拟和验证示例

本节通过一个示例场景帮助理解语义。出于本示例的目的 `k`，在世界状态下，密钥的存在用元组表示，`(k,ver,val)` 其中 `ver`，密钥的最新版本 `k` 具有 `val` 其值。

现在，考虑一组五个交易，所有交易都在世界状态的同一快照上模拟。以下代码片段显示了模拟交易所依据的世界状态的快照，以及每个交易所执行的读取和写入活动的顺序。`T1, T2, T3, T4, and T5`

```
World state: (k1,1,v1), (k2,1,v2), (k3,1,v3), (k4,1,v4), (k5,1,v5)
T1 -> Write(k1, v1'), Write(k2, v2')
T2 -> Read(k1), Write(k3, v3')
T3 -> Write(k2, v2'')
T4 -> Write(k2, v2'''), read(k2)
T5 -> Write(k6, v6'), read(k5)
```

现在，假定这些事务按 T1，..，T5 的顺序排序（可以包含在单个块中或不同块中）

1. `T1` 通过验证，因为它不执行任何读取。此外，键的元组 `k1` 和 `k2` 世界状态被更新为 `(k1,2,v1'), (k2,2,v2')`
2. `T2` 验证失败，因为它读取了一个密钥 `k1`，该密钥已被先前的事务修改- `T1`
3. `T3` 通过验证，因为它不执行读取。此外 `k2`，在世界状态下，键的元组被更新为 `(k2,3,v2'')`
4. `T4` 验证失败，因为它读取了一个密钥 `k2`，该密钥已被先前的事务修改 `T1`
5. `T5` 通过验证，因为它读取了一个密钥，`k5,` 该密钥未被任何先前的事务修改

注意：尚不支持具有多个读写集的事务。

9 Gossip 数据传播协议

Hyperledger Fabric 通过将工作负载划分到事务执行（认可和提交）对等方和事务排序节点之间来优化区块链网络的性能，安全性和可伸缩性。网络操作的这种分离需要安全，可靠和可扩展的数据分发协议，以确保数据的完整性和一致性。为了满足这些要求，Fabric 实施了 **Gossip 数据分发协议**。

9.1 Gossip 协议

对等方利用 Gossip 以可扩展的方式广播分类帐和频道数据。Gossip 消息传递是连续的，并且通道上的每个对等方不断从多个对等方接收最新且一致的分类帐数据。每个闲聊的消息都经过签名，从而使拜占庭参与者可以轻松地识别发送虚假消息的参与者，并可以防止将消息分发到不需要的目标。受延迟，网络分区或其他原因导致丢失块的影响的对等点最终将通过联系拥有这些丢失块的对等点而被同步到当前分类账状态。

基于 Gossip 的数据分发协议在 Fabric 网络上执行三个主要功能：

1. 通过不断识别可用的成员对等点，并最终检测已脱机的对等点，来管理对等点发现和通道成员身份。
2. 在通道上的所有对等点之间分发分类帐数据。数据与通道其余部分不同步的任何对等方都将识别丢失的块，并通过复制正确的数据进行自身同步。
3. 通过允许分类账数据的对等状态传输更新来使新连接的对等节点加速。

基于 Gossip 的广播是由对等方从该信道上的其他对等方接收消息，然后将这些消息转发到该信道上许多随机选择的对等方的操作，其中该数目是可配置的常数。对等方还可以行使拉动机制，而不是等待消息传递。重复此循环，信道成员资格的结果是，分类帐和状态信息不断保持最新并保持同步。为了传播新的数据块，通道上的**领导**对等方从排序服务中提取数据，并开始向其自己组织中的对等方传播 Gossip。

9.2 领导人选举

领导者选举机制用于在每个组织中**选举**一个对等方，该对等方将保持与排序服务的连接，并在其组织的对等方之间发起新到达的块的分发。利用领导者的选择，系统可以有效利用排序服务的带宽。领导者选举模块有两种可能的操作模式：

