应用

受众：架构师、应用程序和智能合约开发人员

应用程序可以通过将交易提交到帐本或查询帐本内容来与区块链网络进行交互。本主题介绍了应用程序如何执行此操作的机制; 在我们的场景中，组织使用应用程序访问 PaperNet，这些应用程序调用定义在商业票据智能合约中的**发行**、**购买**和**兑换**交易。尽管 MagnetoCorp 的应用发行商业票据是基础功能，但它涵盖了所有主要的理解点。

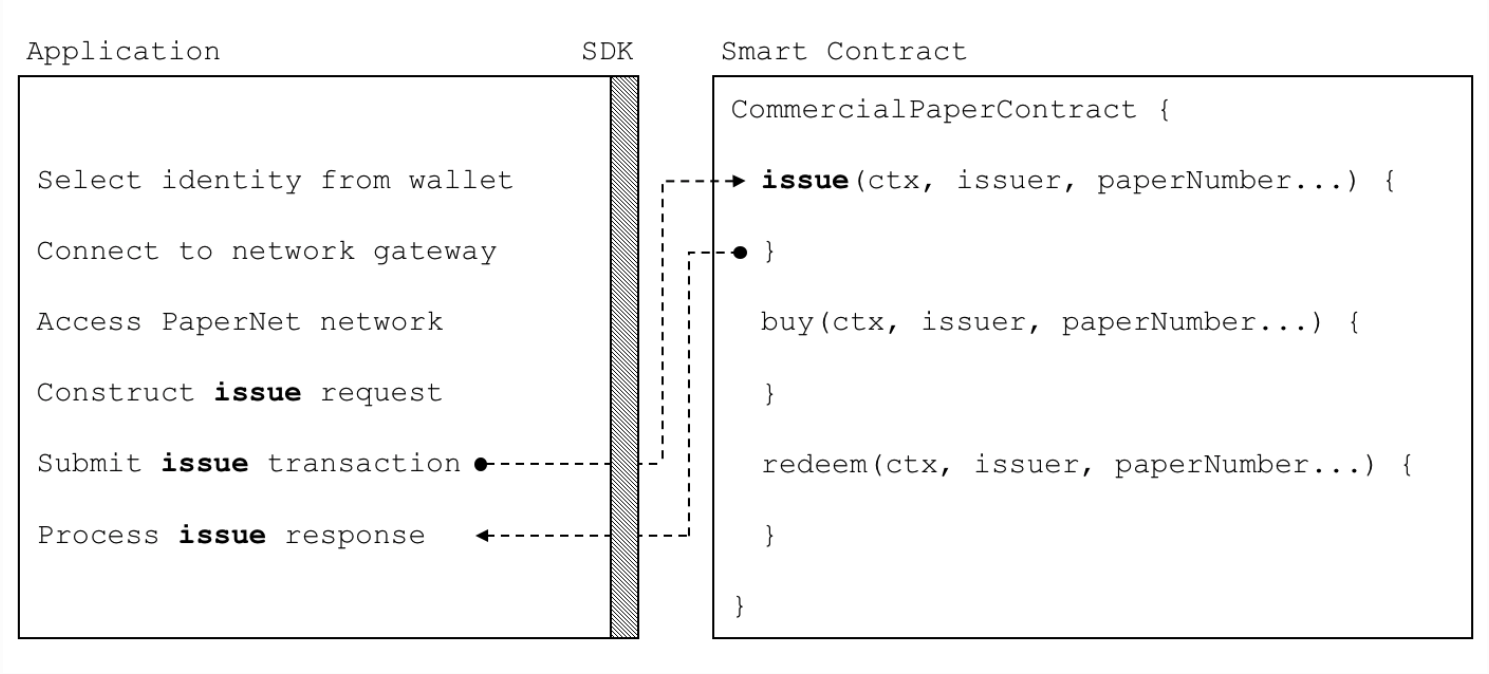
在本主题中，我们将介绍：

- 从应用程序到调用智能合约
- 应用程序如何使用钱包和身份
- 应用程序如何使用网关连接
- 如何访问特定网络
- 如何构造交易请求
- 如何提交交易
- 如何处理交易响应

为了帮助您理解，我们将参考 Hyperledger Fabric 提供的商业票据示例应用程序。您可以下载 并在本地运行它。它是用 JavaScript 和 Java 编写的，但逻辑与语言无关，因此您可以轻松地查看正在发生的事情！（该示例也适用于 Go。）

