[1. 系统管理 2](#_Toc14426)

[1.1. 初始组网 2](#_Toc32480)

[1.1.1. 网络拓扑 2](#_Toc13089)

[1.1.2. 初始部署新网Orderer节点 3](#_Toc18193)

[1.1.3. 初始部署新网peer节点 12](#_Toc24727)

[1.2. 新增orderer节点入网 15](#_Toc7601)

[1.2.1. Orderer节点安装 15](#_Toc27728)

[1.2.2. Orderer节点启动 16](#_Toc17886)

[1.2.3. Orderer节点入网 17](#_Toc9057)

[1.2.4. Orderer节点停止 18](#_Toc23811)

[1.3. 新增机构节点入网 18](#_Toc18315)

[1.3.1. Peer节点安装 18](#_Toc22484)

[1.3.2. Peer节点启动 22](#_Toc13512)

[1.3.3. Peer节点入网 22](#_Toc15411)

[1.3.4. Peer节点停止 23](#_Toc13329)

[2. 可用性监控 24](#_Toc10055)

[2.1. 区块节点浏览 24](#_Toc28065)

[2.2. 组网节点监控 26](#_Toc26960)

[2.2.1. Orderer节点监控 26](#_Toc15870)

[2.2.2. Peer节点监控 27](#_Toc3315)

[2.3. 节点主机监控 27](#_Toc3209)

[2.3.1. 进程/服务、端口 27](#_Toc3607)

[2.3.2. CPU/内存/磁盘 27](#_Toc30574)

[3. 故障排查和恢复 28](#_Toc25744)

[3.1. Orderer节点主机故障排查和恢复 28](#_Toc2086)

[3.1.1. 故障分析 28](#_Toc26914)

[3.1.2. 故障排查标准步骤 29](#_Toc31973)

[3.1.3. 故障恢复 30](#_Toc4007)

[3.2. Peer节点主机故障排查和恢复 30](#_Toc22192)

[3.2.1. 故障分析 30](#_Toc1884)

[3.2.2. 故障排查标准步骤 31](#_Toc6383)

[3.2.3. 故障恢复 31](#_Toc6819)

[3.3. 区块链组网故障排查和恢复 32](#_Toc20144)

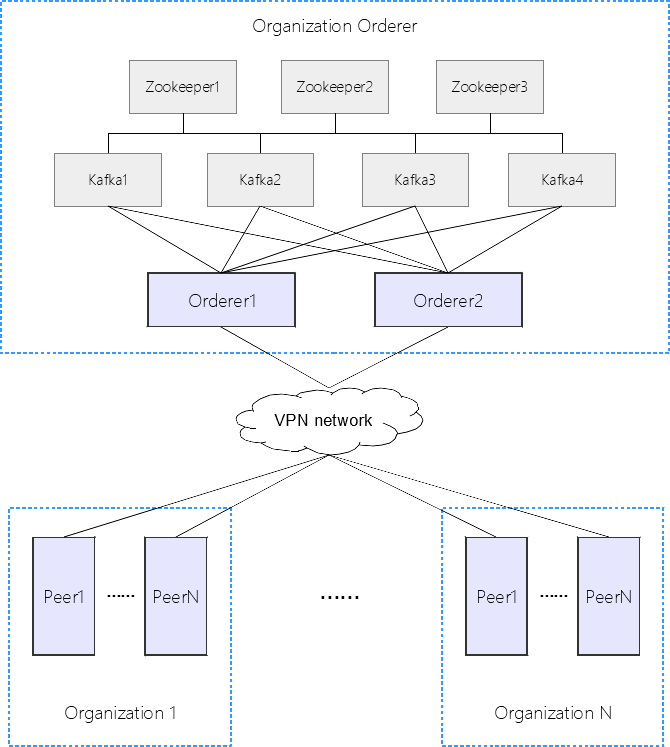
[3.3.1. 故障排查标准步骤 32](#_Toc12728)

[3.3.2. 故障恢复 32](#_Toc19203)

# 系统管理

## 初始组网

### 网络拓扑



区块链网络中包含两类节点：

* Orderer节点：整个网络的共识节点，负责交易打包和区块生成，目前的方案中部署两台Orderer在新网的网络环境中。
* Peer节点：每个入网机构的机构需要部署至少一个peer节点，负责对区块链交易进行签名背书，以及从Orderer同步区块数据到节点本地形成账本的副本。

目前的网络中，新网和每个入网的机构间需要建立私有的数据渠道channel来进行区块链交易。在本方案中，初始组网时新网会作为初始机构首先入网。后续其他机构入网时，需要新网为其创建私有channel，机构再加入该channel实现与新网间的区块链交易。

### 初始部署新网Orderer节点

#### 安装Jdk

把jdk包解压到/usr/local目录下，做好环境变量配置：

export JAVA\_HOME=/usr/local/jdk1.8.0\_91

export JRE\_HOME=/usr/local/jdk1.8.0\_91/jre

export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH

export PATH

#### 安装Zookeeper

在普通用户的当前目录下创建了server目录，把zookeeper的安装包解压到当前目录下面。在zookeeper解压的目录里面进入到conf目录下面，把zoo\_sample.cfg复制一份改名为zoo.cfg：

[ynet@block1 ~]$ cd server/zookeeper-3.4.11/conf/

[ynet@block1 conf]$ ll

-rw-r--r-- 1 ynet ynet 535 Nov 2 02:47 configuration.xsl

-rw-r--r-- 1 ynet ynet 2161 Nov 2 02:47 log4j.properties

-rw-r--r-- 1 ynet ynet 1002 Mar 19 15:31 zoo.cfg

-rw-r--r-- 1 ynet ynet 922 Nov 2 02:47 zoo\_sample.cfg

编辑zoo.cfg文件，配置如下内容：

[ynet@block1 conf]$ cat zoo.cfg

tickTime=2000

initLimit=10

syncLimit=5

dataDir=/appdata/zookeeper

clientPort=2181

#zookeeper server列表

server.1=block1:2888:3888

server.2=block2:2888:3888

server.3=block3:2888:3888

创建/appdata/zookeeper目录，并做好授权，并在这个目录下面创建myid配置文件，写入对应的值，如下：

[ynet@block1 conf]$ ll /appdata/

drwxr-xr-x 3 ynet root 4096 Mar 19 15:52 zookeeper

[ynet@block2 server]$ cd /appdata/zookeeper/

[ynet@block2 zookeeper]$ ll

-rw-rw-r-- 1 ynet ynet 2 Mar 19 15:36 myid

在zookeeper的bin目录下面执行命令行启动server：

zkServer.sh start

默认的日志文件存放在/appdata/zookeeper目录下面。

#### 安装Kafka

软件解压到普通用户的server目录下。

在解压目录下面的config目录下面是kafka存放配置文件的。修改这个目录下面的server.properties文件：

#kafka broker id，4台主机分别为1,2,3,4

broker.id=1

listeners=PLAINTEXT://:9092

#kafka主机名

host.name=block1

num.network.threads=3

num.io.threads=8

socket.send.buffer.bytes=102400

socket.receive.buffer.bytes=102400

socket.request.max.bytes=104857600

log.dirs=/tmp/kafka-logs

num.partitions=1

num.recovery.threads.per.data.dir=1

offsets.topic.replication.factor=1

transaction.state.log.replication.factor=1

transaction.state.log.min.isr=1

log.retention.hours=168

log.segment.bytes=1073741824

log.retention.check.interval.ms=300000

#zookeeper server列表

zookeeper.connect=block1:2181,block2:2181,block2:2181

zookeeper.connection.timeout.ms=6000

group.initial.rebalance.delay.ms=0

启动命令：

nohup /bin/kafka-server-start.sh ./config/server.properties >logs/kafka.log 2>1 &

logs目录需要创建。

kafka默认的日志文件是在/tmp/kafka-logs目录下面。查看日志可以从这里查找。

#### 安装Fabric binaries

将hyperledger-fabric-linux-amd64-1.1.0.tar.gz解压至普通用户的home/username/fabric下，Fabric binaries文件列表如下：

