Hyperledger Fabric

Developing Applications

开发应用程序

This topic covers how to develop a client application and smart contract to solve a business problem using Hyperledger Fabric. In a real world Commercial Paper scenario, involving multiple organizations, you’ll learn about all the concepts and tasks required to accomplish this goal. We assume that the blockchain network is already available.

本主题介绍如何开发客户端应用程序和智能合约，以使用Hyperledger Fabric解决业务问题。在一个真实的商业票据场景中，涉及多个组织，您将了解实现此目标所需的所有概念和任务。我们假设区块链网络已经可用。

The topic is designed for multiple audiences:

该主题面向多个受众：

Solution and application architect

Client application developer

Smart contract developer

Business professional

解决方案和应用程序架构师

客户端应用程序开发人员

智能合约开发者

商务专业人员

You can chose to read the topic in order, or you can select individual sections as appropriate. Individual topic sections are marked according to reader relevance, so whether you’re looking for business or technical information it’ll be clear when a topic is for you.

您可以选择按顺序阅读主题，也可以根据需要选择单独的部分。每个主题部分都根据读者的相关性进行标记，因此无论您是在寻找业务信息还是技术信息，当主题适合您时，都会很清楚。

The topic follows a typical software development lifecycle. It starts with business requirements, and then covers all the major technical activities required to develop an application and smart contract to meet these requirements.

本主题遵循典型的软件开发生命周期。它从业务需求开始，然后涵盖开发应用程序和智能合约以满足这些需求所需的所有主要技术活动。

If you’d prefer, you can try out the commercial paper scenario immediately, following an abbreviated explanation, by running the commercial paper tutorial. You can return to this topic when you need fuller explanations of the concepts introduced in the tutorial.

如果您愿意，可以通过运行商业票据教程，按照简短的解释立即尝试商业票据场景。当需要对本教程中介绍的概念进行更全面的解释时，可以返回本主题。

一、The scenario

一、情景

Audience: Architects, Application and smart contract developers, Business professionals

受众：架构师、应用程序和智能合约开发人员、业务专业人员

In this topic, we’re going to describe a business scenario involving six organizations who use PaperNet, a commercial paper network built on Hyperledger Fabric, to issue, buy and redeem commercial paper. We’re going to use the scenario to outline requirements for the development of commercial paper applications and smart contracts used by the participant organizations.

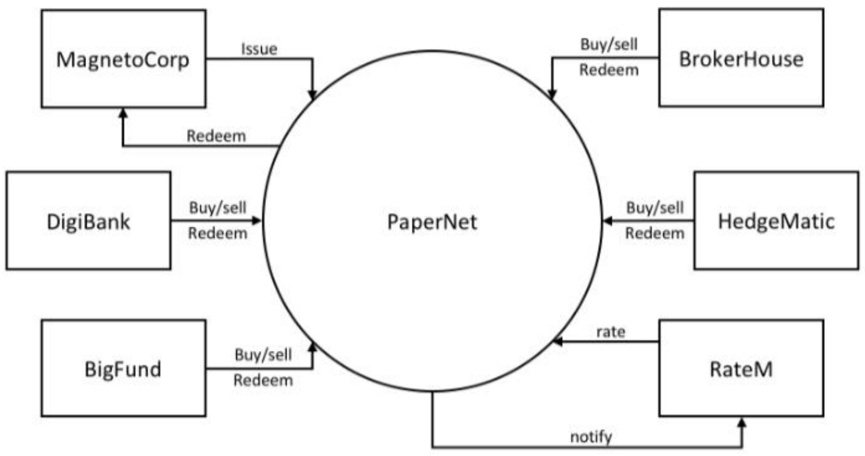
在本主题中，我们将描述一个业务场景，其中有六个组织使用 PaperNet，一个建立在 Hyperledger 结构上的商业票据网络，来发行、购买和兑换商业票据。我们将使用场景来概述参与者组织所使用的商业票据应用程序和智能合约开发的需求。

1、PaperNet network

1、PaperNet 网络

PaperNet is a commercial paper network that allows suitably authorized participants to issue, trade, redeem and rate commercial paper.

Papernet 是一个商业票据网络，允许适当授权的参与者发行、交易、兑换和评级商业票据。



The PaperNet commercial paper network. Six organizations currently use PaperNet network to issue, buy, sell, redeem and rate commercial paper. MagentoCorp issues and redeems commercial paper. DigiBank, BigFund, BrokerHouse and HedgeMatic all trade commercial paper with each other. RateM provides various measures of risk for commercial paper.

Papernet 商业票据网络。目前有六家机构使用 Papernet 网络发行、购买、销售、兑换和赎回商业票据。MagentCorp 发行和赎回商业票据。digibank、bigfund、brokerhouse 和 hedgematic 都是相互交易的商业票据。Ratem 为商业票据提供各种风险度量。

Let’s see how MagnetoCorp uses PaperNet and commercial paper to help its business.

让我们看看 MagnetoCorp 是如何使用 PaperNet 和商业票据来帮助其业务的。

2、Introducing the actors

2、介绍角色

MagnetoCorp is a well-respected company that makes self-driving electric vehicles. In early April 2020, MagnetoCorp won a large order to manufacture 10,000 Model D cars for Daintree, a new entrant in the personal transport market. Although the order represents a significant win for MagnetoCorp, Daintree will not have to pay for the vehicles until they start to be delivered on November 1, six months after the deal was formally agreed between MagnetoCorp and Daintree.

MagnetoCorp 是一家生产自动驾驶电动汽车的著名公司。2020 年 4 月初，Magnetorp 赢得了一大笔订单，为个人运输市场的新进入者 Daintree 生产 10000 辆 D 型汽车。尽管该订单对 MagnetoCorp 来说是一个重大的胜利，但在 Daintree 和 MagnetoCorp 正式达成协议 6 个月后的 11 月 1 日开始交付之前，Daintree 不必支付车辆费用。

To manufacture the vehicles, MagnetoCorp will need to hire 1000 workers for at least 6 months. This puts a short term strain on its finances – it will require an extra 5M USD each month to pay these new employees. Commercial paper is designed to help MagnetoCorp overcome its short term financing needs – to meet payroll every month based on the expectation that it will be cash rich when Daintree starts to pay for its new Model D cars.

为了生产汽车，MagnetoCorp 需要雇佣 1000 名工人至少 6 个月。这给公司的财务带来了短期压力——每月需要额外 500 万美元来支付这些新员工。商业票据旨在帮助 Magnetorp 克服其短期融资需求——每月支付工资，这是基于当 Daintree 开始支付其新的 D 型汽车时，它将现金充裕的预期。

At the end of May, MagnetoCorp needs 5M USD to meet payroll for the extra workers it hired on May 1. To do this, it issues a commercial paper with a face value of 5M USD with a maturity date 6 months in the future – when it expects to see cash flow from Daintree. DigiBank thinks that MagnetoCorp is creditworthy, and therefore doesn’t require much of a premium above the central bank base rate of 2%, which would value 4.95M USD today at 5M USD in 6 months time. It therefore purchases the MagnetoCorp 6 month commercial paper for 4.94M USD – a slight discount compared to the 4.95M USD it is worth. DigiBank fully expects that it will be able to redeem 5M USD from MagnetoCorp in 6 months time, making it a profit of 10K USD for bearing the increased risk associated with this commercial paper. This extra 10K means it receives a 2.4% return on investment – significantly better than the risk free return of 2%.

5 月底，MagnetoCorp 需要 500 万美元来支付 5 月 1 日雇佣的额外工人的工资。为此，该公司发行面值为 500 万美元、到期日为 6 个月的商业票据，预计届时将看到来自戴姆特里的现金流。Digibank 认为 Magnetorp 值得信赖，因此不需要比中央银行基准利率 2%高出很多溢价，后者在 6 个月内将以 500 万美元的价格在今天价值 495 万美元。因此，它以 494 万美元的价格购买了 MagnetoCorp6 个月期的商业票据——与价值 495 万美元相比，略有折扣。Digibank 完全预计，它将能够在 6 个月内从 Magnetorp 赎回 500 万美元，使其在承担与本商业票据相关的风险增加时获得 1 万美元的利润。额外的 10 万意味着它获得了 2.4%的投资

回报，远远好于 2%的无风险回报。

At the end of June, when MagnetoCorp issues a new commercial paper for 5M USD to meet June’s payroll, it is purchased by BigFund for 4.94M USD. That’s because the commercial conditions are roughly the same in June as they are in May, resulting in BigFund valuing MagnetoCorp commercial paper at the same price that DigiBank did in May.

6 月底，MagnetoCorp 以 500 万美元的价格发行了一份新的商业票据，以支付 6 月的工资，BigFund 以 494 万美元的价格收购了磁电机公司。这是因为 6 月份的商业条件与 5 月份大致相同，导致 BigFund 以与 Digibank 5 月份相同的价格对 Magnetorp 商业票据进行估值。

Each subsequent month, MagnetoCorp can issue new commercial paper to meet its payroll obligations, and these may be purchased by DigiBank, or any other participant in the PaperNet commercial paper network - BigFund, HedgeMatic or BrokerHouse. These organizations may pay more or less for the commercial paper depending on two factors - the central bank base rate, and the risk associated with MagnetoCorp. This latter figure depends on a variety of factors such as the production of Model D cars, and the creditworthiness of MagnetoCorp as assessed by RateM, a ratings agency.

随后的每个月，Magnetorp 都可以发行新的商业票据来履行其工资义务，Digibank 或 Papernet 商业票据网络的任何其他参与者（BigFund、Hedgematic 或经纪公司）可以购买这些票据。根据中央银行基准利率和 MagnetoCorp 相关风险这两个因素，这些机构可能会或多或少地支付商业票据的费用。后一个数字取决于多种因素，如 D 型汽车的生产，以及评级机构 Ratem 评估的磁电机公司的信誉。

Let’s pause the MagnetoCorp story for a moment, and develop the client applications and smart contracts that PaperNet uses to issue, buy, sell and redeem commercial paper. We’ll come back to the role of the rating agency, RateM, a little later.

让我们暂停一下 Magnetorp 的故事，开发客户应用程序和智能合约，Papernet 使用它们发行、购买、销售和赎回商业票据。稍后我们将回到评级机构 Ratem 的角色。

二、Analysis

二、分析

Audience: Architects, Application and smart contract developers, Business professionals

受众：架构师、应用程序和智能合约开发人员、业务专业人员

Let’s analyze commercial paper in a little more detail. PaperNet participants such as MagnetoCorp and DigiBank use commercial paper transactions to achieve their business objectives - let’s examine the structure of a commercial paper and the transactions that affect it over time. Later we’ll focus on how money flows between buyers and sellers; for now, let’s focus on the first paper issued by MagnetoCorp.

让我们更详细地分析一下商业票据。像 Magnetorp 和 Digibank 这样的 Papernet 参与者使用商业票据交易来实现他们的业务目标——让我们来检查商业票据的结构以及随着时间推移影响它的交易。稍后，我们将重点讨论资金如何在买卖双方之间流动；现在，让我们

重点讨论 MagnetoCorp 发行的第一张纸。

1、Commercial paper lifecycle

1、商业票据生命周期

A paper 00001 is issued by MagnetoCorp on May 31. Spend a few moments looking at the first state of this paper, with its different properties and values:

MagnetoCorp 于 5 月 31 日发行了一份 00001 号文件。花点时间看一下本文的第一个状态，它具有不同的属性和值：

```
Issuer = MagnetoCorp
Paper = 00001
Owner = MagnetoCorp
Issue date = 31 May 2020
Maturity = 30 November 2020
Face value = 5M USD
Current state = issued
```

This paper state is a result of the issue transaction and it brings MagnetoCorp’s first commercial paper into existence! Notice how this paper has a 5M USD face value for redemption later in the year. See how the Issuer and Owner are the same when paper 00001 is issued. Notice that this paper could be uniquely identified as MagnetoCorp00001 - a composition of the Issuer and Paper properties. Finally, see how the property Current state = issued quickly identifies the stage of MagnetoCorp paper 00001 in its lifecycle.

本文论述的是发行交易的结果，它使 MagnetoCorp 的第一张商业票据得以存在！请注意，今年晚些时候，该票据的面值为 500 万美元。请参阅发行人和所有人在发行 00001 纸时是如何保持一致的。请注意，本文可以唯一地标识为 Magnetorp0001——发行人和纸张属性的组成部分。最后，查看属性 current state=issued 如何快速识别 Magnetorp Paper 00001 在其生命周期中的阶段。

Shortly after issuance, the paper is bought by DigiBank. Spend a few moments looking at how the same commercial paper has changed as a result of this buy transaction:

发行后不久，Digibank 收购了这票据。花点时间看看同一种商业票据是如何因此次买入交易而发生变化的：

```
Issuer = MagnetoCorp
Paper = 00001
Owner = DigiBank
Issue date = 31 May 2020
Maturity date = 30 November 2020
Face value = 5M USD
Current state = trading
```

The most significant change is that of Owner - see how the paper initially owned by MagnetoCorp is now owned by DigiBank. We could imagine how the paper might be subsequently sold to BrokerHouse or HedgeMatic, and the corresponding change to Owner. Note how Current state allow us to easily identify that the paper is now trading.

最重要的变化是所有者的变化——看看 MagnetoCorp 最初拥有的纸张现在是如何被 Digibank 拥有的。我们可以想象这张纸后来如何出售给经纪公司或对冲公司，以及对所有者的相应变化。请注意，当前状态如何使我们能够轻松识别该票据目前正在交易。

After 6 months, if DigiBank still holds the the commercial paper, it can redeem it with MagnetoCorp:

6 个月后，如果 Digibank 仍持有商业票据，则可以向 Magnetorp 赎回：

```
Issuer = MagnetoCorp
Paper = 00001
Owner = MagnetoCorp
Issue date = 31 May 2020
Maturity date = 30 November 2020
Face value = 5M USD
Current state = redeemed
```

This final redeem transaction has ended the commercial paper’s lifecycle - it can be considered closed. It is often mandatory to keep a record of redeemed commercial papers, and the redeemed state allows us to quickly identify these.

这一最终赎回交易结束了商业票据的生命周期——可以认为它已经结束。通常必须保留已赎回商业票据的记录，而且赎回状态允许我们快速识别这些票据。

2、Transactions

2、交易

We’ve seen that paper 00001’s lifecycle is relatively straightforward - it moves between issued, trading and redeemed as a result of an issue, buy, or redeem transaction.

我们已经看到 Paper 00001 的生命周期相对简单——它在发行、购买或赎回交易的结果下在发行、交易和赎回之间移动。

These three transactions are initiated by MagnetoCorp and DigiBank (twice), and drive the state changes of paper 00001. Let’s have a look at the transactions that affect this paper in a little more detail:

这三个交易是由 MagnetoCorp 和 Digibank 发起的（两次），并推动了论文 00001 的状态变化。让我们更详细地了解一下影响本文的事务：

Issue

发行

Examine the first transaction initiated by MagnetoCorp:

检查 MagnetoCorp 发起的第一笔交易：

```
Txn = issue
Issuer = MagnetoCorp
Paper = 00001
Issue time = 31 May 2020 09:00:00 EST
Maturity date = 30 November 2020
Face value = 5M USD
```

See how the issue transaction has a structure with properties and values.

This transaction structure is different to, but closely matches, the structure of paper 00001. That’ s because they are different things - paper 00001 reflects a state of PaperNet that is a result of the issue transaction. It’ s the logic behind the issue transaction (which we cannot see) that takes these properties and creates this paper. Because the transaction creates the paper, it means there’ s a very close relationship between these structures.

查看问题事务如何具有具有属性和值的结构。这种交易结构不同于，但与票据 00001 的结构紧密匹配。这是因为它们是不同的东西——票据 00001 反映了票据交易的纸质网络状态。正是问题事务背后的逻辑（我们看不到）接受了这些属性并创建了本文。因为事务创建了纸张，这意味着这些结构之间有非常密切的关系。

Buy

买

Next, examine the buy transaction which transfers ownership of paper 00001 from MagnetoCorp to DigiBank:

接下来，检查将纸 00001 所有权从 Magnetorp 转移到 Digibank 的买入交易：

```
Txn = buy
Issuer = MagnetoCorp
Paper = 00001
Current owner = MagnetoCorp
New owner = DigiBank
Purchase time = 31 May 2020 10:00:00 EST
Price = 4.94M USD
```

See how the buy transaction has fewer properties that end up in this paper. That’ s because this transaction only modifies this paper. It’ s only New owner = DigiBank that changes as a result of this transaction; everything else is the same. That’ s OK - the most important thing about the buy transaction is the change of ownership, and indeed in this transaction, there’ s an acknowledgement of the current owner of the paper, MagnetoCorp.

请看一下在本文中购买交易如何具有更少的属性。这是因为这个事务只修改了本文。只有新的 owner=digibank 会因为这个事务而改变；其他的都是一样的。没关系——关于买入交易，最重要的是所有权的变更，事实上，在这宗交易中，有一个对目前的所有者 MagnetoCorp 的确认。

You might ask why the Purchase time and Price properties are not captured in paper 00001? This comes back to the difference between the transaction and the paper. The 4.94 M USD price tag is actually a property of the transaction, rather than a property of this paper. Spend a little time thinking about this difference; it is not as obvious as it seems. We’ re going to see later that the ledger will record both pieces of information - the history of all transactions that affect this paper, as well its latest state. Being clear on this separation of information is really important.

你可能会问为什么购买时间和价格属性没有被记录在文件 00001 中？这又回到了交易和票据之间的区别。494 万美元的价格标签实际上是交易的财产，而不是本文的财产。花点时间考虑一下这种差异；它不像看上去那么明显。稍后我们将看到，分类账将记录这两条信息——影响本文的所有交易的历史记录以及最新状态。明确信息的分离是非常重要的。

It’ s also worth remembering that paper 00001 may be bought and sold many times. Although we’ re skipping ahead a little in our scenario, let’ s examine what transactions we might see if paper 00001 changes ownership.

值得记住的是，纸 00001 可能会被买卖多次。尽管在我们的场景中我们跳过了一点，但是让我们来研究一下如果 paper 00001 更改了所有权，我们可能会看到哪些事务。

If we have a purchase by BigFund:

如果 BigFund 购买：

```
Txn = buy
Issuer = MagnetoCorp
Paper = 00001
Current owner = DigiBank
New owner = BigFund
Purchase time = 2 June 2020 12:20:00 EST
Price = 4.93M USD
```

Followed by a subsequent purchase by HedgeMatic

随后由 Hedgematic 进行后续购买

```
Txn = buy
Issuer = MagnetoCorp
Paper = 00001
Current owner = BigFund
New owner = HedgeMatic
Purchase time = 3 June 2020 15:59:00 EST
Price = 4.90M USD
```

See how the paper owners changes, and how in out example, the price changes. Can you think of a reason why the price of MagnetoCorp commercial paper might be falling?

看看纸张所有者是如何变化的，以及进出示例中的价格是如何变化的。你能想到 MagnetoCorp 商业票据价格下跌的原因吗？

Redeem

赎回

The redeem transaction for paper 00001 represents the end of its lifecycle. In our relatively simple example, DigiBank initiates the transaction which transfers the commercial paper back to MagnetoCorp:

票据 00001 的兑换交易代表其生命周期的结束。在我们相对简单的例子中，Digibank 发起了将商业票据转移回 Magnetorp 的交易：

```
Txn = redeem
Issuer = MagnetoCorp
Paper = 00001
Current owner = HedgeMatic
Redeem time = 30 Nov 2020 12:00:00 EST
```

Again, notice how the redeem transaction has very few properties; all of the changes to paper 00001 can be calculated data by the redeem transaction logic: the Issuer will become the new owner, and the Current state will change to redeemed. The Current owner property is specified in our example, so that it can be checked against the current holder of the paper.

同样，请注意兑换交易的属性非常少；对票据 00001 的所有更改都可以通过兑换交易逻辑计算数据：发行人将成为新的所有者，当前状态将更改为兑换。在我们的示例中指定了当前所有者属性，这样就可以对照当前的纸张持有人来检查它。

3、The Ledger

3、分类账

In this topic, we’ve seen how transactions and the resultant paper states are the two most important concepts in PaperNet. Indeed, we’ll see these two fundamental elements in any Hyperledger Fabric distributed ledger – a world state, that contains the current value of all objects, and a blockchain that records the history of all transactions that resulted in the current world state.

在本主题中，我们已经看到了事务和生成的纸面状态如何是 PaperNet 中最重要的两个概念。事实上，我们将在任何超级账本结构分布式账本中看到这两个基本要素——一个包含所有对象当前价值的世界状态，以及一个记录导致当前世界状态的所有交易历史的区块链。

You’re now in a great place translate these ideas into a smart contract. Don’t worry if your programming is a little rusty, we’ll provide tips and pointers to understand the program code. Mastering the commercial paper smart contract is the first big step towards designing your own application. Or, if you’re a business analyst who’s comfortable with a little programming, don’t be afraid to keep dig a little deeper!

你现在处于一个很好的位置，把这些想法转化成一个聪明的合同。别担心，如果您的程序有点生疏，我们将提供一些技巧和指针来理解程序代码。掌握商业票据智能合约是设计自己的应用程序的第一大步。或者，如果你是一个对一点编程感到满意的业务分析师，不要害怕继续深入挖掘！

三、Process and Data Design

三、过程和数据设计

Audience: Architects, Application and smart contract developers, Business professionals

受众：架构师、应用程序和智能合约开发人员、业务专业人员

This topic shows you how to design the commercial paper processes and their related data structures in PaperNet. Our analysis highlighted that modelling PaperNet using states and transactions provided a precise way to understand what’s happening. We’re now going to elaborate on these two strongly related concepts to help us subsequently design the smart contracts and applications of PaperNet.

本主题向您展示如何在 PaperNet 中设计商业票据流程及其相关数据结构。我们的分析强调，使用状态和事务对 PaperNet 进行建模提供了一种准确的方法来了解正在发生的事情。现在，我们将详细介绍这两个密切相关的概念，以帮助我们随后设计智能合约和 Papernet 应用程序。

1、Lifecycle

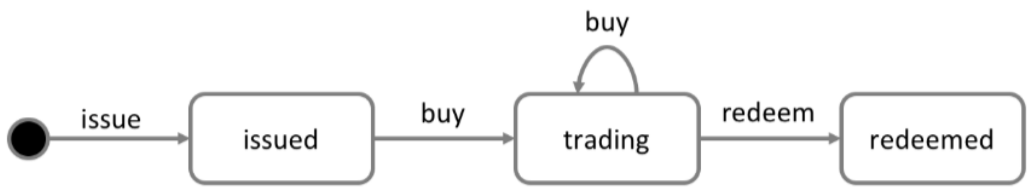
1、生命周期

As we’ve seen, there are two important concepts that concern us when dealing with commercial paper; states and transactions. Indeed, this is true for all blockchain use cases; there are conceptual objects of value, modelled as states, whose lifecycle transitions are described by transactions. An effective analysis of states and transactions is an essential starting point for a successful implementation.

正如我们所看到的，在处理商业票据时，有两个重要的概念与我们有关：状态和交易。事实上，这对所有区块链用例都是正确的；有价值的概念对象，建模为状态，其生命周期转换由事务描述。对状态和事务的有效分析是成功实施的重要起点。

We can represent the life cycle of a commercial paper using a state transition diagram:

我们可以使用状态转换图来表示商业票据的生命周期：



The state transition diagram for commercial paper. Commercial papers transition between issued, trading and redeemed states by means of the issue, buy and redeem transactions.
商业票据的状态转换图。商业票据通过发行、购买和赎回交易在已发行、交易和赎回国家之间过渡。

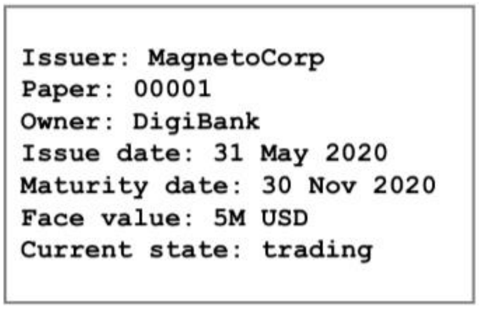
See how the state diagram describes how commercial papers change over time, and how specific transactions govern the life cycle transitions. In Hypledger Fabric, smart contracts implement transaction logic that transition commercial papers between their different states. Commercial paper states are actually held in the ledger world state; so let’s take a closer look at them.

请参阅状态图如何描述商业文件随时间的变化，以及特定事务如何控制生命周期转换。在 Hypleger 结构中，智能合约实现了在不同状态之间转换商业票据的交易逻辑。商业票据状态实际上是在分类帐世界状态中保存的，所以让我们仔细看看它们。

2、Ledger state

2、Ledger 状态

Recall the structure of a commercial paper:
回顾商业票据的结构：



```
Issuer: MagnetoCorp
Paper: 00001
Owner: DigiBank
Issue date: 31 May 2020
Maturity date: 30 Nov 2020
Face value: 5M USD
Current state: trading
```

A commercial paper can be represented as a set of properties, each with a value. Typically, some combination of these properties will provide a unique key for each paper.

