第4章:ネットワークの構築とテスト

Bluemix & Blockchainの学習

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales
David Smits, Senior Certified Architect, IBM Blockchain

計画: 30分のセッションと１～２時間の作業からなる章立て

第1章	ブロックチェーンは何ですか? 概念とアーキテクチャの概要
第2章	構築しようとしている話は何ですか
第2.1章	話のためのアーキテクチャ
第3章	ローカルHyperbelger Fabric V1開発環境の設定
第4章	**ネットワークの構築とテスト**
第5章	管理ユーザー経験
第6章	購入者のサポートとユーザー経験
第7章	販売者のサポートとユーザー経験
第8章	プロバイダーのサポートとユーザー経験
第9章	荷送人のサポートとユーザー経験
第10章	財務会社のサポートとユーザー経験
第11章	デモンストレーションのための結合
第12章	デモンストレーションのためのイベントと自動化

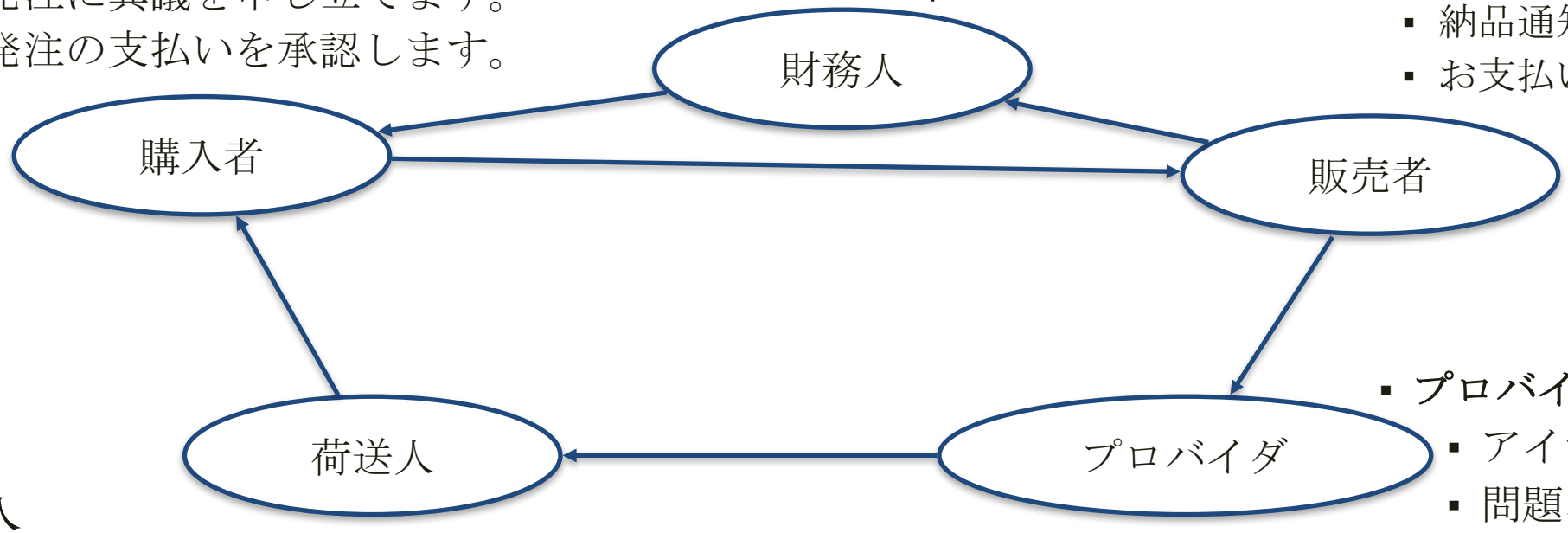
参加者は誰ですか？彼らは何をすることができますか？

Defines the members of the network, the transactions, and who should approve transactions

- 購入者
  - 発注を作成します。
  - 発注を提出します。
  - 貨物を受け取ります。
  - 発注に異議を申し立てます。
  - 発注の支払いを承認します。

- 財務会社
  - Receive Request for Payment
  - Approve and Pay
  - Receive notification of dispute
  - Resolve Dispute

