基于官方示例balance-transfer的改动如下：

1. 将原有chaincode替换成community.go和asset.go。
2. 在docker-compose.yaml文件里为每个peer设置了couchdb。

3.在app.js上加了一个版权交易的路由，该路由首先会查找帖子，然后进行版权交易。

4.实现版权交易的功能由app/ copyrightTransaction.js实现。

5.在app/helper.js里加了一个getkey功能，可以获得当前用户的公钥和私钥，这在版权交易的时候会使用到。

6.在testAPIs.sh中加入了对两个chaincode的安装和实例化，加入了增加帖子和查询帖子，加入了进行版权交易和查询所有版权交易信息等多个测试。

# 证书和公司钥创建步骤

Fabric中有两种类型的公私钥和证书，一种是给节点之前通讯安全而准备的TLS证书，另一种是用户登录和权限控制的用户证书。这些证书本来应该是由CA来颁发，但是目前只有两个社区，所以目前暂时没有启用CA节点，但是Fabric帮我们提供了一个crytogen工具来生成证书。

### 1.1编译cryptogen

编译生成 cryptogen之前我们需要安装一个软件包，否则编译时会报错

sudo apt install libtool libltdl3-dev

Fabric提供了专门编译cryptogen的入口，我们只需要运行以下命令即可：

cd ~/go/src/github.com/hyperledger/fabric

make cryptogen

运行后系统返回如下结果即代表编译成功了

build/bin/cryptogen

CGO\_CFLAGS=" " GOBIN=/home/studyzy/go/src/github.com/hyperledger/fabric/build/bin go install -tags "" -ldflags "-X github.com/hyperledger/fabric/common/tools/cryptogen/metadata.Version=1.0.0" github.com/hyperledger/fabric/common/tools/cryptogen

Binary available as build/bin/cryptogen

我们在build/bin文件夹下就可以看到编译出来的cryptogen程序。

### 1.2配置crypto-config.yaml

examples/e2e\_cli/crypto-config.yaml已经提供了一个Orderer Org和两个Peer Org的配置，该模板中也对字段进行了注释。我们可以把配置修改一下：

[制代码](javascript:void(0);)

OrdererOrgs:

- Name: Orderer

Domain: example.com

Specs:

- Hostname: orderer

PeerOrgs:

- Name: Org1

Domain: org1.example.com

Template:

Count: 1

Users:

Count: 1

- Name: Org2

Domain: org2.example.com

Template:

Count: 1

Users:

Count: 1

[制代码](javascript:void(0);)

Name和Domain就是关于这个组织的名字和域名，这主要是用于生成证书的时候，证书内会包含该信息。而Template Count=1是说我们要生成1套公私钥和证书，因为我们一个组织只需要一个peer节点。最后Users. Count=1是说每个Template下面会有几个普通User（注意，Admin是Admin，不包含在这个计数中），这里配置了1，也就是说我们只需要一个普通用户[User1@org2.example.com](mailto:User1@org2.example.com) 我们可以根据实际需要调整这个配置文件，增删Org Users等。

### 1.3生成公司钥和证书

我们配置好crypto-config.yaml文件后，就可以用cryptogen去读取该文件，并生成对应的公私钥和证书了：

cd examples/e2e\_cli/

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

生成的文件都保存到crypto-config文件夹，我们可以进入该文件夹查看生成了哪些文件：

tree crypto-config

## 二.生成创世区块和Channel配置区块

### 2.1编译生成configtxgen

与前面1.1说到的类似，我们可以通过make命令生成configtxgen程序：

cd ~/go/src/github.com/hyperledger/fabric

make configtxgen

运行后的结果为：

[制代码](javascript:void(0);)

build/bin/configtxgen

CGO\_CFLAGS=" " GOBIN=/home/studyzy/go/src/github.com/hyperledger/fabric/build/bin go install -tags "nopkcs11" -ldflags "-X github.com/hyperledger/fabric/common/configtx/tool/configtxgen/metadata.Version=1.0.0" github.com/hyperledger/fabric/common/configtx/tool/configtxgen

Binary available as build/bin/configtxgen

[制代码](javascript:void(0);)

### 2.2配置configtx.yaml

官方提供的examples/e2e\_cli/configtx.yaml这个文件里面配置了由2个Org参与的Orderer共识配置TwoOrgsOrdererGenesis，以及由2个Org参与的Channel配置：TwoOrgsChannel。Orderer可以设置共识的算法是Solo还是Kafka，以及共识时区块大小，超时时间等，我们使用默认值即可，不用更改。而Peer节点的配置包含了MSP的配置，锚节点的配置。如果我们有更多的Org，或者有更多的Channel，那么就可以根据模板进行对应的修改。

