目录

一.通过自动化脚本启动服务

自动化脚本byfn.sh位于目录$HOME/hyfa/fabric-samples/first-network下，具体用途如下

```
$./byfn.sh -h //查看帮助说明
Usage:
  byfn.sh <mode> [-c <channel name>] [-t <timeout>] [-d <delay>] [-f
<docker-compose-file>] [-s <dbtype>] [-l <language>] [-i <imagetag>] [-v]
    <mode> - one of 'up', 'down', 'restart', 'generate' or 'upgrade'
      - 'up' - bring up the network with docker-compose up
      - 'down' - clear the network with docker-compose down
      - 'restart' - restart the network
      - 'generate' - generate required certificates and genesis block
      - 'upgrade'  - upgrade the network from version 1.1.x to 1.2.x
    -c <channel name> - channel name to use (defaults to "mychannel")
    -t <timeout> - CLI timeout duration in seconds (defaults to 10)
    -d <delay> - delay duration in seconds (defaults to 3)
    -f <docker-compose-file> - specify which docker-compose file use
(defaults to docker-compose-cli.yaml)
    -s <dbtype> - the database backend to use: goleveldb (default) or
couchdb
    -l <language> - the chaincode language: golang (default) or node
    -i <imagetag> - the tag to be used to launch the network (defaults to
"latest")
    -v - verbose mode
  byfn.sh -h (print this message)

Typically, one would first generate the required certificates and
genesis block, then bring up the network. e.g.:

        byfn.sh generate -c mychannel
        byfn.sh up -c mychannel -s couchdb
        byfn.sh up -c mychannel -s couchdb -i 1.2.x
        byfn.sh up -l node
        byfn.sh down -c mychannel
        byfn.sh upgrade -c mychannel

Taking all defaults:
```

```
byfn.sh generate
byfn.sh up
byfn.sh down
```

1.生成组织结构及身份证书

```
$ sudo ./byfn.sh generate
```

- 生成组织结构及身份证书(保存在当前目录的crypto-config目录)

```
drwxr-xr-x 3 root root 4096 Jul 11 11:26 ordererOrganizations
drwxr-xr-x 4 root root 4096 Jul 11 11:26 peerOrganizations
```

- 生成Orderer创世区块文件(保存在当前目录的chanel-artifacts目录下)

```
$HOME/hyfa/fabric-samples/first-network/channel-artifacts/genesis.block
```

- 生成应用通道交易配置文件(保存在当前目录的chanel-artifacts目录下)

```
$HOME/hyfa/fabric-samples/first-network/channel-artifacts/channel.tx
```

- 组织中锚节点更新配置文件(保存在当前目录的chanel-artifacts目录下)

  --检测当前应用通道中新加入的节点

  --跨组织的数据交换

```
Org1MSPanchors.tx
Org2MSPanchors.tx
```

2.创建容器与通道

```
$sudo ./byfn.sh up
```

- 创建相应的docker容器

列出新拉取(pull)的docker镜像

```
dev-peer1.org2.example.com-mycc-1.0-
26c2ef32838554aac4f7ad6f100aca865e87959c9a126e86d764c8d01f8346ab   latest
           526ebbb4eccd        44 seconds ago       147MB
dev-peer0.org1.example.com-mycc-1.0-
384f11f484b9302df90b453200cfb25174305fce8f53f4e94d45ee3b6cab0ce9   latest
           ba6eb425f830        57 seconds ago       147MB
dev-peer0.org2.example.com-mycc-1.0-
15b571b3ce849066b7ec74497da3b27e54e0df1345daff3951b94245ce09c42b   latest
           b582a75829b3        About a minute ago   147MB
```

列出新创建的docker容器名称

```
$sudo  docker ps   |awk  '{print $NF}' |grep -v NAMES|sort
====返回结果====
cli
dev-peer0.org1.example.com-mycc-1.0
dev-peer0.org2.example.com-mycc-1.0
dev-peer1.org2.example.com-mycc-1.0
orderer.example.com
peer0.org1.example.com
peer0.org2.example.com
peer1.org1.example.com
peer1.org2.example.com
```