- 販売者
  - 発注を受け入れます。
  - プロバイダに発注を提出します。
  - 納品通知を受け取ります。
  - お支払いを依頼します。



- 荷送人
  - 配達要求を受け取ります。
  - 配信の通知を送信します。

- プロバイダ
  - アイテム依頼を受信します。
  - 問題、バックオーダーを解決します。
  - 依頼を配信します。
  - 納品通知を受け取ります。

メンバーの定義

- シンプルなネットワークでメンバーには以下があります:
  - 会社名
  - eメールアドレスとして実装する識別子

- "companyName"という単一のフィールドを持つ抽象型メンバーを定義します。

```
18    namespace composer.base
19
20    abstract participant Member {
21      o String companyName
22    }
```

- 次に、メンバーのタイプごとにこの抽象タイプを拡張します。

```
participant Buyer identified by buyerID extends Member{
    o String buyerID
}
```

- 目的は、抽象的な型と異なる種類の定義を別々のファイルに分割する能力を導入することです。 これは、メンテナンスを目的としていますが、作業が完了したらすべてを簡単に組み合わせることができます。

資産の定義

- このチュートリアルで扱う単一の資産は、ここに示すように定義された「発注」です。
- フィールドの多くは日付を格納するためのものなので、何がいつ起こったかはわかります。 日付と理由は取引によって更新され、参加者によって制限されます。
- 角括弧は配列を示すことに注意してください。
- 矢印(- ->)は、以前に定義された他のネットワーククラスへの参照を示すことに注意してください。

```
asset Order identified by orderNumber {
    o String orderNumber
    o String[] items
    o String status
    o Integer amount
    o String created
    o String bought
    o String ordered
    o String dateBackordered
    o String requestShipment
    o String delivered
    o String disputeOpened
    o String disputeResolved
    o String paymentRequested
    o String orderRefunded
    o String paid
    o String[] vendors
    o String dispute
    o String resolve
    o String backorder
    o String refund
    --> Buyer buyer
    --> Seller seller
```

トランザクションの定義

- トランザクションは、資産とメンバと同じモデル言語を使用します。
- ここではトランザクションを命名し、このトランザクションを処理するための要求に伴うものを特定します。
- これはトランザクションクラスです。
- それはCreateOrderという名前です。
- それは1つのフィールド（整数値）を持ちます。
- それは3つの他のインスタンスを参照します。
  - 発注
  - 購入者
  - 販売者

```
transaction CreateOrder {
  o Integer amount
  --> Order order
  --> Buyer buyer
  --> Seller seller
}
```

この情報を参考にして、以下を定義します

- メンバー:
  - 購入者、販売者、 プロバイダー、荷送人、財務会社
- 資産:
  - 発注
- トランザクション:
  - 発注の作成、購入、サプライヤーからの発注、配送の依頼、配達、送金、紛争、解決、支払い請求、支払い、払い戻し。

ネットワークをチェックします

- ステップ1, モデルファイルを更新してください。
- ステップ2, 作成してアーカイブして展開してください。
- ステップ3, コンポーザを読み込んでテストしてください。

- ステップ1, 答えはDocuments / answersフォルダにあります。
- ステップ2, Chapter04フォルダ内から次のコマンドを実行してください。
  - **`buildAndDeploy`**
- ステップ3, 以下に移動してください：
  - モデルファイルを以下からインポートしてください：
    `Chapter04/network/dist/zerotoblockchain-network.bna`
  - モデルをテストしてください。
  - 検査で、Orderオブジェクトであまり起こらないことがわかります。 それは次にあります。

トランザクションを実装するためのコードの記述

- 各トランザクションはロジックを実装する必要があります。 たとえば、発注作成トランザクションは、購入者が発注を作成して販売者に送信する前に保存することを可能にするために存在します。 コードは右側に示されています。

- クラス定義（右下）には、オーダー、オーダーの量、および購入者へのリンクが含まれていることがわかります。 このトランザクションコードでは、発注が販売者にまだ割り当てられていないため、販売者情報は使用されません。

```
/**
 * create an order to purchase
 * @param {org.acme.Z2BTestNetwork.CreateOrder} purchase - the order to be processed
 * @transaction
 */
function CreateOrder(purchase) {
    purchase.order.buyer = purchase.buyer;
    purchase.order.amount = purchase.amount;
    purchase.order.created = new Date().toISOString();
    purchase.order.status = "Order Created";
    return getAssetRegistry('org.acme.Z2BTestNetwork.Order')
        .then(function (assetRegistry) {
            return assetRegistry.update(purchase.order);
        });
}
```

```
transaction CreateOrder {
  o Integer amount
  --> Order order
  --> Buyer buyer
  --> Seller seller
}
```

