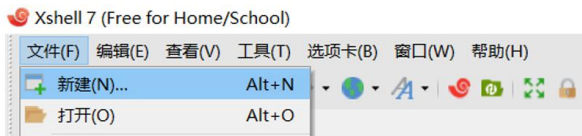



《区块链技术基础与实践》实验指导手册 1

实验名称: Fabric 环境配置与合约执行	实验地点:
实验仪器: Linux 实验环境	
<p>一、实验目的:</p> <ol style="list-style-type: none">1、了解并熟悉 Linux 系统的使用;2、了解 Fabric 的运行环境并进行搭建;3、初步掌握链码(智能合约)的编写;4、掌握 scp 命令的使用;5、了解链码(智能合约)的安装与使用。	
<p>二、实验内容:</p> <p>0 实验前准备</p> <p>本实验采用两台服务器进行操作,一台为 Linux 系统,一台为 windows 系统。在 Linux 系统中进行区块链环境的配置、链码部署,在 windows 系统中进行链码的编写、依赖下载等操作。</p> <p>Linux 系统无法连接互联网,因此代码、配置文件的编写建议在 Windows 系统中编写后再上传至服务器。</p> <p>0.1 windows 开发环境</p> <ol style="list-style-type: none">(1) Golang: 使用的版本为 1.15.15。下载地址: Golang Downloads(2) IDE: 使用 Visual Studio Code、Goland 等均可。 <p>0.2 windows 连接 Linux</p> <ol style="list-style-type: none">(1) 在 Linux 服务器中输入: ifconfig, 即可查看服务器 IP 地址,在后续连接步骤需要用到。 <p>在 windows 系统中,打开 xshell 软件,点击“文件”->“新建”,开始新建服务器连接。</p>  <p>在弹出的窗口中,协议选择“SSH”,然后输入主机(即 Linux 的 IP 地址),端口号输入 22,然后点击“连接”,在随后弹出的对话框中输入 Linux 系统的用户名与密码,即成功连接至 Linux 系统。然后可以在 xshell 中操作 Linux 系统。</p>  <p>0.3 文件上传</p> <p>代码、配置文件在编辑好后需要上传至服务器,可以使用 scp、filezilla、xftp 等方式进行传输,本</p>	

实验中选择 filezilla 工具。

下载地址: [FileZilla Download](#), 选择 filezilla client 进行下载安装。

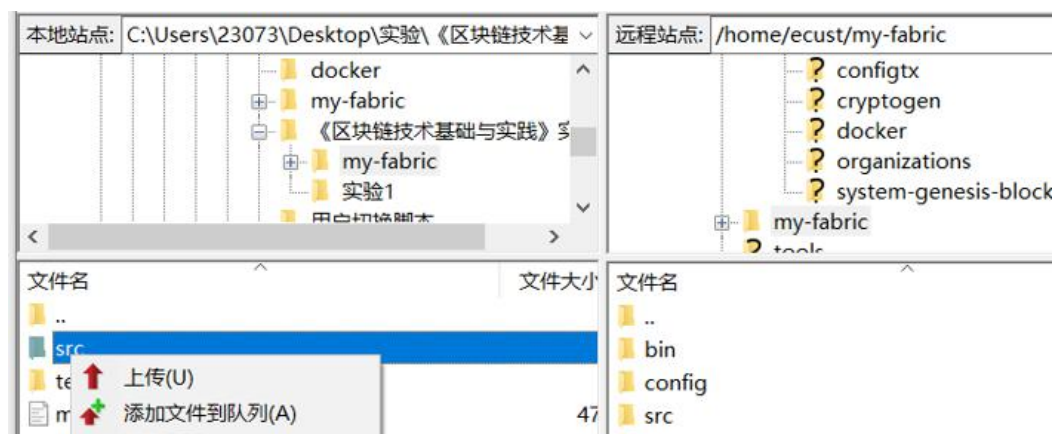
使用方式:

(1) 分别输入服务器 IP 地址、用户名、密码、端口号 22 后, 点击“快速连接”。

主机(H):	172.18.120.220	用户名(U):	ecust	密码(W):	●●●●	端口(P):	22	快速连接(Q)	▼
--------	----------------	---------	-------	--------	------	--------	----	---------	---