- 创建通道mychannel

```
Channel name : mychannel
Creating channel...
+ peer channel create -o orderer.example.com:7050 -c mychannel -f
./channel-artifacts/channel.tx --tls true --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem
+ res=0
+ set +x
2018-07-11 06:42:21.437 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 06:42:21.456 UTC [cli/common] readBlock -> INFO 002 Got status:
&{NOT_FOUND}
2018-07-11 06:42:21.459 UTC [channelCmd] InitCmdFactory -> INFO 003
Endorser and orderer connections initialized
2018-07-11 06:42:21.669 UTC [cli/common] readBlock -> INFO 004 Received
block: 0
===================== Channel 'mychannel' created =====================
```

- 节点加入通道

```
Having all peers join the channel...
+ peer channel join -b mychannel.block
+ res=0
+ set +x
2018-07-11 06:42:21.742 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 06:42:21.867 UTC [channelCmd] executeJoin -> INFO 002
Successfully submitted proposal to join channel
===================== peer0.org1 joined channel 'mychannel'
=====================

+ peer channel join -b mychannel.block
+ res=0
+ set +x
2018-07-11 06:42:24.942 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 06:42:25.053 UTC [channelCmd] executeJoin -> INFO 002
Successfully submitted proposal to join channel
===================== peer1.org1 joined channel 'mychannel'
=====================

+ peer channel join -b mychannel.block
+ res=0
+ set +x
2018-07-11 06:42:28.129 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
```

```
2018-07-11 06:42:28.232 UTC [channelCmd] executeJoin -> INFO 002
Successfully submitted proposal to join channel
===================== peer0.org2 joined channel 'mychannel'
=====================

+ peer channel join -b mychannel.block
+ res=0
+ set +x
2018-07-11 06:42:31.304 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 06:42:31.418 UTC [channelCmd] executeJoin -> INFO 002
Successfully submitted proposal to join channel
===================== peer1.org2 joined channel 'mychannel'
=====================
```

3.生成锚节点配置

```
Updating anchor peers for org1...
+ peer channel update -o orderer.example.com:7050 -c mychannel -f
./channel-artifacts/Org1MSPanchors.tx --tls true --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem
+ res=0
+ set +x
2018-07-11 06:42:34.493 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 06:42:34.504 UTC [channelCmd] update -> INFO 002 Successfully
submitted channel update
===================== Anchor peers updated for org 'Org1MSP' on channel
'mychannel' =====================

Updating anchor peers for org2...
+ peer channel update -o orderer.example.com:7050 -c mychannel -f
./channel-artifacts/Org2MSPanchors.tx --tls true --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem
+ res=0
+ set +x
2018-07-11 06:42:37.574 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 06:42:37.586 UTC [channelCmd] update -> INFO 002 Successfully
submitted channel update
===================== Anchor peers updated for org 'Org2MSP' on channel
'mychannel' =====================
```

4.在节点上安装链码并实例化

```
Installing chaincode on peer0.org1...
+ peer chaincode install -n mycc -v 1.0 -l golang -p
github.com/chaincode/chaincode_example02/go/
+ res=0
+ set +x
2018-07-11 06:42:40.666 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
001 Using default escc
2018-07-11 06:42:40.666 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
002 Using default vscc
2018-07-11 06:42:40.969 UTC [chaincodeCmd] install -> INFO 003 Installed
remotely response:<status:200 payload:"OK" >
===================== Chaincode is installed on peer0.org1
=====================

Install chaincode on peer0.org2...
+ peer chaincode install -n mycc -v 1.0 -l golang -p
github.com/chaincode/chaincode_example02/go/
+ res=0
+ set +x
2018-07-11 06:42:41.051 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
001 Using default escc
2018-07-11 06:42:41.051 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
002 Using default vscc
2018-07-11 06:42:41.240 UTC [chaincodeCmd] install -> INFO 003 Installed
remotely response:<status:200 payload:"OK" >
===================== Chaincode is installed on peer0.org2
=====================

Instantiating chaincode on peer0.org2...
+ peer chaincode instantiate -o orderer.example.com:7050 --tls true --
cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem -C mychannel -n mycc -l golang -v 1.0 -c '{"Args":
["init","a","100","b","200"]}' -P 'AND
('\''Org1MSP.peer'\'','\''Org2MSP.peer'\'')'
+ res=0
+ set +x
2018-07-11 06:42:41.306 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
001 Using default escc
2018-07-11 06:42:41.306 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
002 Using default vscc
===================== Chaincode is instantiated on peer0.org2 on channel
'mychannel' =====================
```

