# Solo多机多节点部署

## 1. 准备工作

所有的节点分离部署, 每台主机上有一个节点, 节点的分布如下表:

| 名称 | IP | Hostname | 组织机构 |
| --- | --- | --- | --- |
| orderer | 192.168.247.129 | orderer.itcast.com | Orderer |
| peer0 | 192.168.247.141 | peer0.orggo.com | OrgGo |
| peer1 | 192.168.247.142 | peer1.orggo.com | OrgGo |
| peer0 | 192.168.247.131 | peer0.orgcpp.com | OrgCpp |
| peer1 | 192.168.247.1451 | peer1.orgcpp.com | OrgCpp |

> 下面的操作在任意一台主机上做都可以, 下面的例子中, 生成证书和创始块、通道文件操作是在 `Orderer节点` 对应的主机上进行的。

### 1.1 准备工作 - 创建工作目录 #

```
# N台主机需要创建一个名字相同的工作目录， 该工作目录名字自己定， 切记名字一定要相同
# 192.168.247.129
$ mkdir ~/testwork
# 192.168.247.141
$ mkdir ~/testwork
# 192.168.247.131
$ mkdir ~/testwork
# 192.168.247.142
$ mkdir ~/testwork
# 192.168.247.145
$ mkdir ~/testwork
```

### 1.2 生成组织节点和用户证书 #

- **编写配置文件**

```
# crypto-config.yaml -> 名字可以改， 一般起名为crypto-config.yaml

OrdererOrgs:
  # ---------------------------------------------------------------------------
  # Orderer
  # ---------------------------------------------------------------------------
  - Name: Orderer
```

```
    Domain: test.com
    Specs:
      - Hostname: orderer

PeerOrgs:
  # ---------------------------------------------------------------------------
  # Org1
  # ---------------------------------------------------------------------------
  - Name: OrgGo
    Domain: orggo.test.com
    EnableNodeOUs: false
    Template:
      Count: 2
    Users:
      Count: 1
  # ---------------------------------------------------------------------------
  # Org2: See "Org1" for full specification
  # ---------------------------------------------------------------------------
  - Name: OrgCpp
    Domain: orgcpp.test.com
    EnableNodeOUs: false
    Template:
      Count: 2
    Users:
      Count: 1
```

- 使用 `cryptogen` 生成证书

```
$ cryptogen generate --config=crypto-config.yaml
```

## 1.3 生成通道文件和创始块文件 #

- 编写配置文件, 名字为 `configtx.yaml` , 该名字不能改, 是固定的.

```
# configtx.yaml -> 名字不能变
---
################################################################################
#
#   Section: Organizations
#
################################################################################
Organizations:
    - &OrdererOrg
        Name: OrdererOrg
        ID: OrdererMSP
        MSPDir: ./crypto-config/ordererOrganizations/test.com/msp

    - &OrgGo
        Name: OrgGoMSP
        ID: OrgGoMSP
        MSPDir: ./crypto-config/peerOrganizations/orggo.test.com/msp
        AnchorPeers:
            - Host: peer0.orggo.test.com
```

```
                Port: 7051

    - &OrgCpp
        Name: OrgCppMSP
        ID: OrgCppMSP
        MSPDir: ./crypto-config/peerOrganizations/orgcpp.test.com/msp
        AnchorPeers:
            - Host: peer0.orgcpp.test.com
              Port: 7051

################################################################################
#
#   SECTION: Capabilities
#
################################################################################
Capabilities:
    Global: &ChannelCapabilities
        V1_1: true
    Orderer: &OrdererCapabilities
        V1_1: true
    Application: &ApplicationCapabilities
        V1_2: true

################################################################################
#
#   SECTION: Application
#
################################################################################
Application: &ApplicationDefaults
    Organizations:

################################################################################
#
#   SECTION: Orderer
#
################################################################################
Orderer: &OrdererDefaults
    # Available types are "solo" and "kafka"
    OrdererType: solo
    Addresses:
        - orderer.test.com:7050
    BatchTimeout: 2s
    BatchSize:
        MaxMessageCount: 10
        AbsoluteMaxBytes: 99 MB
        PreferredMaxBytes: 512 KB
    Kafka:
        Brokers:
            - 127.0.0.1:9092
    Organizations:

################################################################################
#
```

```
#   Profile
#
################################################################################
Profiles:
    TwoOrgsOrdererGenesis:
        Capabilities:
            <<: *ChannelCapabilities
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *OrgGo
                    - *OrgCpp
    TwoOrgsChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *OrgGo
                - *OrgCpp
            Capabilities:
                <<: *ApplicationCapabilities
```