(2) 左侧为本地文件, 右侧为 linux 系统中的文件, 进入对应目录。

在需要上传的文件夹处右键, 选择上传, 即可将文件夹或文件上传至服务器。



0.4 实验过程

为方便同学操作, linux 服务器中, 步骤 2.2 及之前的操作均已完成。本实验从步骤 2.3 开始做起。

1 实验运行环境

1.1 Fabric 版本

Hyperledger Fabric 的主要版本为 1.4.* 和 2.*, 2.* 版本为 2020 年推出的新版本, 较 1.4.* 版本有了较多变化, 本实验采用的 Fabric 版本为 2.1.0。Fabric 利用容器环境运行, 因此可以在 Linux、Windows 等系统上运行。

由于 Fabric 版本之间的差异较大, 在查阅互联网资料时请注意不同版本造成的差异。

1.2 运行环境

(1) **Golang**: 它是一个开源的编程语言, 被广泛用于区块链开发。Fabric 项目就是通过 go 语言编写的, 本实验中也使用 go 语言开发链码。本实验采用的 go 语言版本为 1.15.15。

(2) **Docker**: 它是一个开源的应用容器引擎, 可以将应用打包至镜像中。Fabric 运行在 docker 中的特性使它能安装至不同的系统。本实验采用的 docker 版本为 20.10.7。

(3) **Docker-compose**: 它是用于定义和运行多容器 docker 应用程序的工具。通过 docker-compose, 可以使用 yaml 文件来配置应用程序所需的服务。本实验采用的 docker-compose 版本为 1.29.2。

1.3 安装实验环境

(1) 如果网络畅通, 可以使用 Fabric 官方的一键安装脚本, 完成程序下载与容器下载。

```
curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.1.0 1.4.6 0.4.18
```

(2) 在成功运行脚本后, 会在当前路径下生成 fabric-samples 文件夹, 包括 bin、chaincode、config、fabcar、test-network 等目录。同时也会下载 hyperledger 运行所需的镜像, 包括 fabric-peers、fabric-orderer、fabric-tools、fabric-ca 等。

2 分步骤配置 Fabric 环境

使用 Fabric 源代码提供的脚本来创建智能合约虽然直观, 但无法详细了解背后的操作过程。为增强理解, 本实验使用命令行分步骤完成 Fabric 环境的配置。

首先创建目录 my-fabric，接下来将在该目录中进行环境的配置。

```
mkdir my-fabric
cd my-fabric
```

将 fabric-samples 目录中的 bin、config 文件夹拷贝至 my-fabric 目录下。

```
cp -r ~/fabric-samples/bin/ ~/my-fabric/
cp -r ~/fabric-samples/config/ ~/my-fabric/
```

进入 bin 目录，输入 peer version 检查 Fabric 版本是否为 2.1.0。

```
./bin/peer version
```

执行结果：

```
ecust@ubuntu:~/my-fabric$ ./bin/peer version
peer:
  Version: 2.1.0
  Commit SHA: 1bdf97537
  Go version: go1.14.1
  OS/Arch: linux/amd64
  Chaincode:
    Base Docker Label: org.hyperledger.fabric
    Docker Namespace: hyperledger
```

在 my-fabric 目录下创建 test-network 目录。

```
mkdir test-network
```

本实验的目标是创建一个名为“university.cn”的联盟，具备一个排序节点和 org1、org2、org3 共 3 个组织，每个组织有 2 个节点，2 至 3 位用户。

2.1 密钥材料（cryptogen 操作）

进入 test-network 目录

```
cd test-network
```

查看默认生成密钥材料的模板。

```
../bin/cryptogen showtemplate
```

由于配置文件和实验目标内容有差异，对模板文件进行一定修改。依照设定结构，将模板文件 template.yaml 修改为规定内容（见附件 test-network 文件夹中的 template.yaml 文件）。

```
../bin/cryptogen showtemplate>template.yaml
```