### 2.3生成创世区块

配置修改好后，我们就用configtxgen 生成创世区块。并把这个区块保存到本地channel-artifacts文件夹中：

cd examples/e2e\_cli/

../../build/bin/configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block

### 2.4生成Channel配置区块

../../build/bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel

另外关于锚节点的更新，我们也需要使用这个程序来生成文件：

../../build/bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID mychannel -asOrg Org1MSP

../../build/bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID mychannel -asOrg Org2MSP

最终，我们在channel-artifacts文件夹中，应该是能够看到4个文件。

channel-artifacts/   
├── channel.tx   
├── genesis.block   
├── Org1MSPanchors.tx   
└── Org2MSPanchors.tx

## 三.配置docker-compose文件

前面对节点和用户的公私钥以及证书，还有创世区块都生成完毕，接下来我们就可以分别为两个peer和一个orderer配置docker-compose的yaml文件，分别分发给三台虚拟机以后就可以启动Fabric的Docker环境了。

### 3.1 修改基础配置文件

peer和orderder的基础配置文件在base文件里面。

因为我们只有两个组织，每个组织只有一个peer，所以只需修改base/docker-compose-base.yaml文件，删除peer1.org1.example.com和peer1.org2.example.com。另外在单击模式下，4个peer会映射主机不同的端口，但是我们在多机部署的时候是不需要映射不同端口的，所以将所有peer的端口映射都改为相同的，修改完成的docker-compose-base.yaml文件如下：

[制代码](javascript:void(0);)

# Copyright IBM Corp. All Rights Reserved.

#

# SPDX-License-Identifier: Apache-2.0

#

version: '2'

services:

orderer.example.com:

container\_name: orderer.example.com

image: hyperledger/fabric-orderer

environment:

- ORDERER\_GENERAL\_LOGLEVEL=debug

- ORDERER\_GENERAL\_LISTENADDRESS=0.0.0.0

- ORDERER\_GENERAL\_GENESISMETHOD=file

- ORDERER\_GENERAL\_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block

- ORDERER\_GENERAL\_LOCALMSPID=OrdererMSP

- ORDERER\_GENERAL\_LOCALMSPDIR=/var/hyperledger/orderer/msp

# enabled TLS

- ORDERER\_GENERAL\_TLS\_ENABLED=true

- ORDERER\_GENERAL\_TLS\_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key

- ORDERER\_GENERAL\_TLS\_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt

- ORDERER\_GENERAL\_TLS\_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric

command: orderer

volumes:

- ../channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block

- ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/var/hyperledger/orderer/msp

- ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/:/var/hyperledger/orderer/tls

ports:

- 7050:7050

peer0.org1.example.com:

container\_name: peer0.org1.example.com

extends:

file: peer-base.yaml

service: peer-base

environment:

- CORE\_PEER\_ID=peer0.org1.example.com

- CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

- CORE\_PEER\_CHAINCODELISTENADDRESS=peer0.org1.example.com:7052

- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer0.org1.example.com:7051

- CORE\_PEER\_LOCALMSPID=Org1MSP

volumes:

- /var/run/:/host/var/run/

- ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp:/etc/hyperledger/fabric/msp

- ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls:/etc/hyperledger/fabric/tls

ports:

- 7051:7051

- 7052:7052

- 7053:7053

peer0.org2.example.com:

container\_name: peer0.org2.example.com

extends:

file: peer-base.yaml

service: peer-base

environment:

- CORE\_PEER\_ID=peer0.org2.example.com

- CORE\_PEER\_ADDRESS=peer0.org2.example.com:7051

- CORE\_PEER\_CHAINCODELISTENADDRESS=peer0.org2.example.com:7052

- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer0.org2.example.com:7051

- CORE\_PEER\_LOCALMSPID=Org2MSP

volumes:

- /var/run/:/host/var/run/

- ../crypto-config/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/msp:/etc/hyperledger/fabric/msp

- ../crypto-config/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls:/etc/hyperledger/fabric/tls

ports:

- 7051:7051

- 7052:7052

- 7053:7053

[制代码](javascript:void(0);)

3.2设置orderer节点的docker-compose文件

e2e\_cli提供了多个docker-compose文件，我们可以根据docker-compose-cli来修改

cp docker-compose-cli.yaml docker-compose-orderer.yaml

orderer服务器上我们只需要保留order设置，其他peer和cli设置都可以删除。orderer配置文件如下：

[制代码](javascript:void(0);)

