# 实验六：超级账本项目Fabric实验

超级账本 (Hyperledger) 是由 Linux 基金会于 2015 年启动的区块链开源项目，该项目一经启动，反响热烈，一举成为区块链生态圈的明星项目。目前，已经有超过 200 家的会员单位参与进超级账本项目。超级账本项目旨在让一个组织的组织成员都能享受到去中心化的区块链账本带来的应用级便利，都能够参与到区块链账本的维护和建设中，使区块链账本应用能够在各行各业上线、应用、落地，满足各种各样的安全性需求。

Fabric 是目前超级账本最为成熟和流行的一个项目。与比特币、以太坊等公链系统不同，Fabric 拥有一套完整的联盟链体系，一条区块链的规模相对较小，但其拥有严格的准入规则和身份管理规则，建立了模块化的共识机制、身份管理体系、智能合约、应用客户端、隐私保护模块和证书体系。Fabric系统具有强大的内部成员可信性，这样使得其共识机制可以集中处理交易的排序问题，多采用分布式系统的共识，如 Raft、Kafka 等，效率极高，因此交易速度和吞吐率都在数量级上超越了比特币、以太坊等经典的公链系统。

区块链系统根据面向体是否公开、准入规则、规模大小、去中心化程度，一般分为公有链、联盟链和私有链。其中，私有链的中心化程度较为严重，相比于公有链，联盟链加入了准入的规则，即节点或用户的身份需要经过认证。另外，联盟链实际上是介于公有链和私有链中间的一类区块链，只不过与私有链具有本质上的相似性，即拥有严格的准入规则，但其规模远超私有链，与无准入规则、公开可访问的公有链有本质上的区别。因此，也可以说，联盟链是一种特殊的私有链。

联盟链有以下特点:

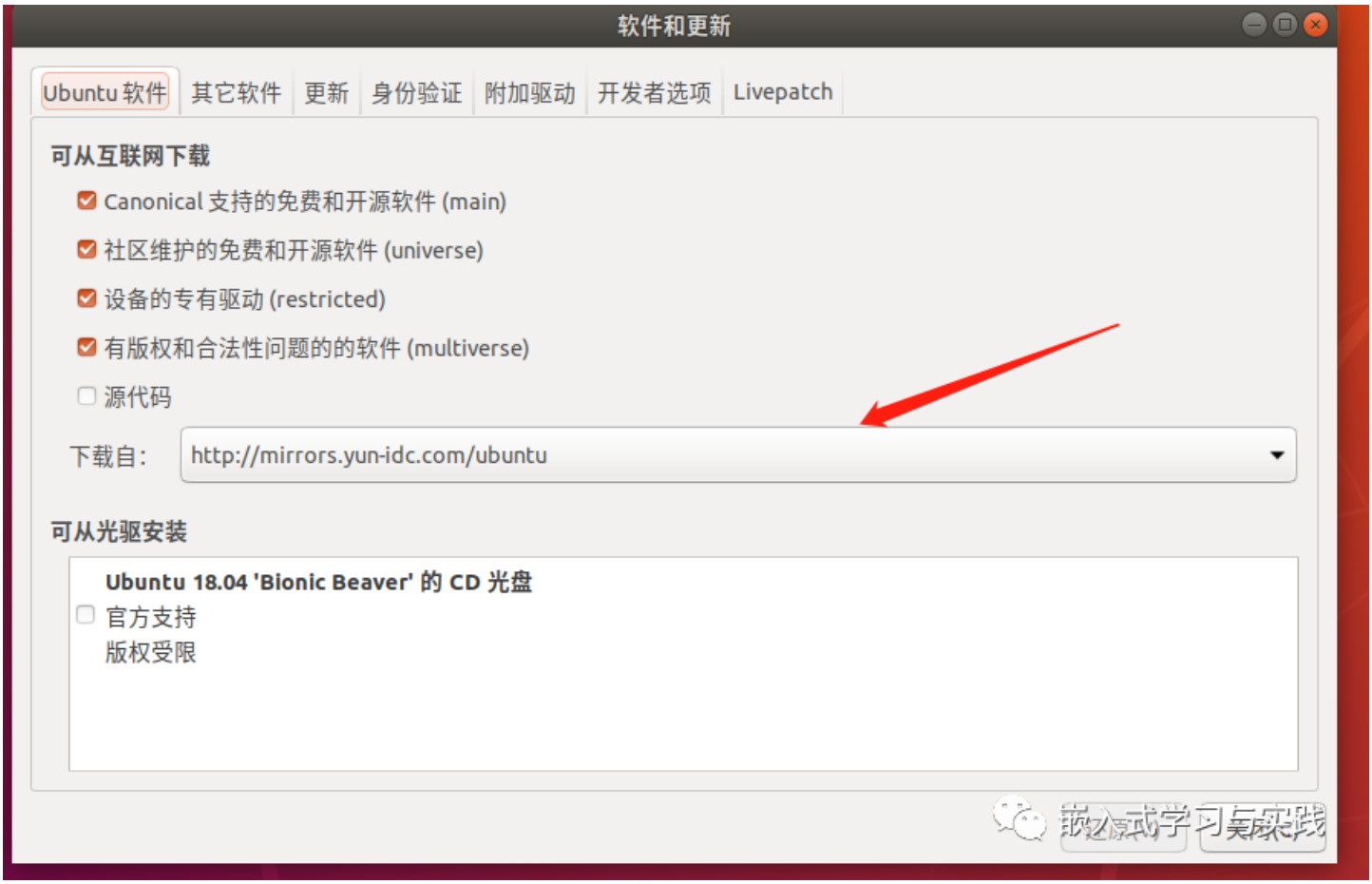
1. 弱中心化 (或称为多中心化、部分去中心化)。在某种意义上，联盟链不像公有链一样归属于任何大众，而是存在于规则严格的联盟组织内部。由于联盟链节点数量大多都是有限的，因此联盟组织内部达成共识相对较容易。这也是联盟链交易确认速度、吞吐率较高的重要原因。
2. 有较强的可控性。公有链的节点是海量的、无许可的，在公有链中，数据只要上链并确认，一般是不可篡改的。但在联盟链中，达成共识比较容易，因此有一定的可控性。
3. 数据默认不公开。联盟链仅供联盟组织内部成员调用其数据。