商业票据可以表示为一组属性，每个属性都有一个值。通常，这些属性的某些组合将为每个纸张提供一个唯一的键。

See how a commercial paper Paper property has value 00001, and the Face value property has value 5M USD. Most importantly, the Current state property indicates whether the commercial paper is issued,trading or redeemed. In combination, the full set of properties make up the state of a commercial paper. Moreover, the entire collection of these individual commercial paper states constitutes the ledger world state.

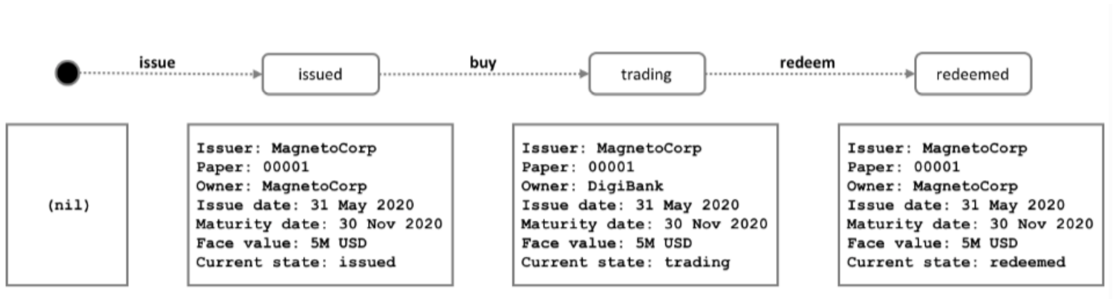
看看商业票据房地产如何价值 00001，面值房地产如何价值 500 万美元。最重要的是，当前的国有资产表明商业票据是发行、交易还是赎回。综合起来，全套的属性构成了商业票据的状态。此外，这些单独的商业票据状态的全部集合构成了分类账世界状态。

All ledger state share this form; each has a set of properties, each with a different value. This multi-property aspect of states is a powerful feature - it allows us to think of a Fabric state as a vector rather than a simple scalar. We then represent facts about whole objects as individual states, which subsequently undergo transitions controlled by transaction logic. A Fabric state is implemented as a key/value pair, in which the value encodes the object properties in a format that captures the object’s multiple properties, typically JSON. The ledger database can support advanced query operations against these properties, which is very helpful for sophisticated object retrieval.

所有分类帐状态都共享此表单；每个都有一组属性，每个属性具有不同的值。状态的多属性方面是一个强大的特性——它允许我们将结构状态看作一个向量，而不是简单的标量。然后，我们将整个对象的事实表示为单个状态，这些状态随后经历由事务逻辑控制的转换。结构状态被实现为一个键/值对，其中值以捕获对象多个属性（通常是 JSON）的格式对对象属性进行编码。分类帐数据库可以支持对这些属性的高级查询操作，这对于复杂的对象检索非常有帮助。

See how MagnetoCorp’s paper 00001 is represented as a state vector that transitions according to different transaction stimuli:

了解磁电机公司的论文 00001 如何表示为一个状态向量，该状态向量根据不同的交易刺激进行转换：



A commercial paper state is brought into existence and transitions as a result of different transactions. Hyperledger Fabric states have multiple properties, making them vectors rather than scalars.

商业票据状态是由于不同的交易而产生和转变的。Hyperledger 结构状态具有多个属性，使它们成为向量而不是标量。

Notice how each individual paper starts with the empty state, which is technically a nil state for the paper, as it doesn’t exist! See how paper 00001 is brought into existence by the issue transaction, and how it is subsequently updated as a result of the buy and redeem transactions.

请注意，每个单独的纸张是如何以空状态开始的，这在技术上是纸张的零状态，因为它不存在！请参阅发行交易如何产生“00001 号文件”，以及购买和赎回交易之后如何更新该文件。

Notice how each state is self-describing; each property has a name and a value. Although all our commercial papers currently have the same properties, this need not be the case for all time, as Hyperledger Fabric supports different states having different properties. This allows the same ledger world state to contain different forms of the same asset as well as different types of asset. It also makes it possible to update a state’s structure; imagine a new regulation that requires an additional data field. Flexible state properties support the fundamental requirement of data evolution over time.

注意每个状态是如何自我描述的；每个属性都有一个名称和一个值。尽管我们所有的商业文件目前都具有相同的属性，但这种情况并不总是如此，因为 Hyperledger 结构支持具有不同属性的不同状态。这允许同一个分类帐世界状态包含同一资产的不同形式以及不同类型的资产。它还使更新一个州的结构成为可能；设想一个新的法规需要一个额外的数据字段。灵活的状态属性支持数据随时间演化的基本要求。

3、State keys

3、状态键

In most practical applications, a state will have a combination of properties that uniquely identify it in a given context – it’s key. The key for a PaperNet commercial paper is formed by a concatenation of the Issuer and paper properties; so for MagnetoCorp’s first paper, it’s MagnetoCorp00001.

在大多数实际应用中，状态将具有在给定上下文中唯一标识它的属性组合——它是关键。PaperNet 商业票据的关键在于发行人和票据属性的结合；因此，对于 Magnetorp 的第一张票据，它是 Magnetorp0001。

A state key allows us to uniquely identify a paper; it is created as a result of the issue transaction and subsequently updated by buy and redeem. Hyperledger Fabric requires each state in a ledger to have a unique key.

状态键允许我们唯一地标识一张纸；它是由发行交易创建的，随后由买入和赎回更新。Hyperledger 结构要求分类帐中的每个状态都有一个唯一的键。

When a unique key is not available from the available set of properties, an application-determined unique key is specified as an input to the transaction that creates the state. This unique key is usually with some form of UUID, which although less readable, is a standard practice. What’s important is that every individual state object in a ledger must have a unique key.

当可用属性集中的唯一键不可用时，应用程序确定的唯一键将指定为创建状态的事务的输入。这个唯一的键通常带有某种形式的 UUID，虽然它不太可读，但却是一种标准的实践。重要的是，分类帐中的每个状态对象都必须具有唯一的键。

4、Multiple states

4、多状态

As we’ve seen, commercial papers in PaperNet are stored as state vectors in a ledger. It’s a reasonable requirement to be able to query different commercial papers from the ledger; for example: find all the papers issued by MagnetoCorp, or: find all the papers issued by MagnetoCorp in the redeemed state.

正如我们所看到的，纸质网络中的商业票据作为状态向量存储在分类账中。从分类帐中查询不同的商业票据是合理的要求，例如：查找磁电机公司发行的所有票据，或：查找磁电机公司在赎回状态下发行的所有票据。

To make these kinds of search tasks possible, it’s helpful to group all related papers together in a logical list. The PaperNet design incorporates the idea of a commercial paper list - a logical container which is updated whenever commercial papers are issued or otherwise changed.

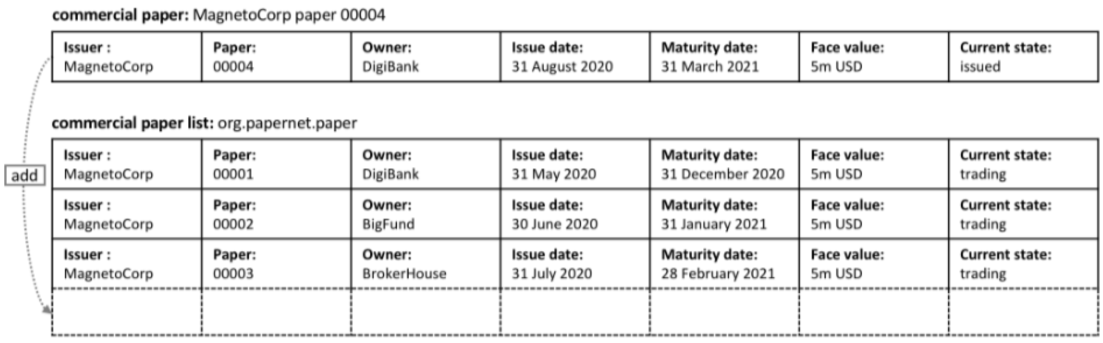
为了使这类搜索任务成为可能，将所有相关的论文分组在一个逻辑列表中是很有帮助的。PaperNet 的设计融合了商业票据清单的理念——一个逻辑容器，每当商业票据发行或以其他方式更改时，都会对其进行更新。

Logical representation

逻辑表示法

It’s helpful to think of all PaperNet commercial papers being in a single list of commercial papers:

考虑到所有纸质商业票据都在单一的商业票据列表中，这很有帮助：



commercial paper: MagnetoCorp paper 00004

Issuer : MagnetoCorp	**Paper:** 00004	**Owner:** DigiBank	**Issue date:** 31 August 2020	**Maturity date:** 31 March 2021	**Face value:** 5m USD	**Current state:** issued

add

commercial paper list: org.papernet.paper

Issuer : MagnetoCorp	**Paper:** 00001	**Owner:** DigiBank	**Issue date:** 31 May 2020	**Maturity date:** 31 December 2020	**Face value:** 5m USD	**Current state:** trading
Issuer : MagnetoCorp	**Paper:** 00002	**Owner:** BigFund	**Issue date:** 30 June 2020	**Maturity date:** 31 January 2021	**Face value:** 5m USD	**Current state:** trading
Issuer : MagnetoCorp	**Paper:** 00003	**Owner:** BrokerHouse	**Issue date:** 31 July 2020	**Maturity date:** 28 February 2021	**Face value:** 5m USD	**Current state:** trading

MagnetoCorp’s newly created commercial paper 00004 is added to the list of existing commercial papers.

磁电机公司新创建的商业票据 00004 被添加到现有商业票据的列表中。

New papers can be added to the list as a result of an issue transaction, and papers already in the list can be updated with buy or redeem transactions. See how the list has a descriptive name: org.papernet.papers; it’s a really good idea to use this kind of DNS name because well-chosen names will make your blockchain designs intuitive to other people. This idea applies equally well to smart contract names.

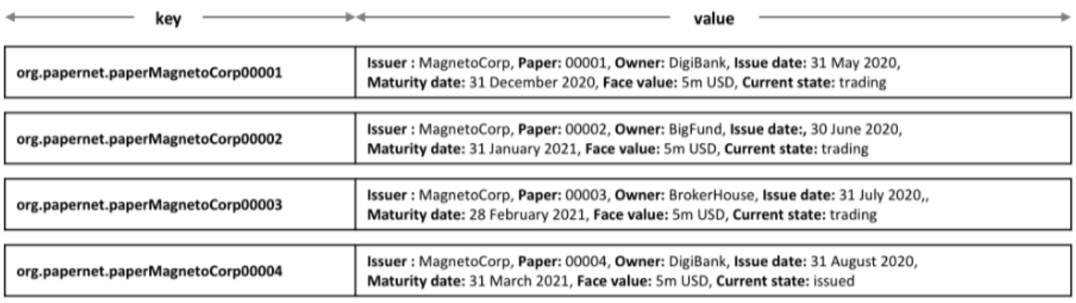
新文件可以作为发行交易的结果添加到列表中，并且列表中已经存在的文件可以通过购买或赎回交易进行更新。看看这个列表如何有一个描述性的名称：org.papernet.papers；使用这种 DNS 名称是一个非常好的主意，因为精心选择的名称将使您的区块链设计对其他人直观。这个想法同样适用于智能合约名称。

Physical representation

物理表示法

While it’s correct to think of a single list of papers in PaperNet – org.papernet.papers – lists are best implemented as a set of individual Fabric states, whose composite key associates the state with its list. In this way, each state’s composite key is both unique and supports effective list query.

虽然在 papernet 中考虑一个单独的文件列表（org.papernet.papers）是正确的，但是列表最好是作为一组单独的结构状态来实现，这些状态的复合键将状态与其列表相关联。这样，每个状态的组合键都是唯一的，并且支持有效的列表查询。



key	value
org.papernet.paperMagnetoCorp00001	**Issuer :** MagnetoCorp, **Paper:** 00001, **Owner:** DigiBank, **Issue date:** 31 May 2020, **Maturity date:** 31 December 2020, **Face value:** 5m USD, **Current state:** trading
org.papernet.paperMagnetoCorp00002	**Issuer :** MagnetoCorp, **Paper:** 00002, **Owner:** BigFund, **Issue date:**, 30 June 2020, **Maturity date:** 31 January 2021, **Face value:** 5m USD, **Current state:** trading
org.papernet.paperMagnetoCorp00003	**Issuer :** MagnetoCorp, **Paper:** 00003, **Owner:** BrokerHouse, **Issue date:** 31 July 2020,, **Maturity date:** 28 February 2021, **Face value:** 5m USD, **Current state:** trading
org.papernet.paperMagnetoCorp00004	**Issuer :** MagnetoCorp, **Paper:** 00004, **Owner:** DigiBank, **Issue date:** 31 August 2020, **Maturity date:** 31 March 2021, **Face value:** 5m USD, **Current state:** issued

Representing a list of PaperNet commercial papers as a set of distinct Hyperledger Fabric states

将 PaperNet 商业票据列表表示为一组不同的 Hyperledger 结构状态

Notice how each paper in the list is represented by a vector state, with a unique composite key formed by the concatenation of org.papernet.paper, Issuer and Paper properties. This structure is helpful for two reasons:

请注意，列表中的每一张纸都是如何由向量状态表示的，其中包含由 org.papernet.paper、issuer 和 paper 属性串联而成的唯一复合键。这种结构有两个原因：

It allows us to examine any state vector in the ledger to determine which list it’s in, without reference to a separate list. It’s analogous to looking at set of sports fans, and identifying which team they support by the colour of the shirt they are wearing. The sports fans self-declare their allegiance; we don’t need a list of fans.

它允许我们检查分类账中的任何状态向量，以确定它在哪个列表中，而不参考单独的列表。这类似于看一组体育迷，根据他们所穿衬衫的颜色来确定他们支持哪支球队。体育迷们自我宣布他们的忠诚；我们不需要一张球迷名单。

Hyperlegder Fabric internally uses a concurrency control mechanism to update a ledger, such that keeping papers in separate state vectors vastly reduces the opportunity for shared-state collisions. Such collisions require transaction re-submission, complicate application design, and decrease performance.

Hyperlegder 结构在内部使用并发控制机制来更新分类账，这样将文件保存在单独的状态向量中就大大减少了共享状态冲突的机会。这种冲突需要重新提交事务，使应用程序设计复杂化，并降低性能。

This second point is actually a key take-away for Hyperledger Fabric; the physical design of state vectors is very important to optimum performance and behaviour. Keep your states separate!

第二点实际上是 Hyperledger 结构的一个关键点；状态向量的物理设计对于优化性能和行为非常重要。把你们的状态分开！

5、Trust relationships

5、信任关系

We have discussed how the different roles in a network, such as issuer, trader or rating agencies as well as different business interests determine who needs to sign off on a transaction. In Fabric, these rules are captured by so-called endorsement policies. The rules can be set on a chaincode granularity, as well as for individual state keys.

我们已经讨论了网络中的不同角色，如发行人、交易员或评级机构，以及不同的商业利益如何决定谁需要签署交易。在结构中，这些规则由所谓的认可策略捕获。可以在链码粒度以及单个状态键上设置规则。

This means that in PaperNet, we can set one rule for the whole namespace that determines which organizations can issue new papers. Later, rules can be set and updated for individual papers to capture the trust relationships of buy and redeem transactions.

这意味着在 PaperNet 中，我们可以为整个名称空间设置一个规则，确定哪些组织可以发布新的文件。稍后，可以为个别票据设置和更新规则，以捕获买入和赎回交易的信任关系。

In the next topic, we will show you how to combine these design concepts to implement the PaperNet commercial paper smart contract, and then an application in exploits it!

在下一个主题中，我们将向您展示如何结合这些设计概念来实现 PaperNet 商业票据智能合约，然后是如何利用它的应用程序！

四、Smart Contract Processing

四、智能合约处理

Audience: Architects, Application and smart contract developers

受众：架构师、应用程序和智能合约开发人员

At the heart of a blockchain network is a smart contract. In PaperNet, the code in the commercial paper smart contract defines the valid states for commercial paper, and the transaction logic that transition a paper from one state to another. In this topic, we’re going to show you how to implement a real world smart contract that governs the process of issuing, buying and redeeming commercial paper.

区块链网络的核心是智能合约。在 PaperNet 中，商业票据智能合约中的代码定义了商业票据的有效状态，以及将票据从一种状态转换为另一种状态的交易逻辑。在本主题中，我们将向您展示如何实施现实世界的智能合约，该合约控制商业票据的发行、购买和赎回过程。

We’re going to cover:

我们将介绍：

What is a smart contract and why it’s important

什么是智能合约，为什么它很重要

How to define a smart contract

如何定义智能合约

How to define a transaction

如何定义事务

How to implement a transaction

如何实现事务

How to represent a business object in a smart contract

如何在智能合约中表示业务对象

How to store and retrieve an object in the ledger

如何在分类帐中存储和检索对象

If you’d like, you can download the sample and even run it locally. It is written in JavaScript, but the logic is quite language independent, so you’ll be easily able to see what’s going on! (The sample will become available for Java and GOLANG as well.)

如果您愿意，可以下载样本，甚至在本地运行它。它是用 JavaScript 编写的，但是逻辑是完全独立于语言的，所以您可以很容易地看到正在发生的事情！（示例也将适用于 Java 和 GOLANG。）

1、Smart Contract

1、智能合约

A smart contract defines the different states of a business object and governs the processes that move the object between these different states. Smart contracts are important because they allow architects and smart contract developers to define the key business processes and data that are shared across the different organizations collaborating in a blockchain network.

智能合约定义业务对象的不同状态，并控制在这些不同状态之间移动对象的流程。智能合约很重要，因为它们允许架构师和智能合约开发者定义在区块链网络中协作的不同组织之间共享的关键业务流程和数据。

In the PaperNet network, the smart contract is shared by the different network participants, such as MagnetoCorp and DigiBank. The same version of the smart contract must be used by all applications connected to the network so that they jointly implement the same shared business processes and data.

在 Papernet 网络中，智能合约由不同的网络参与者共享，如 Magnetorp 和 Digibank。所有连接到网络的应用程序必须使用相同版本的智能合约，以便它们共同实现相同的共享业务流程和数据。

2、Contract class

2、合约类

A copy of the PaperNet commercial paper smart contract is contained in papercontract.js. View it with your browser, or open it in your favourite editor if you’ve downloaded it.

PaperNet 商业票据智能合约的副本包含在 PaperContract.js 中。用你的浏览器查看它，或者在你最喜欢的编辑器中打开它（如果你已经下载了）。

You may notice from the file path that this is MagnetoCorp’s copy of the smart contract. MagnetoCorp and DigiBank must agree the version of the smart contract that they are going to use. For now, it doesn’t matter which organization’s copy you look at, they are all the same.

您可能会从文件路径中注意到这是磁电机公司的智能合约副本。Magnetorp 和 Digibank 必须同意他们将要使用的智能合约版本。现在，不管你看哪个组织的副本，它们都是一样的。

Spend a few moments looking at the overall structure of the smart contract; notice that it’s quite short! Towards the top of papercontract.js, you’ll see that there’s a definition for the commercial paper smart contract:

花点时间看一下智能合约的整体结构；注意它很短！在 paper contract.js 的顶部，您将看到商业票据智能合约的定义：

```
class CommercialPaperContract extends Contract {...}
```

The CommercialPaperContract class contains the transaction definitions for commercial paper - issue, buy and redeem. It’s these transactions that bring commercial papers into existence and move them through their lifecycle. We’ll examine these transactions soon, but for now notice how CommercialPaperContract extends the Hyperledger Fabric Contract class. This built-in class, and the Context class, were brought into scope earlier:

商业票据合同类包含商业票据的交易定义-发行、购买和赎回。正是这些交易使商业票据得以存在并在它们的生命周期中移动。我们很快就会检查这些交易，但现在注意到商业票

据合同如何扩展了超级账本结构合同类。这个内置类和上下文类在前面被引入了作用域：

```
const { Contract, Context } = require('fabric-contract-api');
```

Our commercial paper contract will use built-in features of these classes, such as automatic method invocation, a per-transaction context, transaction handlers, and class-shared state.

我们的商业票据契约将使用这些类的内置特性，例如自动方法调用、每个事务上下文、事务处理程序和类共享状态。

Notice also how the class constructor uses its superclass to initialize itself with an explicit contract name:

还请注意类构造函数如何使用其超类以显式协定名称初始化自身：

```
constructor() {
super('org.papernet.commercialpaper');
}
```

Most importantly, org.papernet.commercialpaper is very descriptive - this smart contract is the agreed definition of commercial paper for all PaperNet organizations.

最重要的是，org.papernet.commercial paper 非常具有描述性——此智能合同是所有 Papernet 组织对商业票据的约定定义。

Usually there will only be one smart contract per file - contracts tend to have different lifecycles, which makes it sensible to separate them. However, in some cases, multiple smart contracts might provide syntactic help for applications, e.g. EuroBond, DollarBond, YenBond, but essentially provide the same function. In such cases, smart contracts and transactions can be disambiguated.

通常每个文件只有一个智能合约——合约往往有不同的生命周期，这使得将它们分开是明智的。但是，在某些情况下，多个智能合约可能为应用程序提供语法帮助，例如 Eurobond、Dollarbond、Yenbond，但本质上提供相同的功能。在这种情况下，智能合约和交易可以消除歧义。

3、Transaction definition

3、交易定义

Within the class, locate the issue method.

在类中，找到 Issue 方法。

```
async issue(ctx, issuer, paperNumber, issueDateTime, maturityDateTime,
faceValue) {...}
```

This function is given control whenever this contract is called to issue a commercial paper. Recall how commercial paper 00001 was created with the following transaction:

每当调用此合同来发行商业票据时，都会对该函数进行控制。回顾商业票据 00001 是如何通过以下交易创建的：

```
Txn = issue
Issuer = MagnetoCorp
Paper = 00001
```

```
Issue time = 31 May 2020 09:00:00 EST
Maturity date = 30 November 2020
Face value = 5M USD
```

We’ve changed the variable names for programming style, but see how these properties map almost directly to the issue method variables.

我们已经更改了编程样式的变量名,但是看看这些属性如何几乎直接映射到问题方法变量。

The issue method is automatically given control by the contract whenever an application makes a request to issue a commercial paper. The transaction property values are made available to the method via the corresponding variables. See how an application submits a transaction using the Hyperledger Fabric SDK in the application topic, using a sample application program.

当申请提出发行商业票据的请求时，合同自动赋予发行方法控制权。事务属性值通过相应的变量提供给方法。在应用程序主题中，使用示例应用程序，查看应用程序如何使用 Hyperledger Fabric SDK 提交事务。

You might have noticed an extra variable in the issue definition - ctx. It’s called the transaction context, and it’s always first. By default, it maintains both per-contract and per-transaction information relevant to transaction logic. For example, it would contain MagnetoCorp’s specified transaction identifier, a MagnetoCorp issuing user’s digital certificate, as well as access to the ledger API.

您可能注意到问题定义 CTX 中有一个额外的变量。它被称为事务上下文，并且总是第一个。默认情况下，它维护与事务逻辑相关的每个合同和每个事务信息。例如，它将包含磁电机公司指定的事务标识符、磁电机公司颁发用户数字证书以及对分类账 API 的访问。

See how the smart contract extends the default transaction context by implementing its own createContext() method rather than accepting the default implementation:

请参阅智能合约如何通过实现自己的 createContext（）方法而不是接受默认实现来扩展默认事务上下文：

```
createContext() {
return new CommercialPaperContext()
}
```

This extended context adds a custom property paperList to the defaults:

此扩展上下文将自定义属性纸质列表添加到默认值：

```
class CommercialPaperContext extends Context {
constructor() {
super();
// All papers are held in a list of papers
this.paperList = new PaperList(this);
}
```

We’ll soon see how ctx.paperList can be subsequently used to help store and retrieve all PaperNet commercial papers.

我们将很快看到 ctx.paperlist 如何随后用于帮助存储和检索所有 Papernet 商业文件。

To solidify your understanding of the structure of a smart contract transaction, locate the buy and redeem transaction definitions, and see if you can see how they map to their corresponding commercial paper transactions.

为了巩固您对智能合约交易结构的理解，请找到买入和赎回交易定义，并查看它们如何映射到相应的商业票据交易。

The buy transaction:

买入交易：

```
async buy(ctx, issuer, paperNumber, currentOwner, newOwner, price, purchaseTime) {...}
Txn = buy
Issuer = MagnetoCorp
Paper = 00001
Current owner = MagnetoCorp
New owner = DigiBank
Purchase time = 31 May 2020 10:00:00 EST
Price = 4.94M USD
The redeem transaction:
async redeem(ctx, issuer, paperNumber, redeemingOwner, redeemDateTime) {...}
Txn = redeem
Issuer = MagnetoCorp
Paper = 00001
Redeemer = DigiBank
Redeem time = 31 Dec 2020 12:00:00 EST
```

In both cases, observe the 1:1 correspondence between the commercial paper transaction and the smart contract method definition. And don’t worry about the async and await keywords - they allow asynchronous JavaScript functions to be treated like their synchronous cousins in other programming languages.

在这两种情况下，观察商业票据交易和智能合约方法定义之间的 1:1 对应关系。不用担心异步和等待关键字——它们允许异步 JavaScript 函数在其他编程语言中被视为它们的同步表亲。

4、Transaction logic

4、交易逻辑

Now that you’ve seen how contracts are structured and transactions are defined, let’s focus on the logic within the smart contract.