# Copyright IBM Corp. All Rights Reserved.

#

# SPDX-License-Identifier: Apache-2.0

#

version: '2'

services:

orderer.example.com:

extends:

file: base/docker-compose-base.yaml

service: orderer.example.com

container\_name: orderer.example.com

[制代码](javascript:void(0);)

### 3.3设置peer节点的docker-compose文件

先为peer0.org1.example.com配置，与创建orderer的配置文件类似，我们也复制一个yaml文件出来进行修改：

cp docker-compose-cli.yaml docker-compose-peer0org1.yaml

去掉orderer的配置，只保留一个peer和cli，因为我们要多级部署，节点与节点之前又是通过主机名通讯，所以需要修改容器中的host文件，也就是extra\_hosts设置。因为之后我们要连接couchdb，所以这里加入couchdb的配置，这里的10.0.2.11:5984是我映射CouchDB后的Linux的IP地址和IP,然后是设置用户名和密码。

同样，cli也需要能够和各个节点通讯，所以cli下面也需要添加extra\_hosts设置，去掉无效的依赖，并且去掉command这一行，因为我们是每个peer都会有个对应的客户端，也就是cli，所以我只需要去手动执行一次命令，而不是自动运行。

修改后的配置文件如下：

[制代码](javascript:void(0);)

# Copyright IBM Corp. All Rights Reserved.

#

# SPDX-License-Identifier: Apache-2.0

#

version: '2'

services:

peer0.org1.example.com:

container\_name: peer0.org1.example.com

environment:

- CORE\_LEDGER\_STATE\_STATEDATABASE=CouchDB

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_COUCHDBADDRESS=10.0.2.11:5984

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_USERNAME=admin

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_PASSWORD=password

extends:

file: base/docker-compose-base.yaml

service: peer0.org1.example.com

extra\_hosts:

- "orderer.example.com:10.0.2.10"

cli:

container\_name: cli

image: hyperledger/fabric-tools

tty: true

environment:

- GOPATH=/opt/gopath

- CORE\_VM\_ENDPOINT=unix:///host/var/run/docker.sock

- CORE\_LOGGING\_LEVEL=DEBUG

- CORE\_PEER\_ID=cli

- CORE\_PEER\_ADDRESS=peer0.org1.example.com:7051

- CORE\_PEER\_LOCALMSPID=Org1MSP

- CORE\_PEER\_TLS\_ENABLED=true

- CORE\_PEER\_TLS\_CERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt

- CORE\_PEER\_TLS\_KEY\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key

- 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

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

volumes:

- /var/run/:/host/var/run/

- ../chaincode/go/:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go

- ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/

- ./scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/

- ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts

depends\_on:

- peer0.org1.example.com

extra\_hosts:

- "orderer.example.com:10.0.2.10"

- "peer0.org1.example.com:10.0.2.11"

- "peer0.org2.example.com:10.0.2.12"

[制代码](javascript:void(0);)

为peer0.org2.example.com配置文件,根据peer0.org1.example.com修改即可

cp docker-compose-peer0org1.yaml docker-compose-peer0org2.yaml

修改后的配置文件如下：

[制代码](javascript:void(0);)

# Copyright IBM Corp. All Rights Reserved.

#

# SPDX-License-Identifier: Apache-2.0

#

version: '2'

services:

peer0.org2.example.com:

container\_name: peer0.org2.example.com

environment:

- CORE\_LEDGER\_STATE\_STATEDATABASE=CouchDB

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_COUCHDBADDRESS=10.0.2.12:5984

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_USERNAME=admin

- CORE\_LEDGER\_STATE\_COUCHDBCONFIG\_PASSWORD=password

extends:

file: base/docker-compose-base.yaml

service: peer0.org2.example.com

extra\_hosts:

- "orderer.example.com:10.0.2.10"

cli:

container\_name: cli

image: hyperledger/fabric-tools

tty: true

environment:

- GOPATH=/opt/gopath

- CORE\_VM\_ENDPOINT=unix:///host/var/run/docker.sock

- CORE\_LOGGING\_LEVEL=DEBUG

- CORE\_PEER\_ID=cli

- CORE\_PEER\_ADDRESS=peer0.org2.example.com:7051

- CORE\_PEER\_LOCALMSPID=Org2MSP

- CORE\_PEER\_TLS\_ENABLED=true

- CORE\_PEER\_TLS\_CERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/server.crt

- CORE\_PEER\_TLS\_KEY\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/server.key

- 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

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

volumes:

