> 我们可以自己组建一个Fabric网路, 网络结构如下:
>
> - 排序节点 1 个
> - 组织个数 2 个, 分别为go和cpp, 每个组织分别有两个peer节点, 用户个数为3

机构名称	组织标识符	组织ID
Go学科	org_go	OrgGoMSP
CPP	org_cpp	OrgCppMSP

一些理论基础:

- 域名
  - baidu.com
  - jd.com
  - taobao.com
- msp
  - Membership service provider (MSP)是一个提供虚拟成员操作的管理框架的组件。
  - 账号
    - 都谁有msp
      - 每个节点都有一个msp账号
      - 每个用户都有msp账号
- 锚节点
  - 代表所属组织和其他组织进行通信的节点

1. 生成fabric证书

1.1 命令介绍

```
$cryptogen --help
```

1.2 证书的文件的生成 - yaml

- **配置文件的模板**

```
# ---------------------------------------------------------------------------
# "OrdererOrgs" - Definition of organizations managing orderer nodes
# ---------------------------------------------------------------------------
OrdererOrgs:    # 排序节点组织信息
  # ---------------------------------------------------------------------------
  # Orderer
  # ---------------------------------------------------------------------------
```

```
  - Name: Orderer   # 排序节点组织的名字
    Domain: example.com # 根域名, 排序节点组织的根域名
    Specs:
      - Hostname: orderer # 访问这台orderer对应的域名为: orderer.example.com
      - Hostname: order2 # 访问这台orderer对应的域名为: order2.example.com
# ---------------------------------------------------------------------------
# "PeerOrgs" - Definition of organizations managing peer nodes
# ---------------------------------------------------------------------------
PeerOrgs:
  # ---------------------------------------------------------------------------
  # Org1
  # ---------------------------------------------------------------------------
  - Name: Org1  # 第一个组织的名字, 自己指定
    Domain: org1.example.com    # 访问第一个组织用到的根域名
    EnableNodeOUs: true         # 是否支持node.js
    Template:                   # 模板, 根据默认的规则生成2个peer存储数据的节点
      Count: 2 # 1. peer0.org1.example.com 2. peer1.org1.example.com
    Users:     # 创建的普通用户的个数
      Count: 3

  # ---------------------------------------------------------------------------
  # Org2: See "Org1" for full specification
  # ---------------------------------------------------------------------------
  - Name: Org2
    Domain: org2.example.com
    EnableNodeOUs: true
    Template:
      Count: 2
    Specs:
      - Hostname: hello
    Users:
      Count: 1
```

> 上边使用的域名, 在真实的生成环境中需要注册备案, 测试环境, 域名自己随便指定就可以

- 根据要求编写好的配置文件, 配置文件名: crypto-config.yaml

```
# crypto-config.yaml
# ---------------------------------------------------------------------------
# "OrdererOrgs" - Definition of organizations managing orderer nodes
# ---------------------------------------------------------------------------
OrdererOrgs:
  # ---------------------------------------------------------------------------
  # Orderer
  # ---------------------------------------------------------------------------
  - Name: Orderer
    Domain: itcast.com
    Specs:
      - Hostname: orderer

# ---------------------------------------------------------------------------
# "PeerOrgs" - Definition of organizations managing peer nodes
# ---------------------------------------------------------------------------
```

```
  PeerOrgs:
    # ---------------------------------------------------------------------------
    # Org1
    # ---------------------------------------------------------------------------
    - Name: OrgGo
      Domain: orggo.itcast.com
      EnableNodeOUs: true
      Template:
        Count: 2
      Users:
        Count: 3

    # ---------------------------------------------------------------------------
    # Org2: See "Org1" for full specification
    # ---------------------------------------------------------------------------
    - Name: OrgCpp
      Domain: orgcpp.itcast.com
      EnableNodeOUs: true
      Template:
        Count: 2
      Users:
        Count: 3

```

- 通过命令生成证书文件

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

2. 创始块文件和通道文件的生成

2.1 命令介绍

```
$ configtxgen --help
  # 输出创始块区块文件的路径和名字
  `-outputBlock string`
  # 指定创建的channel的名字, 如果没指定系统会提供一个默认的名字.
  `-channelID string`
  # 表示输通道文件路径和名字
  `-outputCreateChannelTx string`
  # 指定配置文件中的节点
  `-profile string`
  # 更新channel的配置信息
  `-outputAnchorPeersUpdate string`
  # 指定所属的组织名称
  `-asOrg string`
  # 要想执行这个命令, 需要一个配置文件 configtx.yaml
```