执行命令，使用默认模板生成密钥材料，执行后在 test-network 目录下生成了 organizations 文件夹，其中包含了各个组织、节点的密钥材料。

```
../bin/cryptogen generate --config="./template.yaml" --output="organizations"
```

执行后，会显示如下结果：

```
ecust@ubuntu:~/my-fabric/test-network$ ../bin/cryptogen generate --config="./template.yaml" --output="organizations"
org1.university.cn
org2.university.cn
org3.university.cn
```

2.2 创世区块（configtxgen 操作）

准备好密钥材料后，需要准备通道的相关配置，通道在 Fabric 中就是不同的链，而生成一条链首

先要生成创世区块。这里使用 configtxgen 命令进行配置。

configtxgen 命令依赖于配置文件 configtx.yaml，需要准备好配置文件放在 configx 目录中。

首先创建 configtx 目录。

```
mkdir configtx
```

下面需要对 my-fabric/config 目录下默认的 configtx.yaml 进行修改，然后将其放置在刚刚创建的 my-fabric/test-network/configtx 目录下（见附件 test-network 文件夹中 configtx 文件夹中的 template.yaml 文件）。

下面使用 configtxgen 命令，通过我们自定义的配置文件生成创世区块。

```
../bin/configtxgen -configPath ../configtx -profile ThreeOrgsOrdererGenesis -channelID system-channel -outputBlock ../system-genesis-block/genesis.block
```

执行成功可以看到如下结果：

```
CST [common.tools.configtxgen] doOutputBlock -> INFO 006 Generating genesis block
CST [common.tools.configtxgen] doOutputBlock -> INFO 007 Writing genesis block
```

可以使用如下命令查看创世区块的内容。

```
../bin/configtxgen -inspectBlock ../system-genesis-block/genesis.block
```

2.3 容器操作

在完成密钥材料、创世区块后，使用容器启动整个网络。

首先在 test-network 目录下创建 docker 目录。

```
mkdir docker
```

创建 docker-compose.yaml 文件，然后将其放至 docker 目录中（见附件 test-network 文件夹中 docker 文件夹中的 docker-compose.yaml 文件）。

使用如下命令启动整个网络。

```
sudo docker-compose -f docker/docker-compose.yaml up -d
```

执行结果：

```
ecust@ubuntu:~/my-fabric/test-network$ sudo docker-compose -f docker/docker-compose.yaml up -d
Creating network "docker_university" with the default driver
Creating volume "docker_orderer.university.cn" with default driver
Creating volume "docker_peer0.org1.university.cn" with default driver
Creating volume "docker_peer0.org2.university.cn" with default driver
Creating volume "docker_peer0.org3.university.cn" with default driver
Creating peer0.org3.university.cn ... done
Creating orderer.university.cn ... done
Creating peer0.org2.university.cn ... done
Creating peer0.org1.university.cn ... done
```

使用如下命令可以查看当前正常启动的一个 order 实例和 3 个 peer 实例。

```
sudo docker ps -a
```

执行结果：

```
ecust@ubuntu:~/my-fabric/test-network$ sudo docker ps -a
CONTAINER ID   IMAGE                                COMMAND
3b50f05542cb   hyperledger/fabric-peer:latest     "peer node start"
7046ba8dbcab   hyperledger/fabric-peer:latest     "peer node start"
alce38d21659   hyperledger/fabric-orderer:latest  "orderer"
7481b4dcf01e   hyperledger/fabric-peer:latest     "peer node start"
```

如果在后续步骤中出现错误，可以使用如下命令依次删除所有容器、执行所有步骤，否则可能会由于区块链的持久特性导致无法正常进行。

```
sudo docker rm 容器 ID
```

2.4 创建通道 channel

创建 channel 需要准备通道配置文件、锚节点文件，通过配置文件创建，将组织加入通道。

(1) 准备 channel 配置文件

首先创建目录 channel-artifacts。

```
mkdir channel-artifacts
```

然后执行如下命令，创建通道 channel1

```
../bin/configtxgen -configPath ./configtx -profile ThreeOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel1.tx -channelID channel1
```

执行成功后出现如下结果，并在 channel-artifacts 目录下生成 channel1.tx 文件：

```
[common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
[common.tools.configtxgen] doOutputChannelCreateTx -> INFO 004 Writing new channel tx
```