1. **静态** -系统管理员手动将组织中的对等方配置为领导者。
2. **动态** -同行执行领导者选举程序以选择组织中的一个同行成为领导者。

9.2.1 静态领导人选举

静态领导者选举允许您手动将组织中的一个或多个对等定义为领导者对等体。但是请注意，连接到排序服务的对等点过多可能会导致带宽使用效率低下。要启用静态领导者选举模式，请在的部分中配置以下参数 `core.yaml`：

```
peer:
    # Gossip related configuration
    gossip:
        useLeaderElection: false
        orgLeader: true
```

1.另外，可以配置这些参数并用环境变量覆盖它们：

```
export CORE_PEER_GOSSIP_USELEADERELECTION=false
export CORE_PEER_GOSSIP_ORGLEADER=true
```

注意

以下配置将使对等方保持**待机**模式，即，对等方不会尝试成为领导者：

```
export CORE_PEER_GOSSIP_USELEADERELECTION=false
export CORE_PEER_GOSSIP_ORGLEADER=false
```

2.设置 `CORE_PEER_GOSSIP_USELEADERELECTION` 并 `CORE_PEER_GOSSIP_ORGLEADER` 与 `true` 价值是不明确的，并且将导致错误。

3.在静态配置中，组织管理员负责提供领导者节点的高可用性，以防发生故障或崩溃。

9.2.2 动态领导者选举

动态领导者选举使组织对等方可以**选择**一个对等方，该对等方将连接到排序服务并提取新的模块。该领导者是独立选举组织的同行。

动态选举产生的领导者会向其他同伴发送**心跳**消息，以作为活跃性的证据。如果一个或多个同伴在设定的时间内没有收到**心跳**更新，则他们将选举一位新的领导者。

在网络分区的最坏情况下，组织将有不止一位活跃的领导者来保证弹性和可用性，以允许组织的对等方继续取得进步。在修复网络分区后，其中一位负责人将放弃其领导权。在没有网络分区的稳定状态下，将 **只有**一个活动的领导者连接到排序服务。

以下配置控制领导者**心跳**消息的频率：

```
peer:
    # Gossip related configuration
    gossip:
        election:
            leaderAliveThreshold: 10s
```

为了启用动态领导者选举，需要在以下参数中进行配置 `core.yaml`：

```
peer:
    # Gossip related configuration
    gossip:
        useLeaderElection: true
        orgLeader: false
```

另外，可以配置这些参数并用环境变量覆盖它们：

```
export CORE_PEER_GOSSIP_USELEADERELECTION=true
export CORE_PEER_GOSSIP_ORGLEADER=false
```

9.3 锚点同行

Gossip 使用锚点对等点来确保不同组织中的对等点相互了解。

提交包含对锚点对等体的更新的配置块时，对等点会联系到锚点对等体，并从中了解锚点对等体已知的所有对等体。一旦来自每个组织的至少一个对等方联系到锚定对等方，则锚定对等方将了解该通道中的每个对等方。由于 Gossip 沟通是恒定的，并且因为对等点总是要求被告知他们不认识的任何对等点的存在，因此可以为频道建立成员资格的通用视图。

例如，假设我们在渠道中有三个组织 *A*，*B*，*C*，并且为组织 *C* 定义了一个锚点对等方 *peer0.orgC*。当 *peer1.orgA*（从组织一）接触 *peer0.orgC*，它会告诉它 *peer0.orgA*。当稍后 *peer1.orgB* 与 *peer0.orgC* 联系*时*，后者会告诉前者有关 *peer0.orgA 的信息*。从那时起，组织 *A* 和 *B* 将开始直接交换会员信息，而无需获得任何帮助 *peer0.orgC*。

由于组织之间的通信取决于 Gossip 才能起作用，因此在通道配置中必须至少定义一个锚点对等点。强烈建议每个组织提供自己的一组锚点对等点，以实现高可用性和冗余。注意，锚点对等点不必与领导者对等点相同。

9.3.1 外部和内部端点

为了使 Gossip 有效地工作，对等方需要能够从其自己的组织以及从其他组织的对等方获取对等方的终结点信息。

当对等方被引导时，它将 `peer.gossip.bootstrap` 在其 `core.yaml` 自身中 用来做广告并交换成员信息，从而建立其组织内所有可用对等方的视图。