基本流程

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表：



PaperNet 应用程序调用商业票据智能合约来提交发行交易请求。

应用程序必须遵循六个基本步骤来提交交易：

- 从钱包中选择一个身份
- 连接到网关
- 访问所需的网络
- 构建智能合约的交易请求
- 将交易提交到网络
- 处理响应

您将看到典型应用程序如何使用 Fabric SDK 执行这六个步骤。 您可以在 `issue.js` 文件中找到应用程序代码。请在浏览器中查看，如果您已下载，请在您喜欢的编辑器中打开它。 花一些时间看一下应用程序的整体结构; 尽管有注释和空白，但是它只有100行代码！

钱包

在 `issue.js` 的顶部，您将看到两个 Fabric 类导入代码域：

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

您可以在node SDK 文档中了解 `fabric-network` 类，但是现在，让我们看看如何使用它们将 MagnetoCorp 的应用程序连接到 PaperNet。该应用程序使用 Fabric **Wallet** 类，如下所示：

```
const wallet = await Wallets.newFileSystemWallet('../identity/user/isabella/wallet');
```

了解 `wallet` 如何在本地文件系统中找到钱包。从钱包中检索到的身份显然适用于使用 `issue` 应用程序的 Isabella 用户。钱包拥有一组身份——X.509 数字证书——可用于访问 PaperNet 或任何其他 Fabric 网络。如果您运行该教程，并查看此目录，您将看到 Isabella 的身份凭证。

想想一个钱包里面装着政府身份证，驾照或 ATM 卡的数字等价物。其中的 X.509 数字证书将持有者与组织相关联，从而使他们有权在网络通道中获得权利。例如， `Isabella` 可能是 MagnetoCorp 的管理员，这可能比其他用户更有特权——来自 DigiBank 的 `Balaji` 。 此外，智能合约可以在使用交易上下文的智能合约处理期间检索此身份。

另请注意，钱包不持有任何形式的现金或代币——它们持有身份。

网关

第二个关键类是 Fabric **Gateway**。最重要的是，网关识别一个或多个提供网络访问的 Peer 节点——在我们的例子中是 PaperNet。了解 `issue.js` 如何连接到其网关：

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

`gateway.connect()` 有两个重要参数：

- **connectionProfile**：连接配置文件的文件系统位置，用于将一组 Peer 节点标识为 PaperNet 的网关
- **connectionOptions**：一组用于控制 `issue.js` 与 PaperNet 交互的选项

了解客户端应用程序如何使用网关将自身与可能发生变化的网络拓扑隔离开来。网关负责使用连接配置文件和连接选项将交易提案发送到网络中的正确 Peer 节点。

花一些时间检查连接配置文件 `./gateway/connectionProfile.yaml` 。它使用YAML，易于阅读。

它被加载并转换为 JSON 对象：

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

现在，我们只关注 `channels:` 和 `peers:` 配置部分:(我们稍微修改了细节，以便更好地解释发生了什么。)

```
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
```

看一下 `channel:` 如何识别 `PaperNet:` 网络通道及其两个 Peer 节点。MagnetoCorp 拥有 `peer1.magenetocorp.com`，DigiBank 拥有 `peer2.digibank.com`，两者都有背书节点的角色。通过 `peers:` 键链接到这些 Peer 节点，其中包含有关如何连接它们的详细信息，包括它们各自的网络地址。

连接配置文件包含大量信息——不仅仅是 Peer 节点——而是网络通道，网络排序节点，组织和 CA，因此如果您不了解所有信息，请不要担心！

现在让我们将注意力转向 `connectionOptions` 对象：

```
let connectionOptions = {
    identity: userName,
    wallet: wallet,
    discovery: { enabled:true, asLocalhost: true }
};
```

了解它如何指定应使用 identity、`userName` 和 wallet、`wallet` 连接到网关。 这些是在代码中分配值较早的。

应用程序可以使用其他连接选项来指示 SDK 代表它智能地执行操作。 例如：

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

这里，`commitTimeout` 告诉 SDK 等待100秒以监听是否已提交交易。`strategy:EventStrategies.MSPID_SCOPE_ANYFORTX` 指定 SDK 可以在单个 MagnetoCorp Peer 节点确认交易后通知应用程序，与 `strategy: EventStrategies.NETWORK_SCOPE_ALLFORTX` 相反，`strategy: EventStrategies.NETWORK_SCOPE_ALLFORTX` 要求 MagnetoCorp 和 DigiBank 的所有 Peer 节点确认交易。