可以使用如下命令查看 channel1.tx 文件。

```
../bin/configtxgen -inspectChannelCreateTx ./channel-artifacts/channel1.tx
```

(2) 生成 anchor peer 配置文件

当组织内部存在多个节点时，通道信息会通过锚节点传递，避免大量的节点查询工作。一个组织可以有多个锚节点，这里仅延时每个组织生成一个锚节点。

分别针对 3 个高校生成锚节点。

```
../bin/configtxgen -configPath ./configtx/ -profile ThreeOrgsChannel -channelID channel1 -outputAnchorPeersUpdate=./channel-artifacts/Org1MSPanchors.tx -asOrg=Org1MSP
```

```
../bin/configtxgen -configPath ./configtx/ -profile ThreeOrgsChannel -channelID channel1 -outputAnchorPeersUpdate=./channel-artifacts/Org2MSPanchors.tx -asOrg=Org2MSP
```

```
../bin/configtxgen -configPath ./configtx/ -profile ThreeOrgsChannel -channelID channel1 -outputAnchorPeersUpdate=./channel-artifacts/Org3MSPanchors.tx -asOrg=Org3MSP
```

每次的执行结果如下所示，同时会在 channel-artifacts 目录中生成文件 Org1/2/3MSPanchors.tx：

```
[common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 003 Generating anchor peer update
[common.tools.configtxgen] doOutputAnchorPeersUpdate -> INFO 004 Writing anchor peer update
```

使用如下命令可以查看 Org1/2/3MSPanchors.tx 文件中的内容。

```
../bin/configtxgen -inspectChannelCreateTx ./channel-artifacts/Org1MSPanchors.tx
```

```
../bin/configtxgen -inspectChannelCreateTx ./channel-artifacts/Org2MSPanchors.tx
```

```
../bin/configtxgen -inspectChannelCreateTx ./channel-artifacts/Org3MSPanchors.tx
```

(3) 创建 channel

准备好以上文件后，使用 Org1 来启动创建通道的工作，由于使用了不同的节点身份进行操作，因此在执行代码前需要进行一系列参数配置工作，指定文件名、身份、地址等。

```
#需要默认配置文件
export FABRIC_CFG_PATH="./config"
#使用 Org1 身份
export CORE_PEER_LOCALMSPID="Org1MSP"
#指定 ORDERER CA 证书
export ORDERER_CA=${PWD}/organizations/ordererOrganizations/university.cn/orderers/orderer.university.cn/msp/tlscacerts/tlsca.university.cn-cert.pem
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.university.cn/peers/peer0.org1.university.cn/tls/ca.crt
#指定节点管理员身份
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.university.cn/users/Admin@org1.university.cn/msp
#指定节点地址
export CORE_PEER_ADDRESS=localhost:7051
```

然后执行创建通道的命令

```
./bin/peer channel create -o localhost:7050 -c channel1 --ordererTLSHostnameOverride orderer.university.cn -f ./channel-artifacts/channel1.tx --outputBlock ./channel-artifacts/channel1.block --tls true --cafile $ORDERER_CA
```

执行成功后结果如下，在 channel-artifacts 目录中会生成文件 channel1.block。

```
[channelCmd] InitCmdFactory -> INFO 00d Endorser and orderer connections initialized
[cli.common] readBlock -> INFO 00e Received block: 0
```

（4）加入 channel

接下来把三个组织都加入 channel1 中。

首先执行以下命令：

```
export CORE_PEER_TLS_ENABLED=true
export FABRIC_CFG_PATH="./config"
export ORDERER_CA=${PWD}/organizations/ordererOrganizations/university.cn/orderers/orderer.university.cn/msp/tlscacerts/tlsca.university.cn-cert.pem
```

下面将用户切换脚本保存至 test-network 文件夹下，方便每次调用。（见附件 test-network 目录下的 userOrg1.sh、userOrg2.sh、userOrg3.sh）

将 Org1 加入 channel1。

```
source ./userOrg1.sh
./bin/peer channel join -b ./channel-artifacts/channel1.block
```