5.调用链码对账本数据进行操作

```
Querying chaincode on peer0.org1...
===================== Querying on peer0.org1 on channel 'mychannel'...
=====================
Attempting to Query peer0.org1 ...3 secs
+ peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
+ res=0
+ set +x

100
===================== Query successful on peer0.org1 on channel 'mychannel'
=====================
Sending invoke transaction on peer0.org1 peer0.org2...
+ peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem -C mychannel -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 --peerAddresses
peer0.org2.example.com:7051 --tlsRootCertFiles
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations
/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt -c '{"Args":
["invoke","a","b","10"]}'
+ res=0
+ set +x
2018-07-11 06:43:08.200 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO
001 Chaincode invoke successful. result: status:200
===================== Invoke transaction successful on peer0.org1
peer0.org2 on channel 'mychannel' =====================

Installing chaincode on peer1.org2...
+ peer chaincode install -n mycc -v 1.0 -l golang -p
github.com/chaincode/chaincode_example02/go/
+ res=0
+ set +x
2018-07-11 06:43:08.271 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
001 Using default escc
2018-07-11 06:43:08.271 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
002 Using default vscc
2018-07-11 06:43:08.449 UTC [chaincodeCmd] install -> INFO 003 Installed
remotely response:<status:200 payload:"OK" >
===================== Chaincode is installed on peer1.org2
=====================

Querying chaincode on peer1.org2...
```

```
===================== Querying on peer1.org2 on channel 'mychannel'...
=====================
Attempting to Query peer1.org2 ...3 secs
+ peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
+ res=0
+ set +x

90
===================== Query successful on peer1.org2 on channel 'mychannel'
=====================
```

自动化脚本执行完毕

```
========= All GOOD, BYFN execution completed ===========



 _____   _   _   ____
| ____| | \ | | |  _ \
|  _|   |  \| | | | | |
| |___  | |\  | | |_| |
|_____| |_| \_| |____/
```

6.关闭服务

```
$sudo ./byfn.sh down //通过down参数关闭服务
```

- 同时删除所生成的组织结构及身份证书目录
- 同时删除所生成的chanel-artifacts/目录下的四个配置文件
- 同时删除所有启动的docker容器以及对应的镜像文件

二.通过手动方式启动服务

在实际的应用场景下，我们会经常对某些配置进行个性化定制，以满足实际生产需求，所以，我们在本小节介绍如何通过手动方式启动对应的服务，并通过修改相关配置文件满足实际场景，手动步骤与自动化脚本基本一致。

1.生成组织结构及身份证书

```
$ cd hyfa/fabric-samples/first-network/
```

为fabric网络生成指定拓扑结构的**组织关系和身份证书**

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

> 此命令依赖 `crypto-config.yaml` 配置文件

命令输出

```
org1.example.com
org2.example.com
```

证书和密钥（即MSP）将被输出到目录 `first-network/crypto-config` 的目录中

```
drwxr-xr-x 3 root root 4096 Jul 11 00:31 ordererOrganizations
drwxr-xr-x 4 root root 4096 Jul 11 00:31 peerOrganizations
```