对等体中的 `peer.gossip.bootstrap` 属性 `core.yaml` 用于引导**组织中的** Gossip。如果您使用闲话，通常将组织中的所有对等点配置为指向一组初始的引导对等点（可以指定以空格分隔的对等点列表）。内部端点通常由对等

方本身自动计算，或者只是通过 `core.peer.address` in 显式传递 `core.yaml`。如果需要覆盖此值，则可以导出 `CORE_PEER_GOSSIP_ENDPOINT` 为环境变量。

建立**组织之间的**通信同样需要引导信息。初始的跨组织引导信息是通过上述“锚点”设置提供的。如果要使组织中的其他对等方被其他组织所了解，则需要 `peer.gossip.externalendpoint` 在 `core.yaml` 您的对等方中设置。如果未设置，则不会将对等方的端点信息广播给其他组织中的对等方。

要设置这些属性，请发出：

```
export CORE_PEER_GOSSIP_BOOTSTRAP=<a list of peer endpoints within the peer's org>
export CORE_PEER_GOSSIP_EXTERNALENDPOINT=<the peer endpoint, as known outside the org>
```

9.4 Gossip 讯息

在线对等体通过不断广播“活动”消息来表明其可用性，每个消息都包含**公钥基础结构（PKI）** ID 和消息上发件人的签名。对等方通过收集这些活动消息来维护频道成员身份。如果没有对等方从特定对等方接收到有效消息，则最终从通道成员资格中清除此“死”对等方。由于“活动”消息是经过加密签名的，因此恶意对等端永远无法假冒其他对等端，因为它们缺少根证书颁发机构（CA）授权的签名密钥。

除了自动转发接收到的消息外，状态协调过程还可以在每个通道上的同级之间同步**世界状态**。每个对等方不断从通道上的其他对等方提取数据块，以便在发现差异时修复其自身的状态。由于不需要固定的连接来维护基于 Gossip 的数据分发，因此该过程可靠地为共享分类帐提供了数据一致性和完整性，包括对节点崩溃的容忍度。

因为通道是隔离的，所以一个通道上的对等无法在任何其他通道上发送消息或共享信息。尽管任何对等方都可以属于多个通道，但是分区消息传递通过基于对等方的通道订阅应用消息路由策略，可以阻止将块分发给不在该通道中的对等方。

注意

1.点对点消息的安全性由对等 TLS 层处理，不需要签名。对等方通过其证书（由 CA 分配）进行身份验证。尽管也使用 TLS 证书，但在 Gossip 层中已认证的是对等证书。分类帐块由排序服务签名，然后传递到通道上的领导者对等方。

2.身份验证由对等方的成员资格服务提供者控制。当对等方第一次连接到通道时，TLS 会话将与成员身份绑定。就网络和通道中的成员资格而言，这实质上是向连接的对等方认证每个对等方。

10 经常问的问题

10.1 背书

认可架构：

问题： 网络中有多少对等方需要认可交易？

回答： 认可交易所需的对等体数由链码定义中指定的认可策略决定。

问题： 应用程序客户端是否需要连接到所有对等方？

回答： 客户端只需要连接链码认可策略所需的对等方。

10.2 安全与访问控制

问题：　　如何确保数据隐私？

回答：　　数据隐私涉及多个方面。首先，您可以将网络分成多个通道，其中每个通道代表参与者的一个子集，这些参与者有权查看部署到该通道的链码的数据。

其次，您可以使用私有数据来使分类账数据对渠道中其他组织保持私有。私有数据收集使渠道上已定义的组织子集能够认可，提交或查询私有数据，而不必创建单独的渠道。频道上的其他参与者仅接收数据的哈希。有关更多信息，请参阅在 Fabric 中使用私有数据教程。请注意，关键概念主题还说明了何时使用私有数据而不是通道。

第三，作为 Fabric 使用私有数据对数据进行哈希处理的替代方法，客户端应用程序可以在调用链码之前对数据进行哈希处理或加密。如果您对数据进行哈希处理，则需要提供一种共享源数据的方法。如果加密数据，则需要提供一种共享解密密钥的方法。