トランザクションをテストするためのコードの記述

- このアプリケーションをテストするためにmochaサービスを使用しています。コードは以下のようになります。
- このコードを対話的に探求します。
- 終わったら、npmに作成したものをテストするように指示することができます。これは以下のような結果をもたらすはずです：

```
Finance Network
  #createOrder
    ✓ should be able to create an order (82ms)
  #issueBuyRequest
    ✓ should be able to issue a buy request (40ms)
  #issueOrderFromSupplier
    ✓ should be able to issue a supplier order (50ms)
  #issueRequestShipment
    ✓ should be able to issue a request to ship product (47ms)
  #issueDelivery
    ✓ should be able to record a product delivery (39ms)
  #issueRequestPayment
    ✓ should be able to issue a request to request payment for a product (58ms)
  #issuePayment
    ✓ should be able to record a product payment (48ms)
  #issueDispute
    ✓ should be able to record a product dispute (63ms)
  #issueResolution
    ✓ should be able to record a dispute resolution (48ms)
  #issueBackorder
    ✓ should be able to record a product backorder (53ms)


10 passing (1s)
```

```
describe('#createOrder', () => {

    it('should be able to create an order', () => {
        const factory = businessNetworkConnection.getBusinessNetwork().getFactory();

        // create the buyer
        const buyer = factory.newResource(NS, 'Buyer', buyerID);
        buyer.companyName = 'billybob computing';

        // create the seller
        const seller = factory.newResource(NS, 'Seller', sellerID);
        seller.companyName = 'Simon PC Hardware, Inc';

        // create the order
        let order = factory.newResource(NS, 'Order', orderNo);
        order = createOrderTemplate(order);
        order = addItems(order);
        order.orderNumber = orderNo;

        // create the buy transaction
        const createNew = factory.newTransaction(NS, 'CreateOrder');

        order.buyer = factory.newRelationship(NS, 'Buyer', buyer.$identifier);
        order.seller = factory.newRelationship(NS, 'Seller', seller.$identifier);
        createNew.order = factory.newRelationship(NS, 'Order', order.$identifier);
        createNew.buyer = factory.newRelationship(NS, 'Buyer', buyer.$identifier);
        createNew.seller = factory.newRelationship(NS, 'Seller', seller.$identifier);
        createNew.amount = order.amount;
        // the buyer should of the commodity should be buyer
        //order.buyer.$identifier.should.equal(buyer.$identifier);
        order.status.should.equal('Order Created');
        order.amount.should.equal(orderAmount);
        createNew.amount.should.equal(orderAmount);
        createNew.order.$identifier.should.equal(orderNo);
```

composer-rest-serverの呼び出し

- 第04章から
  - 以下のコマンドを実行してください：
    - **buildAndDeploy**
  - それは完成したネットワークがドッカーにロードされます
  - 以下のコマンドを実行してください：
    - **./start_rest_server.sh**
  - それは、以下の応答を適用します：

```
[? Enter your Fabric Connection Profile Name: hlfv1
[? Enter your Business Network Identifier : zerotoblockchain-network
[? Enter your Fabric username : admin
[? Enter your secret: adminpw
 ? Specify if you want namespaces in the generated REST API: always use namespaces
[? Specify if you want the generated REST API to be secured: No
[? Specify if you want to enable event publication over WebSockets: Yes
[? Specify if you want to enable TLS security for the REST API: No
```

- localhost：3000 / explorerに移動して、新しいRESTful APIを検査してテストしてください。

計画: 30分のセッションと１～２時間の作業からなる章立て

第1章	ブロックチェーンは何ですか? 概念とアーキテクチャの概要
第2章	構築しようとしている話は何ですか
第2.1章	話のためのアーキテクチャ
第3章	ローカルHyperbelger Fabric V1開発環境の設定
第4章	ネットワークの構築とテスト
第5章	**管理ユーザー経験**
第6章	購入者のサポートとユーザー経験
第7章	販売者のサポートとユーザー経験
第8章	プロバイダーのサポートとユーザー経験
第9章	荷送人のサポートとユーザー経験
第10章	財務会社のサポートとユーザー経験
第11章	デモンストレーションのための結合
第12章	デモンストレーションのためのイベントと自動化