既然您已经了解了合同的结构和交易的定义，那么让我们关注智能合同中的逻辑。

Recall the first issue transaction:

召回第一期交易：

```
Txn = issue
Issuer = MagnetoCorp
Paper = 00001
Issue time = 31 May 2020 09:00:00 EST
Maturity date = 30 November 2020
Face value = 5M USD
```

It results in the issue method being passed control:

它会导致问题方法通过控制：

```
async issue(ctx, issuer, paperNumber, issueDateTime, maturityDateTime, faceValue) {
// create an instance of the paper
let paper = CommercialPaper.createInstance(issuer, paperNumber, issueDateTime, maturityDateTime, faceValue);
// Smart contract, rather than paper, moves paper into ISSUED state
paper.setIssued();
// Newly issued paper is owned by the issuer
paper.setOwner(issuer);
// Add the paper to the list of all similar commercial papers in the ledger world state
await ctx.paperList.addPaper(paper);
// Must return a serialized paper to caller of smart contract
return paper.toBuffer();
}
```

The logic is simple: take the transaction input variables, create a new commercial paper paper, add it to the list of all commercial papers using paperList, and return the new commercial paper (serialized as a buffer) as the transaction response.

逻辑很简单：取交易输入变量，创建一张新的商业票据纸，使用纸质列表将其添加到所有商业票据的列表中，并返回新的商业票据（序列化为缓冲区）作为交易响应。

See how paperList is retrieved from the transaction context to provide access to the list of commercial papers. issue(), buy() and redeem() continually re-access ctx.paperList to keep the list of commercial papers up-to-date.

请参阅如何从事务上下文中检索纸质清单，以提供对商业票据列表的访问。issue（)、buy（）和 reduce（）不断地重新访问 ctx.paperlist，以保持商业票据的列表最新。

The logic for the buy transaction is a little more elaborate:

买入交易的逻辑更为详细：

```
async buy(ctx, issuer, paperNumber, currentOwner, newOwner, price, purchaseDateTime) {
// Retrieve the current paper using key fields provided
let paperKey = CommercialPaper.makeKey([issuer, paperNumber]);
let paper = await ctx.paperList.getPaper(paperKey);
// Validate current owner
if (paper.getOwner() !== currentOwner) {
throw new Error('Paper ' + issuer + paperNumber + ' is not owned by ' + currentOwner);
}
// First buy moves state from ISSUED to TRADING
if (paper.isIssued()) {
paper.setTrading();
}
// Check paper is not already REDEEMED
if (paper.isTrading()) {
    paper.setOwner(newOwner);
    } else {
    throw new Error('Paper ' + issuer + paperNumber + ' is not trading. Current
state = ' +paper.getCurrentState());
    }
    // Update the paper
    await ctx.paperList.updatePaper(paper);
    return paper.toBuffer();
    }
```

See how the transaction checks currentOwner and that paper is TRADING before changing the owner with paper.setOwner(newOwner). The basic flow is simple though – check some pre-conditions, set the new owner, update the commercial paper on the ledger, and return the updated commercial paper (serialized as a buffer) as the transaction response.

在使用 paper.setowner(newowner)更改所有者之前，请参阅事务如何检查当前所有者和该纸张的交易情况。不过，基本流程很简单——检查一些前提条件，设置新的所有者，更新分类账上的商业票据，并返回更新后的商业票据（序列化为缓冲区）作为事务响应。

Why don’t you see if you can understand the logic for the redeem transaction?

你为什么不看看你是否能理解兑换交易的逻辑?

5、Representing an object

5、表示一个对象

We’ve seen how to define and implement the issue, buy and redeem transactions using the CommercialPaper and PaperList classes. Let’s end this topic by seeing how these classes work.

我们已经了解了如何使用商业票据和纸质清单类定义和实现问题、购买和兑现交易。让我们看看这些类是如何工作的，以此结束这个主题。

Locate the CommercialPaper class in the paper.js file:

在 paper.js 文件中找到商业论文类：

```
    class CommercialPaper extends State {...}
```

This class contains the in-memory representation of a commercial paper state. See how the createInstance method initializes a new commercial paper with the provided parameters:

此类包含商业票据状态的内存表示形式。请参阅 CreateInstance 方法如何使用提供的参数初始化新的商业票据：

```
    static createInstance(issuer, paperNumber, issueDateTime, maturityDateTime,
faceValue) {
    return new CommercialPaper({ issuer, paperNumber, issueDateTime,
maturityDateTime, faceValue });
    }
```

Recall how this class was used by the issue transaction:

回想一下 Issue 事务如何使用此类：

```
    let paper = CommercialPaper.createInstance(issuer, paperNumber,
issueDateTime, maturityDateTime, faceValue);
```

See how every time the issue transaction is called, a new in-memory instance of a commercial paper is created containing the transaction data.

了解每次调用问题事务时，如何创建包含事务数据的新的商业票据内存实例。

A few important points to note:

需要注意的几个要点：

This is an in-memory representation; we' ll see later how it appears on the ledger.

这是内存中的表示；稍后我们将看到它在分类帐上的显示方式。

The CommercialPaper class extends the State class. State is an application-defined class which creates a common abstraction for a state. All states have a business object class which they represent, a composite key, can be serialized and de-serialized, and so on. State helps our code be more legible when we are storing more than one business object type on the ledger. Examine the State class in the state.js file.

商业票据类扩展了国家级。状态是一个应用程序定义的类，它为状态创建一个公共抽象。所有状态都有一个它们表示的业务对象类，一个组合键，可以序列化和反序列化，等等。当我们在分类帐上存储多个业务对象类型时，状态可以帮助我们的代码更清晰。检查 state.js 文件中的 state 类。

A paper computes its own key when it is created - this key will be used when the ledger is accessed. The key is formed from a combination of issuer and paperNumber.

纸张在创建时会计算自己的密钥-访问分类帐时将使用此密钥。密钥由发行者和纸号组合而成。

```
constructor(obj) {
super(CommercialPaper.getClass(), [obj.issuer, obj.paperNumber]);
Object.assign(this, obj);
}
```

A paper is moved to the ISSUED state by the transaction, not by the paper class. That' s because it' s the smart contract that governs the lifecycle state of the paper. For example, an import transaction might create a new set of papers immediately in the TRADING state.

通过事务而不是通过 Paper 类将纸张移动到“已发行”状态。这是因为它是一个智能合约，控制着论文的生命周期状态。例如，进口交易可能会在交易状态下立即创建一组新的文件。

The rest of the CommercialPaper class contains simple helper methods:

商业论文类的其余部分包含简单的助手方法：

```
getOwner() {
return this.owner;
}
```

Recall how methods like this were used by the smart contract to move the commercial paper through its lifecycle. For example, in the redeem transaction we saw:

回想一下这样的方法是如何被智能合约用来在商业票据的生命周期中移动的。例如，在

兑换交易中，我们看到：

```
if (paper.getOwner() === redeemingOwner) {
paper.setOwner(paper.getIssuer());
paper.setRedeemed();
}
```

6、Access the ledger

6、访问分类

Now locate the PaperList class in the paperlist.js file:

现在 paperlist 定位在 paperlist.js 类文件：

class PaperList extends StateList {

This utility class is used to manage all PaperNet commercial papers in Hyperledger Fabric state database. The PaperList data structures are described in more detail in the architecture topic.

本实用 papernet 舱是用来管理所有的商业文件 hyperledger 结构状态数据库。数据结构中的 paperlist 被描述更详细的架构主题。

Like the CommercialPaper class, this class extends an application-defined StateList class which creates a common abstraction for a list of states - in this case, all the commercial papers in PaperNet.

像这类 commercialpaper 类，扩展类的应用无线电 statelist 创建公共抽象列表——在这个案例中，papernet 所有商业文件。

The addPaper() method is a simple veneer over the StateList.addState() method:

在 addpaper（）方法是一个简单的积木在 statelist.addstate（）方法：

```
async addPaper(paper) {
return this.addState(paper);
}
```

You can see in the StateList.js file how the StateList class uses the Fabric API putState() to write the commercial paper as state data in the ledger:

你可以看到在 statelist.js 文件如何使用 API 的 statelist 类织物 putstate（）写为“商业票据状态数据的分类：

```
async addState(state) {
let key = this.ctx.stub.createCompositeKey(this.name, state.getSplitKey());
let data = State.serialize(state);
await this.ctx.stub.putState(key, data);
}
```

Every piece of state data in a ledger requires these two fundamental elements:

每一块的状态数据在 A 帐需要论文二：基本元素

Key: key is formed with createCompositeKey() using a fixed name and the key of state. The name was assigned when the PaperList object was constructed, and state.getSplitKey() determines each state’s unique key.

关键：关键是形成一个 createcompositekey（）使用固定密钥的名称和状态。这个名字是什么 paperlist 指定当对象（M）和 state.getsplitkey determines 每个州的唯一键。

Data: data is simply the serialized form of the commercial paper state, created using the State.serialize() utility method. The State class serializes and deserializes data using JSON, and the State’ s business object class as required, in our case CommercialPaper, again set when the PaperList object was constructed.

数据：数据是简单的 serialized 商业票据形式的状态，使用 state.serialize（）创建的实用方法。反序列化的类和 serializes 州立使用 JSON 数据，和州的业务对象所需的类。在我们的案例，commercialpaper 一集，当 paperlist 什么构造对象。

Notice how a StateList doesn’ t store anything about an individual state or the total list of states - it delegates all of that to the Fabric state database. This is an important design pattern - it reduces the opportunity for ledger MVCC collisions in Hyperledger Fabric.

知识 statelist 不通知一件事或一个人的州立商店的总清单 - 它是代表美国所有州的结构数据库。这是一个重要的设计模式，它减少了机会，mvcc collisions hyperledger 莱杰在织物。

The StateList getState() and updateState() methods work in similar ways:

在 statelist getstate（）和（）updatestate 方法工作在类似的方式：

```
async getState(key) {
let        ledgerKey        =        this.ctx.stub.createCompositeKey(this.name, State.splitKey(key));
let data = await this.ctx.stub.getState(ledgerKey);
let state = State.deserialize(data, this.supportedClasses);
return state;
}
async updateState(state) {
let key = this.ctx.stub.createCompositeKey(this.name, state.getSplitKey());
let data = State.serialize(state);
await this.ctx.stub.putState(key, data);
}
```

See how they use the Fabric APIs putState(), getState() and createCompositeKey() to access the ledger. We’ ll expand this smart contract later to list all commercial papers in paperNet - what might the method look like to implement this ledger retrieval?

如何使用 API 湖织物 getstate putstate（），（）和（）来访问 createcompositekey 总帐。我们扩展这个智能合同之后的所有商业文件列表的方法是：在 papernet - 实施分类检索这样子的？

That’ s it! In this topic you’ ve understood how to implement the smart contract for PaperNet. You can move to the next sub topic to see how an application calls the smart contract using the Fabric SDK.

很好！你已经在本主题的理解如何实现智能 papernet 合同研究。你可以移动到下一个可用的子主题的应用调用 SDK 使用智能织物的合同。

五、Application

五、应用

Audience: Architects, Application and smart contract developers

受众：架构师、应用程序和智能合约开发人员

An application can interact with a blockhain network by submitting transactions to a ledger or querying ledger content. This topic covers the mechanics of how an application does this; in our scenario, organizations access PaperNet using applications which invoke issue, sell and redeem transactions defined in a commercial paper smart contract. Even though MagnetoCorp’s application to issue a commercial paper is basic, it covers all the major points of understanding.

应用程序可以通过向分类帐提交交易或查询分类帐内容与区块链网络交互。本主题介绍应用程序如何做到这一点的机制；在我们的场景中，组织使用调用商业票据智能合约中定义的发行、销售和兑现交易的应用程序访问 PaperNet。尽管磁电机公司发行商业票据的申请是基本的，但它涵盖了所有主要的理解点。

In this topic, we’re going to cover:

在本主题中，我们将介绍：

The application flow to invoke a smart contract

调用智能合约的应用程序流

How an application uses a wallet and identity

应用程序如何使用钱包和标识

How an application connects using a gateway

应用程序如何使用网关连接

How to access a particular network

如何访问特定网络

How to construct a transaction request

如何构造交易请求

How to submit a transaction

如何提交交易

How to process a transaction response

如何处理事务响应

To help your understanding, we’ll make reference to the commercial paper sample application provided with Hyperledger Fabric. You can download it and run it locally. It is written in JavaScript, but the logic is quite language independent, so you’ll be easily able to see what’s going on! (The sample will become available for Java and GOLANG as well.)

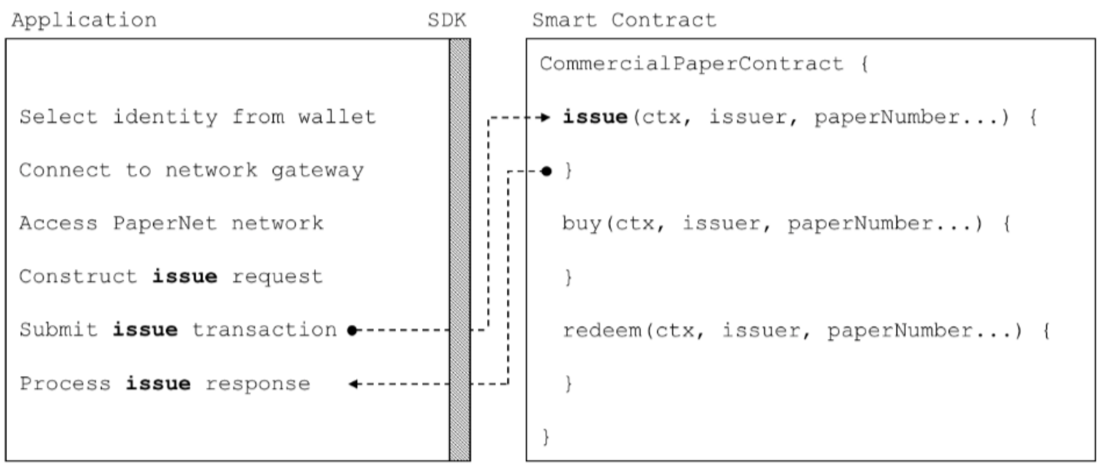
为了帮助您理解，我们将参考随 Hyperledger Fabric 提供的商业纸张示例应用程序。您可以下载并在本地运行它。它是用 JavaScript 编写的，但是逻辑是完全独立于语言的，所以您可以很容易地看到正在发生的事情！（示例也将适用于 Java 和 GOLANG。）

1、Basic Flow

1、基本流程

An application interacts with a blockchain network using the Fabric SDK. Here’s a simplified diagram of how an application invokes a commercial paper smart contract:

应用程序使用 Fabric SDK 与区块链网络交互。以下是应用程序如何调用商业票据智能合约的简化图：



Application
SDK
Smart Contract
Select identity from wallet
Connect to network gateway
Access PaperNet network
Construct issue request
Submit issue transaction
Process issue response
CommercialPaperContract {
issue(ctx, issuer, paperNumber...) {
}
buy(ctx, issuer, paperNumber...) {
}
redeem(ctx, issuer, paperNumber...) {
}
}

A PaperNet application invokes the commercial paper smart contract to submit an issue transaction request.

PaperNet 应用程序调用商业票据智能合约来提交问题事务请求。

An application has to follow six basic steps to submit a transaction:

申请必须遵循六个基本步骤才能提交交易：

Select an identity from a wallet

从钱包中选择身份

Connect to a gateway

连接到网关

Access the desired network

访问所需网络

Construct a transaction request for a smart contract

为智能合约构造事务请求

Submit the transaction to the network

将事务提交到网络

Process the response

处理响应

You’ re going to see how a typical application performs these six steps using the Fabric SDK. You’ ll find the application code in the issue.js file. View it in your browser, or open it in your favourite editor if you’ ve downloaded it. Spend a few moments looking at the overall structure of the application; even with comments and spacing, it’ s only 100 lines of code!

您将看到一个典型的应用程序如何使用 FabricSDK 执行这六个步骤。您可以在 issue.js 文件中找到应用程序代码。在您的浏览器中查看它，或者在您最喜欢的编辑器中打开它（如果已下载）。花点时间看看应用程序的整体结构；即使有注释和间距，也只有 100 行代码！

2、Wallet

2、钱包

Towards the top of issue.js, you’ ll see two Fabric classes are brought into scope:

在 issue.js 的顶部，您将看到两个结构类被纳入范围：

```
const { FileSystemWallet, Gateway } = require('fabric-network');
```

You can read about the fabric-network classes in the node SDK documentation, but for now, let's see how they are used to connect MagnetoCorp's application to PaperNet. The application uses the Fabric Wallet class as follows:

您可以阅读 node sdk 文档中的 fabric network 类，但现在，让我们看看如何使用它们将 magnercorp 的应用程序连接到 papernet。应用程序使用 Fabric Wallet 类，如下所示：

```
const wallet = new FileSystemWallet('../identity/user/isabella/wallet');
```

See how wallet locates a wallet in the local filesystem. The identity retrieved from the wallet is clearly for a user called Isabella, who is using the issue application. The wallet holds a set of identities - X.509 digital certificates - which can be used to access PaperNet or any other Fabric network. If you run the tutorial, and look in this directory, you'll see the identity credentials for Isabella.

查看 wallet 如何在本地文件系统中定位钱包。从钱包中检索到的身份对于使用问题应用程序的 Isabella 用户来说是显而易见的。钱包里有一套身份证——X.509 数字证书——可用于访问纸质网络或任何其他织物网络。如果您运行教程并查看这个目录，您将看到伊莎贝拉的身份凭证。

Think of a wallet holding the digital equivalents of your government ID, driving license or ATM card. The X.509 digital certificates within it will associate the holder with a organization, thereby entitling them to rights in a network channel. For example, Isabella might be an administrator in MagnetoCorp, and this could give her more privileges than a different user - Balaji from DigiBank. Moreover, a smart contract can retrieve this identity during smart contract processing using the transaction context.

想想一个钱包，里面装着与你的政府身份证、驾驶执照或 ATM 卡相当的数字。其中的 X.509 数字证书将使持有人与组织关联，从而使他们有权在网络通道中获得权利。例如，Isabella 可能是磁电机公司的管理员，这可能会给她比 Digibank 的其他用户 Balaji 更多的特权。此外，智能合约可以在使用事务上下文进行智能合约处理期间检索此标识。

Note also that wallets don't hold any form of cash or tokens - they hold identities.

还请注意，钱包不包含任何形式的现金或代币——它们拥有身份。

3、Gateway

3、网关

The second key class is a Fabric Gateway. Most importantly, a gateway identifies one or more peers that provide access to a network - in our case, PaperNet. See how issue.js connects to its gateway:

第二个密钥类是结构网关。最重要的是，网关标识了一个或多个提供网络访问的对等点——在我们的例子中，是 Papernet。查看 issue.js 如何连接到其网关：

```
await gateway.connect(connectionProfile, connectionOptions);
```

gateway.connect() has two important parameters:

gateway.connect（）有两个重要参数：

connectionProfile: the file system location of a connection profile that identifies a set of peers as a gateway to PaperNet

连接配置文件：连接配置文件的文件系统位置，它将一组对等机标识为到 PaperNet 的网关。

connectionOptions: a set of options used to control how issue.js interacts with PaperNet

ConnectionOptions：一组用于控制 issue.js 如何与 PaperNet 交互的选项。

See how the client application uses a gateway to insulate itself from the network topology, which might change. The gateway takes care of sending the transaction proposal to the right peer nodes in the network using the connection profile and connection options.

了解客户机应用程序如何使用网关将自身与网络拓扑隔离，而网络拓扑可能会发生变化。网关负责使用连接配置文件和连接选项将事务建议发送到网络中的右对等节点。

Spend a few moments examining the connection profile ./gateway/connectionProfile.yaml. It uses YAML, making it easy to read.

花点时间检查连接配置文件。/gateway/connection profile.yaml。它使用山药，方便阅读。

It was loaded and converted into a JSON object:

它被加载并转换为 JSON 对象：

```
let connectionProfile = yaml.safeLoad(file.readFileSync('./gateway/connectionProfile.yaml', 'utf8'));
```

Right now, we're only interested in the channels: and peers: sections of the profile: (We've modified the details slightly to better explain what's happening.)

现在，我们只对频道和同行感兴趣：配置文件的部分：(我们稍微修改了一些细节，以便更好地解释正在发生的事情。)

```
channels:
  papernet:
    peers:
      peer1.magnetocorp.com:
        endorsingPeer: true
        eventSource: true

      peer2.digibank.com:
        endorsingPeer: true
        eventSource: true

peers:
  peer1.magnetocorp.com:
    url: grpcs://localhost:7051
    grpcOptions:
      ssl-target-name-override: peer1.magnetocorp.com
      request-timeout: 120
    tlsCACerts:
      path: certificates/magnetocorp/magnetocorp.com-cert.pem

  peer2.digibank.com:
    url: grpcs://localhost:8051
    grpcOptions:
      ssl-target-name-override: peer1.digibank.com
    tlsCACerts:
      path: certificates/digibank/digibank.com-cert.pem
```

See how channel: identifies the PaperNet: network channel, and two of its peers. MagnetoCorp has peer1.magenetocorp.com and DigiBank has peer2.digibank.com, and both have the role of endorsing peers. Link to these peers via the peers: key, which contains details about how to connect to them, including their respective network addresses.

了解 channel：如何识别 papernet:network channel 及其两个对等机。Magnetorp 拥有 peer1.magenetocorp.com 和 digibank 拥有 peer2.digibank.com，两者都具有支持同行的作用。通过 peers:key 链接到这些对等机，其中包含有关如何连接到它们的详细信息，包括它们各自的网络地址。

The connection profile contains a lot of information – not just peers – but network channels, network orderers, organizations, and CAs, so don’t worry if you don’t understand all of it!

连接配置文件包含大量信息，不仅包括对等方，还包括网络渠道、网络订购方、组织和 CA，因此，如果您不完全理解这些信息，请不要担心！

Let’s now turn our attention to the connectionOptions object:

现在让我们将注意力转向 ConnectionOptions 对象：

```
let connectionOptions = {
identity: userName,
wallet: wallet
}
```

See how it specifies that identity, userName, and wallet, wallet, should be used to connect to a gateway. These were assigned values earlier in the code.

查看它如何指定身份、用户名和钱包、钱包应用于连接到网关。这些是在代码前面分配的值。

There are other connection options which an application could use to instruct the SDK to act intelligently on its behalf. For example:

还有其他连接选项，应用程序可以使用这些选项来指示 SDK 为其智能地执行操作。例如：

```
let connectionOptions = {
identity: userName,
wallet: wallet,
eventHandlerOptions: {
commitTimeout: 100,
strategy: EventStrategies.MSPID_SCOPE_ANYFORTX
},
}
```

Here, commitTimeout tells the SDK to wait 100 seconds to hear whether a transaction has been committed. And strategy: EventStrategies.MSPID_SCOPE_ANYFORTX specifies that the SDK can notify an application after a single MagnetoCorp peer has confirmed the transaction, in contrast to strategy: EventStrategies.NETWORK_SCOPE_ALLFORTX which requires that all peers from MagnetoCorp and DigiBank to confirm the transaction.

在这里，commitTimeout 告诉 sdk 等待 100 秒以了解是否提交了事务。和 strategy:eventstrigies.mspid_scope_anyfortx 指定，与 strategy:eventstrigies.network_scope_allfortx 不同，SDK 可以在单个磁电机公司对等机确认交易后通知应用程序，而该策略要求磁电机公司和 digibank 的所有对等机确认交易。

If you’d like to, read more about how connection options allow applications to specify goal-oriented behaviour without having to worry about how it is achieved.

如果您愿意，请阅读更多关于连接选项如何允许应用程序指定面向目标的行为，而不必担心如何实现。

4、Network channel

4、网络通道

The peers defined in the gateway connectionProfile.yaml provide issue.js with access to PaperNet. Because these peers can be joined to multiple network channels, the gateway actually provides the application with access to multiple network channels!

gateway connectionprofile.yaml 中定义的对等方提供了对 papernet 的访问权。因为这些对等点可以连接到多个网络通道，所以网关实际上为应用程序提供了访问多个网络通道的权限！

See how the application selects a particular channel:

查看应用程序如何选择特定通道：

```
const network = await gateway.getNetwork('PaperNet');
```

From this point onwards, network will provide access to PaperNet. Moreover, if the application wanted to access another network, BondNet, at the same time, it is easy:

从这一点开始，网络将提供对纸质网络的访问。此外，如果应用程序希望同时访问另一个网络 BondNet，则很容易：

```
const network2 = await gateway.getNetwork('BondNet');
```

Now our application has access to a second network, BondNet, simultaneously with PaperNet!

现在，我们的应用程序可以访问第二个网络，邦德网，同时访问 Papernet！

We can see here a powerful feature of Hyperledger Fabric - applications can participate in a network of networks, by connecting to multiple gateway peers, each of which is joined to multiple network channels. Applications will have different rights in different channels according to their wallet identity provided in gateway.connect().

