# 环境准备

## Fabric和fabric-samples的版本为1.1.0

git checkout -b release-1.1 origin/release-1.1

## 设置fabric-samples的docker images的版本为1.1.0

进入fabric-samples/first-network目录，修改.env文件IMAGE\_TAG=1.1.0

## 下载命令行处理 JSON 的工具jq

apt-get -y update && apt-get -y install jq

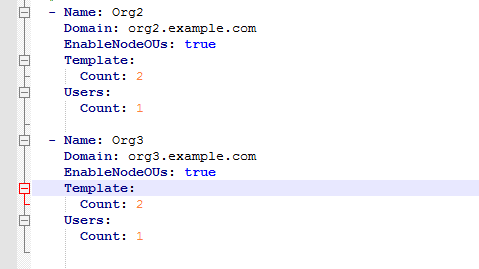
# 在Fabric网络中添加org

这此部分实现在Fabric网络中添加一个新组织，在添加完组织后使用此组织新建一个channel，最后执行chaincode，若全部成功则表示添加新组织成功。

为了方便，测试中的证书和密钥文件都通过Cryptogen工具生成，在生产环境中证书密钥文件都是通过CA服务器生成。配置文件还是通过Configtxgen工具生成。

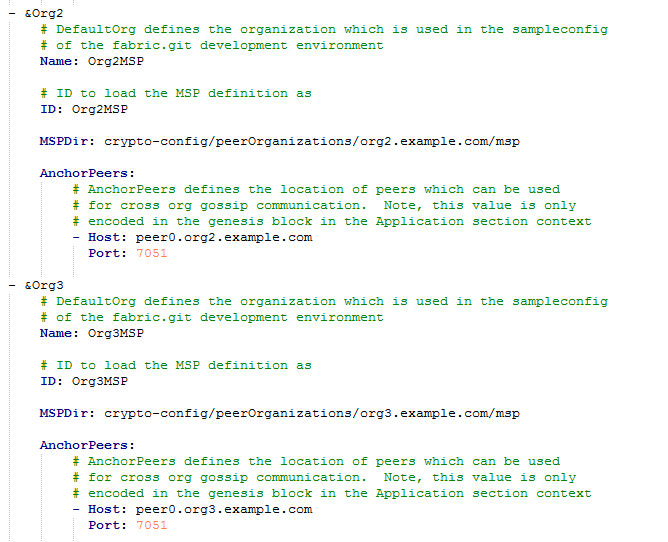
## 修改证书和交易的配置文件

进入fabric-samples/first-network目录，修改生成证书的配置文件crypto-config.yaml，添加新组织org3。如下图：

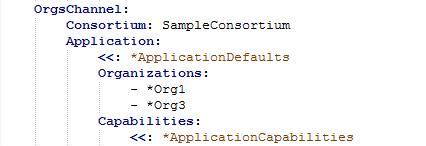


修改交易配置文件，声明新组织org3，再指定org3的证书路径，但是并不将org3加入Fabric网络，也就是在创世channel配置文件中不添加org3。然后再新增一个channel的配置，新增的channel加入的组织中有org3，用于测试结果。具体文件的修改部分如下图：

新增org3声明和指定org3的证书路径：



新增channel的配置文件：



## 生成证书和交易文件

参照《搭建第一个fabric网络》，先生成证书和密钥文件，然后生成创世区块的配置文件。之后生成需要测试的新增channel的配置文件，命令如下：

生成证书和配置文件：

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

export FABRIC\_CFG\_PATH=$PWD

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

export CHANNEL\_NAME=orgmychannel

../bin/configtxgen -profile OrgsChannel -outputCreateChannelTx ./channel-artifacts/orgchannel.tx -channelID $CHANNEL\_NAME