将 Org2 加入 channel1。

```
source ./userOrg2.sh
./bin/peer channel join -b ./channel-artifacts/channel1.block
```

将 Org3 加入 channel1。

```
source ./userOrg3.sh
../bin/peer channel join -b ./channel-artifacts/channel1.block
```

每次执行成功后会看到如下结果：

```
[channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
[channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
```

（5）更新 anchor peer 信息

需要更新三个组织的锚节点信息。

首先执行如下命令：

```
export ORDERER_CA=${PWD}/organizations/ordererOrganizations/university.cn/orderers/orderer.university.cn/msp/tlscacerts/tlsca.university.cn-cert.pem
export FABRIC_CFG_PATH="./config"
```

更新 Org1 的锚节点

```
source ./userOrg1.sh
../bin/peer channel update -o localhost:7050 --ordererTLShostnameOverride orderer.university.cn -c channel1 -f ./channel-artifacts/Org1MSPanchors.tx --tls true --cafile $ORDERER_CA
```

更新 Org2 的锚节点

```
source ./userOrg2.sh
../bin/peer channel update -o localhost:7050 --ordererTLShostnameOverride orderer.university.cn -c channel1 -f ./channel-artifacts/Org2MSPanchors.tx --tls true --cafile $ORDERER_CA
```

更新 Org3 的锚节点

```
source ./userOrg3.sh
../bin/peer channel update -o localhost:7050 --ordererTLShostnameOverride orderer.university.cn -c channel1 -f ./channel-artifacts/Org3MSPanchors.tx --tls true --cafile $ORDERER_CA
```

每次执行成功后会看到如下结果：

```
[channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
[channelCmd] update -> INFO 002 Successfully submitted channel update
```

此时，创建 channel、加入 channel、配置锚节点的步骤已全部完成。需要注意的是，为简化操作，不进入每个 peer 的容器环境中操作，而是通过授予节点操作权限，利用客户端（本机）配置各个节点。

2.5 安装和执行链码

（1）打包源码

在链码开发完成后，需要打包才能安装至各个节点。

Fabric 为用户提供了命令实现链码的打包，该命令会自动下载链码所需要的依赖以及相应的配置文件，该过程需要互联网才能正常执行。由于实验平台所提供的 linux 服务器无法连接互联网，该过程需要在 windows 端手动打包然后再上传至 linux 服务器中。

I 自动打包方式（仅供学习）

在 test-network 目录下执行如下命令，使用 Org1 的身份进行打包（其中“./chaincode/fabcar/go/”为源码所在位置）：

```
source ./userOrg1.sh
```

```
export PATH=${PWD}../../bin:$PATH
export FABRIC_CFG_PATH=$PWD../config/
../bin/peer lifecycle chaincode package fabcar.tar.gz --path ./chaincode/fabcar/go/ --lang golang --label fabcar_1
```

执行成功后，会在当前目录下生成 `fabcar.tar.gz` 文件。

II 手动打包方式

由于 linux 服务器中无法连接外部网络，因此需要在自己的机器上下载代码依赖，编写对应文件，然后将文件上传至服务器进行打包操作。

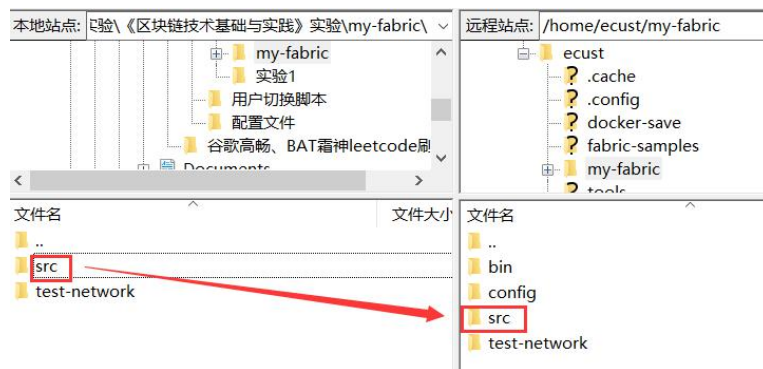
在附件中的 `src` 路径下，执行下载依赖的命令。

```
cd src
```

```
go mod vendor
```