这里我们可以看到 Hyperledger 结构的一个强大功能——应用程序可以通过连接到多个网关对等端（每个网关对等端都连接到多个网络通道）来参与一个网络网络网络。根据 gateway.connect（）中提供的钱包标识，应用程序在不同的通道中具有不同的权限。

5、Construct request

5、构造请求

The application is now ready to issue a commercial paper. To do this, it’s going to use CommercialPaperContract and again, its fairly straightforward to access this smart contract:

申请书现在可以发行商业票据了。要做到这一点，它将使用商业纸质合同，而且，访问此智能合同非常简单：

```
const contract = await network.getContract('papercontract', 'org.papernet.commercialpaper');
```

Note how the application provides a name - papercontract - and an explicit contract name: org.papernet.commercialpaper! We see how a contract name picks out one contract from the papercontract.js chaincode file that contains many contracts. In PaperNet, papercontract.js was installed and instantiated with the name papercontract, and if you’re interested, read how to install and instantiate a chaincode containing multiple smart contracts.

请注意，应用程序如何提供名称（纸质合同）和明确的合同名称：org.papernet.commandpaper！我们可以看到合同名如何从包含许多合同的 papercontract.js chaincode 文件中选择一个合同。在 PaperNet 中，PaperContract.js 是以 PaperContract 的名称安装和实例化的，如果您感兴趣，请阅读如何安装和实例化包含多个智能合约的链码。

If our application simultaneously required access to another contract in PaperNet or BondNet this would be easy:

如果我们的应用程序同时要求访问 PaperNet 或 BondNet 中的另一个合同，这将很容易：

```
const euroContract = await network.getContract('EuroCommercialPaperContract');
const bondContract = await network2.getContract('BondContract');
```

In these examples, note how we didn’t use a qualifying contract name - we have only one smart contract per file, and getContract() will use the first contract it finds.

在这些示例中，请注意我们没有使用限定的合同名称——每个文件只有一个智能合同，getContract（）将使用它找到的第一个合同。

Recall the transaction MagnetoCorp uses to issue its first commercial paper:

回想一下磁电机公司发行其第一份商业票据所用的交易：

```
Txn = issue
Issuer = MagnetoCorp
Paper = 00001
Issue time = 31 May 2020 09:00:00 EST
Maturity date = 30 November 2020
Face value = 5M USD
```

Let’ s now submit this transaction to PaperNet!

现在我们把这笔交易提交给 Papernet!

6、Submit transaction

6、提交交易记录

Submitting a transaction is a single method call to the SDK:

提交事务是对 sdk 的单个方法调用:

```
const issueResponse = await contract.submitTransaction('issue', 'MagnetoCorp', '00001', '2020-05-31', '2020-11-30', '5000000');
```

See how the submitTransaction() parameters match those of the transaction request. It’ s these values that will be passed to the issue() method in the smart contract, and used to create a new commercial paper. Recall its signature:

请参阅 submitTransaction () 参数如何与事务请求的参数匹配。正是这些值将传递给智能合约中的 issue () 方法，并用于创建新的商业票据。收回其签名:

```
async issue(ctx, issuer, paperNumber, issueDateTime, maturityDateTime, faceValue) {...}
```

It might appear that a smart contract receives control shortly after the application issues submitTransaction(), but that’ s not the case. Under the covers, the SDK uses the connectionOptions and connectionProfile details to send the transaction proposal to the right peers in the network, where it can get the required endorsements. But the application doesn’ t need to worry about any of this - it just issues submitTransaction and the SDK takes care of it all!

看起来智能合约在应用程序发出 submitTransaction () 后不久就会收到控制权，但事实并非如此。在封面下，SDK 使用 ConnectionOptions 和 ConnectionProfile 详细信息将事务建议发送到网络中的正确对等方，在那里可以获得所需的认可。但是应用程序不需要担心任何这些问题——它只会发出 SubmitTransaction，而 SDK 会处理所有这些问题!

Let’ s now turn our attention to how the application handles the response!

现在让我们关注应用程序如何处理响应!

7、Process response

7、过程响应

Recall from papercontract.js how the issue transaction returns a commercial paper response:

从 papercontract.js 中回顾问题事务如何返回商业票据响应:

```
return paper.toBuffer();
```

You’ ll notice a slight quirk - the new paper needs to be converted to a buffer before it is returned to the application. Notice how issue.js uses the class method CommercialPaper.fromBuffer() to rehydrate the response buffer as a commercial paper:

您会注意到一个小的怪癖——在将新纸张返回到应用程序之前，需要将其转换为缓冲区。请注意 issue.js 如何使用 class 方法 commercial paper.frombuffer () 将响应缓冲区重新水化为商业文件:

```
let paper = CommercialPaper.fromBuffer(issueResponse);
```

This allows paper to be used in a natural way in a descriptive completion message:

这允许在描述性完成消息中以自然方式使用纸张：

```
console.log(`${paper.issuer}  commercial  paper  :  ${paper.paperNumber} successfully issued for value ${paper.faceValue}`);
```

See how the same paper class has been used in both the application and smart contract - if you structure your code like this, it’ll really help readability and reuse.

看看在应用程序和智能合约中如何使用相同的 Paper 类——如果您像这样构造代码，它将真正有助于可读性和重用性。

As with the transaction proposal, it might appear that the application receives control soon after the smart contract completes, but that’s not the case. Under the covers, the SDK manages the entire consensus process, and notifies the application when it is complete according to the strategy connectionOption. If you’re interested in what the SDK does under the covers, read the detailed transaction flow.

与事务建议一样，应用程序可能在智能合约完成后很快就收到控制权，但事实并非如此。在封面下，SDK 管理整个共识过程，并根据策略连接选项在完成时通知应用程序。如果您对 sdk 在封面下的工作感兴趣，请阅读详细的事务流程。

That’s it! In this topic you’ve understood how to call a smart contract from a sample application by examining how MagnetoCorp’s application issues a new commercial paper in PaperNet. Now examine the key ledger and smart contract data structures are designed by in the architecture topic behind them.

就是这样！在本主题中，您已经了解了如何通过检查 Magencorp 的应用程序如何在 Papernet 中发布新的商业论文来从示例应用程序调用智能合约。现在检查一下关键分类账和智能合约数据结构是由它们背后的架构主题设计的。

六、APIs

六、API

七、Application design elements

七、应用程序设计元素

This section elaborates the key features for client application and smart contract development found in Hyperledger Fabric. A solid understanding of the features will help you design and implement efficient and effective solutions.

本节阐述了在 Hyperledger 结构中发现的客户应用和智能合约开发的关键特性。对这些特性的深入了解将有助于您设计和实现高效的解决方案。

Contract names

合同名称

Chaincode namespace

链代码命名空间

Transaction context

交易上下文

Transaction handlers

交易处理程序

Endorsement policies

背书政策

Connection Profile

连接配置文件

Connection Options

连接选项

Wallet

钱包

Gateway

网关

1、Contract names

1、合同名称

Audience: Architects, application and smart contract developers, administrators

受众：架构师、应用程序和智能合约开发人员、管理员

A chaincode is a generic container for deploying code to a Hyperledger Fabric blockchain network. One or more related smart contracts are defined within a chaincode. Every smart contract has a name that uniquely identifies it within a chaincode. Applications access a particular smart contract within an instantiated chaincode using its contract name.

链码是将代码部署到超级账本结构区块链网络的通用容器。链码中定义了一个或多个相关智能合约。每个智能合约都有一个在链码中唯一标识它的名称。应用程序使用其合同名称访问实例化的链代码中的特定智能合约。

In this topic, we’re going to cover:

在本主题中，我们将介绍：

How a chaincode contains multiple smart contracts

链码如何包含多个智能合约

How to assign a smart contract name

如何分配智能合约名称

How to use a smart contract from an application

如何从应用程序中使用智能合约

The default smart contract

默认智能合约

Chaincode

链码

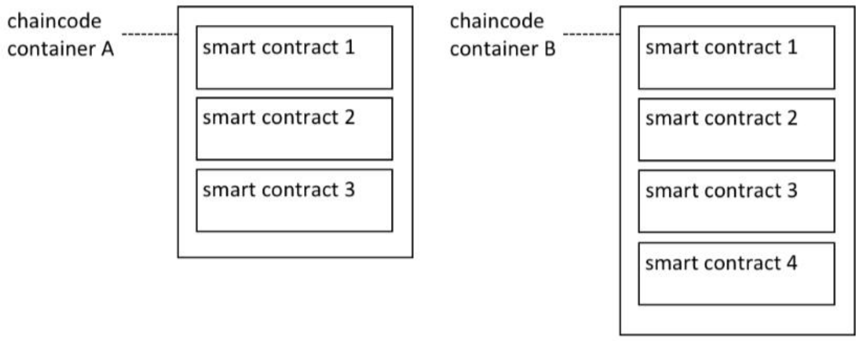
In the Developing Applications topic, we can see how the Fabric SDKs provide high level programming abstractions which help application and smart contract developers to focus on their business problem, rather than the low level details of how to interact with a Fabric network.

在“开发应用程序”主题中，我们可以看到Fabric SDK如何提供高级编程抽象，帮助

应用程序和智能合约开发人员专注于他们的业务问题，而不是如何与 Fabric 网络交互的低级细节。

Smart contracts are one example of a high level programming abstraction, and it is possible to define smart contracts within in a chaincode container. When a chaincode is installed and instantiated, all the smart contracts within it are made available to the corresponding channel.

智能合约是高级编程抽象的一个例子，可以在链码容器中定义智能合约。当安装并实例化一个链码时，其中的所有智能合约都可用于相应的通道。



Multiple smart contracts can be defined within a chaincode. Each is uniquely identified by their name within a chaincode.

可以在一个链码内定义多个智能合约。每一个都由其在链码中的名称唯一标识。

In the diagram above, chaincode A has three smart contracts defined within it, whereas chaincode B has four smart contracts. See how the chaincode name is used to fully qualify a particular smart contract.

在上面的图中，chaincode A 在其中定义了三个智能合约，而 chaincode B 有四个智能合约。查看如何使用链码名称完全限定特定智能合约。

The ledger structure is defined by a set of deployed smart contracts. That’s because the ledger contains facts about the business objects of interest to the network (such as commercial paper within PaperNet), and these business objects are moved through their lifecycle (e.g. issue, buy, redeem) by the transaction functions defined within a smart contract.

分类帐结构由一组已部署的智能合约定义。这是因为分类帐包含有关网络感兴趣的业务对象（如 Papernet 中的商业票据）的事实，并且这些业务对象通过智能合约中定义的交易功能在其生命周期（如发行、购买、赎回）中移动。

In most cases, a chaincode will only have one smart contract defined within it. However, it can make sense to keep related smart contracts together in a single chaincode. For example, commercial papers denominated in different currencies might have contracts EuroPaperContract, DollarPaperContract, YenPaperContract which might need to be kept synchronized with each other in the channel to which they are deployed.

在大多数情况下，一个链码在其中只定义了一个智能合约。然而，将相关的智能合约放在一个链码中是有意义的。例如，以不同货币计价的商业票据可能有合同 EuropaperContract、DollarpaperContract、YenPaperContract，这些合同可能需要在部署它们的渠道中保持同步。

Name

命名

Each smart contract within a chaincode is uniquely identified by its contract name. A smart contract can explicitly assign this name when the class is constructed, or let the Contract class implicitly assign a default name.

链码中的每个智能合约都由其合约名称唯一标识。智能协定可以在构造类时显式分配此名称，或者让协定类隐式分配默认名称。

Examine the papercontract.js chaincode file:

检查 papercontract.js 链码文件：

```
class CommercialPaperContract extends Contract {
constructor() {
// Unique name when multiple contracts per chaincode file
super('org.papernet.commercialpaper');
}
```

See how the CommercialPaperContract constructor specifies the contract name as org.papernet.commercialpaper. The result is that within the papercontract chaincode, this smart contract is now associated with the contract name org.papernet.commercialpaper.

请参阅 CommercialPaperContract 构造函数如何将合同名称指定为 org.papernet.CommercialPaper。结果是，在纸质合同链代码中，该智能合同现在与合同名称 org.papernet.commercialpaper 关联。

If an explicit contract name is not specified, then a default name is assigned – the name of the class. In our example, the default contract name would be CommercialPaperContract.

如果未指定显式协定名称，则会指定默认名称－类的名称。在我们的示例中，默认合同名称将是商业纸质合同。

Choose your names carefully. It' s not just that each smart contract must have a unique name; a well-chosen name is illuminating. Specifically, using an explicit DNS-style naming convention is recommended to help organize clear and meaningful names; org.papernet.commercialpaper conveys that the PaperNet network has defined a standard commercial paper smart contract.

仔细选择你的名字。不仅仅是每个智能合约都必须有一个唯一的名字，一个精心选择的名字也很有启发性。具体来说，建议使用明确的 DNS 风格命名约定来帮助组织清晰而有意义的名称；org.papernet.commercial paper 表示 Papernet 网络已经定义了标准的商业票据智能合约。

Contract names are also helpful to disambiguate different smart contract transaction functions with the same name in a given chaincode. This happens when smart contracts are closely related; their transaction names will tend to be the same. We can see that a transaction is uniquely defined within a channel by the combination of its chaincode and smart contract name.

合同名称还有助于消除给定链码中具有相同名称的不同智能合同事务函数的歧义。当智能合约密切相关时，就会发生这种情况；它们的交易名称往往是相同的。我们可以看到，一个事务是通过其链码和智能合约名称的组合在一个通道中唯一定义的。

Contract names must be unique within a chaincode file. Some code editors will detect multiple definitions of the same class name before deployment. Regardless the chaincode will return an error if multiple classes with the same contract name are explicitly or implicitly specified.

合同名称在链码文件中必须唯一。一些代码编辑器将在部署之前检测到相同类名的多个定义。不管怎样，如果显式或隐式指定了多个具有相同协定名称的类，那么 chaincode 将返回错误。

Application

应用程序

Once a chaincode has been installed on a peer and instantiated on a channel, the smart contracts in it are accessible to an application:

一旦在对等机上安装了链码并在通道上实例化了链码，应用程序就可以访问其中的智能合约：

```
const network = await gateway.getNetwork(`papernet`);
const contract = await network.getContract('papercontract', 'org.papernet.commercialpaper');
const issueResponse = await contract.submitTransaction('issue', 'MagnetoCorp', '00001', '2020-05-31', '2020-11-30', '5000000');
```

See how the application accesses the smart contract with the contract.getContract() method. The papercontract chaincode name org.papernet.commercialpaper returns a contract reference which can be used to submit transactions to issue commercial paper with the contract.submitTransaction() API.

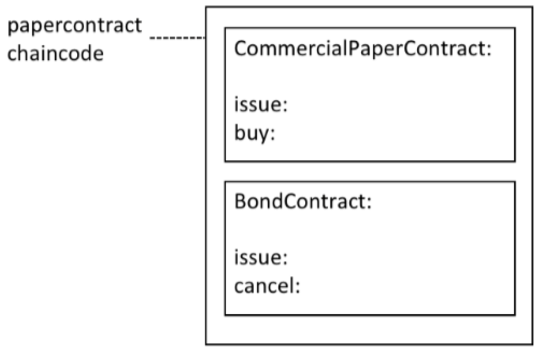
请参阅应用程序如何使用 contract.getcontract()方法访问智能合约。paperContract chaincode name org.papernet.commercial paper 返回一个合同引用，该引用可用于提交事务以使用 contract.submitTransaction () API 发行商业票据。

Default contract

默认合同

The first smart contract defined in a chaincode is the called the default smart contract. A default is helpful because a chaincode will usually have one smart contract defined within it; a default allows the application to access those transactions directly - without specifying a contract name.

链码中定义的第一个智能合约称为默认智能合约。默认值很有用，因为链码通常在其中定义了一个智能合约；默认值允许应用程序直接访问这些事务，而无需指定合约名称。



A default smart contract is the first contract defined in a chaincode.

默认智能合约是链码中定义的第一个合约。

In this diagram, CommercialPaperContract is the default smart contract. Even though we have two smart contracts, the default smart contract makes our previous example easier to write:

在此图中，商业纸质合同是默认的智能合同。尽管我们有两个智能合约，但默认的智能合约使我们以前的示例更容易编写：

```
const network = await gateway.getNetwork(`papernet`);
const contract = await network.getContract('papercontract');
const issueResponse = await contract.submitTransaction('issue', 'MagnetoCorp', '00001', '2020-05-31', '2020-11-30', '5000000');
```

This works because the default smart contract in papercontract is CommercialPaperContract and it has an issue transaction. Note that the issue transaction in BondContract can only be invoked by explicitly addressing it. Likewise, even though the cancel transaction is unique, because BondContract is not the default smart contract, it must also be explicitly addressed.

这是因为纸质合同中的默认智能合同是商业纸质合同，并且它有一个问题交易。注意，BondContract 中的发行事务只能通过显式地寻址来调用。同样，即使取消交易是唯一的，因为 BondContract 不是默认的智能合约，它也必须被明确地处理。

In most cases, a chaincode will only contain a single smart contract, so careful naming of the chaincode can reduce the need for developers to care about chaincode as a concept. In the example code above it feels like papercontract is a smart contract.

在大多数情况下，链码只包含一个智能合约，因此仔细命名链码可以减少开发人员将链码作为一个概念来关注的需求。在上面的示例代码中，感觉 PaperContract 是一个智能合同。

In summary, contract names are a straightforward mechanism to identify individual smart contracts within a given chaincode. Contract names make it easy for applications to find a particular smart contract and use it to access the ledger.

总之，合同名称是一种简单的机制，可以在给定的链代码中标识单个智能合同。合同名称使应用程序很容易找到特定的智能合同，并使用它访问分类帐。

2、Chaincode namespace

2、链代码命名空间

Audience: Architects, application and smart contract developers, administrators

受众：架构师、应用程序和智能合约开发人员、管理员

A chaincode namespace allows it to keep its world state separate from other chaincodes. Specifically, smart contracts in the same chaincode share direct access to the same world state, whereas smart contracts in different chaincodes cannot directly access each other’ s world state. If a smart contract needs to access another chaincode world state, it can do this by performing a chaincode-to-chaincode invocation. Finally, a blockchain can contain transactions which relate to different world states.

chaincode 名称空间允许它将其世界状态与其他 chaincode 分开。具体来说，同一链码中的智能合约共享对同一世界状态的直接访问，而不同链码中的智能合约不能直接访问彼此的世界状态。如果智能合约需要访问另一个链码世界状态，它可以通过执行链码到链码调用来实现这一点。最后，区块链可以包含与不同世界国家相关的交易。

In this topic, we’re going to cover:

在本主题中，我们将介绍：

The importance of namespaces

名称空间的重要性

What is a chaincode namespace

什么是链代码命名空间

Channels and namespaces

通道和命名空间

How to use chaincode namespaces

如何使用 chaincode 名称空间

How to access world states across smart contracts

如何通过智能合约访问世界各国

Design considerations for chaincode namespaces

链代码名称空间的设计注意事项

Motivation

动机

A namespace is a common concept. We understand that Park Street, New York and Park Street, Seattle are different streets even though they have the same name. The city forms a namespace for Park Street, simultaneously providing freedom and clarity.

名称空间是一个常见的概念。我们理解公园街、纽约和西雅图公园街是不同的街道，尽管它们有相同的名称。城市形成了公园街的名称空间，同时提供了自由和清晰。

It’s the same in a computer system. Namespaces allow different users to program and operate different parts of a shared system, without getting in each other’s way. Many programming languages have namespaces so that programs can freely assign unique identifiers, such as variable names, without worrying about other programs doing the same. We’ll see that Hyperledger Fabric uses namespaces to help smart contracts keep their ledger world state separate from other smart contracts.

在计算机系统中是一样的。名称空间允许不同的用户编程和操作共享系统的不同部分，而不会妨碍彼此。许多编程语言都有名称空间，这样程序就可以自由地分配唯一的标识符，例如变量名，而不用担心其他程序也会这样做。我们将看到，HyperledgeFabric 使用名称空间帮助智能合约将其分类帐世界状态与其他智能合约分开。

Scenario

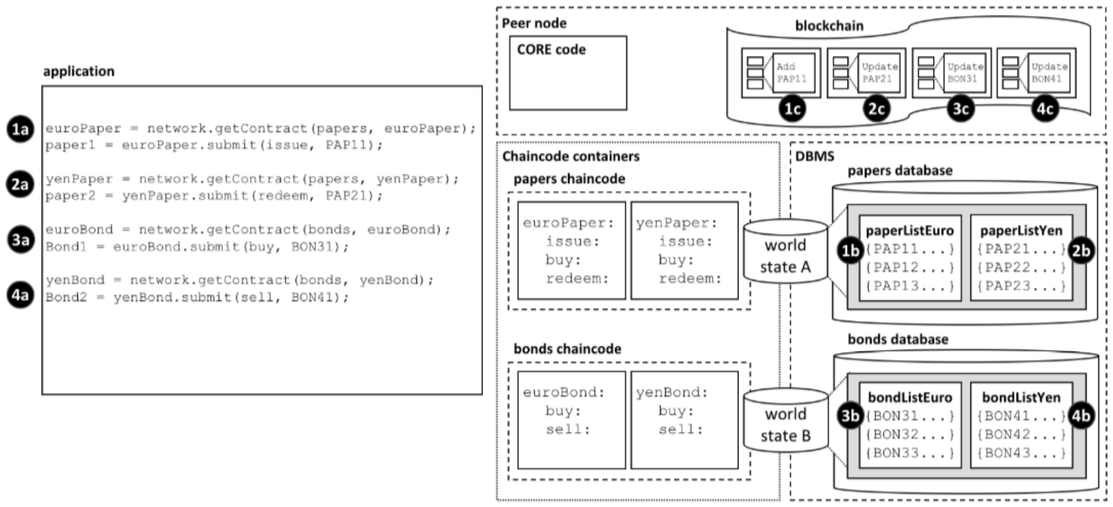
脚本

Let’s examine how the ledger world state organizes facts about business objects that are important to the organizations in a channel using the diagram below. Whether these objects are commercial papers, bonds, or vehicle registrations, and wherever they are in their lifecycle, they are maintained as states within the ledger world state database. A smart contract manages these business objects by interacting with the ledger (world state and blockchain), and in most cases this will involve it querying or updating the ledger world state.

让我们使用下面的图表来检查Ledger World State如何组织有关业务对象的事实，这些业务对象对渠道中的组织很重要。无论这些对象是商业票据、债券或车辆登记，以及它们在其生命周期中的任何位置，它们都作为状态保存在Ledger World State数据库中。智能合约通过与分类账（世界状态和区块链）交互来管理这些业务对象，在大多数情况下，这将涉及到它查询或更新分类账世界状态。

It’s vitally important to understand that the ledger world state is partitioned according to the chaincode of the smart contract that accesses it, and this partitioning, or namespacing is an important design consideration for architects, administrators and programmers.

理解分类帐世界状态是根据访问它的智能合约的链码来划分的，这一点非常重要，而这个划分，或者说名称间距，对于架构师、管理员和程序员来说是一个重要的设计考虑因素。


```
euroPaper = network.getContract(papers, euroPaper);
paper1 = euroPaper.submit(issue, PAP11);

yenPaper = network.getContract(papers, yenPaper);
paper2 = yenPaper.submit(redeem, PAP21);

euroBond = network.getContract(bonds, euroBond);
Bond1 = euroBond.submit(buy, BON31);

yenBond = network.getContract(bonds, yenBond);
Bond2 = yenBond.submit(sell, BON41);
```

The ledger world state is separated into different namespaces according to the chaincode that accesses it. Within a given channel, smart contracts in the same chaincode share the same world state, and smart contracts in different chaincodes cannot directly access each other’s world state. Likewise, a blockchain can contain transactions that relate to different chaincode world states.

分类帐世界状态根据访问它的链代码分为不同的名称空间。在一个给定的通道中，同一个链码中的智能合约共享同一个世界状态，而不同链码中的智能合约不能直接访问彼此的世界状态。同样，区块链可以包含与不同链码世界状态相关的交易。

In our example, we can see four smart contracts defined in two different chaincodes, each of which is in their own chaincode container. The euroPaper and yenPaper smart contracts are defined in the papers chaincode. The situation is similar for the euroBond and yenBond smart contracts – they are defined in the bonds chaincode. This design helps application programmers understand whether they are working with commercial papers or bonds priced in Euros or Yen, and because the rules for each financial product don’t really change for different currencies, it makes sense to manage their deployment in the same chaincode.

在我们的示例中，我们可以看到在两个不同的链码中定义的四个智能合约，每个链码都在各自的链码容器中。Europaper 和 Yenpaper 智能合约在论文链码中定义。欧洲债券和 Yenbond 智能合约的情况类似——它们在债券链代码中定义。这种设计有助于应用程序程序员理解他们是否使用以欧元或日元计价的商业票据或债券，并且由于每个金融产品的规则对于不同的货币并没有真正的改变，因此在同一个链码中管理它们的部署是有意义的。

The diagram also shows the consequences of this deployment choice. The database management system (DBMS) creates different world state databases for the papers and bonds chaincodes and the smart contracts contained within them. World state A and world state B are each held within distinct databases; the data are isolated from each other such that a single world state query (for example) cannot access both world states. The world state is said to be namespaced according to its chaincode.