- /var/run/:/host/var/run/

- ../chaincode/go/:/opt/gopath/src/github.com/hyperledger/fabric/examples/chaincode/go

- ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/

- ./scripts:/opt/gopath/src/github.com/hyperledger/fabric/peer/scripts/

- ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts

depends\_on:

- peer0.org2.example.com

extra\_hosts:

- "orderer.example.com:10.0.2.10"

- "peer0.org2.example.com:10.0.2.11"

- "peer0.org1.example.com:10.0.2.12"

[制代码](javascript:void(0);)

### 3.4分发配置文件

前面4步的操作，我们都是在orderer.example.com上完成的，接下来我们需要将这些文件分发到另外2台服务器上。Linux之间的文件传输，我们可以使用scp命令。

我先登录peer0.org1.example.com，将本地的e2e\_cli文件夹删除：

rm e2e\_cli –R

然后再登录到orderer服务器上，退回到examples文件夹，因为这样可以方便的把其下的e2e\_cli文件夹整个传到peer0.org1服务器上。

scp -r e2e\_cli lxh@10.0.2.11:/home/fabric/go/src/github.com/hyperledger/fabric/examples/

接下来继续使用scp命令将orderer上的文件夹传送给peer0.org2.example.com。

现在所有的配置文件都已经准备完毕了！

# 多机部署方法

## 一.启动orderer节点

在orderer服务器上运行：

cd ~/go/src/github.com/hyperledger/fabric/examples/e2e\_cli  
  
docker-compose -f docker-compose-orderer.yaml up -d

运行完毕后我们可以使用docker ps看到运行了一个名字为orderer.example.com的容器。

## 二.启动peer节点

### 2.1安装couchdb

我们要为每一个peer安装一个couchdb，先切换到peer0.org1服务器上。之前在获取hyperledger fabric镜像的时候已经获得了couchdb的镜像，执行docker images命令可以看到hyperledger/fabric-couchdb这个镜像,但是启动这个镜像的时候经常出现问题自动关闭，我们也可以再pull一个官方发布的couchdb。

docker pull klaemo/couchdb

couchDB在启动的时候需要指定一个本地文件夹映射成CouchDB的数据存储文件夹，所以我们可以在当前用户的目录下创建一个文件夹用于存放数据。

mkdir couchdb

下载完成后，我们只需要执行以下命令即可启用一个CouchDB的实例：

docker run -p 5984:5984 -d --name my-couchdb -e COUCHDB\_USER=admin -e COUCHDB\_PASSWORD=password -v ~/couchdb:/opt/couchdb/data klaemo/couchdb

启动后我们打开浏览器，访问peer0.org1的IP的5984端口的URL，peer0.org1的IP是10.0.2.11，那么URL是：

[http://10.0.2.11:5984/\_utils](http://192.168.100.129:5984/_utils)

这个时候我们就可以看到CouchDB的Web管理界面了。输入用户名admin密码password即可进入。

### 2.2启动peer节点和CLI容器

命令为：

cd ~/go/src/github.com/hyperledger/fabric/examples/e2e\_cli

docker-compose -f docker-compose-peer0org1.yaml up –d

运行完毕后我们使用docker ps应该可以看到3个正在运行的容器。

接下来切回到peer0.org2.example.com服务器。运行相同的命令：

mkdir couchdb

docker run -p 5984:5984 -d --name my-couchdb -e COUCHDB\_USER=admin -e COUCHDB\_PASSWORD=password -v ~/couchdb:/opt/couchdb/data klaemo/couchdb

cd ~/go/src/github.com/hyperledger/fabric/examples/e2e\_cli

docker-compose -f docker-compose-peer0org2.yaml up –d

现在我们整个Fabric网络已经成型，接下来是创建channel和运行ChainCode。

## 三.创建channel

我们切换到peer0.org1.example.com服务器上，使用该服务器上的cli来运行创建Channel和运行ChainCode的操作。先用以下命令进入CLI内部Bash：

docker exec -it cli bash

创建Channel的命令是peer channel create，我们前面创建Channel的配置区块时，指定了Channel的名字是mychannel，那么这里我们必须创建同样名字的Channel。

ORDERER\_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls true --cafile $ORDERER\_CA

执行该命令后，系统会提示：

2017-10-30 18:30:35.210 UTC [channelCmd] readBlock -> DEBU 020 Received block:0

系统会在cli内部的当前目录创建一个mychannel.block文件，这个文件非常重要，接下来其他节点要加入这个Channel就必须使用这个文件。

## 四.各个peer加入channel

还是在peer0.org1的CLI上，我们要将这个Peer加入mychannel就很简单，只需要运行如下命令：

peer channel join -b mychannel.block

系统返回消息：

2017-10-30 18:40:21.405 UTC [channelCmd] executeJoin -> INFO 006 Peer joined the channel!