执行后，在目录下会生成 `vendor` 文件夹，其中包含了代码运行所需的所有依赖。

然后使用 `filezilla` 将文件夹上传至服务器的 `my-fabric` 路径下。



上传成功后，进入 `my-fabric` 目录，将 `src` 文件夹打包压缩为 `code.tar.gz` 文件。

```
cd ..
```

```
tar zcvf code.tar.gz src/
```

然后编写 `metadata.json` 文件，它是一个元数据描述文件，包含链码语言、代码路径、包标签等信息。使用 `vim` 创建该文件

```
vim metadata.json
```

按下键盘上的“i”，进入编辑模式，输入以下内容（其中 `fabcar_1` 为链码自定义的标签值，`type` 为链码所用的语言）：

```
{"path":"","type":"golang","label":"fabcar_1"}
```

输入完成后，按下键盘上的“Esc”键，然后按下“:”键，然后输入 `wq`，回车，即成功保存。

然后将 `code.tar.gz`、`metadata.json` 两个文件打包压缩为 `fabcar.tar.gz` 文件。

```
tar zcvf fabcar.tar.gz code.tar.gz metadata.json
```

(2) 安装链码

需要将打包后的链码安装在 3 个节点上。

首先进入 `test-network` 目录。

```
cd test-network
```

```
export CORE_PEER_TLS_ENABLED=true
```


Org1 节点安装链码。

```
source ./userOrg1.sh
```

```
../bin/peer lifecycle chaincode install ./fabcar.tar.gz
```

显示如下结果，每个节点会生成一致的标识码返回：

```
[cli.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:<status:200  
payload:"\nIfabcar_1:f8c7572e350a616b9e21cfcea9e95312bfc8a2b74bf0d945b3ca2f069d41e863\022\010fab  
car_1" >
```

```
[cli.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: fab  
car_1:f8c7572e350a616b9e21cfcea9e95312bfc8a2b74bf0d945b3ca2f069d41e863
```

Org2 节点安装链码。

```
source ./userOrg2.sh
```

```
../bin/peer lifecycle chaincode install ./fabcar.tar.gz
```

Org3 节点安装链码。

```
source ./userOrg3.sh
```

```
../bin/peer lifecycle chaincode install ./fabcar.tar.gz
```

使用如下命令可以查看节点 1/2/3 上已安装的链码，以 Org1 为例。

```
source ./userOrg1.sh
```

```
../bin/peer lifecycle chaincode queryinstalled
```

查询结果，每个节点上查询出的 Package ID 都是一致的，记录下这个 ID 的值：

```
ecust@ubuntu:~/my-fabric/test-network$ source ./userOrg1.sh  
ecust@ubuntu:~/my-fabric/test-network$ ../bin/peer lifecycle chaincode queryinstalled  
Installed chaincodes on peer:  
Package ID: fabcar_1:f8c7572e350a616b9e21cfcea9e95312bfc8a2b74bf0d945b3ca2f069d41e863, Label: fabcar_1
```

该 Package ID 的值为：

```
fabcar_1:f8c7572e350a616b9e21cfcea9e95312bfc8a2b74bf0d945b3ca2f069d41e863
```

(3) 验证并提交

安装完成后，如果检查确认安装的链码无误，即可向排序节点发布验证信息，需要依赖上一步安装过程中所生成的 ID。

此处的 Package ID 的值为上一步所生成的 Package ID。

```
export PACKAGE_ID=fabcar_1:f8c7572e350a616b9e21cfcea9e95312bfc8a2b74bf0d945b3ca2f069d41e  
863
```

```
export ORDERER_CA=${PWD}/organizations/ordererOrganizations/university.cn/orderers/orderer.univ  
ersity.cn/msp/tlscacerts/tlscacert.university.cn-cert.pem
```

使用 org1 身份，使用配置脚本，验证链码

```
source userOrg1.sh
```

```
../bin/peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride or  
derer.university.cn --tls --cafile ${ORDERER_CA} --channelID channel1 --name fabcar --version 1 --ini  
t-required --package-id ${PACKAGE_ID} --sequence 1
```