该图还显示了这种部署选择的结果。数据库管理系统（DBMS）为论文和债券链码以及其中包含的智能合约创建不同的世界状态数据库。世界状态 A 和世界状态 B 都保存在不同的数据库中；数据彼此隔离，因此单个世界状态查询（例如）无法访问两个世界状态。据说世界国家是根据其链码命名的。

See how world state A contains two lists of commercial papers paperListEuro and paperListYen. The states PAP11 and PAP21 are instances of each paper managed by the euroPaper and yenPaper smart contracts respectively. Because they share the same chaincode namespace, their keys (PAPxyz) must be unique within the namespace of the papers chaincode, a little like a street name is unique within a town. Notice how it would be possible to write a smart contract in the papers chaincode that performed an aggregate calculation over all the commercial papers - whether priced in Euros or Yen - because they share the same namespace. The situation is similar for bonds - they are held within world state B which maps to a separate bonds database, and their keys must be unique.

了解世界 A 州如何包含两个商业文件列表：PaperlistEuro 和 PaperlistYen。PAP11 和 PAP21 是分别由欧洲报纸和 YenPaper 智能合约管理的每份文件的实例。因为它们共享相同的 chaincode 名称空间，所以它们的键（papxyz）在 papers chaincode 的名称空间中必须是唯一的，有点像街道名称在城镇中是唯一的。请注意，如何在论文链代码中编写智能合约，该代码对所有商业论文（无论是以欧元还是日元定价）执行聚合计算，因为它们共享相同的命名空间。债券的情况类似——它们被保存在世界 B 国，B 国映射到一个单独的债券数据库，它们的密钥必须是唯一的。

Just as importantly, namespaces mean that euroPaper and yenPaper cannot directly access world state B, and that euroBond and yenBond cannot directly access world state A. This isolation is helpful, as commercial papers and bonds are very distinct financial instruments; they have different attributes and are subject to different rules. It also means that papers and bonds could have the same keys, because they are in different namespaces. This is helpful; it provides a significant degree of freedom for naming. Use this freedom to name different business objects meaningfully.

同样重要的是，名称空间意味着 Europaper 和 YenPaper 不能直接访问世界 B 国，Eurobond 和 Yenbond 不能直接访问世界 A 国。这种隔离是有帮助的，因为商业票据和债券

是非常独特的金融工具；它们具有不同的属性，并受不同的规则约束。它还意味着文件和绑定可以具有相同的键，因为它们在不同的名称空间中。这很有帮助；它为命名提供了很大的自由度。使用这个自由来有意义地命名不同的业务对象。

Most importantly, we can see that a blockchain is associated with the peer operating in a particular channel, and that it contains transactions that affect both world state A and world state B. That’s because the blockchain is the most fundamental data structure contained in a peer. The set of world states can always be recreated from this blockchain, because they are the cumulative results of the blockchain’s transactions. A world state helps simplify smart contracts and improve their efficiency, as they usually only require the current value of a state. Keeping world states separate via namespaces helps smart contracts isolate their logic from other smart contracts, rather than having to worry about transactions that correspond to different world states. For example, a bonds contract does not need to worry about paper transactions, because it cannot see their resultant world state.

最重要的是，我们可以看到一个区块链与在特定渠道中运行的对等体相关联，并且它包含影响世界状态 A 和世界状态 B 的交易。这是因为区块链是对等体中包含的最基本的数据结构。世界状态集总是可以从此区块链中重新创建，因为它们是区块链交易的累积结果。世界国家有助于简化智能合约并提高效率，因为它们通常只需要一个国家的当前价值。通过名称空间将世界各国分开有助于智能合约将其逻辑与其他智能合约隔离开来，而无需担心与不同世界国家对应的事务。例如，债券合同不需要担心纸质交易，因为它看不到它们的最终世界状态。

It’s also worth noticing that the peer, chaincode containers and DBMS all are logically different processes. The peer and all its chaincode containers are always in physically separate operating system processes, but the DBMS can be configured to be embedded or separate, depending on its type. For LevelDB, the DBMS is wholly contained within the peer, but for CouchDB, it is a separate operating system process.

值得注意的是，对等端、链码容器和 DBMS 都是逻辑上不同的过程。对等端及其所有链码容器始终位于物理上独立的操作系统进程中，但 DBMS 可以配置为嵌入或分离，具体取决于其类型。对于 LEVELDB，DBMS 完全包含在对等端中，但对于 COUCHDB，它是一个单独的操作系统进程。

It’s important to remember that namespace choices in this example are the result of a business requirement to share commercial papers in different currencies but isolate them separate from bonds. Think about how the namespace structure would be modified to meet a business requirement to keep every financial asset class separate, or share all commercial papers and bonds?

重要的是要记住，本例中的名称空间选择是业务要求以不同货币共享商业票据，但将它们与债券分离的结果。想想如何修改名称空间结构以满足业务需求，使每个金融资产类别保持独立，或者共享所有商业票据和债券？

Channels

通道

If a peer is joined to multiple channels, then a new blockchain is created and managed for each channel. Moreover, every time a chaincode is instantiated in a new channel, a new world state database is created for it. It means that the channel also forms a kind of namespace alongside that of the chaincode for the world state.

如果一个对等体加入多个渠道，那么将为每个渠道创建和管理一个新的区块链。此外，每次在新通道中实例化一个链代码时，都会为它创建一个新的世界状态数据库。这意味着该通道还与世界状态的链代码形成了一种名称空间。

However, the same peer and chaincode container processes can be simultaneously joined to multiple channels - unlike blockchains, and world state databases, these processes do not increase with the number of channels joined.

但是，同一个对等和链码容器进程可以同时连接到多个通道——与区块链和世界状态数据库不同，这些进程不会随着加入的通道的数量而增加。

For example, if the papers and bonds chaincodes were instantiated on a new channel, there would a totally separate blockchain created, and two new world state databases created. However, the peer and chaincode containers would not increase; each would just be connected to multiple channels.

例如，如果论文和债券链码在一个新的通道上被实例化，那么将创建一个完全独立的区块链，并创建两个新的世界状态数据库。但是，对等端和链码容器不会增加；每个容器只会连接到多个通道。

Usage

用法

Let' s use our commercial paper example to show how an application uses a smart contract with namespaces. It' s worth noting that an application communicates with the peer, and the peer routes the request to the appropriate chaincode container which then accesses the DBMS. This routing is done by the peer core component shown in the diagram.

让我们使用商业论文示例来演示应用程序如何使用带有名称空间的智能合约。值得注意的是，应用程序与对等端通信，对等端将请求路由到相应的链码容器，然后该容器访问 DBMS。此路由由图中所示的对等核心组件完成。

Here' s the code for an application that uses both commercial papers and bonds, priced in Euros and Yen. The code is fairly self-explanatory:

这是一个应用程序的代码，它使用商业票据和债券，以欧元和日元计价。代码是相当简单的：

```
const euroPaper = network.getContract(papers, euroPaper);
paper1 = euroPaper.submit(issue, PAP11);
const yenPaper = network.getContract(papers, yenPaper);
paper2 = yenPaper.submit(redeem, PAP21);
const euroBond = network.getContract(bonds, euroBond);
bond1 = euroBond.submit(buy, BON31);
const yenBond = network.getContract(bonds, yenBond);
bond2 = yenBond.submit(sell, BON41);
```

See how the application:

查看应用程序如何：

Accesses the euroPaper and yenPaper contracts using the getContract() API specifying the papers chaincode. See interaction points 1a and 2a.

使用 getContract（）API 指定纸张链代码访问 Europaper 和 YenPaper 合同。见交互点 1a 和 2a。

Accesses the euroBond and yenBond contracts using the getContract() API specifying the bonds chaincode. See interaction points 3a and 4a.

使用 getContract（）API 指定债券链代码访问 Eurobond 和 Yenbond 合同。见交互点 3a 和 4a。

Submits an issue transaction to the network for commercial paper PAP11 using the euroPaper contract. See interaction point 1a. This results in the creation of a commercial paper represented by state PAP11 in world state A; interaction point 1b. This operation is captured as a transaction in the blockchain at interaction point 1c.

使用欧洲报纸合同向网络提交商业票据 PAP11 的发行交易。参见交互点 1a。这导致创建了以世界状态 a 中的状态 pap11 为代表的商业票据；交互点 1b。此操作在交互点 1c 被捕获为区块链中的交易。

Submits a redeem transaction to the network for commercial paper PAP21 using the yenPaper contract. See interaction point 2a. This results in the creation of a commercial paper represented by state PAP21 in world state A; interaction point 2b. This operation is captured as a transaction in the blockchain at interaction point 2c.

使用 YenPaper 合同向网络提交商业票据 PAP21 的兑换交易。参见交互点 2a。这导致创建了以世界状态 a 中的状态 pap21 为代表的商业票据；交互点 2b。此操作在交互点 2c 被捕获为区块链中的交易。

Submits a buy transaction to the network for bond BON31 using the euroBond contract. See interaction point 3a. This results in the creation of a bond represented by state BON31 in world state B; interaction point 3b. This operation is captured as a transaction in the blockchain at interaction point 3c.

使用欧洲债券合同向网络提交 Bond31 的买入交易。见交互点 3a。这导致了一种债券的产生，在世界状态 b 中由状态 bon31 表示；交互点 3b。此操作被捕获为在交互点 3c 的区块链中的交易。

Submits a sell transaction to the network for bond BON41 using the yenBond contract. See interaction point 4a. This results in the creation of a bond represented by state BON41 in world state B; interaction point 4b. This operation is captured as a transaction in the blockchain at interaction point 4c.

使用 Yenbond 合同向网络提交 Bond Bon41 的销售交易。见交互点 4a。这导致了一种债券的产生，该债券由世界 B 州的邦 41 代表；交互点 4b。该操作被捕获为交互点 4c 处区块链中的交易。

See how smart contracts interact with the world state:

了解智能合约如何与世界各国互动：

euroPaper and yenPaper contracts can directly access world state A, but cannot directly access world state B. World state A is physically held in the papers database in the database management system (DBMS) corresponding to the papers chaincode.

Europaper 和 YenPaper 合同可以直接访问世界状态 A，但不能直接访问世界状态 B。世界状态 A 物理上保存在纸张链码对应的数据库管理系统（DBMS）中的纸张数据库中。

euroBond and yenBond contracts can directly access world state B, but cannot directly access world state A. World state B is physically held in the bonds database in the database management system (DBMS) corresponding to the bonds chaincode.

Eurobond 和 Yenbond 合同可以直接访问世界状态 B，但不能直接访问世界状态 A。世界状态 B 物理上保存在债券链代码对应的数据库管理系统（DBMS）中的债券数据库中。

See how the blockchain captures transactions for all world states:

查看区块链如何捕获所有世界各国的交易：

Interactions 1c and 2c correspond to transactions create and update commercial papers PAP11 and PAP21 respectively. These are both contained within world state A.

交互 1c 和 2c 分别对应于交易创建和更新商业文件 pap11 和 pap11。它们都包含在世界 A 国。

Interactions 3c and 4c correspond to transactions both update bonds BON31 and BON41. These are both contained within world state B.

交互 3c 和 4c 与更新 Bond31 和 Bon41 的交易相对应。它们都包含在世界 B 州内。

If world state A or world state B were destroyed for any reason, they could be recreated by replaying all the transactions in the blockchain.

如果世界状态 A 或世界状态 B 因任何原因被破坏，可以通过重放区块链中的所有交易来重新创建它们。

Cross chaincode access

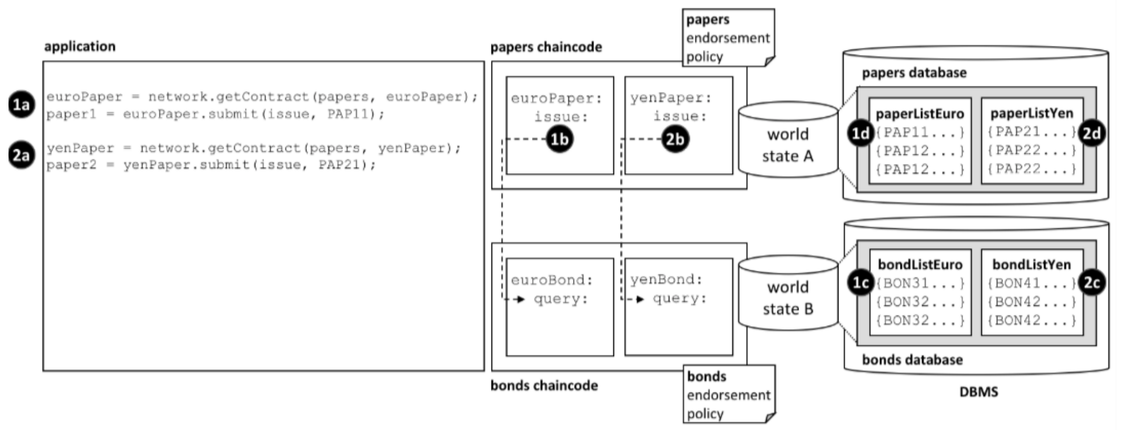
交叉链码访问

As we saw in our example scenario, euroPaper and yenPaper cannot directly access world state B. That’s because we have designed our chaincodes and smart contracts so that these chaincodes and world states are kept separately from each other. However, let’s imagine that euroPaper needs to access world state B.

正如我们在示例场景中看到的，Europaper 和 YenPaper 无法直接访问世界状态 B。这是因为我们设计了链码和智能合约，使这些链码和世界状态彼此独立。然而，让我们设想一下，欧洲报纸需要进入世界 B 州。

Why might this happen? Imagine that when a commercial paper was issued, the smart contract wanted to price the paper according to the current return on bonds with a similar maturity date. In this case it will be necessary for the euroPaper contract to be able to query the price of bonds in world state B. Look at the following diagram to see how we might structure this interaction.

为什么会这样？想象一下，当商业票据发行时，智能合约想要根据具有类似到期日的当前债券收益率来定价。在这种情况下，欧洲报纸合同将有必要能够查询世界 B 国债券的价格。请看下图，了解我们如何构造这种交互。



```
euroPaper = network.getContract(papers, euroPaper);
paper1 = euroPaper.submit(issue, PAP11);

yenPaper = network.getContract(papers, yenPaper);
paper2 = yenPaper.submit(issue, PAP21);
```

How chaincodes and smart contracts can indirectly access another world state - via its chaincode.

链码和智能合约如何通过链码间接访问另一个世界国家。

Notice how:

注意如何：

the application submits an issue transaction in the euroPaper smart contract to issue PAP11. See interaction 1a.

该申请在欧洲报纸智能合约中提交了一份发行交易，以发行 PAP11。见互动 1a。

the issue transaction in the euroPaper smart contract calls the query transaction in the euroBond smart contract. See interaction point 1b.

欧洲报纸智能合约中的发行交易调用欧洲债券智能合约中的查询交易。见交互点 1b。

the queryin euroBond can retrieve information from world state B. See interaction point 1c.

欧洲债券查询可以从世界 B 国检索信息，见交互点 1C。

when control returns to the issue transaction, it can use the information in the response to price the paper and update world state A with information. See interaction point 1d.

当控制权返回到发行事务时，它可以使用响应中的信息为纸张定价，并用信息更新世界状态 A。见交互点 1d。

the flow of control for issuing commercial paper priced in Yen is the same. See interaction points 2a, 2b, 2c and 2d.

发行以日元计价的商业票据的控制流程相同。见交互点 2a、2b、2c 和 2d。

Control is passed between chaincode using the invokeChaincode() API. This API passes control from one chaincode to another chaincode.

使用 InvokeChaincode（）API 在链代码之间传递控件。这个 API 将控制从一个链码传递到另一个链码。

Although we have only discussed query transactions in the example, it is possible to invoke a smart contract which will update the called chaincode’s world state. See the considerations below.

虽然我们在示例中只讨论了查询事务，但是可以调用智能合约来更新被调用的链代码的世界状态。请参阅下面的注意事项。

Considerations

考虑事项

In general, each chaincode will have a single smart contract in it.

一般来说，每个链码中都有一个智能合约。

Multiple smart contracts should only be deployed in the same chaincode if they are very closely related. Usually, this is only necessary if they share the same world state.

如果多个智能合约关系密切，则只能在同一个链代码中部署它们。通常，只有当他们共享同一个世界国家时，这才是必要的。

Chaincode namespaces provide isolation between different world states. In general it makes sense to isolate unrelated data from each other. Note that you cannot choose the chaincode namespace; it is assigned by Hyperledger Fabric, and maps directly to the name of the chaincode.

chaincode 名称空间提供了不同世界状态之间的隔离。一般来说，将不相关的数据彼此隔离是有意义的。注意，您不能选择 chaincode 名称空间；它由 hyperledger 结构分配，并直接映射到 chaincode 的名称。

For chaincode to chaincode interactions using the invokeChaincode() API, both chaincodes must be installed on the same peer.

对于使用 invokeChainCode（）API 进行的链代码到链代码交互，两个链代码必须安装在同一个对等机上。

For interactions that only require the called chaincode’s world state to be queried, the invocation can be in a different channel to the caller’s chaincode.

对于只需要查询被调用链代码的世界状态的交互,调用可以位于调用方链代码的不同通道中。

For interactions that require the called chaincode’s world state to be updated, the invocation must be in the same channel as the caller’s chaincode.

对于需要更新被调用链代码的世界状态的交互,调用必须与调用方的链代码在同一个通道中。

3、Transaction context

3、交易上下文

Content being added in FAB-10440

FAB-10440 中添加的内容

Structure

结构

4、Transaction handlers

4、交易处理程序

Audience: Architects, Application and smart contract developers

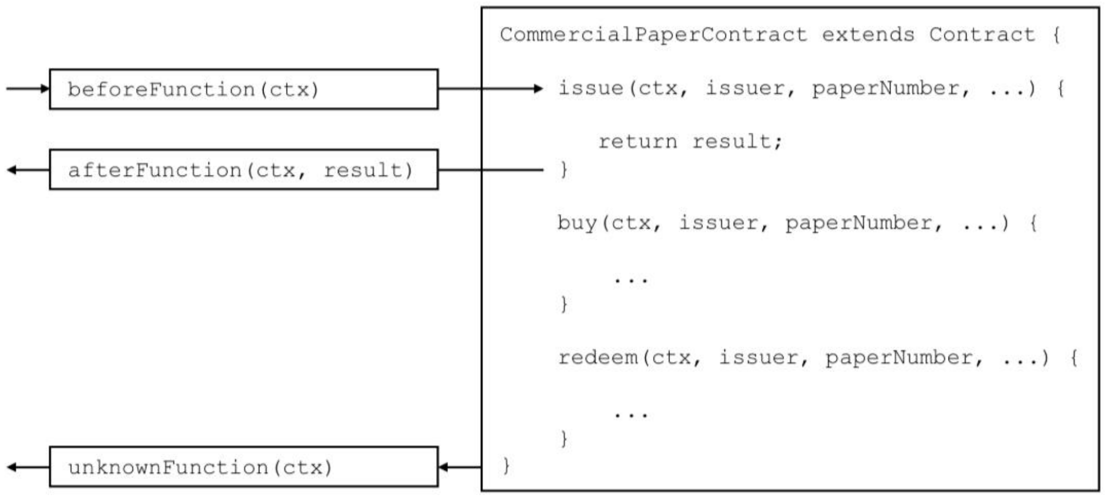
受众：架构师、应用程序和智能合约开发人员

Transaction handlers allow smart contract developers to define common processing at key points during the interaction between an application and a smart contract. Transaction handlers are optional but, if defined, they will receive control before or after every transaction in a smart contract is invoked. There is also a specific handler which receives control when a request is made to invoke a transaction not defined in a smart contract.

事务处理程序允许智能合约开发人员在应用程序和智能合约之间的交互过程中在关键点定义公共处理。事务处理程序是可选的，但如果定义了，它们将在调用智能合约中的每个事务之前或之后接收控制。还有一个特定的处理程序，它在请求调用智能合约中未定义的事务时接收控制。

Here’s an example of transaction handlers for the commercial paper smart contract sample:

以下是商业票据智能合约示例的事务处理程序示例：



```
CommercialPaperContract extends Contract {

    issue(ctx, issuer, paperNumber, ...) {

        return result;
    }

    buy(ctx, issuer, paperNumber, ...) {

        ...
    }

    redeem(ctx, issuer, paperNumber, ...) {

        ...
    }
}
```

Before, After and Unknown transaction handlers. In this example, BeforeFunction() is called before the issue, buy and redeem transactions. AfterFunction() is called after the issue, buy and redeem transactions. UnknownFunction() is only called if a request is made to invoke a transaction not defined in the smart contract. (The diagram is simplified by not repeating BeforeFunction and AfterFunction boxes for each transaction.

事务处理程序之前、之后和未知。在本例中，在发行、购买和兑换交易之前调用 beforeFunction()。AfterFunction() 在发行、购买和兑换交易之后调用。只有在请求调用智能合约中未定义的事务时，才会调用 UnknownFunction()。(该图通过不对每个事务重复 beforefunction 和 afterfunction 框来简化。

Types of handler

处理程序类型

There are three types of transaction handlers which cover different aspects of the interaction between an application and a smart contract:

有三种类型的事务处理程序，它们涵盖应用程序和智能合约之间交互的不同方面：

Before handler: is called before every smart contract transaction is invoked. The handler will usually modify the transaction context to be used by the transaction. The handler has access to the full range of Fabric APIs; for example, it can issue getState() and putState().

在调用每个智能合约事务之前调用 before handler:。处理程序通常会修改事务要使用的事务上下文。处理程序可以访问所有的结构 API；例如，它可以发出 getState()和 putstate ()。

After handler: is called after every smart contract transaction is invoked. The handler will usually perform post-processing common to all transactions, and also has full access to the Fabric APIs.

after handler：在调用每个智能合约事务之后调用。该处理程序通常执行所有事务通用的后处理，并且还具有对结构 API 的完全访问权。

Unknown handler: is called if an attempt is made to invoke a transaction that is not defined in a smart contract. Typically, the handler will record the failure for subsequent processing by an administrator. The handler has full access to the Fabric APIs.

如果试图调用智能合约中未定义的事务，则调用未知的处理程序。通常，处理程序将记录失败，以便管理员进行后续处理。处理程序可以完全访问结构 API。

Defining a handler

定义处理程序

Transaction handlers are added to the smart contract as methods with well defined names. Here’ s an example which adds a handler of each type:

事务处理程序作为定义良好的名称的方法添加到智能合约中。下面是一个示例，其中添加了每种类型的处理程序：

```
CommercialPaperContract extends Contract {
...
async beforeTransaction(ctx) {
// Write the transaction ID as an informational to the console
console.info(ctx.stub.getTxID());
};
async afterTransaction(ctx, result) {
// This handler interacts with the ledger
ctx.stub.cpList.putState(...);
};
async unknownTransaction(ctx) {
// This handler throws an exception
throw new Error('Unknown transaction function');
};
}
```

The form of a transaction handler definition is the similar for all handler types, but notice how the afterTransaction(ctx, result) also receives any result returned by the transaction.

事务处理程序定义的形式对于所有处理程序类型都是类似的，但是请注意 AfterTransaction（CTX，result）如何接收事务返回的任何结果。

Handler processing

处理程序处理

Once a handler has been added to the smart contract, it can be invoked during transaction processing. During processing, the handler receives ctx, the transaction context, performs some processing, and returns control as it completes. Processing continues as follows:

一旦将处理程序添加到智能合约中，就可以在事务处理期间调用它。在处理过程中，处

理程序接收事务上下文CTX，执行一些处理，并在完成时返回控件。处理继续如下：

Before handler: If the handler completes successfully, the transaction is called with the updated context. If the handler throws an exception, then the transaction is not called and the smart contract fails with the exception error message.

在处理程序之前：如果处理程序成功完成，则使用更新的上下文调用事务。如果处理程序引发异常，则不会调用事务，智能协定将失败，并显示异常错误消息。

After handler: If the handler completes successfully, then the smart contract completes as determined by the invoked transaction. If the handler throws an exception, then the transaction fails with the exception error message.

在处理程序之后：如果处理程序成功完成，那么智能合约将按照被调用事务确定的方式完成。如果处理程序引发异常，则事务将失败，并显示异常错误消息。

Unknown handler: The handler should complete by throwing an exception with the required error message. If an Unknown handler is not specified, or an exception is not thrown by it, there is sensible default processing; the smart contract will fail with an unknown transaction error message.

未知处理程序：处理程序应通过引发带有所需错误消息的异常来完成。如果未指定未知的处理程序，或者未引发异常，则会进行合理的默认处理；智能合约将失败，并显示未知的事务错误消息。

If the handler requires access to the function and parameters, then it is easy to do this:

如果处理程序需要访问函数和参数，那么很容易做到这一点：

```
async beforeTransaction(ctx) {
// Retrieve details of the transaction
let txnDetails = ctx.stub.getFunctionAndParameters();
console.info(`Calling function: ${txnDetails.fcn} `);
console.info(util.format(`Function arguments : %j ${stub.getArgs()} ``);
}
```

Multiple handlers

多个处理程序

It is only possible to define at most one handler of each type for a smart contract. If a smart contract needs to invoke multiple functions during before, after or unknown handling, it should coordinate this from within the appropriate function.

