**实验9 End2End案例详解**

**【实验介绍】**

在上一个实验中，我们完成了Fabric基本环境的安装与配置，并成功运行了官方提供的End2End网络案例，对Fabric的架构特点有了初步的认识。

本次实验仍然使用End2End案例，End2End案例几乎涉及了 Fabric 网络中超过80%的情况，有种案例即生产的即视感。End2End案例涉及了节点证书文件生成、节点用户管理和节点频道管理以及频道及智能合约相关操作，是入门学习的必备。本实验的设计目的是帮助读者理解End2End案例的启动逻辑，理解Fabric中智能合约的逻辑。

（1）End2End：

End2End案例中实现了点对点的网络，网络中有两个组织分别为org1和org2，每个组织中有两个节点，分别为peer0、pee1; 由3个zookeeper、4个kafka和1个orderer实现排序服务，其结构如图9-1所示：



图 9-1 End2End结构图

End2End的全部配置都在docker-compose-cli.yaml文件中，配置过程中首先运行zookeeper集群，再运行kafka集群，最后运行orderer和peer，此顺序不可打乱，End2End功能在script.sh文件中，启动End2End时将自动运行script.sh中的所有功能，直到打印出运行成功。在后续的实验过程中，我们将详细分析script.sh文件中的内容。

End2End案例中全部文件都在fabric/examples/e2e\_cli目录下，其中的文件功能解释如下：

base：存放配置提炼的公有部分，有两个文件，分别为docker-compose-base.yaml和peer-base.yaml；

channel-artifacts：存放生成的通道和创世纪块等文件，包括有channel.tx、genesis.block、Org1MSPanchors.tx和Org2MSPanchors.tx；

crypto-config：存放生成的公私钥和证书等文件；

scripts：其中只有一个script.sh文件，该文件是案例的运行功能的集合，运行后会自动执行全部功能，直到完成；

configtx.yaml：通道适配文件；

crypto-config.yaml：生成的公私钥和证书的配置文件；

docker-compose-cli.yaml：Fabric网络Docker运行配置文件；

download-dockerimages.sh：下载Fabric镜像执行文件；

generateArtifacts.sh：生成公私钥和证书的执行文件；

network\_setup.sh：案例运行的入口文件。

End2End从network\_setup.sh文件开始执行，执行networkup()函数，检查crypto-config目录是否存在，如果不存在则调用generateArtifacts.sh生成公私钥和证书等文件，再根据docker-compose-cli.yaml的配置内容由docker运行zookeeper、kafka、orderer、peer和cli，启动Fabric网络环境。最后，在cli中运行script.sh文件，首先提示START-E2E，表示将开始执行案例，在不出错的情况下，文件将逐行自动执行，直到提示END-E2E证明运行成功。

其运行过程大致如下：

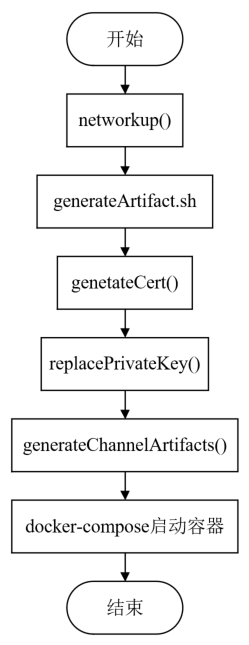


图 9-2 End2End运行过程图

本实验调用合约主要用到的是peer命令，通过peer chaincode命令在peer节点上执行链码相关的操作，例如安装（install）、实例化（instantiate）、调用（invoke）、打包（package）、查询（query）以及升级链码（upgrade）。不同的子命令选项对应于peer节点上不同的操作，query方法，不更改底层的链码状态，不会在交易的上下文内运行，仅用于读取区块链的状态，它的调用不会记录在区块链上；invoke方法用于修改区块链状态，所有的创建、更新和删除操作都被封装在invoke方法中，此方法的调用将会被作为交易，记录在区块链中

peer chaincode query 子命令选项利用链码查询节点上账本中的状态值。对于不同的子命令还有对应的命令标志，此处以peer chaincode query为例来演示命令标志的使用方法。

|  |  |  |
| --- | --- | --- |
| -C | --channelID string | 指定命令执行的通道 |
| -n | --name string | 链码名称 |
| -c | --ctor string | JSON格式的函数构造消息 |