[ynet@block4 ~]$ tree fabric

fabric

├── bin

│   ├── configtxgen //创始区块生成工具

│   ├── configtxlator //channel修改工具

│   ├── cryptogen //pki证书生成工具

│   ├── get-docker-images.sh

│   ├── orderer //orderer节点运行程序

│   └── peer //peer节点运行程序

└── config

├── configtx.yaml //创始区块定义文件

├── core.yaml //peer配置文件

└── orderer.yaml //orderer配置文件

2 directories, 9 files

#### 生成msp证书文件

定义新网部署orderer和peer的msp结构模板crypto-config.yaml，如下

OrdererOrgs:

- Name: Orderer #orderer机构定义

Domain: blockchain.com

Specs:

- Hostname: orderer0 #orderer节点主机名定义

- Hostname: orderer1 #orderer节点主机名定义

PeerOrgs:

- Name: xinwang #peer机构定义

Domain: xinwang.blockchain.com

EnableNodeOUs: false

Template:

Count: 2 #peer节点数量

Users:

Count: 1 #为客户端sdk生成证书的数量

生成orderer和peer的msp文件

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

生成msp的文件结构如下

[ynet@block4 bin]$ tree crypto-config

crypto-config

├── ordererOrganizations

│   └── blockchain.com

│   ├── ca #orderer机构ca

│   │   ├── 910be7544d285c7420b22f5b8f66000848147b4e83c4500b8725d926ece39170\_sk

│   │   └── ca.blockchain.com-cert.pem

│   ├── msp #orderer机构msp

│   │   ├── admincerts

│   │   │   └── Admin@blockchain.com-cert.pem

│   │   ├── cacerts

│   │   │   └── ca.blockchain.com-cert.pem

│   │   └── tlscacerts

│   │   └── tlsca.blockchain.com-cert.pem

│   ├── orderers

│   │   ├── orderer0.blockchain.com #orderer节点0的msp

│   │   │   ├── msp

│   │   │   │   ├── admincerts

│   │   │   │   │   └── Admin@blockchain.com-cert.pem

│   │   │   │   ├── cacerts

│   │   │   │   │   └── ca.blockchain.com-cert.pem

│   │   │   │   ├── keystore

│   │   │   │   │   └── 5f07fadd2d67fe84bc70f72711b445b24b1f555687e810832057191511b9441a\_sk

│   │   │   │   ├── signcerts

│   │   │   │   │   └── orderer0.blockchain.com-cert.pem

│   │   │   │   └── tlscacerts

│   │   │   │   └── tlsca.blockchain.com-cert.pem

│   │   │   └── tls

│   │   │   ├── ca.crt

│   │   │   ├── server.crt

│   │   │   └── server.key

│   │   └── orderer1.blockchain.com #orderer节点1的msp

│   │   ├── msp

│   │   │   ├── admincerts

│   │   │   │   └── Admin@blockchain.com-cert.pem

│   │   │   ├── cacerts

│   │   │   │   └── ca.blockchain.com-cert.pem

│   │   │   ├── keystore

│   │   │   │   └── 7d76a636df1355ab86d4c9aaef5a38d6d88ef7b154f11e5fc1b6da5ece27ef17\_sk

│   │   │   ├── signcerts

│   │   │   │   └── orderer1.blockchain.com-cert.pem

│   │   │   └── tlscacerts

│   │   │   └── tlsca.blockchain.com-cert.pem

│   │   └── tls

│   │   ├── ca.crt

│   │   ├── server.crt

│   │   └── server.key

│   ├── tlsca #orderer机构的tls传输层ca

│   │   ├── fb0804d06dd049e8e15a2c896a86adc4bd0ff52df8c82346f14f6c20af503a14\_sk

│   │   └── tlsca.blockchain.com-cert.pem

│   └── users

│   └── Admin@blockchain.com #orderer机构管理员msp

│   ├── msp

│   │   ├── admincerts

│   │   │   └── Admin@blockchain.com-cert.pem

│   │   ├── cacerts

│   │   │   └── ca.blockchain.com-cert.pem

│   │   ├── keystore

│   │   │   └── 576c81218833925f8f1a2477d97f09d6e0d4e16d78ad651fe11ba833ffa63bcd\_sk

│   │   ├── signcerts

│   │   │   └── Admin@blockchain.com-cert.pem

│   │   └── tlscacerts

│   │   └── tlsca.blockchain.com-cert.pem

│   └── tls

│   ├── ca.crt

│   ├── client.crt

│   └── client.key

└── peerOrganizations

└── xinwang.blockchain.com

   ├── ca #peer机构xinwang的ca

   │   ├── a7e09e2f1102d6938883be9c1b58c3c0a065c91e6ad80e6c1cf69229f5b435de\_sk

    │   └── ca.xinwang.blockchain.com-cert.pem

   ├── msp #peer机构xinwang的msp

   │   ├── admincerts

   │   │   └── Admin@xinwang.blockchain.com-cert.pem

   │   ├── cacerts

   │   │   └── ca.xinwang.blockchain.com-cert.pem

   │   └── tlscacerts

   │   └── tlsca.xinwang.blockchain.com-cert.pem

   ├── peers

   │   ├── peer0.xinwang.blockchain.com #xinwang机构的peer节点0的msp

   │   │   ├── msp

   │   │   │   ├── admincerts

   │   │   │   │   └── Admin@xinwang.blockchain.com-cert.pem

   │   │   │   ├── cacerts

   │   │   │   │   └── ca.xinwang.blockchain.com-cert.pem

   │   │   │   ├── keystore

   │   │   │   │   └── 99c147206cc650bb47e70dcf2a355dd926240efc572c3e60588758c2c4990e90\_sk