对于智能合约，只能为每种类型定义至多一个处理程序。如果智能合约需要在处理之前、之后或未知的过程中调用多个函数，则它应该在适当的函数内对此进行协调。

5、Endorsement policies

5、背书政策

Audience: Architects, Application and smart contract developers

受众：架构师、应用程序和智能合约开发人员

Endorsement policies define the smallest set of organizations that are required to endorse a transaction in order for it to be valid. To endorse, an organization’s endorsing peer needs to run the smart contract associated with the transaction and sign its outcome. When the ordering service sends the transaction to the committing peers, they will each individually check whether the endorsements in the transaction fulfill the endorsement policy. If this is not the case, the transaction is invalidated and it will have no effect on world state.

认可策略定义了为使交易有效而认可交易所需的最小组织集合。要进行认可，组织的认可对等方需要运行与事务关联的智能合约并签署其结果。当订购服务将交易发送给提交对等方时，他们将分别检查交易中的背书是否满足背书策略。如果不是这样的话，事务将失效，并且对世界状态没有影响。

Endorsement policies work at two different granularities: they can be set for an entire namespace, as well as for individual state keys. They are formulated using basic logic expressions such as AND and OR. For example, in PaperNet this could be used as follows: the endorsement policy for a paper that has been sold from MagnetoCorp to DigiBank could be set to AND(MagnetoCorp.peer, DigiBank.peer), requiring any changes to this paper to be endorsed by both MagnetoCorp and DigiBank.

认可策略在两个不同的粒度上工作：它们可以为整个名称空间以及单个状态键设置。它们是使用诸如和或之类的基本逻辑表达式来表示的。例如，在 PaperNet 中，这可以如下使用：对于从 Magnetorp 出售给 Digibank 的纸张的背书政策可以设置为和(Magnetorp.Peer，Digibank.Peer)，要求对该纸张的任何更改都必须经过 Magnetorp 和 Digibank 的背书。

7、Connection Profile

7、连接配置文件

Audience: Architects, application and smart contract developers

受众：架构师、应用程序和智能合约开发人员

A connection profile describes a set of components, including peers, orderers and certificate authorities in a Hyperledger Fabric blockchain network. It also contains channel and organization information relating to these components. A connection profile is primarily used by an application to configure a gateway that handles all network interactions, allowing it it to focus on business logic. A connection profile is normally created by an administrator who understands the network topology.

连接配置文件描述了一组组件，包括超级账本结构区块链网络中的对等方、订购方和证书颁发机构。它还包含与这些组件相关的渠道和组织信息。连接配置文件主要由应用程序用于配置处理所有网络交互的网关，使其能够专注于业务逻辑。连接配置文件通常由了解网络拓扑的管理员创建。

In this topic, we’re going to cover:

在本主题中，我们将介绍：

Why connection profiles are important

为什么连接配置文件很重要

How applications use a connection profile

应用程序如何使用连接配置文件

How to define a connection profile

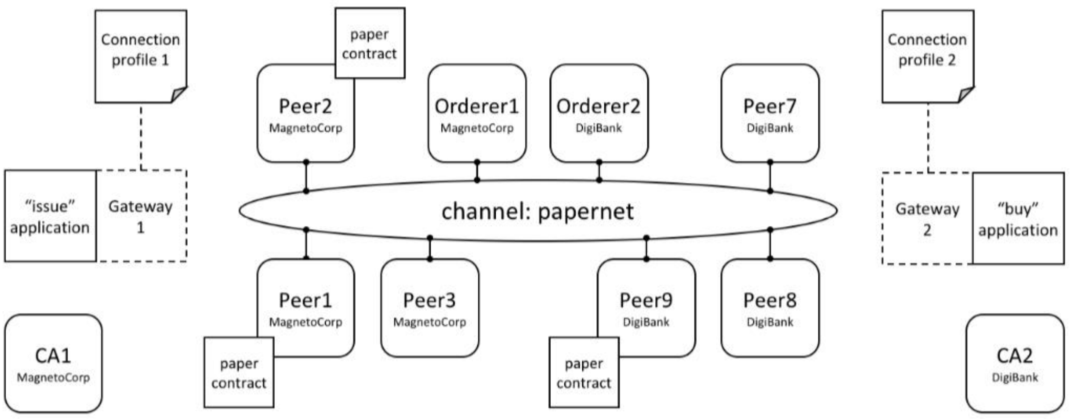
如何定义连接配置文件

Scenario

脚本

A connection profile is used to configure a gateway. Gateways are important for many reasons, the primary being to simplify an application’s interaction with a network channel.

连接配置文件用于配置网关。网关之所以重要，有许多原因，主要是为了简化应用程序与网络通道的交互。



Two applications, issue and buy, use gateways 1&2 configured with connection profiles 1&2. Each profile describes a different subset of MagnetoCorp and DigiBank network components. Each connection profile must contain sufficient information for a gateway to interact with the network on behalf of the issue and buy applications. See the text for a detailed explanation.

发行和购买两个应用程序，使用配置了连接配置文件 1 和 2 的网关 1 和 2。每个配置文件描述了 Magnetorp 和 Digibank 网络组件的不同子集。每个连接配置文件必须包含足够的信息，以便网关代表发行和购买应用程序与网络进行交互。有关详细说明，请参阅文本。

A connection profile contains a description of a network view, expressed in a technical syntax, which can either be JSON or YAML. In this topic, we use the YAML representation, as it’s easier for you to read. Static gateways need more information than dynamic gateways because the latter can use service discovery to dynamically augment the information in a connection profile.

连接配置文件包含网络视图的描述，以技术语法表示，可以是 JSON 或 YAML。在本主题中，我们使用 yaml 表示，因为它更容易阅读。静态网关比动态网关需要更多的信息，因为后者可以使用服务发现动态地增加连接配置文件中的信息。

A connection profile should not be an exhaustive description of a network channel; it just needs to contain enough information sufficient for a gateway that’s using it. In the network above, connection profile 1 needs to contain at least the endorsing organizations and peers for the issue transaction, as well as identifying the peers that will notify the gateway when the transaction has been committed to the ledger.

连接配置文件不应该是对网络通道的详尽描述；它只需要包含足够的信息，以供使用它的网关使用。在上面的网络中，连接配置文件 1 需要至少包含问题事务的认可组织和对等

方，以及标识将在事务提交到分类帐时通知网关的对等方。

It’s easiest to think of a connection profile as describing a view of the network. It could be a comprehensive view, but that’s unrealistic for a few reasons:

最简单的方法是将连接配置文件描述为网络视图。这可能是一个全面的观点，但由于以下几个原因，这是不现实的：

Peers, orderers, certificate authorities, channels, and organizations are added and removed according to demand.

对等点、订购者、证书颁发机构、通道和组织将根据需要进行添加和删除。

Components can start and stop, or fail unexpectedly (e.g. power outage).

部件可以启动和停止，或意外故障（例如断电)。

A gateway doesn’t need a view of the whole network, only what’s necessary to successfully handle transaction submission or event notification for example.

网关不需要查看整个网络，只需要成功处理事务提交或事件通知所必需的内容。

Service Discovery can augment the information in a connection profile. Specifically, dynamic gateways can be configured with minimal Fabric topology information; the rest can be discovered.

服务发现可以增加连接配置文件中的信息。具体来说，动态网关可以用最少的结构拓扑信息进行配置；其余的可以被发现。

A static connection profile is normally created by an administrator who understands the network topology in detail. That’s because a static profile can contain quite a lot of information, and an administrator needs to capture this in the corresponding connection profile. In contrast, dynamic profiles minimize the amount of definition required, and therefore can be a better choice for developers who want to get going quickly, or administrators who want to create a more responsive gateway. Connection profiles are created in either the YAML or JSON format using an editor of choice.

静态连接配置文件通常由详细了解网络拓扑的管理员创建。这是因为静态配置文件可以包含很多信息，管理员需要在相应的连接配置文件中捕获这些信息。相反，动态概要文件将所需的定义量减至最少，因此对于希望快速入门的开发人员或希望创建响应更迅速的网关的管理员来说，这是一个更好的选择。连接配置文件使用所选的编辑器以 yaml 或 json 格式创建。

Usage

用法

We’ll see how to define a connection profile in a moment; let’s first see how it is used by a sample MagnetoCorp issue application:

稍后我们将了解如何定义连接配置文件；首先让我们了解一下示例 Magnetorp 发行应用程序如何使用它：

```
const yaml = require('js-yaml');
const { Gateway } = require('fabric-network');
const connectionProfile = yaml.safeLoad(fs.readFileSync('../gateway/paperNet.yaml', 'utf8'));
const gateway = new Gateway();
await gateway.connect(connectionProfile, connectionOptions);
```

After loading some required classes, see how the paperNet.yaml gateway file is loaded from the file system, converted to a JSON object using the yaml.safeLoad() method, and used to configure a gateway using its connect() method.

在加载了一些必需的类之后，请参阅如何从文件系统加载 papernet.yaml 网关文件，如何使用 yaml.safeload（）方法转换为 JSON 对象，以及如何使用其 connect（）方法配置网关。

By configuring a gateway with this connection profile, the issue application is providing the gateway with the relevant network topology it should use to process transactions. That’s because the connection profile contains sufficient information about the PaperNet channels, organizations, peers, orderers and CAs to ensure transactions can be successfully processed.

通过使用此连接配置文件配置网关，问题应用程序向网关提供了处理事务应使用的相关网络拓扑。这是因为连接配置文件包含关于 PaperNet 通道、组织、对等方、订购方和 CA 的足够信息，以确保事务能够成功处理。

It’s good practice for a connection profile to define more than one peer for any given organization – it prevents a single point of failure. This practice also applies to dynamic gateways; to provide more than one starting point for service discovery.

对于一个连接配置文件来说，为任何给定的组织定义多个对等点是一个很好的实践——它可以防止单点故障。此实践也适用于动态网关；为服务发现提供多个起点。

A DigiBank buy application would typically configure its gateway with a similar connection profile, but with some important differences. Some elements will be the same, such as the channel; some elements will overlap, such as the endorsing peers. Other elements will be completely different, such as notification peers or certificate authorities for example.

Digibank Buy 应用程序通常会使用类似的连接配置文件配置其网关，但有一些重要的区别。一些元素将是相同的，例如通道；一些元素将重叠，例如认可对等方。其他元素将完全不同，例如通知对等方或证书颁发机构。

The connectionOptions passed to a gateway complement the connection profile. They allow an application to declare how it would like the gateway to use the connection profile. They are interpreted by the SDK to control interaction patterns with network components, for example to select which identity to connect with, or which peers to use for event notifications. Read about the list of available connection options and when to use them.

传递给网关的 ConnectionOptions 补充了连接配置文件。它们允许应用程序声明希望网关如何使用连接配置文件。它们被 SDK 解释为控制与网络组件的交互模式，例如选择要连接的标识或用于事件通知的对等方。阅读可用连接选项列表以及何时使用它们。

Structure

结构

To help you understand the structure of a connection profile, we’re going to step through an example for the network shown above. Its connection profile is based on the PaperNet commercial paper sample, and stored in the GitHub repository. For convenience, we’ve reproduced it below. You will find it helpful to display it in another browser window as you now read about it:

为了帮助您了解连接配置文件的结构，我们将逐步介绍上面所示的网络示例。其连接配置文件基于PaperNet商业纸张示例，并存储在Github存储库中。为了方便起见，我们在下面复制了它。您会发现在另一个浏览器窗口中显示它很有帮助，因为您现在已经了解了它：

Line 9: name: ″papernet.magnetocorp.profile.sample″

第9行：名称：“papernet.magnercorp.profile.sample”

This is the name of the connection profile. Try to use DNS style names; they are a very easy way to convey meaning.

这是连接配置文件的名称。尝试使用DNS样式的名称；它们是一种非常容易表达意思的方法。

Line 16: x-type: ″hlfv1″

第16行：X型：“HLFv1”

Users can add their own x- properties that are “application-specific” - just like with HTTP headers. They are provided primarily for future use.

用户可以添加自己的“特定于应用程序”的x属性，就像使用HTTP头一样。它们主要供将来使用。

Line 20: description: ″Sample connection profile for documentation topic″

第20行：描述：“文档主题的示例连接配置文件”

A short description of the connection profile. Try to make this helpful for the reader who might be seeing this for the first time!

连接配置文件的简短描述。试着让这对第一次看到它的读者有所帮助！

Line 25: version: ″1.0″

第25行：版本：“1.0”

The schema version for this connection profile. Currently only version 1.0 is supported, and it is not envisioned that this schema will change frequently.

此连接配置文件的架构版本。目前只支持版本1.0，不希望此模式经常更改。

Line 32: channels:

第32行：频道：

This is the first really important line. channels: identifies that what follows are all the channels that this connection profile describes. However, it is good practice to keep different channels in different connection profiles, especially if they are used independently of each other.

这是第一条非常重要的路线。通道:标识以下是此连接配置文件描述的所有通道。但是，在不同的连接配置文件中保留不同的通道是一种良好的做法，尤其是当它们彼此独立使用时。

Line 36: papernet:

第36行：papernet：

Details of papernet, the first channel in this connection profile, will follow.

以下是连接配置文件中的第一个通道Papernet的详细信息。

Line 41: orderers:

第41行：订购方：

Details of all the orderers for papernet follow. You can see in line 45 that the orderer for this channel is orderer1.magnetocorp.example.com. This is just a logical name; later in the connection profile (lines 134 - 147), there will be details of how to connect to this orderer. Notice that orderer2.digibank.example.com is not in this list; it makes sense that applications use their own organization’s orderers, rather than those from a different organization.

以下是 Papernet 的所有订购者的详细信息。您可以在第 45 行中看到，此频道的订购方是 order1.maverecorp.example.com。这只是一个逻辑名称；稍后在连接配置文件（第 134-147 行）中，将详细介绍如何连接到该医嘱者。请注意，order2.digibank.example.com 不在此列表中；应用程序使用自己组织的 order，而不是来自不同组织的 order，这是有意义的。

Line 49: peers:

第 49 行：对等体：

Details of all the peers for papernet will follow.

以下是 Papernet 所有同行的详细信息。

You can see three peers listed from MagnetoCorp: peer1.magnetocorp.example.com, peer2.magnetocorp.example.com and peer3.magnetocorp.example.com. It’s not necessary to list all the peers in MagnetoCorp, as has been done here. You can see only one peer listed from DigiBank: peer9.digibank.example.com; including this peer starts to imply that the endorsement policy requires MagnetoCorp and DigiBank to endorse transactions, as we’ll now confirm. It’s good practice to have multiple peers to avoid single points of failure.

您可以看到 Magnetorp 列出的三个对等机：peer1.magnercorp.example.com、peer2.magnercorp.example.com 和 peer3.magnercorp.example.com。没有必要像这里所做的那样列出磁电机公司的所有同行。您只能看到 Digibank 中列出的一个对等方：peer9.digibank.example.com；包括这个对等方开始暗示背书政策要求 Magnetorp 和 Digibank 对交易进行背书，正如我们现在确认的那样。拥有多个同龄人以避免单点失败是一种良好的实践。

Underneath each peer you can see four non-exclusive roles: endorsingPeer, chaincodeQuery, ledgerQuery and eventSource. See how peer1 and peer2 can perform all roles as they host papercontract. Contrast to peer3, which can only be used for notifications, or ledger queries that access the blockchain component of the ledger rather than the world state, and hence do not need to have smart contracts installed. Notice how peer9 should not be used for anything other than endorsement, because those roles are better served by MagnetoCorp peers.

在每个对等项下，您可以看到四个非独占角色：认可对等项、链码查询、LedgerQuery 和 EventSource。了解 Peer1 和 Peer2 如何在托管纸质合同时执行所有角色。与 Peer3 相反，Peer3 只能用于通知，或用于访问分类账的区块链组件而不是世界状态的分类账查询，因此不需要安装智能合约。请注意，Peer9 不应该被用于除背书以外的任何其他用途，因为磁电机公司的同行更好地服务于这些角色。

Again, see how the peers are described according to their logical names and their roles. Later in the profile, we’ll see the physical information for these peers.

同样，看看如何根据逻辑名称和角色来描述对等体。在稍后的配置文件中，我们将看到

这些对等机的物理信息。

Line 97: organizations:

第 97 行：组织：

Details of all the organizations will follow, for all channels. Note that these organizations are for all channels, even though papernet is currently the only one listed. That’s because organizations can be in multiple channels, and channels can have multiple organizations. Moreover, some application operations relate to organizations rather than channels. For example, an application can request notification from one or all peers within its organization, or all organizations within the network - using connection options. For this, there needs to be an organization to peer mapping, and this section provides it.

所有渠道的所有组织详情如下。请注意，这些组织适用于所有渠道，尽管目前仅列出 Papernet。这是因为组织可以有多个渠道，渠道可以有多个组织。此外，一些应用程序操作与组织相关，而不是与渠道相关。例如，应用程序可以使用连接选项从其组织中的一个或所有对等方或网络中的所有组织请求通知。为此，需要有一个组织到对等映射，本节提供了它。

Line 101: MagnetoCorp:

第 101 行：磁电机公司：

All peers that are considered part of MagnetoCorp are listed: peer1, peer2 and peer3. Likewise for Certificate Authorities. Again, note the logical name usages, the same as the channels: section; physical information will follow later in the profile.

所有被视为 Magnetorp 一部分的对等机都被列出：Peer1、Peer2 和 Peer3。同样适用于证书颁发机构。同样，请注意逻辑名称用法，与 channels:section 相同；物理信息将在稍后的概要文件中显示。

Line 121: DigiBank:

第 121 行：数字银行：

Only peer9 is listed as part of DigiBank, and no Certificate Authorities. That’s because these other peers and the DigiBank CA are not relevant for users of this connection profile.

只有 peer9 被列为 digibank 的一部分，没有证书颁发机构。这是因为这些其他对等端和 Digibank CA 与此连接配置文件的用户不相关。

Line 134: orderers:

第 134 行：订购方：

The physical information for orderers is now listed. As this connection profile only mentioned one orderer for papernet, you see orderer1.magnetocorp.example.com details listed. These include its IP address and port, and gRPC options that can override the defaults used when communicating with the orderer, if necessary. As with peers:, for high availability, specifying more than one orderer is a good idea.

现在将列出订购者的物理信息。由于这个连接配置文件只提到了 PaperNet 的一个订购者，所以您可以看到列出的 order1.maverecorp.example.com 详细信息。其中包括其 IP 地址和端口，以及 GRPC 选项，这些选项可以在必要时覆盖与订购方通信时使用的默认值。与同行一样，对于高可用性，指定多个订购者是一个好主意。

Line 152: peers:

第 152 行：同行：

The physical information for all previous peers is now listed. This connection profile has three peers for MagnetoCorp: peer1, peer2, and peer3; for DigiBank, a single peer peer9 has its information listed. For each peer, as with orderers, their IP address and port is listed, together with gRPC options that can override the defaults used when communicating with a particular peer, if necessary.

现在列出了以前所有对等机的物理信息。此连接配置文件有三个 Magnetorp 对等机：Peer1、Peer2 和 Peer3；对于 Digibank，单个对等机 Peer9 列出了其信息。对于每一个对等机，与订购者一样，它们的 IP 地址和端口都会被列出，并且 GRPC 选项可以覆盖与特定对等机通信时使用的默认值（如果需要）。

Line 194: certificateAuthorities:

第 194 行：证书授权：

The physical information for certificate authorities is now listed. The connection profile has a single CA listed for MagnetoCorp, ca1-magnetocorp, and its physical information follows. As well as IP details, the registrar information allows this CA to be used for Certificate Signing Requests (CSR). These are used to request new certificates for locally generated public/private key pairs.

现在列出了证书颁发机构的物理信息。连接配置文件有一个为 Magnetorp、CA1 Magnetorp 列出的 CA，其物理信息如下。以及 IP 详细信息，注册信息允许将此 CA 用于证书签名请求（CSR）。这些用于为本地生成的公钥/私钥对请求新证书。

Now you’ve understood a connection profile for MagnetoCorp, you might like to look at a corresponding profile for DigiBank. Locate where the profile is the same as MagnetoCorp’s, see where it’s similar, and finally where it’s different. Think about why these differences make sense for DigiBank applications.

现在您已经了解了磁电机公司的连接配置文件，您可能需要查看 Digibank 的相应配置文件。找到与 Magnetorp 相同的配置文件，查看其相似之处，最后找出不同之处。想想为什么这些差异对于 Digibank 应用程序是有意义的。

That’s everything you need to know about connection profiles. In summary, a connection profile defines sufficient channels, organizations, peers, orderers and certificate authorities for an application to configure a gateway. The gateway allows the application to focus on business logic rather than the details of the network topology.

这就是关于连接配置文件的所有信息。总之，连接配置文件为应用程序配置网关定义了足够的通道、组织、对等方、订购方和证书颁发机构。网关允许应用程序关注业务逻辑，而不是网络拓扑的细节。

Sample

样品

This file is reproduced inline from the GitHub commercial paper sample.

此文件是从 Github 商业纸样本中直接复制的。

```
1: ---
2: #
```

```
# [Required]. A connection profile contains information about a set of network
# components. It is typically used to configure gateway, allowing applications
# interact with a network channel without worrying about the underlying
# topology. A connection profile is normally created by an administrator who
# understands this topology.
#
name: "papernet.magnetocorp.profile.sample"
#
# [Optional]. Analogous to HTTP, properties with an "x-" prefix are deemed
# "application-specific", and ignored by the gateway. For example, property
# "x-type" with value "hlfv1" was originally used to identify a connection
# profile for Fabric 1.x rather than 0.x.
#
x-type: "hlfv1"
#
# [Required]. A short description of the connection profile
#
description: "Sample connection profile for documentation topic"
#
# [Required]. Connection profile schema version. Used by the gateway to
# interpret these data.
#
version: "1.0"
#
# [Optional]. A logical description of each network channel; its peer and
# orderer names and their roles within the channel. The physical details of
# these components (e.g. peer IP addresses) will be specified later in the
# profile; we focus first on the logical, and then the physical.
#
channels:
  #
  # [Optional]. papernet is the only channel in this connection profile
  #
  papernet:
    #
    # [Optional]. Channel orderers for PaperNet. Details of how to connect to
    # them is specified later, under the physical "orderers:" section
    #
    orderers:
    #
    # [Required]. Orderer logical name
```

```
#
  - orderer1.magnetocorp.example.com
#
# [Optional]. Peers and their roles
#
peers:
#
# [Required]. Peer logical name
#
  peer1.magnetocorp.example.com:
    #
    # [Optional]. Is this an endorsing peer? (It must have chaincode
    # installed.) Default: true
    #
    endorsingPeer: true
    #
    # [Optional]. Is this peer used for query? (It must have chaincode
    # installed.) Default: true
    #
    chaincodeQuery: true
    #
    # [Optional]. Is this peer used for non-chaincode queries? All peers
    # support these types of queries, which include queryBlock(),
    # queryTransaction(), etc. Default: true
    #
    ledgerQuery: true
    #
    # [Optional]. Is this peer used as an event hub? All peers can produce
    # events. Default: true
    #
    eventSource: true
  #
  peer2.magnetocorp.example.com:
    endorsingPeer: true
    chaincodeQuery: true
    ledgerQuery: true
    eventSource: true
  #
  peer3.magnetocorp.example.com:
    endorsingPeer: false
    chaincodeQuery: false
    ledgerQuery: true
    eventSource: true
```

```
      #
      peer9.digibank.example.com:
        endorsingPeer: true
        chaincodeQuery: false
        ledgerQuery: false
        eventSource: false
#
# [Required]. List of organizations for all channels. At least one organization
# is required.
#
organizations:
  #
  # [Required]. Organizational information for MagnetoCorp
  #
  MagnetoCorp:
    #
    # [Required]. The MSPID used to identify MagnetoCorp
    #
    mspid: MagnetoCorpMSP
    #
    # [Required]. The MagnetoCorp peers
    #
    peers:
      - peer1.magnetocorp.example.com
      - peer2.magnetocorp.example.com
      - peer3.magnetocorp.example.com
    #
    # [Optional]. Fabric-CA Certificate Authorities.
    #
    certificateAuthorities:
      - ca-magnetocorp
  #
  # [Optional]. Organizational information for DigiBank
  #
  DigiBank:
    #
    # [Required]. The MSPID used to identify DigiBank
    #
    mspid: DigiBankMSP
    #
    # [Required]. The DigiBank peers
    #
    peers:
      - peer9.digibank.example.com
#
# [Optional]. Orderer physical information, by orderer name
#
orderers:
  #
  # [Required]. Name of MagnetoCorp orderer
  #
  orderer1.magnetocorp.example.com:
    #
    # [Required]. This orderer's IP address
    #
    url: grpc://localhost:7050
    #
    # [Optional]. gRPC connection properties used for communication
    #
    grpcOptions:
      ssl-target-name-override: orderer1.magnetocorp.example.com
#
# [Required]. Peer physical information, by peer name. At least one peer is
# required.
#
peers:
  #
  # [Required]. First MagetoCorp peer physical properties
  #
  peer1.magnetocorp.example.com:
    #
    # [Required]. Peer's IP address
    #
    url: grpc://localhost:7151
    #
    # [Optional]. gRPC connection properties used for communication
    #
    grpcOptions:
      ssl-target-name-override: peer1.magnetocorp.example.com
      request-timeout: 120001
  #
  # [Optional]. Other MagnetoCorp peers
  #
  peer2.magnetocorp.example.com:
    url: grpc://localhost:7251
    grpcOptions:
      ssl-target-name-override: peer2.magnetocorp.example.com
      request-timeout: 120001
  #
  peer3.magnetocorp.example.com:
    url: grpc://localhost:7351
    grpcOptions:
      ssl-target-name-override: peer3.magnetocorp.example.com
      request-timeout: 120001
  #
  # [Required]. Digibank peer physical properties
  #
  peer9.digibank.example.com:
    url: grpc://localhost:7951
    grpcOptions:
      ssl-target-name-override: peer9.digibank.example.com
      request-timeout: 120001
#
# [Optional]. Fabric-CA Certificate Authority physical information, by name.
# This information can be used to (e.g.) enroll new users. Communication is via
# REST, hence options relate to HTTP rather than gRPC.
#
certificateAuthorities:
  #
  # [Required]. MagnetoCorp CA
  #
  ca1-magnetocorp:
    #
    # [Required]. CA IP address
    #
    url: http://localhost:7054
    #
    # [Optioanl]. HTTP connection properties used for communication
    #
    httpOptions:
      verify: false
    #
    # [Optional]. Fabric-CA supports Certificate Signing Requests (CSRs). A
    # registrar is needed to enroll new users.
    #
    registrar:
      - enrollId: admin
        enrollSecret: adminpw
    #
    # [Optional]. The name of the CA.
    #
    caName: ca-magnetocorp
```

7、Connection Options

7、连接选项

Audience: Architects, administrators, application and smart contract developers

受众：架构师、管理员、应用程序和智能合约开发人员

Connection options are used in conjunction with a connection profile to control precisely how a gateway interacts with a network. Using a gateway allows an application to focus on business logic rather than network topology.