- 通过命令 `configtxgen` 生成创始块和通道文件

```
# 我们先创建一个目录 channel-artifacts 存储生成的文件，目的是为了和后边的配置文件模板的配置项保持一致
$ mkdir channel-artifacts
# 生成通道文件
$ configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block
# 生成创始块文件
$ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID testchannel
```

# 2 部署 orderer 排序节点

## 2.1 编写配置文件 #

> 编写启动 `orderer` 节点容器使用的配置文件 - `docker-compose.yaml`

```
version: '2'

services:

```

```
  orderer.test.com:
    container_name: orderer.test.com
    image: hyperledger/fabric-orderer:latest
    environment:
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=testwork_default
      - ORDERER_GENERAL_LOGLEVEL=INFO
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LISTENPORT=7050
      - 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/test.com/orderers/orderer.test.com/msp:/var/hyperledger/orderer/msp
    - ./crypto-config/ordererOrganizations/test.com/orderers/orderer.test.com/tls/:/var/hyperledger/orderer/tls
    networks:
        default:
          aliases:
            - testwork  # 这个名字使用当前配置文件所在的目录 的名字
    ports:
      - 7050:7050
```

注意的细节:

- networks 的名字要跟 当前配置文件所在的目录名 相同
- 环境变量 CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=testwork_default 的名字是 网络名_default

## 2.2 启动orderer容器 #

通过上面编写好的docker-compose配置文件就可以启动 orderer 容器了

```
$ docker-compose up -d
Creating network "testwork_default" with the default driver
Creating orderer.test.com ... done
# 检测是否启动成功
$ docker-compose  ps
      Name         Command   State           Ports
-------------------------------------------------------------
orderer.test.com   orderer   Up      0.0.0.0:7050->7050/tcp
```

# 3 部署 peer0.orggo 节点

## 3.1 准备工作 #

- 切换到 `peer0.orggo` 主机 - `192.168.247.141`
- 进入到工作目录中:

```
$ cd ~/testwork
```

- 拷贝文件

> 将 `orderer` 节点所在宿主机上生成的 `crypto-config` 和 `channel-artifacts` 目录拷贝到当前 `testwork` 目录中。
>
> 我们可以通过 `scp` 命令实现远程拷贝, 从 `orderer` 节点宿主机拷贝到当前 `peer0.orggo` 节点.
>
> - orderer节点宿主机 `IP: 192.168.247.129` , 登录用户名: `itcast`

```
# 通过scp命令远程拷贝
# -r : 表示要拷贝的是目录, 执行递归操作
# itcast@192.168.247.129:/home/itcast/testwork/channel-artifacts
#   itcast@192.168.247.129: 从192.168.247.129上拷贝数据, 登录用户名为itcast
#   /home/itcast/testwork/channel-artifacts: 要拷贝192.168.247.129上itcast用户的哪个目录
#   ./ : 远程目录拷贝到本地的什么地方
$ scp -r itcast@192.168.247.129:/home/itcast/testwork/channel-artifacts  ./
$ scp -r itcast@192.168.247.129:/home/itcast/testwork/crypto-config  ./
# 查看拷贝结果
$ tree ./ -L 1
.
├── channel-artifacts
└── crypto-config
```