   │   │   │   ├── signcerts

   │   │   │   │   └── peer0.xinwang.blockchain.com-cert.pem

   │   │   │   └── tlscacerts

   │   │   │   └── tlsca.xinwang.blockchain.com-cert.pem

   │   │   └── tls

   │   │   ├── ca.crt

   │   │   ├── server.crt

   │   │   └── server.key

   │   └── peer1.xinwang.blockchain.com #xinwang机构的peer节点1的msp

   │   ├── msp

   │   │   ├── admincerts

   │   │   │   └── Admin@xinwang.blockchain.com-cert.pem

   │   │   ├── cacerts

   │   │   │   └── ca.xinwang.blockchain.com-cert.pem

   │   │   ├── keystore

   │   │   │   └── e3a3a60e2d1ca615ee2cf5e901e47297e61b383538bf5f6f02ac04481b6a5bbc\_sk

   │   │   ├── signcerts

   │   │   │   └── peer1.xinwang.blockchain.com-cert.pem

   │   │   └── tlscacerts

   │   │   └── tlsca.xinwang.blockchain.com-cert.pem

   │   └── tls

   │   ├── ca.crt

   │   ├── server.crt

   │   └── server.key

   ├── tlsca

   │   ├── 5ef4feffd18dcdf7d22420560032ce6a1167ca1d71f9c47a6472e136ec2f7be4\_sk

   │   └── tlsca.xinwang.blockchain.com-cert.pem

   └── users

   ├── Admin@xinwang.blockchain.com #xinwang机构的管理员的msp

   │   ├── msp

   │   │   ├── admincerts

   │   │   │   └── Admin@xinwang.blockchain.com-cert.pem

   │   │   ├── cacerts

   │   │   │   └── ca.xinwang.blockchain.com-cert.pem

   │   │   ├── keystore

   │   │   │   └── 79fd5f71dc594291202b1f5856a02fa69c74e3080e482ef3b6d15e24983b4260\_sk

   │   │   ├── signcerts

   │   │   │   └── Admin@xinwang.blockchain.com-cert.pem

   │   │   └── tlscacerts

   │   │   └── tlsca.xinwang.blockchain.com-cert.pem

   │   └── tls

   │   ├── ca.crt

   │   ├── client.crt

   │   └── client.key

   └── User1@xinwang.blockchain.com #xinwang机构的客户端sdk使用的msp

   ├── msp

   │   ├── admincerts

   │   │   └── User1@xinwang.blockchain.com-cert.pem

   │   ├── cacerts

   │   │   └── ca.xinwang.blockchain.com-cert.pem

   │   ├── keystore

   │   │   └── 032a73f393a0d8d73dad25af0fbc0bdc5117accf5e458d5bc240e3bc316e77a3\_sk

   │   ├── signcerts

   │   │   └── User1@xinwang.blockchain.com-cert.pem

   │   └── tlscacerts

   │   └── tlsca.xinwang.blockchain.com-cert.pem

   └── tls

   ├── ca.crt

   ├── client.crt

   └── client.key

#### 生成创始区块和channel定义

创始区块和channel定义在configtx.yaml文件中，内容如下

Profiles:

OrgsOrdererGenesis: #创始区块结构定义

Capabilities:

<<: \*ChannelCapabilities

Orderer:

<<: \*OrdererDefaults

Organizations:

- \*OrdererOrg

Capabilities:

<<: \*OrdererCapabilities

Consortiums:

SampleConsortium:

Organizations:

- \*Org1

OrgsChannel: #渠道channel定义

Consortium: SampleConsortium

Application:

<<: \*ApplicationDefaults

Organizations:

- \*Org1

Capabilities:

<<: \*ApplicationCapabilities

Organizations:

- &OrdererOrg

Name: OrdererOrg #Orderer机构名

ID: OrdererMSP #Orderer机构MSPID

MSPDir: #Orderer机构msp路径

AdminPrincipal: Role.ADMIN

- &Org1

Name: Xingwang #机构名

ID: XinwangMSP #机构MSPID

MSPDir: #机构msp路径

AdminPrincipal: Role.ADMIN

AnchorPeers:

- Host: peer0.xinwang.blockchain.com #xinwang的peer锚节点主机名

Port: 7051

Orderer: &OrdererDefaults

OrdererType: kafka

Addresses: #orderer节点列表

- orderer0.blockchain.com:7050

- orderer1.blockchain.com:7050

BatchTimeout: 2s

BatchSize:

MaxMessageCount: 10

AbsoluteMaxBytes: 10 MB

PreferredMaxBytes: 512 KB

MaxChannels: 0

Kafka:

Brokers: #kafka server列表

- block1:9092

- block2:9092

- block3:9092

- block4:9092

Organizations:

Application: &ApplicationDefaults

Organizations:

Capabilities:

Channel: &ChannelCapabilities

V1\_1: true

Orderer: &OrdererCapabilities

V1\_1: true

Application: &ApplicationCapabilities

V1\_1: true

生成创始区块

configtxgen -profile OrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block

生成channel定义

configtxgen -profile OrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel

#### 配置orderer.yaml

将1.1.2.5节生成的msp文件夹crypto-config/ordererOrganizations/blockchain.com/orderers/orderer0.blockchain.com和crypto-config/ordererOrganizations/blockchain.com/orderers/orderer1.blockchain.com分别复制到两台orderer主机的fabric/bin目录下。

将1.1.2.6节生成的创始区块channel-artifacts/genesis.block复制到两台orderer主机的fabric/bin目录下。

将orderer.yaml复制到fabric/bin/orderer的同级目录下，配置其中的参数：

......

# orderer节点监听ip

ListenAddress: 0.0.0.0

# orderer节点监听端口

ListenPort: 7050

# orderer节点tls通讯层私钥证书

TLS:

Enabled: true

PrivateKey: crypto-config/ordererOrganizations/blockchain.com/orderers/orderer.blockchain.com/tls/server.key

Certificate: crypto-config/ordererOrganizations/blockchain.com/orderers/orderer.blockchain.com/tls/server.crt

RootCAs:

- crypto-config/ordererOrganizations/blockchain.com/orderers/orderer.blockchain.com/tls/ca.crt

ClientAuthRequired: false

ClientRootCAs:

......

# 创始区块路径

GenesisFile: channel-artifacts/genesis.block

# Orderer节点msp路径

LocalMSPDir: crypto-config/ordererOrganizations/blockchain.com/orderers/orderer.blockchain.com/msp

# Orderer节点MSPID

LocalMSPID: OrdererMSP

......

FileLedger:

# ledger账本文件存储路径

Location: /data/hyperledger/production/orderer

......