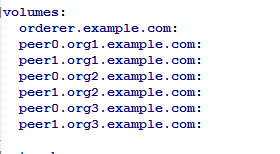
## 在网络中新增org3两个节点

修改网络节点文件，新增org3两个节点，并根据生成的证书文件配置节点的具体信息。

在base/docker-compose-base.yaml文件内新增org3的两个节点信息。

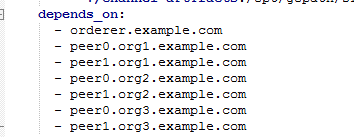


在docker-compose-cli.yaml文件内新增org3两个节点信息。





在cli容器的依赖中添加两个节点。



## 启动Fabric网络，测试新增org3

启动网络后，用新增的org3组织的管理员身份登录peer0.org3.example.com节点，然后创建新增的channel进行测试。命令如下：

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

docker exec -it cli bash

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp

CORE\_PEER\_ADDRESS=peer0.org3.example.com:7051

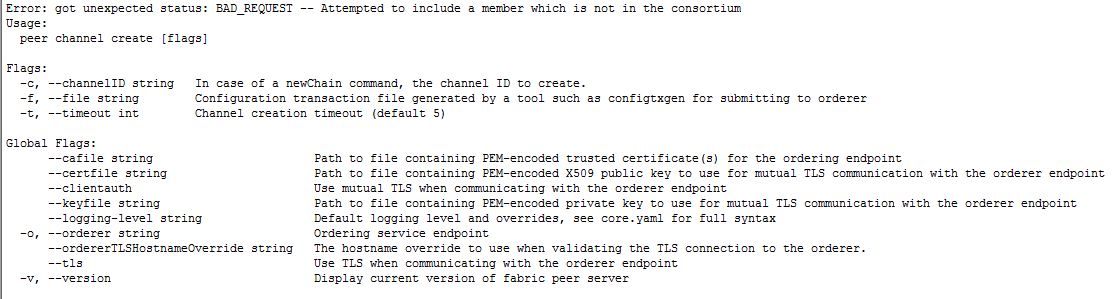
CORE\_PEER\_LOCALMSPID="Org3MSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt

export CHANNEL\_NAME=orgmychannel

peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/orgchannel.tx --tls $CORE\_PEER\_TLS\_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

在创建新的channel时报错，提示进行的操作正在使用网络中不存在的成员。测试结果正确，因为此时org3虽然在配置文件中声明，并且在创建的channel中又使用到了org3，但是还未添加org3至fabric网络。后面将执行如何将已存在的org3组织动态的加入fabric网络中。报错如下图：



## 在网络配置文件添加org

此小节详细描述获取解析网络配置文件，然后在获取的配置文件中添加org3的配置信息，最后形成新的配置文件的具体操作。因为在cli容器里下载获取配置文件，为了方便cli中的文件到主机查看，后面的对文件的操作均在主机和docker容器共同可见使用的channel-artifacts文件夹中。

### 生成org3的证书和配置文件

在前面步骤中，有一行命令如下：

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

因为消费了已经添加org3的crypto-config.yaml文件，所以在执行此文件时已经生成我们所需的证书文件，目录为：first-network/crypto-config/peerOrganizations/org3.example.com。所以在这里不需要再次生成，后面直接使用。

生成配置文件的命令如下：

export FABRIC\_CFG\_PATH=$PWD

../bin/configtxgen -printOrg Org3MSP -profile ./configtx.yaml > channel-artifacts/org3.json

此命令生成org3的配置文件org3.json并存于channel-artifacts。

### 获取网络创世区块配置文件

进入cli容器并切换至channel-artifacts目录

docker exec -it cli bash

cd channel-artifacts

设置操作节点为orderer节点，设置证书为orderer节点对应的证书，然后才有权限获取网络的创世区块配置文件。

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

export CORE\_PEER\_LOCALMSPID="OrdererMSP"

export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp

export 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

获取创世区块的配置文件，将获取的到创世区块二进制文件存于config\_block.pb。

peer channel fetch config config\_block.pb -o orderer.example.com:7050 -c testchainid --tls --cafile $ORDERER\_CA

### 修改创世区块配置文件新增org3配置文件

将创世区块二进制文件转换为json格式文件

configtxlator proto\_decode --input config\_block.pb --type common.Block>config\_old.json

用jq将生成的json文件转换为所需格式的创世区块json格式的配置文件。

jq .data.data[0].payload.data.config config\_old.json >config.json

在创世区块配置文件中添加org3的配置文件的数据

jq -s '.[0] \* {"channel\_group":{"groups":{"Consortiums":{"groups": {"SampleConsortium":{"groups":{"Org3MSP":.[1]}}}}}}}' config.json org3.json > modified\_config.json