## 3.2 编写 配置文件 #

> 编写启动 `peer0-orggo` 节点的配置文件 - `docker-compose.yaml`

```
# docker-compose.yaml
version: '2'

services:
    peer0.orggo.test.com:
      container_name: peer0.orggo.test.com
      image: hyperledger/fabric-peer:latest
      environment:
        - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
        - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=testwork_default
        - CORE_LOGGING_LEVEL=INFO
        #- CORE_LOGGING_LEVEL=DEBUG
        - CORE_PEER_GOSSIP_USELEADERELECTION=true
        - CORE_PEER_GOSSIP_ORGLEADER=false
        - CORE_PEER_PROFILE_ENABLED=true
```

```
16          - CORE_PEER_LOCALMSPID=OrgGoMSP
17          - CORE_PEER_ID=peer0.orggo.test.com
18          - CORE_PEER_ADDRESS=peer0.orggo.test.com:7051
19          - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.orggo.test.com:7051
20          - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.orggo.test.com:7051
21          # TLS
22          - CORE_PEER_TLS_ENABLED=true
23          - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
24          - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
25          - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
26        volumes:
27          - /var/run/:/host/var/run/
28          - ./crypto-
    config/peerOrganizations/orggo.test.com/peers/peer0.orggo.test.com/msp:/etc/hyperledger/fa
    bric/msp
29          - ./crypto-
    config/peerOrganizations/orggo.test.com/peers/peer0.orggo.test.com/tls:/etc/hyperledger/fa
    bric/tls
30        working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
31        command: peer node start
32        networks:
33          default:
34            aliases:
35              - testwork
36        ports:
37          - 7051:7051
38          - 7053:7053
39        extra_hosts:  # 声明域名和IP的对应关系
40          - "orderer.test.com:192.168.247.129"
41          - "peer0.orgcpp.test.com:192.168.247.131"
42
43      cli:
44        container_name: cli
45        image: hyperledger/fabric-tools:latest
46        tty: true
47        stdin_open: true
48        environment:
49          - GOPATH=/opt/gopath
50          - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
51          #- CORE_LOGGING_LEVEL=DEBUG
52          - CORE_LOGGING_LEVEL=INFO
53          - CORE_PEER_ID=cli
54          - CORE_PEER_ADDRESS=peer0.orggo.test.com:7051
55          - CORE_PEER_LOCALMSPID=OrgGoMSP
56          - CORE_PEER_TLS_ENABLED=true
57          -
    CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrga
    nizations/orggo.test.com/peers/peer0.orggo.test.com/tls/server.crt
58          -
    CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrgan
    izations/orggo.test.com/peers/peer0.orggo.test.com/tls/server.key
```

```
59          -
    CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peer
    Organizations/orggo.test.com/peers/peer0.orggo.test.com/tls/ca.crt
60          -
    CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrga
    nizations/orggo.test.com/users/Admin@orggo.test.com/msp
61        working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
62        command: /bin/bash
63        volumes:
64            - /var/run/:/host/var/run/
65            - ./chaincode/:/opt/gopath/src/github.com/chaincode
66            - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
67            - ./channel-
    artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts
68        depends_on:   # 启动顺序
69          - peer0.orggo.test.com
70
71        networks:
72            default:
73              aliases:
74                - testwork
75        extra_hosts:
76          - "orderer.test.com:192.168.247.129"
77          - "peer0.orggo.test.com:192.168.247.141"
78          - "peer0.orgcpp.test.com:192.168.247.131"
```

## 3.3 启动容器 #

- 启动容器

```
1   $ docker-compose up -d
2   Creating network "testwork_default" with the default driver
3   Creating peer0.orgGo.test.com ... done
4   Creating cli                  ... done
5   # 查看启动状态
6   $ docker-compose ps
7           Name               Command       State                       Ports

8   ------------------------------------------------------------------------------------
    ---------
9   cli                    /bin/bash         Up

10  peer0.orgGo.test.com   peer node start   Up      0.0.0.0:7051->7051/tcp, 0.0.0.0:7053-
    >7053/tcp
```