#### 启动Orderer节点

nohup ./orderer >logs/orderer.log 2>1 &

### 初始部署新网peer节点

#### 安装Docker并导入镜像

准备工作

yum install -y yum-utils device-mapper-persistent-data lvm2 bzip2 libtool-ltdl-devel

yum-config-manager --add-repo <https://download.docker.com/linux/centos/docker-ce.repo>

安装命令

yum install docker-ce

usermod -aG docker ynet

systemctl start docker

systemctl restart docker

导入Fabric chaincode运行环境镜像

docker load --input fabric-ccenvx86\_64-1.1.0.tar

docker load --input fabric-javaenvx86\_64-1.1.0.tar

docker load --input fabric-baseos.tar

docker load --input fabric-baseimage.tar

查看已安装的image

docker images | grep -E \*\(1.1.0\|0.4.6\)

hyperledger/fabric-javaenv x86\_64-1.1.0 cd35b19d5858 6 weeks ago 1.52GB

hyperledger/fabric-ccenv x86\_64-1.1.0 031f3d67d318 6 weeks ago 1.38GB

hyperledger/fabric-baseimage x86\_64-0.4.6 dbe6787b5747 2 months ago 1.37GB

hyperledger/fabric-baseos x86\_64-0.4.6 220e5cf3fb7f 2 months ago 151MB

#### 安装Fabric binaries

同1.1.2.4节内容

#### 配置core.yaml

将1.1.2.5节生成的msp定义文件夹crypto-config/peerOrganizations/xinwang.blockchain.com/peers/peer0.xinwang.blockchain.com/msp

和crypto-config/peerOrganizations/org1.example.com/peers/peer1.xinwang.blockchain.com/msp分别复制到两台peer主机的fabric/bin目录下。

将core.yaml复制到fabric/bin/peer的同级目录下，配置其中的参数：

......

# peer节点监听ip和端口

listenAddress: 0.0.0.0:7051

# peer节点间p2p通讯的ip和端口

address: 0.0.0.0:7051

......

#peer节点tls传输层证书路径

tls:

enabled: true

clientAuthRequired: false

cert:

file: crypto-config/peerOrganizations/blockchain.xinwang.com/peers/peer1.blockchain.xinwang.com/tls/server.crt

key:

file: crypto-config/peerOrganizations/blockchain.xinwang.com/peers/peer1.blockchain.xinwang.com/tls/server.key

rootcert:

file: crypto-config/peerOrganizations/blockchain.xinwang.com/peers/peer1.blockchain.xinwang.com/tls/ca.crt

clientRootCAs:

files:

- crypto-config/peerOrganizations/blockchain.xinwang.com/peers/peer1.blockchain.xinwang.com/tls/ca.crt

......

# ledger账本文件存储路径

fileSystemPath: /data/hyperledger/production

......

# peer节点msp路径

mspConfigPath: crypto-config/peerOrganizations/blockchain.xinwang.com/peers/peer1.blockchain.xinwang.com/msp

# peer节点MSPID

localMspId: XinWangMSP

#### 启动Peer节点

nohup ./peer node start >logs/orderer.log 2>1 &

#### 安装并实例化chaincode

在Peer节点安装并实例化智能合约Chaincode：

#### 安装chaincode

./peer chaincode install -n acctflow -v 1.0 -p github.com/hyperledger/chaincode/acctflow

#### 实例化chaincode

./peer chaincode instantiate -o orderer1.blockchain.com:7050 --tls true --cafile crypto-config/ordererOrganizations/blockchain.com/orderers/orderer1.blockchain.com/msp/tlscacerts/tlsca.blockchain.com-cert.pem -C mychannel -n acctflow -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P "OR ('XinwangMSP.member')"

## 新增orderer节点入网

本例适用于在新网已按照1.1.2搭建orderer节点的基础上，增加额外的orderer节点。本例中假设新增orderer节点域名为orderer2.blockchain.com。

### Orderer节点安装

#### 安装Fabric binaries

同1.1.2.4节内容

#### 生成msp证书文件

首先使用openssl或ssh-keygen等工具为orderer2生成节点的公私钥对和tls公私钥对，然后需要使用1.1.2.5节生成的orderer root ca和orderer tls ca为其签发节点证书和tls证书。

如果使用openssl，参考命令如下：

echo "generate ECDSA private key"

#生成key

openssl ecparam -name secp384r1 -out secp384r1.pem

openssl ecparam -in secp384r1.pem -genkey -noout -out ./keystore/private.key

#openssl ecparam -out ./keystore/private.key -name prime256v1 -genkey

echo "request cert"

#证书req

openssl req -new -sha256 \

-config ../../openssl.cnf -extensions v3\_req -subj "/C=US/ST=California/L=San Francisco/O=blockchain.com/CN=$PEER\_NAME.blockchain.com" \

-key ./keystore/private.key \

-out ./signcerts/cert.csr

echo "sign by ca"

#ca签名

openssl x509 -req -sha256 \

-extfile ../../openssl.cnf -extensions v3\_sign \

-in ./signcerts/cert.csr \

-out ./signcerts/cert.crt \

-CA ./cacerts/ca.blockchain.com-cert.pem -CAkey ../../ca/$CA\_NAME -CAcreateserial

复制orderer1.blockchain.com节点的msp目录结构，将keystore替换为orderer2的私钥，signcerts替换为orderer2的证书，tls中的server.key和server.crt替换为orderer2的tls私钥和证书。

Orderer2.blockchain.com/

├── msp

│   ├── admincerts

│   │   └── Admin@blockchain.com-cert.pem

│   ├── cacerts

│   │   └── ca.blockchain.com-cert.pem

│   ├── keystore

│   │   └── private.key

│   ├── signcerts

│   │   └── orderer2.blockchain.com-cert.pem

│   └── tlscacerts

│   └── tlsca.blockchain.com-cert.pem

└── tls

├── ca.crt

├── server.crt

└── server.key

7 directories, 8 files

该msp目录即可作为orderer2节点的msp定义。

#### 配置orderer.yaml

同1.1.2.7节内容

### Orderer节点启动

nohup ./orderer >logs/orderer.log 2>1 &

### Orderer节点入网

在orderer1.blockchain.com节点上执行如下步骤：

echo "Installing jq"

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

echo "fetch channel config"

peer channel fetch config config\_block.pb -o orderer.blockchain.com:7050 -c ${CHANNEL\_NAME} --tls --cafile ${ORDERER\_CA}

echo "convert to json format"

configtxlator proto\_decode --input config\_block.pb --type common.Block | jq .data.data[0].payload.data.config > "config.json"

得到channel定义文件config.json，打开编辑，找到其中的OrdererAddresses定义：

"OrdererAddresses": {

"mod\_policy": "/Channel/Orderer/Admins",

"value": {

"addresses": [

"orderer0.blockchain.com:7050",

"orderer1.blockchain.com:7050"

]

},

"version": "0"

}

在addresses列表中新增orderer2.blockchain.com，如下：

"OrdererAddresses": {

"mod\_policy": "/Channel/Orderer/Admins",

"value": {

"addresses": [

"orderer0.blockchain.com:7050",

"orderer1.blockchain.com:7050",

"orderer2.blockchain.com:7050"

]

},

"version": "0"

}

将修改后的config.json另存为modify.json，再打包生成config\_update\_in\_envelope.pb

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

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

configtxlator compute\_update --channel\_id "${CHANNEL\_NAME}" --original original\_config.pb --updated modified\_config.pb > config\_update.pb

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

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

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

更新区块链网络的channel定义：

peer channel update -f config\_update\_in\_envelope.pb -c ${CHANNEL\_NAME} -o orderer.blockchain.com:7050 --tls --cafile ${ORDERER\_CA}

### Orderer节点停止

kill $(ps aux | grep '[o]rderer' | awk '{print $2}')