本章实验中的 Fabric 便是联盟链项目的典型例子。

## 一、Fabric环境的配置

1. **替换软件源**

方法一：在“软件与更新”中选择合适的软件源即可（推荐）




方法二：手动修改/etc/apt/source.list，替换完成后用"sudo apt update"和"sudo apt upgrade"命令对系统软件包进行更新

2. **更新Ubuntu系统环境**

```
1 sudo apt-get update
2 sudo apt-get upgrade
```

3. **安装Go语言环境**

- 下载go语言安装包

```
1 wget https://dl.google.com/go/go1.18.1.linux-amd64.tar.gz
2 sudo tar -xvf go1.18.1.linux-amd64.tar.gz
3 sudo mv go /usr/local
4 mkdir go
5 mkdir go/src
```

- 配置go语言环境变量

```
1 vim ~/.bashrc
```

```
# 添加如下内容
export GOROOT=/usr/local/go
export GOPATH=$HOME/go
export PATH=$GOPATH/bin:$GOROOT/bin:$PATH
export HUBPATH=/root/Cross-chain/CrossChainHub # 1.6步提前执行
# 保存更改
source ~/.bashrc
# 确认是否配置成功
go version
```

### 4. 安装docker与docker-compose

```
# 卸载旧版本
sudo apt-get remove docker docker-engine docker.io containerd runc
# 更新apt软件包索引
sudo apt-get update
# 安装软件包以允许apt通过HTTPS使用存储库
sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common
# 添加Docker的官方GPG密钥
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# 验证密钥
sudo apt-key fingerprint 0EBFCD88
# X86_64/amd64平台下设置存储库
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
# 安装docker
sudo apt-get install docker-ce docker-ce-cli containerd.io
# 安装docker-compose
# 下载运行文件
# https://github.com/docker/compose/releases/download/1.28.6/docker-compose-Linux-x86_64
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.6/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
# 配置权限
sudo chmod +x /usr/local/bin/docker-compose
# 查看版本信息
docker version
```

```
31 docker-compose version
```