第四，通过将访问控制构建到链码逻辑中，可以将数据访问限制为组织中的某些角色。

第五，可以通过对等方上的文件系统加密对静态分类帐数据进行加密，而通过 TLS 对传输中的数据进行加密。

问题：　　定购者是否看到交易数据？

回答：　　不可以，排序者只能排序交易，而不能打开交易。如果您根本不希望数据通过排序者，则可以使用 Fabric 的私有数据功能。或者，您可以在调用链码之前对客户端应用程序中的数据进行哈希处理或加密。如果加密数据，则需要提供一种共享解密密钥的方法。

10.3 应用程序侧编程模型

问题：　　应用程序客户如何知道交易的结果？

回答：　　交易模拟结果由背书人在投标响应中返回给客户。如果有多个背书，客户可以检查所有回复是否相同，并提交结果和背书以进行排序和承诺。最终，提交对等方将验证或使事务无效，并且客户端会通过事件知道 SDK 提供给应用程序客户端的结果。

问题：　　如何查询分类帐数据？

回答：　　在链码中，您可以基于键进行查询。可以按范围查询键，并且可以对组合键进行建模以启用针对多个参数的等效查询。例如，（owner，asset_id）的组合键可用于查询某个实体拥有的所有资产。这些基于密钥的查询可用于针对分类帐的只读查询，以及用于更新分类帐的事务。

如果您在链码中将资产数据建模为 JSON 并使用 CouchDB 作为状态数据库，则还可以使用链码中的 CouchDB JSON 查询语言对链码数据值执行复杂的丰富查询。应用程序客户端可以执行只读查询，但是通常不会将这些响应作为事务的一部分提交给排序服务。

问题：　　如何查询历史数据以了解数据来源？

回答：　　chaincode API `GetHistoryForKey()` 将返回键值的历史记录。

问题：　　如何保证查询结果正确，尤其是当被查询的对等体正在恢复并赶上块处理时？

回答：　　客户端可以查询多个对等方，比较其块体高度，比较其查询结果，并在较高的块体高度处优先使用这些对等体。

10.4 Chaincode（智能合约和数字资产）

问题：　　Hyperledger Fabric 是否支持智能合约逻辑？

回答：　　是。我们称此功能为 Chaincode。这是我们对智能合约方法/算法的解释，具有其他功能。链码是部署在网络上的程序代码，由链验证程序在共识过程中一起执行和验证。开发人员可以使用链码来开发业务合同，资产定义以及集中管理的分散式应用程序。

问题：　　如何创建商业合同？

回答：　　通常有两种开发业务合同的方法：第一种方法是将单个合同编码为独立的链码实例；第二种方法（可能也是更有效的方法）是使用链码创建去中心化的应用程序，这些应用程序管理一种或多种类型的业务合同的生命周期，并让最终用户实例化这些应用程序中的合同实例。

问题：　　如何创建资产？

回答：　　用户可以使用链码（用于业务规则）和成员资格服务（用于数字令牌）来设计资产以及管理资产的逻辑。

在大多数区块链解决方案中，有两种流行的方法来定义资产：无状态 UTXO 模型，其中账户余额被编码为过去的交易记录；以及帐户模型，其中帐户余额保留在分类帐上的状态存储空间中。

每种方法都有其自身的优点和缺点。这项区块链技术并不主张任何一种。相反，我们的首要要求之一是确保可以轻松实施这两种方法。

问题：　　编写链码支持哪些语言？

回答：　　链码可以用任何编程语言编写并在容器中执行。当前，支持 Golang，node.js 和 java chaincode。

问题：　　Hyperledger Fabric 是否有本国货币？

回答：　　不会。但是，如果您的链网确实需要本币，则可以使用 chaincode 开发自己的本币。本机货币的一个共同属性是，每次交易在其链上进行处理时，都会进行一定数量的交易（定义将调用该货币的链码）。