## 新增机构节点入网

本例适用于新网已经按照1.12节和1.1.3节步骤完成orderer和peer的部署和区块链网络搭建，一个新的机构即将接入区块链网络，并建立与新网的一对一的私有数据渠道channel，执行区块链交易。本例中假设构名为bankA。

### Peer节点安装

#### 安装Docker并导入镜像

同1.1.3.1节内容

#### 安装Fabric binaries

同1.1.3.2节内容

#### 生成msp证书文件

定义机构bankA的msp结构模板crypto-config.yaml，如下：

PeerOrgs:

- Name: banka #peer机构定义

Domain: banka.blockchain.com

EnableNodeOUs: false

Template:

Count: 1 #peer节点数量

Users:

Count: 1 #为客户端sdk生成证书的数量

生成orderer和peer的msp文件：

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

生成msp的文件结构如下：

crypto-config/

└── peerOrganizations

└── banka.blockchain.com

├── ca

│   ├── 176c3f60481441e1dda2052759a1ebfc9b24d098fd4e927188a3c810f5e8df4f\_sk

│   └── ca.banka.blockchain.com-cert.pem

├── msp

│   ├── admincerts

│   │   └── Admin@banka.blockchain.com-cert.pem

│   ├── cacerts

│   │   └── ca.banka.blockchain.com-cert.pem

│   └── tlscacerts

│   └── tlsca.banka.blockchain.com-cert.pem

├── peers

│   └── peer0.banka.blockchain.com

│   ├── msp

│   │   ├── admincerts

│   │   │   └── Admin@banka.blockchain.com-cert.pem

│   │   ├── cacerts

│   │   │   └── ca.banka.blockchain.com-cert.pem

│   │   ├── keystore

│   │   │   └── 8b08060b903c477d16f1978af2747d186940b09fec730548b5f311f4b7f3e929\_sk

│   │   ├── signcerts

│   │   │   └── peer0.banka.blockchain.com-cert.pem

│   │   └── tlscacerts

│   │   └── tlsca.banka.blockchain.com-cert.pem

│   └── tls

│   ├── ca.crt

│   ├── server.crt

│   └── server.key

├── tlsca

│   ├── e9b7cd4a55fbfe79f3400f1824b1852dd04b46391168aa0ea42d8d2647c8fed3\_sk

│   └── tlsca.banka.blockchain.com-cert.pem

└── users

├── Admin@banka.blockchain.com

│   ├── msp

│   │   ├── admincerts

│   │   │   └── Admin@banka.blockchain.com-cert.pem

│   │   ├── cacerts

│   │   │   └── ca.banka.blockchain.com-cert.pem

│   │   ├── keystore

│   │   │   └── 4ba54b00d077977ef87a496946dcbcc4d3caa3518914eb6600df2f82b78f7454\_sk

│   │   ├── signcerts

│   │   │   └── Admin@banka.blockchain.com-cert.pem

│   │   └── tlscacerts

│   │   └── tlsca.banka.blockchain.com-cert.pem

│   └── tls

│   ├── ca.crt

│   ├── client.crt

│   └── client.key

└── User1@banka.blockchain.com

├── msp

│   ├── admincerts

│   │   └── User1@banka.blockchain.com-cert.pem

│   ├── cacerts

│   │   └── ca.banka.blockchain.com-cert.pem

│   ├── keystore

│   │   └── 61162ed9a31cc66ed5dc4c73f5ffa0caacced09d90af97354988eed2b4e05291\_sk

│   ├── signcerts

│   │   └── User1@banka.blockchain.com-cert.pem

│   └── tlscacerts

│   └── tlsca.banka.blockchain.com-cert.pem

└── tls

├── ca.crt

├── client.crt

└── client.key

34 directories, 31 files

#### 配置core.yaml

将1.3.1.3节生成的msp定义文件夹crypto-config/peerOrganizations/banka.blockchain.com/peers/peer0.banka.blockchain.com/msp复制到peer主机的fabric/bin目录下。

将core.yaml复制到fabric/bin/peer的同级目录下，配置其中的参数：

......

# peer节点监听ip和端口

listenAddress: 0.0.0.0:7051

# peer节点间p2p通讯的ip和端口

address: 0.0.0.0:7051

......

#peer节点tls传输层证书路径

tls:

enabled: true

clientAuthRequired: false

cert:

file: crypto-config/peerOrganizations/blockchain.banka.com/peers/peer0.blockchain.banka.com/tls/server.crt

key:

file: crypto-config/peerOrganizations/blockchain.banka.com/peers/peer0.blockchain.banka.com/tls/server.key

rootcert:

file: crypto-config/peerOrganizations/blockchain.banka.com/peers/peer0.blockchain.banka.com/tls/ca.crt

clientRootCAs:

files:

- crypto-config/peerOrganizations/blockchain.banka.com/peers/peer0.blockchain.banka.com/tls/ca.crt

......

# ledger账本文件存储路径

fileSystemPath: /data/hyperledger/production

......

# peer节点msp路径

mspConfigPath: crypto-config/peerOrganizations/blockchain.banka.com/peers/peer0.blockchain.banka.com/msp

# peer节点MSPID

localMspId: BankAMSP

### Peer节点启动

nohup ./peer node start >logs/orderer.log 2>1 &

### Peer节点入网

BankA的peer节点入网后需要加入新网为其创建的一对一私有数据渠道channel，需要执行如下步骤：

BankA将ca证书、admin证书、tls ca证书打包成msp文件目录，结构如下：

crypto-config/peerOrganizations/banka.blockchain.com/msp/

├── admincerts

│   └── Admin@banka.blockchain.com-cert.pem

├── cacerts

│   └── ca.banka.blockchain.com-cert.pem

└── tlscacerts

└── tlsca.banka.blockchain.com-cert.pem

BankA将打包的msp发送新网，新网在peer节点创建双方的一对一私有channel，配置文件如下：

Profiles:

OrgsChannel: #渠道channel定义

Consortium: SampleConsortium

Application:

<<: \*ApplicationDefaults

Organizations:

- \*XinwangOrg

- \*BankAOrg

Capabilities:

<<: \*ApplicationCapabilities

Organizations:

- &XinwangOrg

Name: Xingwang #机构名

ID: XinwangMSP #机构MSPID

MSPDir: #机构msp路径

AdminPrincipal: Role.ADMIN

AnchorPeers:

- Host: peer0.xinwang.blockchain.com #xinwang的peer锚节点主机名

Port: 7051

- &BankAOrg

Name: BankA #机构名

ID: BankAMSP #机构MSPID

MSPDir: #机构msp路径

AdminPrincipal: Role.ADMIN

AnchorPeers:

- Host: peer0.banka.blockchain.com #bankA的peer锚节点主机名

Port: 7051

Application: &ApplicationDefaults

Organizations:

Capabilities:

Channel: &ChannelCapabilities

V1\_1: true

Orderer: &OrdererCapabilities

V1\_1: true

Application: &ApplicationCapabilities

V1\_1: true

新网在peer节点生成channel定义：

configtxgen -profile OrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel

新网将生成好的渠道定义文件channel.tx发送给BankA，BankA将peer节点加入channel中，命令如下：

./peer channel join -b mychannel.block

在Peer节点安装并实例化智能合约Chaincode：

#### 安装chaincode

./peer chaincode install -n acctflow -v 1.0 -p github.com/hyperledger/chaincode/acctflow