### 将创世区块配置文件和修改后转为 protobuf 格式

将修改后的创世区块文件config.json和添加org3后的文件modified\_config.json 转为 protobuf 格式

configtxlator proto\_encode --input config.json --type common.Config > original\_config.pb

configtxlator proto\_encode --input modified\_config.json --type common.Config > modified\_config.pb

### 计算两个pb文件之后得出差异文件

configtxlator compute\_update --channel\_id testchainid --original original\_config.pb --updated modified\_config.pb > config\_update.pb

### 将差异文件转换为最终的新配置文件

需要为差异文件的数据进行补全，先将pb的二进制文件转化为json文件，然后添加信息，之后再将添加后的json文件转换为pb文件，命令如下：

configtxlator proto\_decode --input config\_update.pb --type common.ConfigUpdate > config\_update.json

echo '{"payload":{"header":{"channel\_header":{"channel\_id":"testchainid", "type":2}},"data":{"config\_update":'$(cat config\_update.json)'}}}' > config\_update\_in\_envelope.json

configtxlator proto\_encode --input config\_update\_in\_envelope.json --type common.Envelope > org3\_update\_in\_envelope.pb

## 为新配置签名

再次设置orderer节点证书位置等信息

export CORE\_PEER\_LOCALMSPID="OrdererMSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

export CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/users/Admin@example.com/msp

export 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

orderer节点对新配置签名

peer channel signconfigtx -f org3\_update\_in\_envelope.pb

## 提交签名后的配置交易至 orderer

peer channel update -f org3\_update\_in\_envelope.pb -c testchainid -o orderer.example.com:7050 --tls --cafile $ORDERER\_CA

## 测试结果

再次执行命令测试，检测org3所在的channel是否能创建成功，命令如下：

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp

CORE\_PEER\_ADDRESS=peer0.org3.example.com:7051

CORE\_PEER\_LOCALMSPID="Org3MSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt

export CHANNEL\_NAME=orgmychannel

peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/orgchannel.tx --tls $CORE\_PEER\_TLS\_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

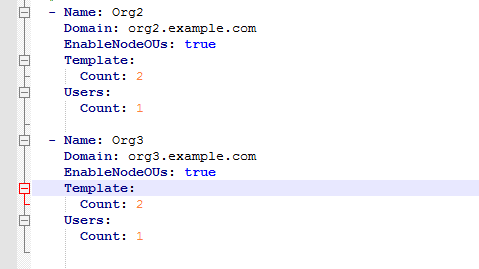
若channel创建成功，则表明org3在网络中动态添加成功。

# 在已有channel中添加org

此部分描述的是在已经创建好的channel中添加org（组织），具体的操作方式很多与在网络中添加org相似。网络中是向ID为的channel添加组织，这里是向ID为的channel添加组织。网络中修改后的文件只需给orderer节点签名，这里则需要给channel中所有的org签名。

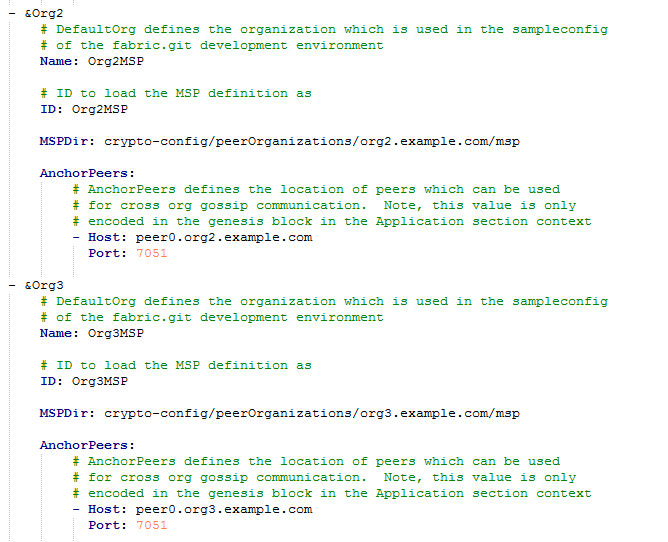
## 修改证书和交易的配置文件

进入fabric-samples/first-network目录，修改生成证书的配置文件crypto-config.yaml，添加新组织org3。如下图：



修改交易配置文件，声明新组织org3，再指定org3的证书路径，但是并不将org3加入Fabric网络，也就是在创世channel配置文件中不添加org3。然后再新增一个channel的配置，新增的channel加入的组织中有org3，用于测试结果。具体文件的修改部分如下图：