10.5 最新版本中的差异

问题：　　在哪里可以找到版本之间突出的区别是什么？

回答：　　任何后续版本之间的差异与一起提供唱片集。

问题：　　上面未回答的技术问题可以从哪里获得帮助？

回答：　　请使用 StackOverflow。

10.6 排序服务

问题：　　**我已经启动并正在运行排序服务，并且想要切换共识算法。我怎么做？**

回答：　　明确不支持此功能。

问题：　　**排序系统通道是什么？**

回答：　　排序者系统通道（有时称为排序系统通道）是排序者最初引导时使用的通道。它用于编排频道创建。排序系统通道定义了联盟和新通道的初始配置。在渠道创建时，联盟中的组织定义，`/Channel` 小组的价值观和政策以及 `/Channel/Orderer` 小组的价值观和政策都被组合在一起，形成了新的初始渠道定义。

问题：　　**如果我更新我的应用程序频道，是否应该更新我的排序系统频道？**

回答：　　创建应用程序通道后，将独立于任何其他通道（包括排序系统通道）对其进行管理。取决于修改，可能需要更改，也可能不需要更改以移植到其他通道。通常，MSP 更改应在所有通道之间同步，而策略更改则更有可能特定于特定通道。

问题：　　**我可以让一个组织同时扮演排序和应用程序的角色吗？**

回答：　　尽管这是可行的，但强烈建议不要这样做。默认情况下，该 `/Channel/Orderer/BlockValidation` 策略允许排序组织的任何有效证书进行签名。如果组织同时充当排序和应用程序角色，则应更新此策略以将块签名者限制为授权排序的证书子集。

问题：　　**我想为 Fabric 写一个共识实现。我该从哪里开始？**

回答：　　共识插件需要实现共识包中 定义的 `Consenter` 和 `Chain` 接口。已经针对这些接口构建了两个插件：raft 和 kafka。您可以研究它们以为自己的实现提供线索。排序服务代码可在排序程序包下找到。

问题：　　**我想更改排序服务配置，例如批次超时，在启动网络后，我该怎么办？**

回答：　　这属于重新配置网络。请查阅有关 configtxlator 的主题 。

10.6.1 独奏

问题：　　**如何在生产中部署 Solo？**

回答：　　Solo 不用于生产。它不是容错，也永远不会容错。不推荐使用 Raft，可以在以后的版本中将其删除。

10.6.2卡夫卡

问题：　　**如何从排序服务中删除节点？**

回答：　　这是一个两步过程：
将节点的证书添加到相关排序者的 MSP CRL 中，以防止对等/客户端连接到它。
通过使用标准的 Kafka 访问控制措施（例如 TLS CRL 或防火墙），防止节点连接到 Kafka 群集。

问题：　　**我以前从未部署过 Kafka / ZK 集群，我想使用基于 Kafka 的排序服务。我该如何进行？**

回答：　　Hyperledger Fabric 文档假定读者通常具有设置，配置和管理 Kafka 集群的操作专业知识（请参阅警告提示）。如果您坚持在没有此类专业知识的情况下继续进行操作，则在尝试基于 Kafka 的排序服务之前，*至少*应完成《Kafka 快速入门》指南的前 6 个步骤 。您还可以查阅此样本配置文件，以获取有关 Kafka / ZooKeeper 合理默认值的简要说明。

问题：　　**在哪里可以找到使用基于 Kafka 的排序服务的网络的 Docker 组合？**

回答：　　请参阅端到端 CLI 示例。

问题：　　**为什么在基于 Kafka 的排序服务中存在 ZooKeeper 依赖关系？**

回答：　　Kafka 在内部将其用于经纪人之间的协调。

问题：　　**我正在尝试遵循 BYFN 示例并收到“服务不可用”错误，我该怎么办？**

回答：　　检查排序服务的日志。“由于同意者错误而导致拒绝发送请求”日志消息通常表示与 Kafka 群集存在连接问题。确保 Kafka 群集设置正确，并且可以通过排序服务的节点访问。

10.6.3 BFT

问题：　　**BFT 版本的排序服务何时可用？**

回答：　　尚未设置日期。我们正在努力在 1.x 周期内发布一个版本，即它将在 Fabric 中进行次要版本升级。跟踪 FAB-33 进行更新。