### 5. 配置代理

```
1  # 设置go代理
2  go env -w GO111MODULE="on"
3  go env -w GOPROXY=https://mirrors.aliyun.com/goproxy/
4  # 设置docker代理
5  sudo vim /etc/docker/daemon.json
6  # 添加如下内容
7  {
8    "registry-mirrors": ["https://registry.docker-
   cn.com","https://6kx4zyno.mirror.aliyuncs.com","http://hub-
   mirror.c.163.com","https://docker.mirrors.ustc.edu.cn"]
9  }
10 # 重启生效
11 systemctl daemon-reload
12 systemctl restart docker
```

### 6. Fabric的安装

```
1  # 创建并进入工作目录
2  mkdir -p $GOPATH/src/github.com/hyperledger/
3  cd $GOPATH/src/github.com/hyperledger/
4  # 从Github克隆源码
5  git clone https://github.com/hyperledger/fabric.git
6  cd fabric
7  git checkout v1.4.3
8  # 执行scripts目录中的bootstrap.sh脚本会自动下载fabric-samples和fabric镜像（需要较长时
9  cd scripts/
10 ./bootstrap.sh
11 # 脚本完成后会发现当前目录有了fabric-samples目录，之后执行
12 docker images
```

# 二、first-network 的启动和 Channel 的配置

节点分类：Fabric 中的节点主要分为两大类: Peer 节点和 Orderer 节点。其中，Peer 节点的主要功能是背书，即先模拟执行客户端提交的交易提案，若执行通过，则对该交易进行背书并返还给客户端，还具备存储区块链链上的数据的功能。Orderer 节点也叫排序节点，主要负责对背书后的交易进行排序共识，并打包发送给 Peer 节点上链。每个Peer 节点可以担任多种角色，如 Anchor Peer (锚节点，允许被其他 Peer 节点发现)、Endorser Peer (背书节点，模拟执行客户端提交的交易提案对交易

进行背书并返给客户端)、Leader Peer (主节点，具有和 Orderer 节点进行通信的功能)、Committer Peer (记账节点，广播、同步、存储区块链数据)。

通道(Channel)：通道是 Fabric 中非常重要的概念，对一般用户来说，通道是指应用通道。通道的主要作用是对不同的账本进行隔离，可以理解为，有几条通道就有几条独立运行的区块链。

### 1. 启动 first network

本实验要求跟随指导书的指令引导，搭建第一个 Fabric 网络。首先定位到以下目录:

```
cd ~
cd go/src/github.com/hyperledger/fabric/scripts/fabric-samples/first-network/
```

生成证书、密钥。

```
../bin/cryptogen generate --config=./crypto-config.yaml
```

执行如下命令:

```
tree crypto-config
```

生成 ordererOrganizations 和 peerOrganizations 两个组织。其中，peerOrganizations 中包含 org1.example.com 和 org2.example.com 两个组织。具体命令如下:

```
cd crypto-config/
ls
sudo apt install tree
tree
```

生成 Orderer 节点的 genesis block

```
cd.
export FABRIC_CFG_PATH=$PWD
../bin/configtxgen -profile TwoOrgsOrdererGenesis -channelID byfn-sys-channel -outputBlock ./channel-artifacts/genesis.block
```

创建 channel transaction 的配置

```
export CHANNEL_NAME=mychannel && ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL_NAME
```

在 Channel上定义org1的 Anchor Peer。

```
../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org1MSP
```

输入相关命令，在Channel 上定义 org2 的Anchor Peer，然后输出channel-artifacts文件夹包含的文件。

```
../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL_NAME -asOrg Org2MSP
tree channel-artifacts/
```

查看排序创世块和通道创世块的命令分别如下:

```
../bin/configtxgen -profile TwoOrgsOrdererGenesis -inspectBlock ./channel-artifacts/genesis.block
../bin/configtxgen -profile TwoOrgsOrdererGenesis -inspectChannelCreateTx ./channel-artifacts/channel.tx
```

会以JSON格式解析给出区块的内容

使用 docker-compose，根据配置文件来启动网络

```
docker-compose -f docker-compose-cli.yaml up -d
```

2. **在CLI容器中配置 Channel**

加入CLI容器。

```
docker exec -it cli bash
```

将节点切换到 peer0.org1。

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
```

下面进行 Channel 创建，应该与前面创的 Channel 配置保持一样的名字.