- ordererOrganizations

  包括构成Orderer组织(1个Orderer节点)的身份信息

- peerOrganizations

  所有的Peer节点组织(2个组织, 4个节点)的相关身份信息. 其中最关键的是MSP目录, 代表了实体的身份信息

```
$ tree  -L 6 crypto-config //通过tree命令查看目录树
```

```
crypto-config
├── ordererOrganizations
|   └── example.com
|       ├── ca
|       |   ├── ca.example.com-cert.pem
|       |   └──
ded244dc5aa11c2b42e43fa4de6b952259078c78131fc86031a02baff21abd78_sk
|       ├── msp
|       |   ├── admincerts
|       |   |   └── Admin@example.com-cert.pem
|       |   ├── cacerts
|       |   |   └── ca.example.com-cert.pem
|       |   └── tlscacerts
|       |       └── tlsca.example.com-cert.pem
|       ├── orderers
|       |   └── orderer.example.com
|       |       ├── msp
|       |       |   ├── admincerts
|       |       |   ├── cacerts
|       |       |   ├── keystore
|       |       |   ├── signcerts
|       |       |   └── tlscacerts
```

```
|       |       └── tls
|       |           ├── ca.crt
|       |           ├── server.crt
|       |           └── server.key
|       ├── tlsca
|       |   ├──
1e1955762d0669ba18f078a04ac7b19918f40554aeddd6841dc6c059eefbbcf9_sk
|       |   └── tlsca.example.com-cert.pem
|       └── users
|           └── Admin@example.com
|               ├── msp
|               |   ├── admincerts
|               |   ├── cacerts
|               |   ├── keystore
|               |   ├── signcerts
|               |   └── tlscacerts
|               └── tls
|                   ├── ca.crt
|                   ├── client.crt
|                   └── client.key
└── peerOrganizations
    ├── org1.example.com    # 第一个组织的相关材料，每个组织会生成单独的根证书
    |   ├── ca              #存放组织的根证书和对应的私钥文件，默认采用EC算法，证书为
自签名。为根证书。
    |   |   ├──
15d058166d13e584fca35523a71a12b2fbaf6de5072d3661801476dfbfb3ea4f_sk
    |   |   └── ca.org1.example.com-cert.pem
    |   ├── msp             #存放代表该组织的身份信息
    |   |   ├── admincerts #组织管理员的身份验证证书，被根证书签名
    |   |   |   └── Admin@org1.example.com-cert.pem
    |   |   ├── cacerts     #组织的根证书，同CA目录下文件
    |   |   |   └── ca.org1.example.com-cert.pem
    |   |   ├── config.yaml
    |   |   └── tlscacerts #用于TLS的CA 证书，自签名
    |   |       └── tlsca.org1.example.com-cert.pem
    |   ├── peers        # 存放属于该组织的所有Peer节点
    |   |   ├── peer0.org1.example.com  #第一个peer的信息，包括其msp证书和tls 证
书两类
    |   |   |   ├── msp
    |   |   |   |   ├── admincerts#组织管理员的身份验证证书。Peer将基于这些证书来认
证交易签署者是否为管理员身份
    |   |   |   |   ├── cacerts    #存放组织的根证书
    |   |   |   |   ├── config.yaml
    |   |   |   |   ├── keystore    #本节点的身份私钥，用来签名
    |   |   |   |   ├── signcerts   #验证本节点签名的证书，被组织根证书签名
    |   |   |   |   └── tlscacerts #TLS 连接用的身份证书，即组织TLS 证书
    |   |   |   └── tls    # 存放tls相关的证书和私钥
    |   |   |       ├── ca.crt       #组织的根证书
    |   |   |       ├── server.crt   #验证本节点签名的证书，被组织根证书签名
```

```
    |   |   |       └── server.key  #本节点的身份私钥，用来签名
    |   |   └── peer1.org1.example.com
    |   |       ├── msp
    |   |       |   ├── admincerts
    |   |       |   ├── cacerts
    |   |       |   ├── config.yaml
    |   |       |   ├── keystore
    |   |       |   ├── signcerts