新增org3声明和指定org3的证书路径：



## 生成证书和交易文件

参照《搭建第一个fabric网络》，先生成证书和密钥文件，然后生成创世区块的配置文件，之后生成mychannel的配置文件，命令如下：

生成证书和配置文件：

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

export FABRIC\_CFG\_PATH=$PWD

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

export CHANNEL\_NAME=mychannel

../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL\_NAME

../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org1MSP

../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org2MSP

## 在网络中新增org3两个节点

此部分和在Fabric网络中添加org中的操作一样。在此不再重复

## 启动Fabric网络，测试新增org3

这里测试还未进行任何操作的情况下，使用org3的身份权限情况下进入mychannel操作chaincod会出现什么。首先是启动网络，然后使用org1身份创建mychannel和创建、初始化、执行chaincode。

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

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

docker exec -it cli bash

export CHANNEL\_NAME=mychannel

peer channel create -o orderer.example.com:7050 -c $CHANNEL\_NAME -f ./channel-artifacts/channel.tx --tls $CORE\_PEER\_TLS\_ENABLED --cafile /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 join -b mychannel.block

peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go

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')"

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

执行成功显示查询的结果，现在切换至org3的peer0节点，使用org3用户身份连接mychannel然后安装chaincode执行chaincode。

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.example.com/users/Admin@org3.example.com/msp

CORE\_PEER\_ADDRESS=peer0.org3.example.com:7051

CORE\_PEER\_LOCALMSPID="Org3MSP"

CORE\_PEER\_TLS\_ROOTCERT\_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.example.com/peers/peer0.org3.example.com/tls/ca.crt

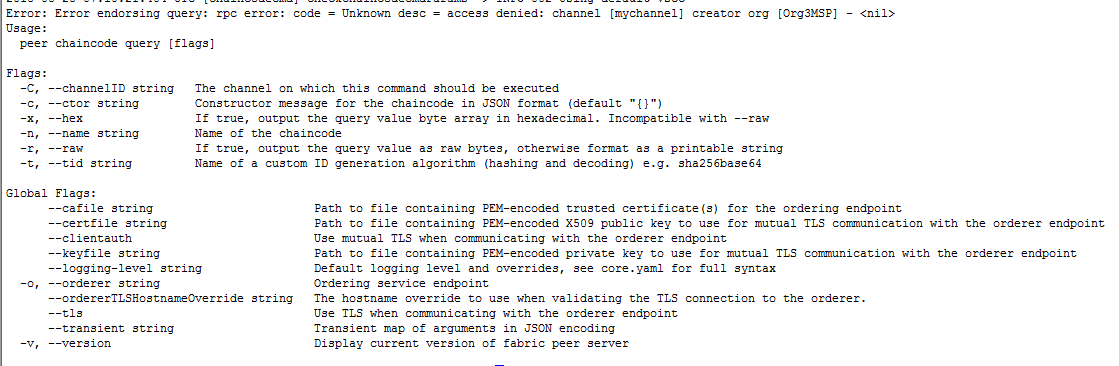
export CHANNEL\_NAME=mychannel

peer channel join -b mychannel.block

peer chaincode install -n mycc -v 1.0 -p github.com/chaincode/chaincode\_example02/go

peer chaincode query -C $CHANNEL\_NAME -n mycc -c '{"Args":["query","a"]}'

执行查询语句之后，报错如下图，表明mychannel中还没有or3组织，测试成功。



## 在mychannel配置文件添加org

此小节也可参考在Fabric网络中添加org中的部分，具体的操作只是将channel的ID由fabric网络的testchainid修改为mychannel。相同的部分我就不重复描述，着重描述不同的地方。