#### 实例化chaincode

./peer chaincode instantiate -o orderer1.blockchain.com:7050 --tls true --cafile crypto-config/ordererOrganizations/blockchain.com/orderers/orderer1.blockchain.com/msp/tlscacerts/tlsca.blockchain.com-cert.pem -C mychannel -n acctflow -v 1.0 -c '{"Args":["init","a","100","b","200"]}' -P "OR ('XinwangMSP.member')"

### Peer节点停止

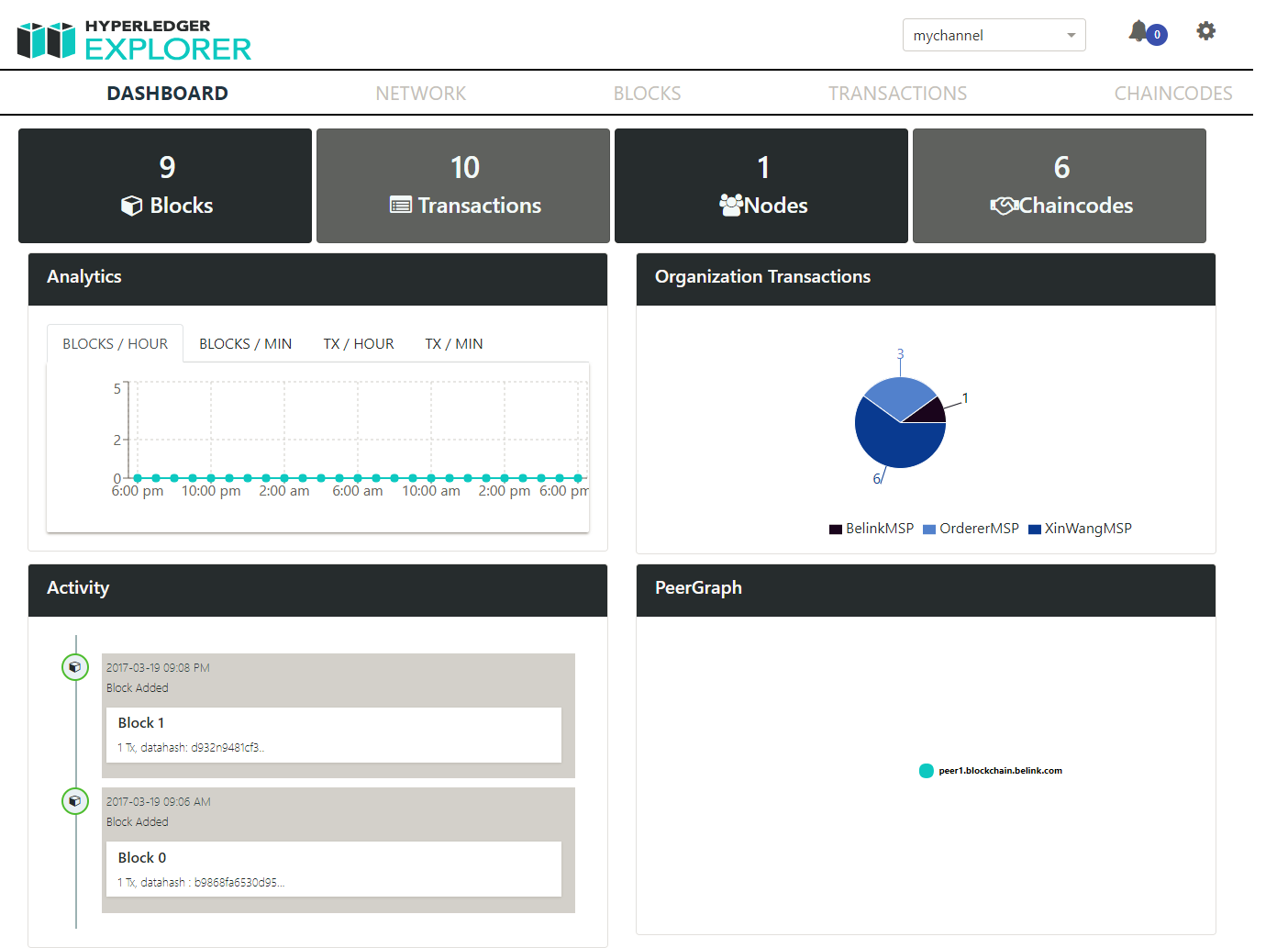
停止peer节点

kill $(ps aux | grep '[p]eer node' | awk '{print $2}')

# 可用性监控

## 区块节点浏览

通过Blockchain-explorer管理工具可以提供对区块链网络的实时信息查看功能，包括区块高度、最新生成区块、区块内交易和交易内容。



Blockchain-explorer建议搭建到peer节点所在主机，步骤如下：

环境准备：

安装nodejs 6.9.x (不支持7.0以上)

安装PostgreSQL 9.5或以上版本

下载代码：

git clone https://github.com/hyperledger/blockchain-explorer.git.

cd blockchain-explorer

初始化postgres数据库：

sudo -u postgres psql

\i app/db/explorerpg.sql

\i app/db/updatepg.sql

检查数据库内容：

\l view created fabricexplorer database

\d view created tables

修改config.yaml配置文件，将peer节点的端口、tls证书，以及admin的私钥和证书路径配置在文件中：

{

"network-config": {

"org1": {

"name": "XinwangOrg",

"mspid": "XinwangMSP",

"peer1": {

"requests": "grpcs://127.0.0.1:7051",

"events": "grpcs://127.0.0.1:7053",

"server-hostname": "peer0.xinwang.blockchain.com",

"tls\_cacerts": "crypto-config/peer

Organizations/xinwang.blockchain.com/peers/peer0.xinwang.blockchain.com/tls/ca.crt"

},

"admin": {

"key": "crypto-config/peerOrganiza

tions/xinwang.blockchain.com/users/Admin@xinwang.blockchain.com/msp/keystore",

"cert": "crypto-config/peerOrganiz

ations/xinwang.blockchain.com/users/Admin@xinwang.blockchain.com/msp/signcerts"

}

}

},

"host": "localhost",

"port": "8080", #浏览器访问端口

"channel": ["mychannel"], #需要浏览的channel列表

"keyValueStore": "/tmp/fabric-client-kvs",

"eventWaitTime": "30000",

"users":[

{

"username":"admin",

"secret":"adminpw"

}

],

"pg": {

"host": "127.0.0.1",

"port": "5432",

"database": "fabricexplorer",

"username": "hppoc",

"passwd": "password"

},

"license": "Apache-2.0"

}

构建项目：

cd blockchain-explorer/app/test

npm install

npm run test

cd blockchain-explorer

npm install

cd client/

npm install

npm test -- -u --coverage

npm run build

启动项目：

cd blockchain-explorer/

./start.sh

查看日志：

tail -f log.log

通过浏览器访问http://localhost:8080，即可查看区块链网络信息。

## 组网节点监控

### Orderer节点监控

测试网络响应时间：通过ping工具测试访问orderer节点的相应时间

ping orderer0.blockchain.com

ping orderer1.blockchain.com

测试Orderer节点打包执行状态：通过命令行查询channel的config信息，如果查询正常，可以认为Orderer目前工作正常

peer channel fetch config config\_block.pb -o orderer0.blockchain.com:7050 -c mychannel --tls --cafile crypto-config/ordererOrganizations/blockchain.com/tlsca/tlsca.blockchain.com-cert.pem

peer channel fetch config config\_block.pb -o orderer1.blockchain.com:7050 -c mychannel --tls --cafile crypto-config/ordererOrganizations/blockchain.com/tlsca/tlsca.blockchain.com-cert.pem

正常的返回：

2018-06-04 16:20:57.718 CST [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized

2018-06-04 16:20:57.722 CST [main] main -> INFO 002 Exiting.....