    |   |       |   └── tlscacerts
    |   |       └── tls
    |   |           ├── ca.crt
    |   |           ├── server.crt
    |   |           └── server.key
    |   ├── tlsca
    |   |   ├──
d45d38d7d48e3e41332c66d673869080be030c0e76d47cc6c8d6909f89f401e8_sk
    |   |   └── tlsca.org1.example.com-cert.pem
    |   └── users
    |       ├── Admin@org1.example.com
    |       |   ├── msp
    |       |   |   ├── admincerts
    |       |   |   ├── cacerts
    |       |   |   ├── keystore
    |       |   |   ├── signcerts
    |       |   |   └── tlscacerts
    |       |   └── tls
    |       |       ├── ca.crt
    |       |       ├── client.crt
    |       |       └── client.key
    |       └── User1@org1.example.com
    |           ├── msp
    |           |   ├── admincerts
    |           |   ├── cacerts
    |           |   ├── keystore
    |           |   ├── signcerts
    |           |   └── tlscacerts
    |           └── tls
    |               ├── ca.crt
    |               ├── client.crt
    |               └── client.key
    └── org2.example.com
        ├── ca
        |   ├──
5d261ccc23b0a7345dc3ee387a4714c6ec3261f957d40e82198b1d6141b715a9_sk
        |   └── ca.org2.example.com-cert.pem
        ├── msp
        |   ├── admincerts
        |   |   └── Admin@org2.example.com-cert.pem
        |   ├── cacerts
```

```
        |   |   └── ca.org2.example.com-cert.pem
        |   ├── config.yaml
        |   └── tlscacerts
        |       └── tlsca.org2.example.com-cert.pem
        ├── peers
        |   ├── peer0.org2.example.com
        |   |   ├── msp
        |   |   |   ├── admincerts
        |   |   |   ├── cacerts
        |   |   |   ├── config.yaml
        |   |   |   ├── keystore
        |   |   |   ├── signcerts
        |   |   |   └── tlscacerts
        |   |   └── tls
        |   |       ├── ca.crt
        |   |       ├── server.crt
        |   |       └── server.key
        |   └── peer1.org2.example.com
        |       ├── msp
        |       |   ├── admincerts
        |       |   ├── cacerts
        |       |   ├── config.yaml
        |       |   ├── keystore
        |       |   ├── signcerts
        |       |   └── tlscacerts
        |       └── tls
        |           ├── ca.crt
        |           ├── server.crt
        |           └── server.key
        ├── tlsca
        |   ├──
c16ccfac347d187e318e38de956036528f29e542c4cb2451dafd0d33b44efebf_sk
        |   └── tlsca.org2.example.com-cert.pem
        └── users
            ├── Admin@org2.example.com
            |   ├── msp
            |   |   ├── admincerts
            |   |   ├── cacerts
            |   |   ├── keystore
            |   |   ├── signcerts
            |   |   └── tlscacerts
            |   └── tls
            |       ├── ca.crt
            |       ├── client.crt
            |       └── client.key
            └── User1@org2.example.com
                ├── msp
                |   ├── admincerts
                |   ├── cacerts
```

```
|   ├── keystore
|   ├── signcerts
|   └── tlscacerts
└── tls
    ├── ca.crt
    ├── client.crt
    └── client.key
```