正常运行后可以看到生成的交易结果信息：

```
[chaincodeCmd] ClientWait -> INFO 001 txid [7059e6cc5889a2eaac725835e44a6923ebcb94f52f497  
5232fef5cf55d8642d5] committed with status (VALID) at
```

使用 org1 身份，检查提交状态

```
source userOrg1.sh
```

```
../bin/peer lifecycle chaincode checkcommitreadiness --channelID channel1 --name fabcar --version 1 --sequence 1 --init-required
```

执行结果为：

```
Chaincode definition for chaincode 'fabcar', version '1', sequence '1' on channel 'channel1' approval status by org:  
Org1MSP: true  
Org2MSP: false  
Org3MSP: false
```

当在三个节点都验证了链码后，结果将变为全部通过。

使用 org2 身份，使用配置脚本，验证链码

```
source userOrg2.sh
```

```
../bin/peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.university.cn --tls --cafile ${ORDERER_CA} --channelID channel1 --name fabcar --version 1 --init-required --package-id ${PACKAGE_ID} --sequence 1
```

使用 org3 身份，使用配置脚本，验证链码

```
source userOrg3.sh
```

```
../bin/peer lifecycle chaincode approveformyorg -o localhost:7050 --ordererTLSHostnameOverride orderer.university.cn --tls --cafile ${ORDERER_CA} --channelID channel1 --name fabcar --version 1 --init-required --package-id ${PACKAGE_ID} --sequence 1
```

当三个组织均验证后，检查验证状态，三个组织都变为 true。然后可以开始提交链码的操作。

```
Chaincode definition for chaincode 'fabcar', version '1', sequence '1' on channel 'channel1' approval status by org:  
Org1MSP: true  
Org2MSP: true  
Org3MSP: true
```

未提交前，检查提交状态(此时未提交，会有错误)。

```
source userOrg1.sh
```

```
../bin/peer lifecycle chaincode querycommitted --channelID channel1 --name fabcar
```

产生报错信息：

```
Error: query failed with status: 404 - namespace fabcar is not defined
```

下面开始提交链码的操作。

配置 CA 证书。

```
export ORDERER_CA=${PWD}/organizations/ordererOrganizations/university.cn/orderers/orderer.university.cn/msp/tlscacerts/tlsca.university.cn-cert.pem
```

```
export PEER0_ORG1_CA=${PWD}/organizations/peerOrganizations/org1.university.cn/peers/peer0.org1.university.cn/tls/ca.crt
```

```
export PEER0_ORG2_CA=${PWD}/organizations/peerOrganizations/org2.university.cn/peers/peer0.org2.university.cn/tls/ca.crt
```

```
export PEER0_ORG3_CA=${PWD}/organizations/peerOrganizations/org3.university.cn/peers/peer0.org3.university.cn/tls/ca.crt
```

配置节点地址。

```
export PEER0_ORG1_ADDRESS=localhost:7051
export PEER0_ORG2_ADDRESS=localhost:9051
export PEER0_ORG3_ADDRESS=localhost:11051
```

使用 org1 身份，使用配置脚本提交链码

```
source userOrg1.sh
```

```
../bin/peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.university.cn --tls true --cafile ${ORDERER_CA} --channelID channel1 --name fabcar --peerAddresses ${PEER0_ORG1_ADDRESS} --tlsRootCertFiles ${PEER0_ORG1_CA} --peerAddresses ${PEER0_ORG2_ADDRESS} --tlsRootCertFiles ${PEER0_ORG2_CA} --peerAddresses ${PEER0_ORG3_ADDRESS} --tlsRootCertFiles ${PEER0_ORG3_CA} --version 1 --sequence 1 --init-required
```

成功执行后结果如下：

```
ClientWait -> INFO 001 txid [f47aa4774fef59d207b48de8efela5dc3080171672e8f054325bc64363098d9] committed with status (VALID) at localhost:7051
ClientWait -> INFO 002 txid [f47aa4774fef59d207b48de8efela5dc3080171672e8f054325bc64363098d9] committed with status (VALID) at localhost:11051
ClientWait -> INFO 003 txid [f47aa4774fef59d207b48de8efela5dc3080171672e8f054325bc64363098d9] committed with status (VALID) at localhost:9051
```