### 生成org3的证书和配置文件

参考在Fabric网络中添加org

### 获取网络创世区块配置文件

进入cli容器并切换至channel-artifacts目录

docker exec -it cli bash

cd channel-artifacts

设置操作节点为orderer节点或者设置为mychannel配置文件中的组织对应的节点，设置证书为对应节点对应的证书，然后才有权限获取网络的创世区块配置文件。如设置为mychannel中包含的org2的节点和证书：

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

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

export 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

获取mychannel的配置文件，将获取的到mychannel区块二进制文件存于config\_block.pb。

peer channel fetch config config\_block.pb -o orderer.example.com:7050 -c mychannel --tls --cafile $ORDERER\_CA

### 修改**mychannel**区块配置文件新增org3配置文件

将mychannel区块二进制文件转换为json格式文件

configtxlator proto\_decode --input config\_block.pb --type common.Block>config\_old.json

用jq将生成的json文件转换为所需格式的mychannel区块json格式的配置文件。

jq .data.data[0].payload.data.config config\_old.json >config.json

在创世区块配置文件中添加org3的配置文件的数据

jq -s '.[0] \* {"channel\_group":{"groups":{"Application":{"groups": {"Org3MSP":.[1]}}}}}' config.json org3.json > modified\_config.json

### 将创世区块配置文件和修改后转为 protobuf 格式

configtxlator proto\_encode --input config.json --type common.Config > original\_config.pb

configtxlator proto\_encode --input modified\_config.json --type common.Config > modified\_config.pb

### 计算两个pb文件之后得出差异文件

configtxlator compute\_update --channel\_id mychannel --original original\_config.pb --updated modified\_config.pb > config\_update.pb

### 将差异文件转换为最终的新配置文件

configtxlator proto\_decode --input config\_update.pb --type common.ConfigUpdate > config\_update.json

echo '{"payload":{"header":{"channel\_header":{"channel\_id":"mychannel", "type":2}},"data":{"config\_update":'$(cat config\_update.json)'}}}' > config\_update\_in\_envelope.json

configtxlator proto\_encode --input config\_update\_in\_envelope.json --type common.Envelope > org3\_update\_in\_envelope.pb

## 为新配置签名

为配置交易签名，需要 channel 中的大多数 Org 对其进行签名。对于 mychannel 而言，已有了 org1，org2，因此新增 org3 时需要 org1、org2 都签名。签名操作于 cli 中完成，通过更改环境变量，改变签名者的身份。切换至 Org1进行签名，Org1 有两个 peer，需采用Anchor Peer，签名完之后切换至Org2进行签名，也采用Anchor Peer。

切换至peer0.org1.example.com命令如下：

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

切换至peer0.org2.example.com命令如下：

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

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

每次切换之后都要进行签名，命令如下：

peer channel signconfigtx -f org3\_update\_in\_envelope.pb

## 提交签名后的配置交易至 orderer

export 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 update -f org3\_update\_in\_envelope.pb -c mychannel -o orderer.example.com:7050 --tls --cafile $ORDERER\_CA

## 测试结果

再切换至org3的节点执行chaincode执行成功，至此在已创建的channel中添加org成功。

# 在chaincode中背书策略添加org

因为要在chaincode的中添加org，需要改变chaincode的背书策略，因此在这里是用升级chaincode的方式来实现添加org。升级chaincode需要chaincode的创建者来执行并且只执行一次，所以我们使用实例化chaincode的peer和用户来执行升级操作，命令如下：

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

然后执行升级chaincode的操作，需要先安装chaincode，然后进行升级操作，升级中指定的背书策略中添加org3。安装chaincode的操作需要在所有节点进行，否则在升级操作之后，chaincode则无法正常执行。

安装chaincode命令：

peer chaincode install -n mycc -v 2.0 -p github.com/chaincode/chaincode\_example02/go

升级chaincode命令：

peer chaincode upgrade -o orderer.example.com:7050 --tls $CORE\_PEER\_TLS\_ENABLED --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 mychannel -n mycc -v 2.0 -c '{"Args":["init","a","90","b","210"]}' -P "OR ('Org1MSP.peer','Org2MSP.peer','Org3MSP.peer')"