Cryptogen 按照配置文件中指定的结构生成了对应的组织和密钥、证书文件

其中最关键的是各个资源下的msp 目录内容，存储了生成的代表MSP 身份的各种证书文件，一般包括：

- admincerts： 管理员的身份证书文件
- cacerts： 信任的根证书文件
- key store： 节点的签名私钥文件
- signcerts： 节点的签名身份证书文件
- tlscacerts: TLS 连接用的证书
- intermediatecerts （可选）： 信任的中间证书
- crls （可选）： 证书撤销列表
- config.yaml （可选）： 记录OrganizationalUnitIdentifiers 信息，包括根证书位置和ID信息

这些身份文件随后可以分发到对应的Orderer 节点和Peer 节点上，并放到对应的MSP路径下，用于签名使用。

2.创建创世区块并启动Orderer

指定使用 `configtx.yaml` 文件中定义的 `TwoOrgsOrdererGenesis` 模板, 生成Orderer服务系统通道的初始区块文件

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

命令输出

```
2018-07-11 00:52:00.625 PDT [common/tools/configtxgen] main -> WARN 001
Omitting the channel ID for configtxgen is deprecated.  Explicitly passing
the channel ID will be required in the future, defaulting to 'testchainid'.
2018-07-11 00:52:00.625 PDT [common/tools/configtxgen] main -> INFO 002
Loading configuration
2018-07-11 00:52:00.632 PDT [common/tools/configtxgen/encoder]
NewChannelGroup -> WARN 003 Default policy emission is deprecated, please
include policy specificiations for the channel group in configtx.yaml
2018-07-11 00:52:00.632 PDT [common/tools/configtxgen/encoder]
NewOrdererGroup -> WARN 004 Default policy emission is deprecated, please
include policy specificiations for the orderer group in configtx.yaml
2018-07-11 00:52:00.632 PDT [common/tools/configtxgen/encoder]
NewOrdererOrgGroup -> WARN 005 Default policy emission is deprecated,
please include policy specificiations for the orderer org group OrdererOrg
in configtx.yaml
2018-07-11 00:52:00.634 PDT [msp] getMspConfig -> INFO 006 Loading NodeOUs
2018-07-11 00:52:00.634 PDT [common/tools/configtxgen/encoder]
NewOrdererOrgGroup -> WARN 007 Default policy emission is deprecated,
please include policy specificiations for the orderer org group Org1MSP in
configtx.yaml
2018-07-11 00:52:00.635 PDT [msp] getMspConfig -> INFO 008 Loading NodeOUs
2018-07-11 00:52:00.635 PDT [common/tools/configtxgen/encoder]
NewOrdererOrgGroup -> WARN 009 Default policy emission is deprecated,
please include policy specificiations for the orderer org group Org2MSP in
configtx.yaml
2018-07-11 00:52:00.635 PDT [common/tools/configtxgen] doOutputBlock ->
INFO 00a Generating genesis block
2018-07-11 00:52:00.635 PDT [common/tools/configtxgen] doOutputBlock ->
INFO 00b Writing genesis block
```

```
$ ls channel-artifacts/
-rw-r--r-- 1 root root 12655 Jul 11 00:52 channel-artifacts/genesis.block
//生成的创世区块
```

3.生成应用通道交易配置文件

```
$ export CHANNEL_NAME=mychannel //设置临时变量
```

指定使用 `configtx.yaml` 配置文件中的 `TwoOrgsChannel` 模板, 来生成新建通道的配置交易文件, `TwoOrgsChannel` 模板指定了Org1和Org2都属于后面新建的应用通道

```
$ sudo ../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx
./channel-artifacts/channel.tx -channelID $CHANNEL_NAME
```

命令输出

```
2018-07-11 00:54:21.557 PDT [common/tools/configtxgen] main -> INFO 001
Loading configuration
2018-07-11 00:54:21.563 PDT [common/tools/configtxgen]
doOutputChannelCreateTx -> INFO 002 Generating new channel configtx
2018-07-11 00:54:21.563 PDT [common/tools/configtxgen/encoder]
NewApplicationGroup -> WARN 003 Default policy emission is deprecated,
please include policy specificiations for the application group in
configtx.yaml
2018-07-11 00:54:21.563 PDT [msp] getMspConfig -> INFO 004 Loading NodeOUs
2018-07-11 00:54:21.564 PDT [common/tools/configtxgen/encoder]
NewApplicationOrgGroup -> WARN 005 Default policy emission is deprecated,
please include policy specificiations for the application org group Org1MSP
in configtx.yaml
2018-07-11 00:54:21.564 PDT [msp] getMspConfig -> INFO 006 Loading NodeOUs
2018-07-11 00:54:21.564 PDT [common/tools/configtxgen/encoder]
NewApplicationOrgGroup -> WARN 007 Default policy emission is deprecated,
please include policy specificiations for the application org group Org2MSP
in configtx.yaml
2018-07-11 00:54:21.565 PDT [common/tools/configtxgen]
doOutputChannelCreateTx -> INFO 008 Writing new channel tx
```

```
$ ll channel-artifacts/channel.tx
-rw-r--r-- 1 root root 346 Jul 11 00:54 channel-artifacts/channel.tx //生成
的通道文件
```

4.生成各组织的锚节点更新文件

锚节点配置更新文件用来对组织的锚节点进行配置

同样基于 `configtx.yaml` 配置文件生成新建通道文件, 每个组织都需要分别生成且注意指定对应的组织名称

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

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

命令输出

```
2018-07-11 00:57:34.629 PDT [common/tools/configtxgen] main -> INFO 001
Loading configuration
2018-07-11 00:57:34.635 PDT [common/tools/configtxgen]
doOutputAnchorPeersUpdate -> INFO 002 Generating anchor peer update
2018-07-11 00:57:34.635 PDT [common/tools/configtxgen]
doOutputAnchorPeersUpdate -> INFO 003 Writing anchor peer update
2018-07-11 00:57:40.028 PDT [common/tools/configtxgen] main -> INFO 001
Loading configuration
2018-07-11 00:57:40.037 PDT [common/tools/configtxgen]
doOutputAnchorPeersUpdate -> INFO 002 Generating anchor peer update
2018-07-11 00:57:40.037 PDT [common/tools/configtxgen]
doOutputAnchorPeersUpdate -> INFO 003 Writing anchor peer update
```