再次检查提交状态。

```
source userOrg1.sh
```

```
../bin/peer lifecycle chaincode querycommitted --channelID channel1 --name fabcar
```

结果如下：

```
ecust@ubuntu:~/my-fabric/test-network$ ../bin/peer lifecycle chaincode querycommitted --channelID channel1 --name fabcar
Committed chaincode definition for chaincode 'fabcar' on channel 'channel1':
Version: 1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vsccl, Approvals: [Org1MSP: true, Org2MSP: true, Org3MSP: true]
```

(4) 执行链码（初始化三个节点的链码）

配置 CA 证书

```
export ORDERER_CA=${PWD}/organizations/ordererOrganizations/university.cn/orderers/orderer.university.cn/msp/tlscacerts/tlsca.university.cn-cert.pem
export PEER0_ORG1_CA=${PWD}/organizations/peerOrganizations/org1.university.cn/peers/peer0.org1.university.cn/tls/ca.crt
export PEER0_ORG2_CA=${PWD}/organizations/peerOrganizations/org2.university.cn/peers/peer0.org2.university.cn/tls/ca.crt
export PEER0_ORG3_CA=${PWD}/organizations/peerOrganizations/org3.university.cn/peers/peer0.org3.university.cn/tls/ca.crt
```

配置节点地址

```
export PEER0_ORG1_ADDRESS=localhost:7051
export PEER0_ORG2_ADDRESS=localhost:9051
export PEER0_ORG3_ADDRESS=localhost:11051
```

使用 org1 身份，使用配置脚本提交链码

```
source userOrg1.sh
```

```
../bin/peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.university.cn --tls true --cafile ${ORDERER_CA} -C channel1 -n fabcar --peerAddresses ${PEER0_ORG1_ADDRESS} --tlsRootCertFiles ${PEER0_ORG1_CA} --peerAddresses ${PEER0_ORG2_ADDRESS} --tlsRootCertFiles ${PEER0_ORG2_CA} --peerAddresses ${PEER0_ORG3_ADDRESS} --tlsRootCertFiles ${PEER0_ORG3_CA}
```

```
0_ORG3_CA} --isInit -c '{"function":"InitLedger","Args":[]}'
```

成功执行后结果如下：

```
[chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```

（5）查询结果（执行链码）

初始化链码后，可以调用链码。

使用 Org1 身份，调用链码（任意身份执行的效果一样）。

```
source userOrg1.sh
```

```
../bin/peer chaincode query -C channel1 -n fabcar -c '{"Args":["queryAllCars"]}'
```

成功执行后结果如下：

```
[{"Key":"CAR0","Record":{"make":"Toyota","model":"Prius","colour":"blue","owner":"Tomoko"}}, {"Key":"CAR1","Record":{"make":"Ford","model":"Mustang","colour":"red","owner":"Brad"}}, {"Key":"CAR2","Record":{"make":"Hyundai","model":"Tucson","colour":"green","owner":"Jin Soo"}}, {"Key":"CAR3","Record":{"make":"Volkswagen","model":"Passat","colour":"yellow","owner":"Max"}}, {"Key":"CAR4","Record":{"make":"Tesla","model":"S","colour":"black","owner":"Adriana"}}, {"Key":"CAR5","Record":{"make":"Peugeot","model":"205","colour":"purple","owner":"Michel"}}, {"Key":"CAR6","Record":{"make":"Chery","model":"S22L","colour":"white","owner":"Aarav"}}, {"Key":"CAR7","Record":{"make":"Fiat","model":"Punto","colour":"violet","owner":"Pari"}}, {"Key":"CAR8","Record":{"make":"Tata","model":"Nano","colour":"indigo","owner":"Valeria"}}, {"Key":"CAR9","Record":{"make":"Holden","model":"Barina","colour":"brown","owner":"Shotaro"}}]
```

任课教师签名：

2021 年 月 日