连接选项与连接配置文件一起使用，以精确控制网关与网络的交互方式。使用网关允许应用程序关注业务逻辑，而不是网络拓扑。

In this topic, we’re going to cover:

在本主题中，我们将介绍：

Why connection options are important

为什么连接选项很重要

How an application uses connection options

应用程序如何使用连接选项

What each connection option does

每个连接选项的作用

When to use a particular connection option

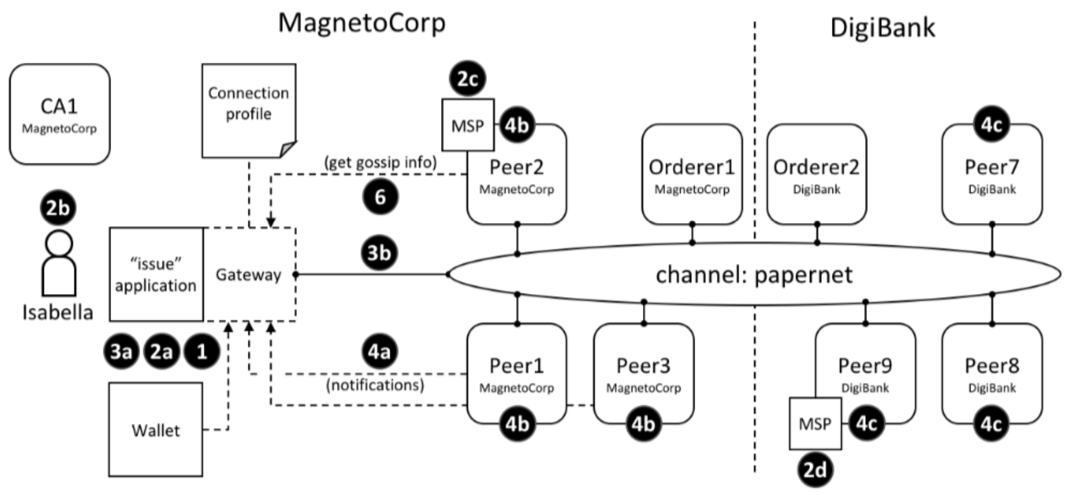
何时使用特定连接选项

Scenario

脚本

A connection option specifies a particular aspect of a gateway’s behaviour. Gateways are important for many reasons, the primary being to allow an application to focus on business logic and smart contracts, while it manages interactions with the many components of a network.

连接选项指定网关行为的特定方面。网关之所以重要，有许多原因，主要是允许应用程序专注于业务逻辑和智能合约，同时管理与网络许多组件的交互。



The different interaction points where connection options control behaviour. These options are explained fully in the text.

连接选项控制行为的不同交互点。这些选项在文本中进行了详细说明。

One example of a connection option might be to specify that the gateway used by the issue application should use identity Isabella to submit transactions to the papernet network. Another might be that a gateway should wait for all three nodes from MagnetoCorp to confirm a transaction has been committed returning control. Connection options allow applications to specify the precise behaviour of a gateway’s interaction with the network. Without a gateway, applications need to do a lot more work; gateways save you time, make your application more readable, and less error prone.

连接选项的一个示例可能是指定 Issue 应用程序使用的网关应使用 Identity Isabella 向 PaperNet 网络提交事务。另一种可能是，网关应该等待磁电机公司的所有三个节点确认已提交事务并返回控制权。连接选项允许应用程序指定网关与网络交互的精确行为。如果没有网关，应用程序需要做更多的工作；网关可以节省您的时间，使应用程序更具可读性，并且不易出错。

Usage

用法

We’ll describe the full set of connection options available to an application in a moment; let’s first see see how they are specified by the sample MagnetoCorp issue application:

我们稍后将描述应用程序可用的完整连接选项集；让我们先看看示例 Magnetorp 发行应用程序如何指定这些选项：

```
const userName = 'User1@org1.example.com';
const wallet = new FileSystemWallet('../identity/user/isabella/wallet');
const connectionOptions = {
  identity: userName,
  wallet: wallet,
  eventHandlerOptions: {
    commitTimeout: 100,
    strategy: EventStrategies.MSPID_SCOPE_ANYFORTX
    }
  };
await gateway.connect(connectionProfile, connectionOptions);
```

See how the identity and wallet options are simple properties of the connectionOptions object. They have values userName and wallet respectively, which were set earlier in the code. Contrast these options with the eventHandlerOptions option which is an object in its own right. It has two properties: commitTimeout: 100 (measured in seconds) and strategy: EventStrategies.MSPID_SCOPE_ANYFORTX.

查看标识和钱包选项如何是 ConnectionOptions 对象的简单属性。它们分别具有值 username 和 wallet，这些值在代码前面设置。将这些选项与 EventHandlerOptions 选项进行对比，后者本身就是一个对象。它有两个属性：commitTimeout:100（以秒为单位）和 strategy:eventstreategies.mspid_scope_anyfortx。

See how connectionOptions is passed to a gateway as a complement to connectionProfile; the network is identified by the connection profile and the options specify precisely how the gateway should interact with it. Let’s now look at the available options.

请参阅 ConnectionOptions 如何作为 ConnectionProfile 的补充传递到网关；网络由连接配置文件标识，并且这些选项精确地指定网关应如何与之交互。现在我们来看看可用的选项。

Options

选项

Here’s a list of the available options and what they do.

以下是可用选项的列表以及它们的作用。

wallet identifies the wallet that will be used by the gateway on behalf of the application. See interaction 1; the wallet is specified by the application, but it’s actually the gateway that retrieves identities from it.

Wallet 标识网关代表应用程序使用的钱包。参见交互 1；钱包是由应用程序指定的，但它实际上是从中检索身份的网关。

A wallet must be specified; the most important decision is the type of wallet to use, whether that’s file system, in-memory, HSM or database.

必须指定钱包；最重要的决定是要使用的钱包类型，无论是文件系统、内存、HSM 还是数据库。

identity is the user identity that the application will use from wallet. See interaction 2a; the user identity is specified by the application and represents the user of the application, Isabella, 2b. The identity is actually retrieved by the gateway.

Identity 是应用程序将从 Wallet 使用的用户标识。参见交互 2a；用户标识由应用程序指定并表示应用程序的用户 isabella，2b。该标识实际上由网关检索。

In our example, Isabella’s identity will be used by different MSPs (2c, 2d) to identify her as being from MagnetoCorp, and having a particular role within it. These two facts will correspondingly determine her permission over resources, such as being able to read and write the ledger, for example.

在我们的示例中，不同的 MSP（2c，2d）将使用 Isabella 的身份来识别她来自磁电机

公司，并在其中扮演特定的角色。这两个事实将相应地决定她对资源的权限，例如能够读写分类帐。

A user identity must be specified. As you can see, this identity is fundamental to the idea that Hyperledger Fabric is a permissioned network - all actors have an identity, including applications, peers and orderers, which determines their control over resources. You can read more about this idea in the membership services topic.

必须指定用户标识。正如您所看到的，这种身份是 Hyperledger 结构是一个被许可的网络这一理念的基础——所有参与者都有一个身份，包括应用程序、对等方和订购方，这决定了他们对资源的控制。您可以在会员服务主题中了解更多关于这个想法的信息。

clientTlsIdentity is the identity that is retrieved from a wallet (3a) and used for secure communications (3b) between the gateway and different channel components, such as peers and orderers.

clienttlsidentity 是从钱包（3a）中检索并用于网关和不同通道组件（如对等方和订购方）之间的安全通信（3b）的标识。

Note that this identity is different to the user identity. Even though clientTlsIdentity is important for secure communications, it is not as foundational as the user identity because its scope does not extend beyond secure network communications.

请注意，此标识与用户标识不同。尽管 clientlsidentity 对于安全通信很重要，但它并不像用户标识那样基础，因为它的作用域不超过安全网络通信。

clientTlsIdentity is optional. You are advised to set it in production environments. You should always use a different clientTlsIdentity to identity because these identities have very different meanings and lifecycles. For example, if your clientTlsIdentity was compromised, then so would your identity; it’s more secure to keep them separate.

clientlsidentity 是可选的。建议您在生产环境中进行设置。您应该始终使用不同的 clientlsidentity 来标识，因为这些标识具有非常不同的含义和生命周期。例如，如果你的客户身份受到影响，那么你的身份也会受到影响；让他们分开更安全。

eventHandlerOptions.commitTimeout is optional. It specifies, in seconds, the maximum amount of time the gateway should wait for a transaction to be committed by any peer (4a) before returning control to the application. The set of peers to use for notification is determined by the eventHandlerOptions.strategy option. If a commitTimeout is not specified, the gateway will use a timeout of 300 seconds.

eventHandlerOptions.commitTimeout 是可选的。它以秒为单位指定在将控制权返回到应用程序之前，网关应等待任何对等方（4a）提交事务的最长时间。用于通知的对等机集由 eventhandlerOptions.strategy 选项确定。如果未指定 commitTimeout，则网关将使用 300 秒的超时。

eventHandlerOptions.strategy is optional. It identifies the set of peers that a gateway should use to listen for notification that a transaction has been committed. For example, whether to listen for a single peer, or all peers, from its organization. It can take one of the following values:

eventhandlerOptions.strategy 是可选的。它标识一组对等机，网关应使用这些对等

机来侦听已提交事务的通知。例如，是否从其组织中侦听单个对等方或所有对等方。它可以采用以下值之一：

EventStrategies.MSPID_SCOPE_ANYFORTX Listen for any peer within the user’s organization. In our example, see interaction points 4b; any of peer 1, peer 2 or peer 3 from MagnetoCorp can notify the gateway.

eventstrigies.mspid_scope_anyfortx 侦听用户组织内的任何对等方。在我们的示例中，请参见交互点 4b；任何来自 Magnetorp 的对等 1、对等 2 或对等 3 都可以通知网关。

EventStrategies.MSPID_SCOPE_ALLFORTX This is the default value. Listen for all peers within the user’s organization. In our example peer, see interaction point 4b. All peers from MagnetoCorp must all have notified the gateway; peer 1, peer 2 and peer 3. Peers are only counted if they are known/discovered and available; peers that are stopped or have failed are not included.

eventstrigies.mspid_scope_allfortx 这是默认值。倾听用户组织中的所有对等方。在我们的示例中，对等机，请参见交互点 4b。Magnetorp 的所有对等机都必须通知网关；对等机 1、对等机 2 和对等机 3。只有在已知/发现和可用的情况下才计算对等；不包括已停止或已失败的对等。

EventStrategies.NETWORK_SCOPE_ANYFORTX Listen for any peer within the entire network channel. In our example, see interaction points 4b and 4c; any of peer 1-3 from MagnetoCorp or peer 7-9 of DigiBank can notify the gateway.

eventstrigies.network_scope_anyfortx 在整个网络通道中监听任何对等点。在我们的示例中，请参见交互点 4b 和 4c；来自 Magnetorp 的任何对等 1-3 或 Digibank 的对等 7-9 都可以通知网关。

EventStrategies.NETWORK_SCOPE_ALLFORTX Listen for all peers within the entire network channel. In our example, see interaction points 4b and 4c. All peers from MagnetoCorp and DigiBank must notify the gateway; peers 1-3 and peers 7-9. Peers are only counted if they are known/discovered and available; peers that are stopped or have failed are not included.

eventstrigies.network_scope_allfortx 侦听整个网络通道中的所有对等点。在我们的示例中，请参见交互点 4b 和 4c。Magnetorp 和 Digibank 的所有对等方都必须通知网关；对等方 1-3 和对等方 7-9。只有在已知/发现和可用的情况下才计算对等；不包括已停止或已失败的对等。

<PluginEventHandlerFunction> The name of a user-defined event handler. This allows a user to define their own logic for event handling. See how to define a plugin event handler, and examine a sample handler.

<pluginEventHandlerFunction>用户定义的事件处理程序的名称。这允许用户为事件处理定义自己的逻辑。请参阅如何定义插件事件处理程序，并检查示例处理程序。

A user-defined event handler is only necessary if you have very specific event handling requirements; in general, one of the built-in event strategies will be sufficient. An example of a user-defined event handler might be to wait for more than half the peers in an organization to confirm a transaction has been committed.

只有当您有非常具体的事件处理需求时，用户定义的事件处理程序才是必需的；一般来说，一个内置的事件策略就足够了。用户定义的事件处理程序的一个例子可能是等待组织中超过一半的对等方确认事务已提交。

If you do specify a user-defined event handler, it does not affect your application logic; it is quite separate from it. The handler is called by the SDK during processing; it decides when to call it, and uses its results to select which peers to use for event notification. The application receives control when the SDK has finished its processing.

如果您确实指定了一个用户定义的事件处理程序，它不会影响您的应用程序逻辑；它与应用程序逻辑是完全独立的。该处理程序在处理过程中由 sdk 调用；它决定何时调用它，并使用其结果来选择用于事件通知的对等方。当 sdk 完成处理后，应用程序接收控制。

If a user-defined event handler is not specified then the default values for EventStrategies are used.

如果未指定用户定义的事件处理程序，则使用 EventStrategies 的默认值。

discovery.enabled is optional and has possible values true or false. The default is true. It determines whether the gateway uses service discovery to augment the network topology specified in the connection profile. See interaction point 6; peer’s gossip information used by the gateway.

discovery.enabled 是可选的，可能的值为 true 或 false。默认值为 true。它确定网关是否使用服务发现来扩充连接配置文件中指定的网络拓扑。参见交互点 6；网关使用的对等方的八卦信息。

This value will be overridden by the INITIALIIZE-WITH-DISCOVERY environment variable, which can be set to true or false.

此值将被 initialize-with-discovery 环境变量覆盖，该变量可以设置为 true 或 false。

discovery.asLocalhost is optional and has possible values true or false. The default is true. It determines whether IP addresses found during service discovery are translated from the docker network to the local host.

discovery.aslocalhost 是可选的，其值可能为 true 或 false。默认值为 true。它确定在服务发现期间找到的 IP 地址是否从 Docker 网络转换到本地主机。

Typically developers will write applications that use docker containers for their network components such as peers, orderers and CAs, but that do not run in docker containers themselves. This is why true is the default; in production environments, applications will likely run in docker containers in the same manner as network components and therefore address translation is not required. In this case, applications should either explicitly specify false or use the environment variable override.

通常，开发人员会编写应用程序，这些应用程序将 Docker 容器用于其网络组件，如对等端、订购方和 CA，但这些应用程序本身并不在 Docker 容器中运行。这就是默认值为 true 的原因；在生产环境中，应用程序可能以与网络组件相同的方式在 Docker 容器中运行，因此不需要进行地址转换。在这种情况下，应用程序应该显式地指定 false 或使用环境变量 override。

This value will be overridden by the DISCOVERY-AS-LOCALHOST environment variable, which can be set to true or false.

这个值将被 discovery-as-localhost 环境变量覆盖，该变量可以设置为 true 或 false。

Considerations

考虑事项

The following list of considerations is helpful when deciding how to choose connection options.

以下注意事项列表在决定如何选择连接选项时很有用。

eventHandlerOptions.commitTimeout and eventHandlerOptions.strategy work together. For example, commitTimeout: 100 and strategy: EventStrategies.MSPID_SCOPE_ANYFORTX means that the gateway will wait for up to 100 seconds for any peer to confirm a transaction has been committed. In contrast, specifying strategy: EventStrategies.NETWORK_SCOPE_ALLFORTX means that the gateway will wait up to 100 seconds for all peers in all organizations.

eventhandlerOptions.commitTimeout 和 eventhandlerOptions.strategy 一起工作。例如，commitTimeout:100 和 strategy:EventStrategies.mspid_scope_anyfortx 意味着网关将等待最多 100 秒，以便任何对等方确认已提交事务。相反，指定 strategy:eventstridges.network_scope_allfortx 意味着网关将为所有组织中的所有对等方等待 100 秒。

The default value of eventHandlerOptions.strategy: EventStrategies.MSPID_SCOPE_ALLFORTX will wait for all peers in the application's organization to commit the transaction. This is a good default because applications can be sure that all their peers have an up-to-date copy of the ledger, minimizing concurrency issues.

eventHandlerOptions.strategy 的默认值：eventstrigies.mspid_scope_allfortx 将等待应用程序组织中的所有对等方提交事务。这是一个很好的默认值，因为应用程序可以确保他们的所有同行都有一个最新的分类账副本，从而最小化并发问题。

However, as the number of peers in an organization grows, it becomes a little unnecessary to wait for all peers, in which case using a pluggable event handler can provide a more efficient strategy. For example the same set of peers could be used to submit transactions and listen for notifications, on the safe assumption that consensus will keep all ledgers synchronized.

但是，随着组织中对等方数量的增加，有点不需要等待所有对等方，在这种情况下，使用可插拔事件处理程序可以提供更有效的策略。例如，在一致同意将保持所有分类账同步的安全假设下，可以使用同一组对等方提交交易并听取通知。

Service discovery requires clientTlsIdentity to be set. That's because the peers exchanging information with an application need to be confident that they are exchanging information with entities they trust. If clientTlsIdentity is not set, then discovery will not be obeyed, regardless of whether or not it is set.

服务发现要求设置 clienttlsidentity。这是因为对等方与应用程序交换信息时需要确信他们正在与信任的实体交换信息。如果没有设置 clientlsidentity，那么无论是否设置了发现，都不会遵守发现。

Although applications can set connection options when they connect to the gateway, it can be necessary for these options to be overridden by an administrator. That's because options relate to network interactions, which can vary over time. For example, an administrator trying to understand the effect of using service discovery on network performance.

尽管应用程序可以在连接到网关时设置连接选项，但管理员可能需要覆盖这些选项。这是因为选项与网络交互有关，而网络交互会随时间而变化。例如，管理员试图了解使用服务发现对网络性能的影响。

A good approach is to define application overrides in a configuration file which is read by the application when it configures its connection to the gateway.

一种好的方法是在配置文件中定义应用程序覆盖，当应用程序配置其与网关的连接时，该配置文件会读取该文件。

Because the discovery options enabled and asLocalHost are most frequently required to be overridden by administrators, the environment variables INITIALIIZE-WITH-DISCOVERY and DISCOVERY-AS-LOCALHOST are provided for convenience. The administrator should set these in the production runtime environment of the application, which will most likely be a docker container.

由于启用的发现选项和 as localhost 通常需要由管理员重写，因此为方便起见，提供了环境变量 initialize-with-discovery 和 discovery-as-localhost。管理员应该在应用程序的生产运行时环境中设置这些，很可能是 Docker 容器。

8、Wallet

8、钱包

Audience: Architects, application and smart contract developers

受众：架构师、应用程序和智能合约开发人员

A wallet contains a set of user identities. An application run by a user selects one of these identities when it connects to a channel. Access rights to channel resources, such as the ledger, are determined using this identity in combination with an MSP.

钱包包含一组用户身份信息。用户运行的应用程序在连接到通道时选择其中一个标识。对渠道资源（如分类账）的访问权限是通过使用此标识和 MSP 来确定的。

In this topic, we’re going to cover:

在本主题中，我们将介绍：

Why wallets are important

为什么钱包很重要

How wallets are organized

钱包的组织方式

Different types of wallet

不同类型的钱包

Wallet operations

钱包操作

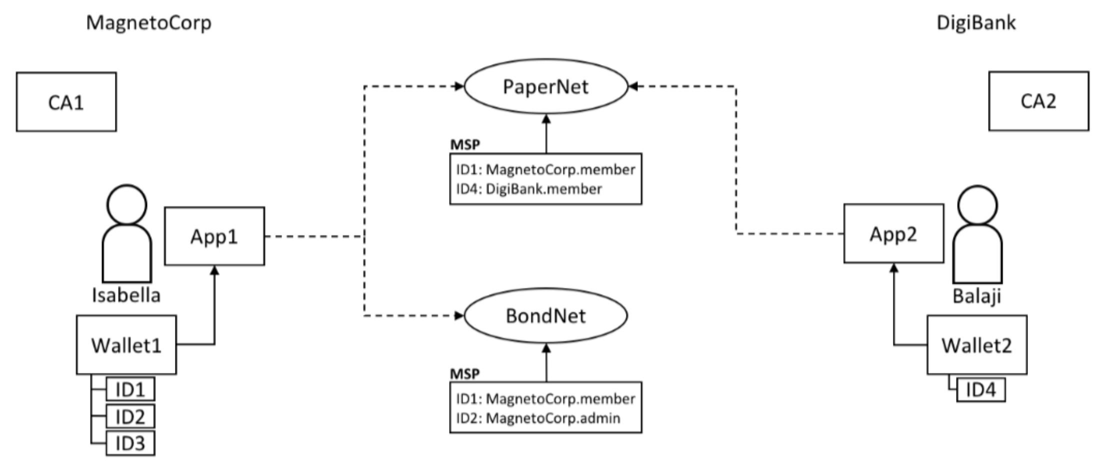
Scenario

脚本

When an application connects to a network channel such as PaperNet, it selects a user identity to do so, for example ID1. The channel MSPs associate ID1 with a role within a particular organization, and this role will ultimately determine the application’s rights over channel resources. For example, ID1 might identify a user as a member of the MagnetoCorp organization who can read and write to the ledger, whereas ID2 might identify an administrator in MagnetoCorp who can add a new organization to a consortium.

当应用程序连接到网络通道（如 Papernet）时，它会选择用户标识进行连接，例如 ID1。

通道 MSP 将 ID1 与特定组织中的角色关联，并且此角色最终将确定应用程序对通道资源的权限。例如，ID1 可以将用户标识为磁电机公司组织的成员，该组织可以读写分类账，而 ID2 可以标识磁电机公司的管理员，该管理员可以将新组织添加到联合体中。



Two users, Isabella and Balaji have wallets containing different identities they can use to connect to different network channels, PaperNet and BondNet.

Isabella 和 Balaji 这两个用户的钱包里有不同的身份，他们可以用来连接不同的网络渠道、纸网和邦德网。

Consider the example of two users; Isabella from MagnetoCorp and Balaji from DigiBank. Isabella is going to use App 1 to invoke a smart contract in PaperNet and a different smart contract in BondNet. Similarly, Balaji is going to use App 2 to invoke smart contracts, but only in PaperNet. (It’s very easy for applications to access multiple networks and multiple smart contracts within them.)

以两个用户为例:Magnetorp 的 Isabella 和 Digibank 的 Balaji。Isabella 将使用 App1 在 Papernet 中调用智能合约，在 BondNet 中调用不同的智能合约。同样，Balaji 也将使用 App2 来调用智能合约，但只在 Papernet 中使用。(应用程序很容易访问多个网络和其中的多个智能合约。)

See how:

看看如何：

MagnetoCorp uses CA1 to issue identities and DigiBank uses CA2 to issue identities. These identities are stored in user wallets.

Magnetorp 使用 CA1 发布身份信息，Digibank 使用 CA2 发布身份信息。这些身份信息存储在用户钱包中。

Balaji’s wallet holds a single identity, ID4 issued by CA2. Isabella’s wallet has many identities, ID1, ID2 and ID3, issued by CA1. Wallets can hold multiple identities for a single user, and each identity can be issued by a different CA.

巴拉吉的钱包里有一个身份证，由 CA2 签发的 ID4。伊莎贝拉的钱包有许多身份，ID1，ID2 和 ID3，由 CA1 发行。钱包可以为单个用户保存多个身份信息，每个身份信息都可以由不同的 CA 颁发。

Both Isabella and Balaji connect to PaperNet, and its MSPs determine that Isabella is a member of the MagnetoCorp organization, and Balaji is a member of the DigiBank organization, because of the respective CAs that issued their identities. (It is possible for an organization to use multiple CAs, and for a single CA to support multiple organizations.)