5.创建容器，启动网络

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

-f: 指定docker-compose文件

-d：Detached mode,后台运行容器

命令输出

```
Creating network "net_byfn" with the default driver
Creating volume "net_peer0.org2.example.com" with default driver
Creating volume "net_peer1.org2.example.com" with default driver
Creating volume "net_peer1.org1.example.com" with default driver
Creating volume "net_peer0.org1.example.com" with default driver
Creating volume "net_orderer.example.com" with default driver
Creating peer1.org1.example.com ...
Creating peer1.org2.example.com ...
Creating peer0.org2.example.com ...
Creating orderer.example.com ...
Creating peer0.org1.example.com ...
Creating orderer.example.com
Creating peer0.org2.example.com
Creating peer0.org1.example.com
Creating peer1.org1.example.com
Creating peer1.org2.example.com ... done
Creating cli ...
Creating cli ... done
```

查看创建的docker容器

生成的docker容器名称

```
cli
peer1.org2.example.com
peer1.org1.example.com
peer0.org2.example.com
peer0.org1.example.com
orderer.example.com
```

6.创建通道并配置

执行如下命令进入到CLI容器中(后继操作都在容器中执行)

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

直接进入容器该路径下,**相关操作不要更换路径**

```
root@138a8115a08c:/opt/gopath/src/github.com/hyperledger/fabric/peer# pwd
/opt/gopath/src/github.com/hyperledger/fabric/peer
```

6.1 配置变量

```
export CHANNEL_NAME=mychannel
```

6.2 创建通道

```
$ 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/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem
```

命令返回:

```
2018-07-11 11:44:50.198 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 11:44:50.228 UTC [cli/common] readBlock -> INFO 002 Got status:
&{NOT_FOUND}
2018-07-11 11:44:50.232 UTC [channelCmd] InitCmdFactory -> INFO 003
Endorser and orderer connections initialized
2018-07-11 11:44:50.436 UTC [cli/common] readBlock -> INFO 004 Received
block: 0
```

> 该命令自动在本地生成与该应用通道同名的初始区块 **mychannel.block**, 只有拥有该文件才可以加入创建的应用通道中

参数说明: -o: 指定orderer节点的地址

-c: 指定要创建的应用通道的名称(必须与在创建应用通道交易配置文件时的通道名称保持一致)

-f: 指定创建应用通道时所使用的应用通道交易配置文件

--tls: 开启TLS验证 --cafile: 指定TLS_CA证书路径

查看生成的文件

```
root@138a8115a08c:/opt/gopath/src/github.com/hyperledger/fabric/peer# ll
total 36
drwxr-xr-x 5 root root  4096 Jul 11 11:44 ./
drwxr-xr-x 3 root root  4096 Jul 11 08:00 ../
drwxrwxr-x 2 1000 1000  4096 Jul 11 07:57 channel-artifacts/
drwxr-xr-x 4 root root  4096 Jul 11 07:31 crypto/
-rw-r--r-- 1 root root 15671 Jul 11 11:44 mychannel.block
drwxrwxr-x 2 1000 1000  4096 Jul 10 07:59 scripts/
```

6.3 当前节点加入通道

应用通道所包含组织的成员节点可以加入通道中

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

join命令: 将本Peer节点加入到某个应用通道中

命令返回

```
2018-07-11 13:20:01.980 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 13:20:02.084 UTC [channelCmd] executeJoin -> INFO 002
Successfully submitted proposal to join channel
```

查看加入列表

```
$ peer channel list
Channels peers has joined:
mychannel
```

6.4更新锚节点

锚节点通过广播的方式通知有新节点加入

- 使用Org1的管理员身份更新锚节点配置(默认)

```
$ 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/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem
```