修改cli的环境变量，使其指向peer0.org2

[制代码](javascript:void(0);)

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

[制代码](javascript:void(0);)

这样peer0.org1和peer0.org2就都已经加入channel了。

注：这一切都是我在peer0.org1的cli内完成的，这是由于加入channel依赖 [channel-ID].block文件，而且在peer0.org1服务器的cli内也留有了peer0.org2的证书文件，因此两个节点的加入都在一个cli内完成了。其实当一个节点创建channel以后可以把[channel-ID].block文件从容器内拷贝到主机，再分发给其他节点，这样其他节点就可以通过[channel-ID].block加入channel了。当cli容器被销毁时该容器下所有的文件都会被销毁，但是如果备份了[channel-ID].block，那么即使所有的peer和cli被销毁了，也可以再次通过[channel-ID].block加入channel、同步区块。

## 五.更新锚节点

关于AnchorPeer，其实我的每个组织只有一个peer节点，所以更新不更新锚节点也不重要。

对于Org1来说，peer0.org1是锚节点，我们需要切换到peer0.org1服务器上并更新锚节点：

[制代码](javascript:void(0);)

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 mychannel -f ./channel-artifacts/Org1MSPanchors.tx --tls true --cafile $ORDERER\_CA

[制代码](javascript:void(0);)

另外对于Org2，peer0.org2是锚节点，切换到peer0.org2服务器上然后执行如下命令：

[制代码](javascript:void(0);)

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 mychannel -f ./channel-artifacts/Org2MSPanchors.tx --tls true --cafile $ORDERER\_CA

[制代码](javascript:void(0);)

## 

## 六.chaincode的安装与运行

以上，整个Fabric网络和Channel都准备完毕，接下来我们来安装和运行ChainCode。社区联盟的chaincode代码已经在上一章贴上了。

### 6.1安装chaincode

在cli上为每一个peer安装链上代码

，用peer chaincode install命令可以安装指定的ChainCode并对其命名：

[制代码](javascript:void(0);)

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 chaincode install -n mycc -v 1.0 -p github.com/hyperledger/fabric/examples/chaincode/go/community

[制代码](javascript:void(0);)

安装的过程其实就是对CLI中指定的代码进行编译打包，并把打包好的文件发送到Peer，等待接下来的实例化。

切换到peer0.org2服务器，给peer0.org2安装链上代码：

[制代码](javascript:void(0);)

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/hyperledger/fabric/examples/chaincode/go/community

[制代码](javascript:void(0);)

### 6.2实例化chaincode

实例化链上代码主要是在Peer所在的机器上对前面安装好的链上代码进行包装，生成对应Channel的Docker镜像和Docker容器。并且在实例化时我们可以指定背书策略。我们在peer0.org1服务器的cli内运行以下命令完成实例化：

[制代码](javascript:void(0);)

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 chaincode instantiate -o orderer.example.com:7050 --tls true --cafile $ORDERER\_CA -C mychannel -n mycc -v 1.0 -c '{"Args":["init"]}' -P "OR ('Org1MSP.member','Org2MSP.member')"

[制代码](javascript:void(0);)

使用docker ps可以看到有新的容器 dev-peer0.org1.example.com-mycc-1.0正在运行。

### 6.3在一个peer发起交易

我们可以调用initLedger功能，该功能会初始化两个帖子。对应的代码：

peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile $ORDERER\_CA -C mychannel -n mycc -c '{"Args":["initLedger"]}'

### 6.4在另一个节点上查询帖子

前面的操作都是在org1下面做的，那么处于同一个区块链（同一个Channel下）的org2，是否会看org1的更改呢？切换到peer0.org2服务器，由于mycc已经在前面org1的时候实例化了，也就是说对应的区块已经生成了，所以在org2不能再次初始化。我们直接运行查询命令：

[制代码](javascript:void(0);)

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 query -C mychannel -n mycc -c '{"Args":["queryPost","POST1"]}'

[制代码](javascript:void(0);)

因为peer0.org2也需要生成Docker镜像，创建对应的容器，才能通过容器返回结果，所以等的时间会稍长一点。我们回到Ubuntu终端，执行docker ps，可以看到peer0.org2也多了一个容器：dev-peer0.org2.example.com-mycc-1.0