Isabella 和 Balaji 都连接到了 Papernet，其 MSP 确定 Isabella 是 Magnetorp 组织的成员，而 Balaji 是 Digibank 组织的成员，因为发布了各自身份的 CA。（一个组织可以使用多个 CA，一个 CA 可以支持多个组织。）

Isabella can use ID1 to connect to both PaperNet and BondNet. In both cases, when Isabella uses this identity, she is recognized as a member of MangetoCorp.

Isabella 可以使用 ID1 连接到 PaperNet 和 BondNet。在这两种情况下，当伊莎贝拉使用这个身份时，她被公认为 MangetoCorp 的成员。

Isabella can use ID2 to connect to BondNet, in which case she is identified as an administrator of MagnetoCorp. This gives Isabella two very different privileges: ID1 identifies her as a simple member of MagnetoCorp who can read and write to the BondNet ledger, whereas ID2 identities her as a MagnetoCorp administrator who can add a new organization to BondNet.

Isabella 可以使用 ID2 连接到 BondNet，在这种情况下，她被认定为 Magnetorp 的管理员。这给了伊莎贝拉两个非常不同的特权：ID1 将她认定为磁电机公司的简单成员，磁电机公司可以读写邦德网分类账，而 ID2 将她认定为磁电机公司的管理员，磁电机公司可以向邦德网添加新的组织。

Balaji cannot connect to BondNet with ID4. If he tried to connect, ID4 would not be recognized as belonging to DigiBank because CA2 is not known to BondNet’s MSP.

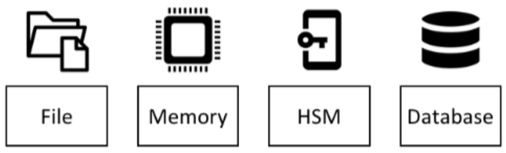
Balaji 无法使用 ID4 连接到 BondNet。如果他尝试连接，ID4 将不会被认为是属于 Digibank，因为 BondNet 的 MSP 不知道 CA2。

Types

类型

There are different types of wallets according to where they store their identities:

根据钱包的存储位置，钱包有不同的类型：



The four different types of wallet: File system, In-memory, Hardware Security Module (HSM) and CouchDB.

四种不同类型的钱包：文件系统、内存、硬件安全模块（HSM）和 CouchDB。

FileSystem: This is the most common place to store wallets; file systems are pervasive, easy to understand, and can be network mounted. They are a good default choice for wallets.

文件系统：这是最常见的存储钱包的地方；文件系统无处不在，易于理解，并且可以通过网络安装。它们是钱包的默认选择。

Use the FileSystemWallet class to manage file system wallets.

使用 FileSystemWallet 类管理文件系统钱包。

In-memory: A wallet in application storage. Use this type of wallet when your application is running in a constrained environment without access to a file system; typically a web browser. It’s worth remembering that this type of wallet is volatile; identities will be lost after the application ends normally or crashes.

内存:应用程序存储器中的钱包。当应用程序在没有访问文件系统(通常是 Web 浏览器)的受限环境中运行时，请使用此类型的钱包。值得记住的是，这种类型的钱包是不稳定的;应用程序正常结束或崩溃后，身份将丢失。

Use the InMemoryWallet class to manage in-memory wallets.

使用 InMemoryWallet 类管理内存中的钱包。

Hardware Security Module: A wallet stored in an HSM. This ultra-secure, tamper-proof device stores digital identity information, particularly private keys. HSMs can be locally attached to your computer or network accessible. Most HSMs provide the ability to perform on-board encryption with private keys, such that the private key never leave the HSM.

硬件安全模块：存储在 HSM 中的钱包。这种超安全、防篡改的设备存储数字身份信息，特别是私钥。HSMS 可以本地连接到您的计算机或网络。大多数 HSM 都提供了使用私钥执行车载加密的能力，这样私钥就永远不会离开 HSM。

Currently you should use the FileSystemWallet class in combination with the HSMWalletMixin class to manage HSM wallets.

目前，您应该结合使用 filesystemwallet 类和 hsmwalletmixin 类来管理 hsm 钱包。

CouchDB: A wallet stored in Couch DB. This is the rarest form of wallet storage, but for those users who want to use the database back-up and restore mechanisms, CouchDB wallets can provide a useful option to simplify disaster recovery.

CouchDB：放在沙发数据库中的钱包。这是最罕见的钱包存储形式，但是对于那些想要使用数据库备份和恢复机制的用户来说，CouchDB 钱包可以提供一个有用的选项来简化灾难恢复。

Use the CouchDBWallet class to manage CouchDB wallets.

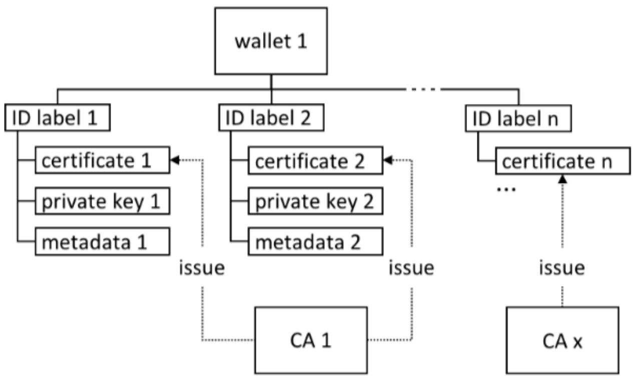
使用 couchdbwallet 类管理 couchdb 钱包。

Structure

结构

A single wallet can hold multiple identities, each issued by a particular Certificate Authority. Each identity has a standard structure comprising a descriptive label, an X.509 certificate containing a public key, a private key, and some Fabric-specific metadata. Different wallet types map this structure appropriately to their storage mechanism.

一个钱包可以保存多个身份信息，每个身份信息由特定的证书颁发机构颁发。每个标识都有一个标准结构，包括描述性标签、包含公钥的 X.509 证书、私钥和一些特定于结构的元数据。不同的钱包类型将这种结构适当地映射到它们的存储机制。



A Fabric wallet can hold multiple identities with certificates issued by a different Certificate Authority. Identities comprise certificate, private key and Fabric metadata.

一个 Fabric 钱包可以持有由不同证书颁发机构颁发的证书的多个身份。标识包括证书、私钥和结构元数据。

There’ s a couple of key class methods that make it easy to manage wallets and identities:

有几个关键的类方法可以很容易地管理钱包和身份：

```
const identity = X509WalletMixin.createIdentity('Org1MSP', certificate, key);
await wallet.import(identityLabel, identity);
```

See how the X509WalletMixin.createIdentity() method creates an identity that has metadata Org1MSP, a certificate and a private key. See how wallet.import() adds this identity to the wallet with a particular identityLabel.

请参阅 x509walletmixin.createidentity（）方法如何创建具有元数据 org1Msp、证书和私钥的标识。请参阅 wallet.import()如何将此标识添加到具有特定标识标签的钱包中。

The Gateway class only requires the mspid metadata to be set for an identity - Org1MSP in the above example. It currently uses this value to identify particular peers from a connection profile, for example when a specific notification strategy is requested. In the DigiBank gateway file networkConnection.yaml, see how Org1MSP notifications will be associated with peer0.org1.example.com:

在上面的示例中，网关类只需要为标识-org1Msp 设置 mspid 元数据。它当前使用此值从连接配置文件中标识特定对等方，例如请求特定通知策略时。在 digibank gateway 文件 networkconnection.yaml 中，查看 org1msp 通知如何与 peer0.org1.example.com 关联：

```
organizations:
Org1:
mspid: Org1MSP
peers:
- peer0.org1.example.com
```

You really don’ t need to worry about the internal structure of the different wallet types, but if you’ re interested, navigate to a user identity folder in the commercial paper sample:

您确实不需要担心不同钱包类型的内部结构，但是如果您感兴趣，请导航到商业票据示例中的用户标识文件夹：

```
magnetocorp/identity/user/isabella/
                                    wallet/
                                           User1@org1.example.com/
                                                         User1@org.example.com
                                                         c75bd6911aca8089...-priv
                                                         c75bd6911aca8089...-pub
```

You can examine these files, but as discussed, it’s easier to use the SDK to manipulate these data.

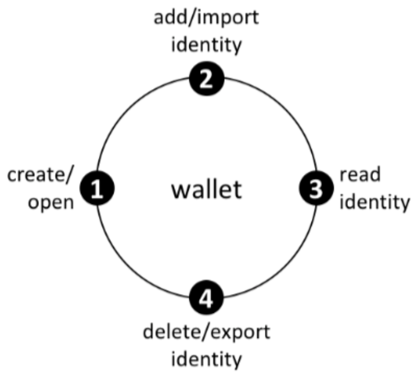
您可以检查这些文件，但如前所述，使用 SDK 来操作这些数据更容易。

Operations

操作

The different wallet classes are derived from a common Wallet base class which provides a standard set of APIs to manage identities. It means that applications can be made independent of the underlying wallet storage mechanism; for example, File system and HSM wallets are handled in a very similar way.

不同的钱包类派生自一个通用的钱包基本类，该类提供了一组用于管理身份的标准 API。这意味着应用程序可以独立于底层钱包存储机制；例如，文件系统和 HSM 钱包的处理方式非常相似。



Wallets follow a lifecycle: they can be created or opened, and identities can be read, added, deleted and exported.

钱包遵循一个生命周期：它们可以被创建或打开，标识可以被读取、添加、删除和导出。

An application can use a wallet according to a simple lifecycle. Wallets can be opened or created, and subsequently identities can be added, read, updated, deleted and exported. Spend a little time on the different Wallet methods in the JSDOC to see how they work; the commercial paper tutorial provides a nice example in addToWallet.js:

应用程序可以根据简单的生命周期使用钱包。可以打开或创建钱包，然后可以添加、读取、更新、删除和导出标识。在 jsdoc 中花点时间研究不同的 wallet 方法，看看它们是如何工作的；商业论文教程在 addtowallet.js 中提供了一个很好的示例：

```
const wallet = new FileSystemWallet('../identity/user/isabella/wallet');
const cert = fs.readFileSync(path.join(credPath, '.../User1@org1.example.com-cert.pem')).toString();
const key = fs.readFileSync(path.join(credPath, '.../_sk')).toString();
const identityLabel = 'User1@org1.example.com';
const identity = X509WalletMixin.createIdentity('Org1MSP', cert, key);
await wallet.import(identityLabel, identity);
```

Notice how:

注意如何：

When the program is first run, a wallet is created on the local file system at .../isabella/wallet.

当程序首次运行时，在本地文件系统上创建一个钱包，地址为…/isabella/wallet。

a certificate cert and private key are loaded from the file system.

从文件系统加载证书证书和私钥。

a new identity is created with cert, key and Org1MSP using X509WalletMixin.createIdentity().

使用 cert、key 和 org1Msp 使用 x509walletmixin.createIdentity（）创建新标识。

the new identity is imported to the wallet with wallet.import() with a label User1@org1.example.com.

新标识将通过 wallet.import（）导入钱包，标签为 user1@org1.example.com。

That's everything you need to know about wallets. You've seen how they hold identities that are used by applications on behalf of users to access Fabric network resources. There are different types of wallets available depending on your application and security needs, and a simple set of APIs to help applications manage wallets and the identities within them.

这就是你需要知道的关于钱包的一切。您已经看到了它们如何保存应用程序代表用户访问结构网络资源所使用的标识。根据您的应用程序和安全需求，有不同类型的钱包可用，还有一组简单的 API 来帮助应用程序管理钱包及其内部的标识。

9、Gateway

9、网关

Audience: Architects, application and smart contract developers

受众：架构师、应用程序和智能合约开发人员

A gateway manages the network interactions on behalf of an application, allowing it to focus on business logic. Applications connect to a gateway and then all subsequent interactions are managed using that gateway's configuration.

网关代表应用程序管理网络交互，使其能够专注于业务逻辑。应用程序连接到网关，然后使用该网关的配置管理所有后续交互。

In this topic, we're going to cover:

在本主题中，我们将介绍：

Why gateways are important

为什么网关很重要

How applications use a gateway

应用程序如何使用网关

How to define a static gateway

如何定义静态网关

How to define a dynamic gateway for service discovery

如何定义服务发现的动态网关

Using multiple gateways

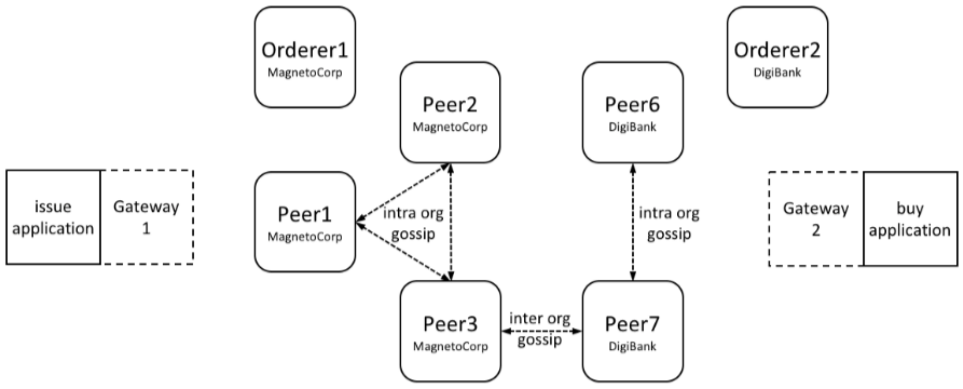
使用多个网关

Scenario

脚本

A Hyperledger Fabric network channel can constantly change. The peer, orderer and CA components, contributed by the different organizations in the network, will come and go. Reasons for this include increased or reduced business demand, and both planned and unplanned outages. A gateway relieves an application of this burden, allowing it to focus on the business problem it is trying to solve.

一个超级账本结构的网络渠道可以不断变化。网络中不同组织提供的对等、订购者和 CA 组件将来去自如。原因包括增加或减少业务需求，以及计划内和计划外停机。网关减轻了应用程序的这种负担，使其能够将重点放在要解决的业务问题上。



A MagnetoCorp and DigiBank applications (issue and buy) delegate their respective network interactions to their gateways. Each gateway understands the network channel topology comprising the multiple peers and orderers of two organizations MagnetoCorp and DigiBank, leaving applications to focus on business logic. Peers can talk to each other both within and across organizations using the gossip protocol.

Magnetorp 和 Digibank 应用程序（发行和购买）将各自的网络交互委托给各自的网关。每个网关都了解由两个组织的多个对等方和订购方组成的网络通道拓扑，Magnetorp 和 Digibank，使应用程序专注于业务逻辑。同事们可以使用八卦协议在组织内部和跨组织之间进行交谈。

A gateway can be used by an application in two different ways:

应用程序可以通过两种不同的方式使用网关：

Static: The gateway configuration is completely defined in a connection profile. All the peers, orderers and CAs available to an application are statically defined in the connection profile used to configure the gateway. For peers, this includes their role as an endorsing peer or event notification hub, for example. You can read more about these roles in the connection profile topic.

静态：网关配置完全在连接配置文件中定义。在用于配置网关的连接配置文件中，静态定义了应用程序可用的所有对等端、订购方和 CA。例如，对于对等方，这包括他们作为认可对等方或事件通知中心的角色。您可以在连接配置文件主题中了解有关这些角色的更多信息。

The SDK will use this static topology, in conjunction with gateway connection options, to manage the transaction submission and notification processes. The connection profile must contain enough of the network topology to allow a gateway to interact with the network on behalf of the application; this includes the network channels, organizations, orderers, peers and their roles.

SDK 将结合网关连接选项使用此静态拓扑来管理事务提交和通知过程。连接配置文件必须包含足够的网络拓扑，以允许网关代表应用程序与网络交互；这包括网络通道、组织、订购者、对等方及其角色。

Dynamic: The gateway configuration is minimally defined in a connection profile. Typically, one or two peers from the application’s organization are specified, and they use service discovery to discover the available network topology. This includes peers, orderers, channels, instantiated smart contracts and their endorsement policies. (In production environments, a gateway configuration should specify at least two peers for availability.)

动态：网关配置在连接配置文件中定义最少。通常，会指定应用程序组织中的一个或两个对等方，它们使用服务发现来发现可用的网络拓扑。这包括对等方、订购方、渠道、实例化的智能合约及其背书策略。(在生产环境中，网关配置应至少指定两个对等方作为可用性。)

The SDK will use all of the static and discovered topology information, in conjunction with gateway connection options, to manage the transaction submission and notification processes. As part of this, it will also intelligently use the discovered topology; for example, it will calculate the minimum required endorsing peers using the discovered endorsement policy for the smart contract.

SDK 将使用所有静态和发现的拓扑信息以及网关连接选项来管理事务提交和通知过程。作为其中的一部分，它还将智能地使用发现的拓扑；例如，它将使用发现的智能合约认可策略计算认可对等方所需的最小值。

You might ask yourself whether a static or dynamic gateway is better? The trade-off is between predictability and responsiveness. Static networks will always behave the same way, as they perceive the network as unchanging. In this sense they are predictable - they will always use the same peers and orderers if they are available. Dynamic networks are more responsive as they understand how the network changes - they can use newly added peers and orderers, which brings extra resilience and scalability, at potentially some cost in predictability. In general it’s fine to use dynamic networks, and indeed this the default mode for gateways.

您可能会问自己静态网关还是动态网关更好？权衡的是可预测性和响应性。静态网络总是以相同的方式运行，因为它们认为网络是不变的。从这个意义上说，它们是可预测的——如果它们可用，它们将始终使用相同的对等方和订购方。动态网络在理解网络如何变化的过程中反应更为迅速——它们可以使用新添加的对等点和订购方，这将带来额外的弹性和可扩展性，并且在可预测性方面可能会付出一些代价。一般来说，使用动态网络是很好的，实际上这是网关的默认模式。

Note that the same connection profile can be used statically or dynamically. Clearly, if a profile is going to be used statically, it needs to be comprehensive, whereas dynamic usage requires only sparse population.

请注意，相同的连接配置文件可以静态或动态地使用。显然，如果一个概要文件要静态地使用，它需要是全面的，而动态使用只需要稀疏的填充。

Both styles of gateway are transparent to the application; the application program design does not change whether static or dynamic gateways are used. This also means that some applications may use service discovery, while others may not. In general using dynamic discovery means less definition and more intelligence by the SDK; it is the default.

这两种类型的网关对应用程序都是透明的;应用程序设计不会改变是使用静态网关还是动态网关。这也意味着一些应用程序可能使用服务发现,而其他应用程序可能不使用。通常,使用动态发现意味着 SDK 的定义更少,智能更多;它是默认的。

Connect

连接

When an application connects to a gateway, two options are provided. These are used in subsequent SDK processing:

当应用程序连接到网关时,提供了两个选项。这些在后续的 SDK 处理中使用:

```
await gateway.connect(connectionProfile, connectionOptions);
```

Connection profile: connectionProfile is the gateway configuration that will be used for transaction processing by the SDK, whether statically or dynamically. It can be specified in YAML or JSON, though it must be converted to a JSON object when passed to the gateway:

连接配置文件:ConnectionProfile 是用于 SDK 事务处理的网关配置,无论是静态的还是动态的。它可以在 yaml 或 json 中指定,但在传递到网关时必须转换为 json 对象:

```
let connectionProfile = yaml.safeLoad(fs.readFileSync('../gateway/paperNet.yaml', 'utf8'));
```

Read more about connection profiles and how to configure them.

阅读有关连接配置文件及其配置方法的更多信息。

Connection options: connectionOptions allow an application to declare rather than implement desired transaction processing behaviour. Connection options are interpreted by the SDK to control interaction patterns with network components, for example to select which identity to connect with, or which peers to use for event notifications. These options significantly reduce application complexity without compromising functionality. This is possible because the SDK has implemented much of the low level logic that would otherwise be required by applications; connection options control this logic flow.

连接选项:ConnectionOptions 允许应用程序声明而不是实现所需的事务处理行为。连接选项由 SDK 解释,以控制与网络组件的交互模式,例如选择要连接的标识或用于事件通知的对等方。这些选项显著降低了应用程序的复杂性,同时又不影响功能。这是可能的,因为 SDK 已经实现了应用程序所需的许多低级逻辑;连接选项控制着这个逻辑流。

Read about the list of available connection options and when to use them.

阅读可用连接选项列表以及何时使用它们。

Static

静态的

Static gateways define a fixed view of a network. In the MagnetoCorp scenario, a gateway might identify a single peer from MagnetoCorp, a single peer from DigiBank, and a MagentoCorp orderer. Alternatively, a gateway might define all peers and orderers from MagnetCorp and DigiBank. In both cases, a gateway must define a view of the network sufficient to get commercial paper transactions endorsed and distributed.

静态网关定义网络的固定视图。在 Magnetorp 场景中，网关可能会识别 Magnetorp 中的单个对等机、Digibank 中的单个对等机和 Magentorp 订购程序。或者，网关可以定义 magnetcorp 和 digibank 中的所有对等方和订购方。在这两种情况下，网关必须定义一个足以使商业票据交易得到认可和分发的网络视图。

Applications can use a gateway statically by explicitly specifying the connect option discovery: { enabled:false } on the gateway.connect() API. Alternatively, the environment variable setting FABRIC_SDK_DISCOVERY=false will always override the application choice.

应用程序可以通过在 gateway.connect（）API 上显式指定 connect 选项 discovery:enabled:false 来静态使用网关。或者，环境变量设置 fabric_sdk_discovery=false 将始终覆盖应用程序选项。

Examine the connection profile used by the MagnetoCorp issue application. See how all the peers, orderers and even CAs are specified in this file, including their roles.

检查磁电机公司问题应用程序使用的连接配置文件。查看如何在此文件中指定所有对等方、订购方甚至 CA，包括它们的角色。

It’s worth bearing in mind that a static gateway represents a view of a network at a moment in time. As networks change, it may be important to reflect this in a change to the gateway file. Applications will automatically pick up these changes when they re-load the gateway file.

值得记住的是，静态网关表示某个时刻的网络视图。随着网络的变化，在对网关文件的更改中反映这一点可能很重要。应用程序将在重新加载网关文件时自动接收这些更改。

Dynamic

动态的

Dynamic gateways define a small, fixed starting point for a network. In the MagnetoCorp scenario, a dynamic gateway might identify just a single peer from MagnetoCorp; everything else will be discovered! (To provide resiliency, it might be better to define two such bootstrap peers.)

动态网关为网络定义一个小的、固定的起点。在 Magnetorp 场景中，动态网关可能只识别 Magnetorp 中的一个对等机；其他一切都将被发现！(为了提供弹性，最好定义两个这样的引导对等机。)

If service discovery is selected by an application, the topology defined in the gateway file is augmented with that produced by this process. Service discovery starts with the gateway definition, and finds all the connected peers and orderers within the MagnetoCorp organization using the gossip protocol. If anchor peers have been defined for a channel, then service discovery will use the gossip protocol across organizations to discover components within the connected organization. This process will also discover smart contracts installed on peers and their endorsement policies defined at a channel level. As with static gateways, the discovered network must be sufficient to get commercial paper transactions endorsed and distributed.

如果应用程序选择了服务发现，则网关文件中定义的拓扑将与此进程生成的拓扑进行扩

充。服务发现从网关定义开始，并使用八卦协议查找 Magnetorp 组织内所有连接的对等方和订购方。如果已经为一个通道定义了锚定对等端，那么服务发现将跨组织使用八卦协议来发现连接的组织中的组件。此过程还将发现安装在对等端上的智能合约及其在渠道级别定义的认可策略。与静态网关一样，发现的网络必须足以使商业票据交易得到批准和分发。

Dynamic gateways are the default setting for Fabric applications. They can be explicitly specified using the connect option discovery: { enabled:true } on the gateway.connect() API. Alternatively, the environment variable setting FABRIC_SDK_DISCOVERY=true will always override the application choice.

动态网关是结构应用程序的默认设置。可以使用 gateway.connect（）api 上的 connect 选项 discovery:enabled:true 显式指定它们。或者，环境变量设置 fabric_sdk_discovery=true 将始终覆盖应用程序选项。

A dynamic gateway represents an up-to-date view of a network. As networks change, service discovery will ensure that the network view is an accurate reflection of the topology visible to the application. Applications will automatically pick up these changes; they do not even need to re-load the gateway file.

动态网关表示网络的最新视图。随着网络的变化，服务发现将确保网络视图是应用程序可见拓扑的准确反映。应用程序将自动接收这些更改；它们甚至不需要重新加载网关文件。

Multiple gateways

多个网关

Finally, it is straightforward for an application to define multiple gateways, both for the same or different networks. Moreover, applications can use the name gateway both statically and dynamically.

最后，应用程序可以很容易地为相同或不同的网络定义多个网关。此外，应用程序可以静态和动态地使用名称网关。

It can be helpful to have multiple gateways. Here are a few reasons:

拥有多个网关可能会有所帮助。以下是几个原因：

Handling requests on behalf of different users.

代表不同的用户处理请求。

Connecting to different networks simultaneously.

同时连接到不同的网络。

Testing a network configuration, by simultaneously comparing its behaviour with an existing configuration.

测试网络配置，同时将其行为与现有配置进行比较。