命令返回

```
2018-07-11 12:00:12.677 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 12:00:12.690 UTC [channelCmd] update -> INFO 002 Successfully
submitted channel update
```

- 使用Org2的管理员身份更新锚节点配置

变更脚本变量(比如当前变量CORE_PEER_LOCALMSPID默认为Org1MSP)

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

执行更新锚节点配置命令

```
$ 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/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem
```

命令返回

```
2018-07-11 12:05:37.325 UTC [channelCmd] InitCmdFactory -> INFO 001
Endorser and orderer connections initialized
2018-07-11 12:05:37.338 UTC [channelCmd] update -> INFO 002 Successfully
submitted channel update
```

如果需要加入其它组织，变更脚本变量后执行更新锚节点操作即可，至此，手动配置网络完成, 可以测试链码ChainCode

> 切换：pee1.org1.example.com

```
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_ADDRESS=peer1.org1.example.com:7051
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabri
c/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.exa
mple.com/msp
CORE_PEER_TLS_ROOTCERT_FILE=/opt/gopath/src/github.com/hyperledger/fa
bric/peer/crypto/peerOrganizations/org1.example.com/peers/peer1.org1.exa
mple.com/tls/ca.crt
```

三.测试链码

进去CLI容器进行相关运维操作

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

Peer加入应用通道后, 可以执行链码相关操作,进行测试 链码在调用之前, 必须先经过安装和实例化两个步骤, 部署到Peer节点上.

1.查看并环境变量

```
$ echo $CORE_PEER_LOCALMSPID #确认当前指定的org
Org1MSP
$ echo $CORE_PEER_ADDRESS  #确认当前指定的peer
peer0.org1.example.com:7051
$ export CHANNEL_NAME=mychannel #设置通道名称
```

2.安装并实例化链码

2.1 链码使用须知

-- 将其安装在指定的节点上

-- 安装完成后要对其进行实例化

-- 调用链码(查询, 执行事务)

2.2 安装并实例化链码

- 安装链码

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

参数说明:

-n: 指定要安装的链码的名称

-v: 指定链码的版本

-p: 指定要安装的链码的所在路径,**注意不要写绝对路径**

命令返回:

```
2018-07-11 13:01:11.090 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
001 Using default escc
2018-07-11 13:01:11.090 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
002 Using default vscc
2018-07-11 13:01:11.351 UTC [chaincodeCmd] install -> INFO 003 Installed
remotely response:<status:200 payload:"OK" >
```

- 实例化链码

```
$ peer chaincode instantiate -o orderer.example.com:7050 \
--tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem \
-C $CHANNEL_NAME \
-n mycc -v 1.0 \
-c '{"Args":["init","a", "100", "b","200"]}' \
-P "OR('Org1MSP.peer','Org2MSP.peer')"
```

参数说明:

-o: 指定Oderer节点地址

--tls: 开启TLS验证

--cafile: 指定TLS_CA证书路

-n: 指定要实例化的链码名称

-v: 指定要实例化的链码的版本号

-C: 指定通道名称

-c: 实例化链码时指定的参数

（"init","a", "100", "b","200":设置a账户初始化金额为100，b账户的初始金额为200）

-P: 指定背书策略(指定交易在哪些节点上面进行签名)

命令返回:

```
2018-07-11 13:21:13.234 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
001 Using default escc
2018-07-11 13:21:13.234 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO
002 Using default vscc
```

3.代码调用测试

3.1 查询账户余额

```
$ peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","a"]}'
#查询a账户余额
100
$ peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","b"]}'
#查询b账户余额
200
```

3.2 转账测试

```
peer chaincode invoke -o orderer.example.com:7050 \
--tls --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizati
ons/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.c
om-cert.pem  \
-C $CHANNEL_NAME -n mycc \
-c '{"Args":["invoke","a","b","10"]}'
```

参数说明: -o: 指定orderer节点地址 --tls: 开启TLS验证 --cafile: 指定TLS_CA证书路径 -n: 指定链码名称 -C: 指定通道名称 -c: 指定调用链码的所需参数 func invoke(accoutF string, accountT stirng, amount string)

3.3 查询当前余额

```
$ peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","a"]}'
#查询a账户余额
90
$ peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","b"]}'
#查询b账户余额
210
```