```
export CHANNEL_NAME=mychannel
peer channel create -o orderer.example.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/channel.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

“-c”标志指定 Channel 的名字， “-f”标志指定配置交易，本例中是 channel.tx。

这时可以看到目录中多了一个创世区块 mychannel.block，包含 channel.tx 的配置信息，从而可以加入 Channel.

下面加入 peer0.org1的通道，进行链码的相关配置

```
peer channel join -b mychannel.block
```

输入相关命令，将peer0.org2.example.com:7051也加入 Channel.

```
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
peer channel join -b mychannel.block
```

更新 peer0.orgl.example.com 的 Anchor Peer 信息。

```
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
peer channel update -o orderer.example.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org1MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

输入相关命令，以更新另一个 Anchor Peer 的信息

```
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
peer channel update -o orderer.example.com:7050 -c $CHANNEL_NAME -f ./channel-artifacts/Org2MSPanchors.tx --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
```

# 三、链码的安装和实例化

链码是 Fabric 中的智能合约，与以太坊类似，链码的发布、询问和写入都是 Fabric中的基本交易类型。下面讲解 Fabric 作为联盟链是如何处理交易的。

每笔交易都要由应用客户端首先提交交易提案，具体步骤如下。

①应用客户端发布一个新的交易提案，把相关链码标识、链码参数、背书策略发给背书Peer节点。

② Peer 节点收到应用客户端发布的交易提案后，验证客户端本身的身份和其所具备的权限是否支持其发布这笔交易提案。验证通过后，Peer 节点模拟执行此交易，通过后，对此交易进行背书，返还给客户端。

③客户端收到背书 Peer 节点返回的信息后，判断提案结果是否一致，以及是否符合相应的背书策略，如果不一致或不符合，那么停止，否则，客户端把数据打包到一起组成一个交易并签名，发给 Orderer 节点。

④ Orderer 节点收到交易信息后，将交易放入共识排序序列。此后将一批交易进行打包记录为新的区块，并发送给 Peer 节点。

⑤记账 Peer 节点收到区块后，会检查每笔交易的正确性和有效性，检查交易的输入,输出是否符合当前区块链的世界状态。完成后，将区块添置于当前本地的区块链的最后方,并修改世界状态。

依然在CLI容器中进行。

将节点切换到 peer0.org1

```
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
```

本实验提供链码示例

```
peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode_example02/go
```

将节点切换到 peer0.org2，同样安装上述链码

```
CORE_PEER_LOCALMSPID="Org2MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ADDRESS=peer0.org2.example.com:7051
peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode_example02/go
```

将节点切换到 peer0.org1，实例化链码。

```
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
CORE_PEER_ADDRESS=peer0.org1.example.com:7051
export CHANNEL_NAME=mychannel
peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL_NAME -n mycc -v 1.0 -c '{"Args":["init","a", "100", "b","200"]}' -P "OR ('Org1MSP.peer','Org2MSP.peer')"
```

对链码进行 Query 操作，查询 a 的值，正确输出应为 100

```
peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","a"]}'
```

对链码进行 Invoke 操作，从 a 向 b 转账 10.

```
peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C $CHANNEL_NAME -n mycc --peerAddresses peer0.org1.example.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt -c '{"Args":["invoke","a","b","10"]}'
```

对链码进行 query 操作，查询 a 的值，因为已经发生了转账，正确输出应为 90。

```
peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","a"]}'
```

# 练习：

1. 将节点切换到 peer0.org2,将 peer0.org2 作为背书节点,再次从a 向b 转账 10，并查询a 和 b 各自的余额，请自行输入命令。

2. 重新实例化上述链码，要求背书策略为“org1 和 org2 中都要有节点背书”。执行从a到 b 转账 10 的 Invoke 操作，并对 john 的余额进行查询。