## 3.4 对peer0.orggo节点的操作 #

- 进入到客户端容器中

```
1   $ docker exec -it cli bash
```

- 创建通道

```
$ peer channel create -o orderer.test.com:7050 -c testchannel -f ./channel-artifacts/channel.tx --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/test.com/msp/tlscacerts/tlsca.test.com-cert.pem
$ ls
channel-artifacts  crypto  `testchannel.block`  --> 生成的通道块文件
```

- 将当前节点加入到通道中

```
$ peer channel join -b testchannel.block
```

- 安装链码

```
$ peer chaincode install -n testcc -v 1.0 -l golang -p github.com/chaincode
```

- 初始化链码

```
$ peer chaincode instantiate -o orderer.test.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/test.com/msp/tlscacerts/tlsca.test.com-cert.pem -C testchannel -n testcc -v 1.0 -l golang -c '{"Args":["init","a","100","b","200"]}' -P "AND ('OrgGoMSP.member', 'OrgCppMSP.member')"
```

- 查询

```
$ peer chaincode query -C testchannel -n testcc -c '{"Args":["query","a"]}'
$ peer chaincode query -C testchannel -n testcc -c '{"Args":["query","b"]}'
```

- 将生成的通道文件 `testchannel.block` 从cli容器拷贝到宿主机

```
# 从客户端容器退出到宿主机
$ exit
# 拷贝操作要在宿主机中进行
$ docker cp cli:/opt/gopath/src/github.com/hyperledger/fabric/peer/testchannel.block ./
```

# 4 部署 peer0.orgcpp 节点

## 4.1 准备工作 #

- 切换到 `peer0.orgcpp` 主机 - `192.168.247.131`
- 进入到工作目录

```
$ cd ~/testwork
```

- 远程拷贝文件

```
# 从主机192.168.247.141的zoro用户下拷贝目录crypto-config到当前目录下
$ scp -r zoro@192.168.247.141:/home/zoro/testwork/crypto-config  ./
# 链码拷贝
$ scp -r zoro@192.168.247.141:/home/zoro/testwork/chaincode  ./
# 从主机192.168.247.141的zoro用户下拷贝文件testchannel.block到当前目录下
$ scp zoro@192.168.247.141:/home/zoro/testwork/testchannel.block  ./
# 查看结果
$ tree ./ -L 1
./
├── chaincode
├── crypto-config
└── testchannel.block
```

- 为了方便操作可以将 通道块文件 放入到客户端容器挂载的目录中

```
# 创建目录
$ mkdir channel-artifacts
# 移动
$ mv testchannel.block channel-artifacts/
```