异常的返回：Orderer节点无法访问

Error: failed to create deliver client: orderer client failed to connect to orderer.blockchain.com:7050: failed to create new connection: context deadline exceeded

### Peer节点监控

测试网络响应时间：通过ping工具测试访问orderer节点的相应时间

ping peer0.xinwang.blockchain.com

ping peer1.xinwang.blockchain.com

测试peer节点背书执行状态：通过命令行执行调用peer节点的系统级chaincode，执行查询区块链信息的接口GetChainInfo，如果调用正常，可以认为peer节点能够正常执行chaincode

export CORE\_PEER\_ADDRESS=peer0.xinwang.blockchain.com:7051

peer chaincode query -C mychannel -n qscc -c '{"Args":["GetChainInfo", "mychannel"]}'

export CORE\_PEER\_ADDRESS=peer1.xinwang.blockchain.com:7051

peer chaincode query -C mychannel -n qscc -c '{"Args":["GetChainInfo", "mychannel"]}'

正常的返回：

2018-06-04 16:51:05.073 CST [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc

2018-06-04 16:51:05.073 CST [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc

Query Result:\*\*\*\*\*\*\*\*\*\*

2018-06-04 16:51:05.076 CST [main] main -> INFO 003 Exiting.....

异常的返回：Peer节点无法访问

Error: Error getting endorser client chaincode: endorser client failed to connect to peer0.xinwang.blockchain.com:7051: failed to create new connection: context deadline exceeded

## 节点主机监控

### 进程/服务、端口

检查Zookeeper进程状态和监听端口

ps -ef | grep zookeeper

netstat -anp| grep 2182

检查Kafka进程状态查询和监听端口

ps -ef | grep kafka

netstat -anp| grep 9092

检查Orderer节点进程状态查询和监听端口

ps -ef | grep orderer

netstat -anp| grep 7050

检查Peer节点进程状态查询和监听端口

ps -ef | grep peer

netstat -anp| grep 7051

### CPU/内存/磁盘

检查orderer节点的cpu和内存占用

ps -o %cpu,%mem,cmd -e | grep '[o]rderer'

1.4 0.7 ./orderer

检查peer节点的cpu和内存占用

ps -o %cpu,%mem,cmd -e | grep '[p]eer node'

1.5 0.6 ./peer node start

检查orderer或peer节点的磁盘占用

df -h

# 故障排查和恢复

## Orderer节点主机故障排查和恢复

### 故障分析

Orderer节点在区块链网络中负责将peer节点背书的交易进行排序，打包生成区块数据，并广播到全网所有peer节点。

为防止单点故障导致Orderer服务不可用，1.1.1节的网络拓扑中部署了两台Orderer节点主机。单台Orderer节点故障不会影响整个区块链网络的正常服务。如果两台Orderer同时故障，区块链网络将停止打包生成区块的操作，导致client无法提交已经背书的交易，client程序会抛出orderer无法访问的异常，如下：

2018-05-29 07:39:04,174 main DEBUG Channel:2548 - Channel mychannel sending transaction to orderer(s) with TxID 7f01ab59c8a99f3da598a160f884a14e1fc6b9fbcaa85896b95ed75d1a3b1e2e

2018-05-29 07:39:04,174 main DEBUG Orderer:135 - Order.sendTransaction name: orderer1, url: grpcs://orderer1.blockchain.com:17050

2018-05-29 07:39:05,548 grpc-default-executor-2 ERROR OrdererClient:157 - Received error on channel mychannel, orderer orderer1, url grpcs://orderer1.blockchain.com:17050, UNAVAILABLE: io exception

io.grpc.StatusRuntimeException: UNAVAILABLE: io exception

一般情况下Orderer节点的故障原因可能包括但不限于以下几种情况：

1. 软件系统故障，例如系统内存或磁盘空间不足，或者未知的程序错误，导致Orderer进程服务停止；
2. 网络故障，导致Orderer一段时间离线，无法访问；
3. 硬件设备故障，停电宕机；
4. 磁盘数据损坏，无法读写账本数据。

### 故障排查标准步骤

检查Zookeeper进程状态和监听端口

ps -ef | grep zookeeper

netstat -anp| grep 2182

检查Kafka进程状态查询和监听端口

ps -ef | grep kafka

netstat -anp| grep 9092

检查Orderer节点进程状态查询和监听端口

ps -ef | grep orderer

netstat -anp| grep 7050

检查orderer节点的cpu和内存占用

ps -o %cpu,%mem,cmd -e | grep '[o]rderer'

1.4 0.7 ./orderer

检查orderer节点的磁盘占用

df -h

测试Orderer节点响应时间

ping orderer0.blockchain.com

ping orderer1.blockchain.com

测试Orderer节点打包执行状态

peer channel fetch config config\_block.pb -o orderer0.example.com:7050 -c mychannel --tls --cafile crypto-config/ordererOrganizations/blockchain.com/tlsca/tlsca.blockchain.com-cert.pem

peer channel fetch config config\_block.pb -o orderer1.example.com:7050 -c mychannel --tls --cafile crypto-config/ordererOrganizations/blockchain.com/tlsca/tlsca.blockchain.com-cert.pem

正常的返回：

2018-06-04 16:20:57.718 CST [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized

2018-06-04 16:20:57.722 CST [main] main -> INFO 002 Exiting.....