2.2 创始块/通道文件的生成

- **配置文件的编写** - 参考模板

```yaml

---
################################################################################
#
#   Section: Organizations
#
#   - This section defines the different organizational identities which will
#   be referenced later in the configuration.
#
################################################################################
Organizations:          # 固定的不能改
    - &OrdererOrg       # 排序节点组织，自己起个名字
        Name: OrdererOrg    # 排序节点的组织名
        ID: OrdererMSP      # 排序节点组织的ID
        MSPDir: crypto-config/ordererOrganizations/example.com/msp # 组织的msp账号信息

    - &Org1         # 第一个组织，名字自己起
        Name: Org1MSP # 第一个组织的名字
        ID: Org1MSP     # 第一个组织的ID
        MSPDir: crypto-config/peerOrganizations/org1.example.com/msp
        AnchorPeers: # 锚节点
            - Host: peer0.org1.example.com  # 指定一个peer节点的域名
              Port: 7051                    # 端口不要改

    - &Org2
        Name: Org2MSP
        ID: Org2MSP
        MSPDir: crypto-config/peerOrganizations/org2.example.com/msp
        AnchorPeers:
            - Host: peer0.org2.example.com
              Port: 7051

################################################################################
#
#   SECTION: Capabilities，在fabric1.1之前没有，设置的时候全部设置为true
#
################################################################################
Capabilities:
    Global: &ChannelCapabilities
        V1_1: true
    Orderer: &OrdererCapabilities
        V1_1: true
    Application: &ApplicationCapabilities
        V1_2: true

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

```

```
54  ################################################################################
55  #
56  #   SECTION: Orderer
57  #
58  ################################################################################
59  Orderer: &OrdererDefaults
60      # Available types are "solo" and "kafka"
61      # 共识机制 == 排序算法
62      OrdererType: solo   # 排序方式
63      Addresses:          # orderer节点的地址
64          - orderer.example.com:7050  # 端口不要改
65
66      # BatchTimeout,MaxMessageCount,AbsoluteMaxBytes只要一个满足, 区块就会产生
67      BatchTimeout: 2s    # 多长时间产生一个区块
68      BatchSize:
69          MaxMessageCount: 10     # 交易的最大数据量, 数量达到之后会产生区块, 建议100左右
70          AbsoluteMaxBytes: 99 MB # 数据量达到这个值, 会产生一个区块, 32M/64M
71          PreferredMaxBytes: 512 KB
72      Kafka:
73          Brokers:
74              - 127.0.0.1:9092
75      Organizations:
76
77  ################################################################################
78  #
79  #   Profile
80  #
81  ################################################################################
82  Profiles:   # 不能改
83      TwoOrgsOrdererGenesis:  # 区块名字, 随便改
84          Capabilities:
85              <<: *ChannelCapabilities
86          Orderer:
87              <<: *OrdererDefaults
88              Organizations:
89                  - *OrdererOrg
90              Capabilities:
91                  <<: *OrdererCapabilities
92          Consortiums:
93              SampleConsortium:   # 这个名字可以改
94                  Organizations:
95                      - *Org1
96                      - *Org2
97      TwoOrgsChannel: # 通道名字, 可以改
98          Consortium: SampleConsortium    # 这个名字对应93行
99          Application:
100             <<: *ApplicationDefaults
101             Organizations:
102                 - *Org1
103                 - *Org2
104             Capabilities:
105                 <<: *ApplicationCapabilities
106
```

- 按照要求编写的配置文件

```
# configtx.yaml
---
################################################################################
#
#   Section: Organizations
#
################################################################################
Organizations:
    - &OrdererOrg
        Name: OrdererOrg
        ID: OrdererMSP
        MSPDir: crypto-config/ordererOrganizations/itcast.com/msp

    - &org_go
        Name: OrgGoMSP
        ID: OrgGoMSP
        MSPDir: crypto-config/peerOrganizations/orggo.itcast.com/msp
        AnchorPeers:
            - Host: peer0.orggo.itcast.com
              Port: 7051

    - &org_cpp
        Name: OrgCppMSP
        ID: OrgCppMSP
        MSPDir: crypto-config/peerOrganizations/orgcpp.itcast.com/msp
        AnchorPeers:
            - Host: peer0.orgcpp.itcast.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.itcast.com:7050
    BatchTimeout: 2s
    BatchSize:
        MaxMessageCount: 100
        AbsoluteMaxBytes: 32 MB
        PreferredMaxBytes: 512 KB
    Kafka:
        Brokers:
            - 127.0.0.1:9092
    Organizations:

################################################################################
#
#   Profile
#
################################################################################
Profiles:
    ItcastOrgsOrdererGenesis:
        Capabilities:
            <<: *ChannelCapabilities
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            SampleConsortium:
                Organizations:
                    - *org_go
                    - *org_cpp
    ItcastOrgsChannel:
        Consortium: SampleConsortium
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *org_go
                - *org_cpp
            Capabilities:
                <<: *ApplicationCapabilities