## 4.2 编写配置文件 #

> 编写启动 peer0.orgcpp 节点的配置文件 docker-compose.yaml

```
# docker-compose.yaml
version: '2'
services:
    peer0.orgcpp.test.com:
      container_name: peer0.orgcpp.test.com
      image: hyperledger/fabric-peer:latest
      environment:
        - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
        - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=testwork_default
        - CORE_LOGGING_LEVEL=INFO
        #- CORE_LOGGING_LEVEL=DEBUG
        - CORE_PEER_GOSSIP_USELEADERELECTION=true
        - CORE_PEER_GOSSIP_ORGLEADER=false
        - CORE_PEER_PROFILE_ENABLED=true
        - CORE_PEER_LOCALMSPID=OrgCppMSP
        - CORE_PEER_ID=peer0.orgcpp.test.com
        - CORE_PEER_ADDRESS=peer0.orgcpp.test.com:7051
        - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.orgcpp.test.com:7051
        - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.orgcpp.test.com:7051
        # TLS
        - CORE_PEER_TLS_ENABLED=true
        - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
        - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
        - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
      volumes:
        - /var/run/:/host/var/run/
```

```
        - ./crypto-config/peerOrganizations/orgcpp.test.com/peers/peer0.orgcpp.test.com/msp:/etc/hyperledger/fabric/msp
        - ./crypto-config/peerOrganizations/orgcpp.test.com/peers/peer0.orgcpp.test.com/tls:/etc/hyperledger/fabric/tls
      working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
      command: peer node start
      networks:
        default:
          aliases:
            - testwork
      ports:
        - 7051:7051
        - 7053:7053
      extra_hosts:  # 声明域名和IP的对应关系
        - "orderer.test.com:192.168.247.129"
        - "peer0.orggo.test.com:192.168.247.141"

    cli:
      container_name: cli
      image: hyperledger/fabric-tools:latest
      tty: true
      stdin_open: true
      environment:
        - GOPATH=/opt/gopath
        - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
        #- CORE_LOGGING_LEVEL=DEBUG
        - CORE_LOGGING_LEVEL=INFO
        - CORE_PEER_ID=cli
        - CORE_PEER_ADDRESS=peer0.orgcpp.test.com:7051
        - CORE_PEER_LOCALMSPID=OrgCppMSP
        - CORE_PEER_TLS_ENABLED=true
        - CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/orgcpp.test.com/peers/peer0.orgcpp.test.com/tls/server.crt
        - CORE_PEER_TLS_KEY_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/orgcpp.test.com/peers/peer0.orgcpp.test.com/tls/server.key
        - CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/orgcpp.test.com/peers/peer0.orgcpp.test.com/tls/ca.crt
        - CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/orgcpp.test.com/users/Admin@orgcpp.test.com/msp
      working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
      command: /bin/bash
      volumes:
          - /var/run/:/host/var/run/
          - ./chaincode/:/opt/gopath/src/github.com/chaincode
          - ./crypto-config:/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
          - ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/fabric/peer/channel-artifacts
```

```
    depends_on:   # 启动顺序
      - peer0.orgcpp.test.com

    networks:
        default:
          aliases:
            - testwork
    extra_hosts:
      - "orderer.test.com:192.168.247.129"
      - "peer0.orggo.test.com:192.168.247.141"
      - "peer0.orgcpp.test.com:192.168.247.131"
```

> 注意: 该配置文件中已经将 映射删掉了

## 4.3 启动当前节点 #

- 启动客户端容器

```
$ docker-compose up -d
Creating network "testwork_default" with the default driver
Creating peer0.orgcpp.test.com ... done
Creating cli                   ... done
# 查看启动情况
$ docker-compose ps
        Name               Command       State                      Ports

-------------------------------------------------------------------------------------
cli                    /bin/bash         Up

peer0.orgcpp.test.com   peer node start   Up      0.0.0.0:7051->7051/tcp,
0.0.0.0:7053->7053/tcp
```

## 4.4 对peer0.orgcpp节点的操作 #

- 进入到操作该节点的客户端中

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

- 加入到通道中

```
$ peer channel join -b ./channel-artifacts/testchannel.block
```

- 安装链码

```
$ peer chaincode install -n testcc -v 1.0 -l golang -p github.com/chaincode
```

- 查询

```
$ peer chaincode query -C testchannel -n testcc -c '{"Args":["query","a"]}'
$ peer chaincode query -C testchannel -n testcc -c '{"Args":["query","b"]}'
```

- 交易

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

# 5. 其余节点的部署

> 关于其余节点的部署, 在此不再过多赘述, 部署方式请参考 第 4 章内容, 步骤是完全一样的

# 6. 链码的打包

> 我们在进行多机多节点部署的时候, 所有的peer节点都需要安装链码, 有时候会出现链码安装失败的问题, 提示链码的指纹（哈希）不匹配，我们可以通过以下方法解决

1. 通过客户端在第1个peer节点中安装好链码之后， 将链码打包

```
$ peer chaincode package -n testcc -p github.com/chaincode -v 1.0 mycc.1.0.out
    -n: 链码的名字
    -p: 链码的路径
    -v: 链码的版本号
    -mycc.1.0.out: 打包之后生成的文件
```

2. 将打包之后的链码从容器中拷贝出来

```
$ docker cp cli:/xxxx/mycc.1.0.out ./
```

3. 将得到的打包之后的链码文件拷贝到其他的peer节点上
4. 通过客户端在其他peer节点上安装链码

```
$ peer chaincode install mycc.1.0.out
```