异常的返回：Orderer节点无法访问

Error: failed to create deliver client: orderer client failed to connect to orderer.blockchain.com:7050: failed to create new connection: context deadline exceeded

检查Orderer日志文件中的错误信息：Orderer日志在启动节点时定义了输出位置和文件名

nohup ./orderer >logs/orderer.log 2>1 &

常见的错误日志：

|  |  |
| --- | --- |
| 错误信息 | 可能的错误原因 |
| Transactions of type ORDERER\_TRANSACTION cannot be Broadcast | 无法对外广播Orderer交易：当前Orderer节点通讯异常 |
| Transactions of type CONFIG cannot be Broadcast | 无法对外广播CONFIG交易：当前Orderer节点通讯异常 |
| Unable to bootstrap orderer. Error reading genesis block file | 无法启动Orderer节点：创始区块文件损坏 |
| Error reading from channel, cause was：... | 无法读取渠道数据：当前Orderer节点通讯异常 |
| Block number should have been M but was N | 全网区块高度大于当前Orderer节点区块高度：当前Orderer节点通讯异常，导致区块同步速度慢 |

### 故障恢复

对于一般的软件故障、系统或网络故障，只需重启Orderer服务即可恢复服务，重启后Orderer可继续提供区块打包服务。

对于硬件或磁盘故障等情况，可能需要更换主机设备重新安装Orderer服务，安装步骤请参见1.1.2.4～1.1.2.8节内容，其中MSP和Orderer域名直接使用原主机的配置方案即可。如果原MSP丢失，请使用Orderer ca重新为其颁发证书，执行步骤请参见1.2.1.2节。

由于Orderer节点只负责区块链中的交易打包和区块广播，因此没有需要迁移的持久化数据，Orderer节点的OrdererLedger中存储的是已经广播给peer的历史区块数据。

## Peer节点主机故障排查和恢复

### 故障分析

Peer节点是区块链网络中由联盟机构部署的背书节点，peer节点负责从Orderer节点同步区块数据到本地，在本地维护一个BlockLedger的副本，同时负责对client提交的交易进行签名背书，并为接入Peer节点的client提供查询服务。

为防止单点故障导致Peer节点服务不可用，1.1.1节的网络拓扑中为每个联盟机构都部署了两台以上的peer节点主机。单台Peer节点故障不会影响整个区块链网络的正常服务。如果一个机构的所有Peer节点同时故障，peer节点将无法提供交易背书和查询服务，账本同步也将停止。此时client程序会抛出peer节点无法访问的异常，如下：

2018-05-29 09:07:00,092 main ERROR Channel:2413 - Sending proposal to peer0 failed because of: gRPC failure=Status{code=UNAVAILABLE, description=io exception, cause=io.netty.channel.AbstractChannel$AnnotatedConnectException: Connection refused: no further information: peer0.blockchain.belink.com/101.251.195.188:7051

一般情况下Peer节点的故障原因可能包括但不限于以下几种情况：

1. 软件系统故障，例如系统内存或磁盘空间不足，或者未知的程序错误，导致Peer进程服务停止；
2. 网络故障，导致Peer一段时间离线，无法访问；
3. 硬件设备故障，停电宕机；
4. 磁盘数据损坏，无法读写账本数据。

### 故障排查标准步骤

检查Peer节点进程状态查询和监听端口

ps -ef | grep peer

netstat -anp| grep 7051

检查peer节点的cpu和内存占用

ps -o %cpu,%mem,cmd -e | grep '[p]eer node'

1.5 0.6 ./peer node start

检查peer节点的磁盘占用

df -h

测试Peer节点网络响应时间

ping peer0.xinwang.blockchain.com

ping peer1.xinwang.blockchain.com

测试peer节点背书执行状态

export CORE\_PEER\_ADDRESS=peer0.xinwang.blockchain.com:7051

peer chaincode query -C mychannel -n qscc -c '{"Args":["GetChainInfo", "mychannel"]}'

export CORE\_PEER\_ADDRESS=peer1.xinwang.blockchain.com:7051

peer chaincode query -C mychannel -n qscc -c '{"Args":["GetChainInfo", "mychannel"]}'

正常的返回：

2018-06-04 16:51:05.073 CST [chaincodeCmd] checkChaincodeCmdParams -> INFO 001 Using default escc

2018-06-04 16:51:05.073 CST [chaincodeCmd] checkChaincodeCmdParams -> INFO 002 Using default vscc

Query Result:\*\*\*\*\*\*\*\*\*\*

2018-06-04 16:51:05.076 CST [main] main -> INFO 003 Exiting.....

异常的返回：Peer节点无法访问

Error: Error getting endorser client chaincode: endorser client failed to connect to peer0.xinwang.blockchain.com:7051: failed to create new connection: context deadline exceeded

检查Peer日志文件中的错误信息：Peer日志在启动节点时定义了输出位置和文件名

nohup ./peer node start >logs/orderer.log 2>1 &

常见的错误日志：

|  |  |
| --- | --- |
| 错误信息 | 可能的错误原因 |
| Error while unmarshaling proposal response payload | 无法解包背书请求：client与peer间通讯丢包导致 |
| Error getting orderer endpoint | 无法连接orderer节点：peer与orderer节点间通讯异常 |
| Error sending transaction | 无法发送交易：当前peer离线 |
| Error connecting due to ... | 无法建立连接：当前peer离线 |
| Error loadding config block | 无法加载channel定义区块：peer与orderer节点间通讯异常 |
| Failed to get Peer Endpoint | 无法连接其他peer节点：当前peer离线，或对方peer通讯异常 |
| Failed to return new GRPC server | 无法创建GRPC通讯server：当前peer离线 |

### 故障恢复

对于一般的软件故障、系统或网络故障，只需重启Peer服务即可恢复服务，重启后会自动同步错过的区块数据。

对于硬件或磁盘故障等情况，可能需要更换主机设备重新安装Peer服务，安装步骤请参见1.3.1节内容，其中MSP和Peer域名直接使用原主机的配置方案即可。如果原MSP丢失，请使用原机构ca重新为其颁发证书，执行步骤请参见1.2.1.2节。

对于重新部署的peer节点，接入网后会从创世区块开始重新下载历史的区块数据，生成本地的账本副本，如果历史区块数量较大，该过程会比较耗时，区块同步结束后该peer节点的区块高度将于全网保持一致，此时peer节点才能正常执行交易背书操作。

## 区块链组网故障排查和恢复

区块链组网的整体故障可能的原因是：

* 全部Orderer节点离线，导致网络无法打包生成新的区块；
* 联盟机构的所有peer节点离线，导致该机构所在的channel无法执行背书交易。

在全网故障的情况下，只要至少有一个peer节点保存有账本的完整副本，当重新组网后，其他peer节点就可自动进行账本的同步，直到全网账本保持一致。

### 故障排查标准步骤

Orderer节点故障排查请参见3.1.2节。

Peer节点故障排查请参见3.2.2节。

### 故障恢复

对于Orderer节点全部离线的情况，按照3.1.3节所述恢复流程逐个恢复Orderer节点即可，只要第一个Orderer节点恢复在线状态，区块链网络即可恢复区块打包的功能。

对于非新网的联盟机构Peer节点全部离线的情况，只会影响该机构与新网间的私有channel，故障期间该机构无法执行读写链操作，但新网仍然可对channel数据进行读写，同时其他的channel不受影响。机构peer节点的恢复步骤请参见3.2.3节，恢复后该机构peer会自动同步故障期间写入的区块数据。

对于新网peer节点全部离线的情况，由于新网参与了所有一对一私有channel，故障期间新网将无法对所有channel执行读写链操作，但其他机构仍然可对channel数据进行读写。新网peer节点的恢复步骤请参见3.2.3节，恢复后peer会自动同步故障期间写入的区块数据。