如果您愿意，请阅读更多 有关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的信息。

网络通道

在网关 `connectionProfile.yaml` 中定义的 Peer 节点提供 `issue.js` 来访问 PaperNet。 由于这些 Peer 节点可以连接到多个网络通道，因此网关实际上为应用程序提供了对多个网络通道的访问！

了解应用程序如何选择特定通道：

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

从这一点开始，`network` 将提供对 PaperNet 的访问。 此外，如果应用程序想要访问另一个网络，`BondNet`，同时，它很容易：

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

现在，我们的应用程序可以访问第二个网络 `BondNet`，同时可以访问 `PaperNet`！

我们在这里可以看到 Hyperledger Fabric 的一个强大功能——应用程序可以通过连接到多个网关 Peer 节点来加入**网络中的网络**，每个网关 Peer 节点都连接到多个网络通道。 根据 `gateway.connect()` 提供的钱包标识，应用程序将在不同的通道中拥有不同的权限。

构造请求

该应用程序现在准备**发行**商业票据。要做到这一点，它将再次使用 `CommercialPaperContract`，它可以非常直接地访问这个智能合约：

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

请注意应用程序如何提供名称——`papercontract`——以及可选的合约命名空间：`org.papernet.commercialpaper`！ 我们看到如何从包含许多合约的 `papercontract.js` 链码文件中选出一个合约名称。在 PaperNet 中，`papercontract.js` 已安装并使用名称 `papercontract` 部署到了通道，如果您有兴趣，请如何部署包含多个智能合约的链代码。

如果我们的应用程序同时需要访问 PaperNet 或 BondNet 中的另一个合约，这将很容易：

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

在这些例子中，注意我们是如何不使用一个有效的合约名字——每个文件我们只有一个智能合约，并且 `getContract()` 将会使用它找到的第一个合约。

回想一下 MagnetoCorp 用于发行其第一份商业票据的交易：

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

我们现在将此交易提交给 PaperNet！

提交交易

提交一个交易是对 SDK 的单个方法调用：

```
const issueResponse = await contract.submitTransaction('issue', 'MagnetoCorp', '00001', '2020-05-
```

了解 `submitTransaction()` 参数如何与交易请求匹配。它们的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。回想一下它的签名：

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

那可能会显示一个智能合约会在应用程序触发了 `submitTransaction()` 之后很快地收到控制，但是并不是那样的。在外表下，SDK 使用了 `connectionOptions` 和 `connectionProfile` 来将交易提案发送给网络中正确的节点，从那里可以得到所需的背书。但是应用程序并不用担心——它仅仅是触发了 `submitTransaction` 然后 SDK 会除了接下来所有的事情！

我们注意到，`submitTransaction` API 包含了监听交易提交的一个流程。监听提交是必须的，因为如果没有它的话，你将不会知道你的交易是否被成功地排序了，验证了并且提交到了账本上。

现在让我们将注意力转向应用程序如何处理响应！

处理响应

回想一下 `papercontract.js` 如何**发行**交易返回一个商业票据响应：

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

您会注意到一个轻微的怪癖——新`票据`需要在返回到应用程序之前转换为缓冲区。请注意 `issue.js` 如何使用类方法 `CommercialPaper.fromBuffer()` 将响应缓冲区重新转换为商业票据：

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

这样可以在描述性完成消息中以自然的方式使用`票据`：

```
console.log(`${paper.issuer} commercial paper : ${paper.paperNumber} successfully issued for valu
```

了解如何在应用程序和智能合约中使用相同的 `paper` 类——如果您像这样构建代码，它将真正有助于可读性和重用。

与交易提案一样，智能合约完成后，应用程序可能会很快收到控制权，但事实并非如此。SDK 负责管理整个共识流程，并根据`策略`连接选项在应用程序完成时通知应用程序。 如果您对 SDK 的内容感兴趣，请阅读详细的交易流程。

就是这样！在本主题中，您已了解如何通过检查 MagnetoCorp 的应用程序如何在 PaperNet 中发行新的商业票据，从示例应用程序调用智能合约。现在检查关键账本和智能合约数据结构是由它们背后的架构主题设计的。