```

- **执行命令生成文件**

> -profile 后边的参数从configtx.yaml中的Profiles 里边的配置项

- 生成创始块文件

```
$ configtxgen -profile ItcastOrgsOrdererGenesis -outputBlock ./genesis.block
- 在当前目录下得到一个文件: genesis.block
```

- 生成通道文件

```
$ configtxgen -profile ItcastOrgsChannel -outputCreateChannelTx channel.tx -channelID itcastchannel
```

- 生成锚节点更新文件

> 这个操作是可选的

```
# 每个组织都对应一个锚节点的更新文件
# go组织锚节点文件
$ configtxgen -profile ItcastOrgsChannel -outputAnchorPeersUpdate GoMSPanchors.tx -channelID itcastchannel -asOrg OrgGoMSP
# cpp组织锚节点文件
$ configtxgen -profile ItcastOrgsChannel -outputAnchorPeersUpdate CppMSPanchors.tx -channelID itcastchannel -asOrg OrgCppMSP
```

```
# 查看生成的文件
$ tree -L 1
.
├── channel-artifacts
├── channel.tx  ----------> 生成的通道文件
├── configtx.yaml
├── CppMSPanchors.tx -----> 生成的cpp组织锚节点文件
├── crypto-config
├── crypto-config.yaml
├── genesis.block --------> 生成的创始块文件
└── GoMSPanchors.tx ------> 生成的go组织锚节点文件
```

3. docker-compose文件的编写

3.1 客户端角色需要使用的环境变量

```
# 客户端docker容器启动之后, go的工作目录
- GOPATH=/opt/gopath    # 不需要修改
# docker容器启动之后, 对应的守护进程的本地套接字, 不需要修改
- CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
- CORE_LOGGING_LEVEL=INFO    # 日志级别
- CORE_PEER_ID=cli           # 当前客户端节点的ID, 自己指定
- CORE_PEER_ADDRESS=peer0.org1.example.com:7051 # 客户端连接的peer节点
- CORE_PEER_LOCALMSPID=Org1MSP  # 组织ID
- CORE_PEER_TLS_ENABLED=true    # 通信是否使用tls加密
- CORE_PEER_TLS_CERT_FILE=      # 证书文件
 /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
```

```
- CORE_PEER_TLS_KEY_FILE=        # 私钥文件
 /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.key
-CORE_PEER_TLS_ROOTCERT_FILE=    # 根证书文件
 /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
# 指定当前客户端的身份
- CORE_PEER_MSPCONFIGPATH=
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
```

3.2 orderer节点需要使用的环境变量

```
- ORDERER_GENERAL_LOGLEVEL=INFO # 日志级别
- ORDERER_GENERAL_LISTENADDRESS=0.0.0.0 # orderer节点监听的地址
- ORDERER_GENERAL_GENESISMETHOD=file    # 创始块的来源，指定file来源就是文件中
# 创始块对应的文件，这个不需要改
- ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
- ORDERER_GENERAL_LOCALMSPID=OrdererMSP # orderer节点所属的组的ID
- ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp  # 当前节点的msp账号路径
# enabled TLS
- ORDERER_GENERAL_TLS_ENABLED=true  # 是否使用tls加密
- ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key     # 私钥
- ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt   # 证书
- ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]         # 根证书
```

3.3 peer节点需要使用的环境变量

```
- CORE_PEER_ID=peer0.orggo.test.com # 当前peer节点的名字，自己起
# 当前peer节点的地址信息
- CORE_PEER_ADDRESS=peer0.orggo.test.com:7051
# 启动的时候，指定连接谁，一般写自己就行
- CORE_PEER_GOSSIP_BOOTSTRAP=peer0.orggo.test.com:7051
# 为了被其他节点感知到，如果不设置别的节点不知有该节点的存在
- CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.orggo.test.com:7051
- CORE_PEER_LOCALMSPID=OrgGoMSP
# docker的本地套接字地址，不需要改
- CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
# 当前节点属于哪个网络
- CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=network_default
- CORE_LOGGING_LEVEL=INFO
- CORE_PEER_TLS_ENABLED=true
- CORE_PEER_GOSSIP_USELEADERELECTION=true   # 释放自动选举leader节点
- CORE_PEER_GOSSIP_ORGLEADER=false          # 当前不是leader
- CORE_PEER_PROFILE_ENABLED=true    # 在peer节点中有一个profile服务
- 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
```