命令peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'中的通道为mychannel，链码名称为mycc，调用函数的参数为["query","a"]，此参数列表将由链码内的函数进行解析。此命令的含义为，向通道mychannel发送通过链码mycc查询节点账本的变量a的值的指令。

**【实验要求】**

1. 理解End2End案例的启动逻辑
2. 手动调用案例中的合约
3. 理解Fabric中智能合约的逻辑

**【实验准备】**

**理解End2End案例的启动逻辑**

这一小节的内容主要是介绍End2End案例的启动逻辑，作为一个了解，不影响后续的操作，允许直接进入下一小节的内容。

首先查看案例的入口文件，network\_setup.sh，可以看到：

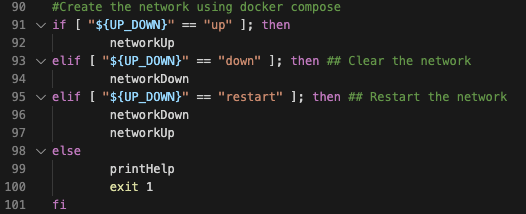


图 9-3 network\_setup.sh

network\_setup.sh先检查参数，然后根据参数“up”，“down”，“restart”调用不同函数networkUp，networkDown，networkUp+networkDown。

networkUp命令，执行 generateArtifacts.sh文件：

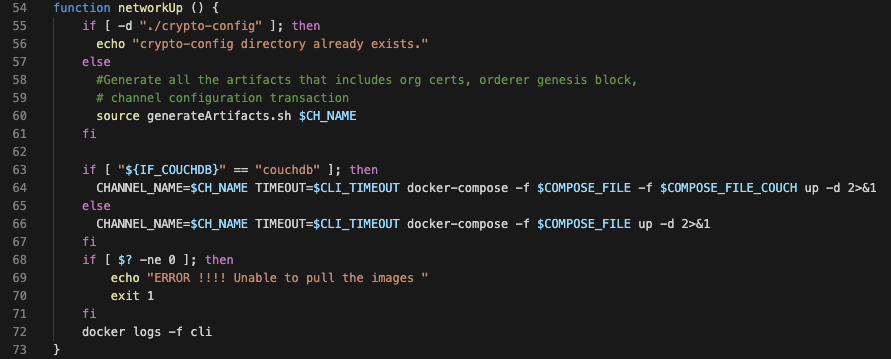


图 9-4 networkUp函数

程序运行generateArtifacts.sh文件，运行完毕后，再调用compose-docker 命令。

我们分析一下generateArtifacts文件，该文件主要执行了三个步骤：

(1) generateCerts() 函数

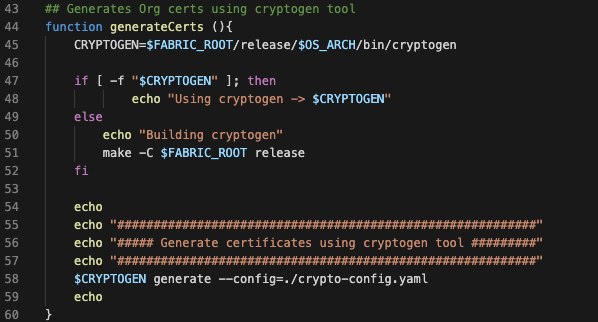


图 9-5 generateCerts()函数

(2) replacePrivateKey() 函数

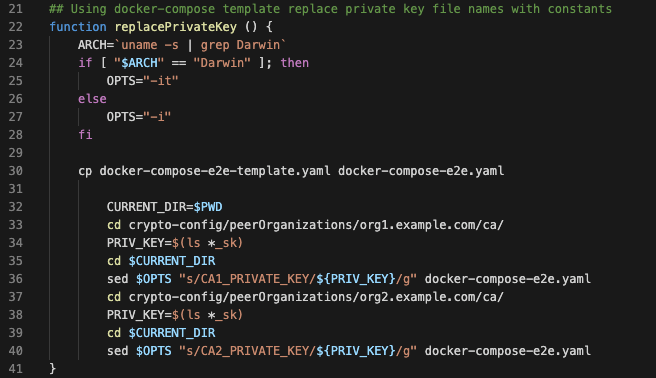


图 9-6 replacePrivateKey()函数

(3) generateChannelArtifacts()函数

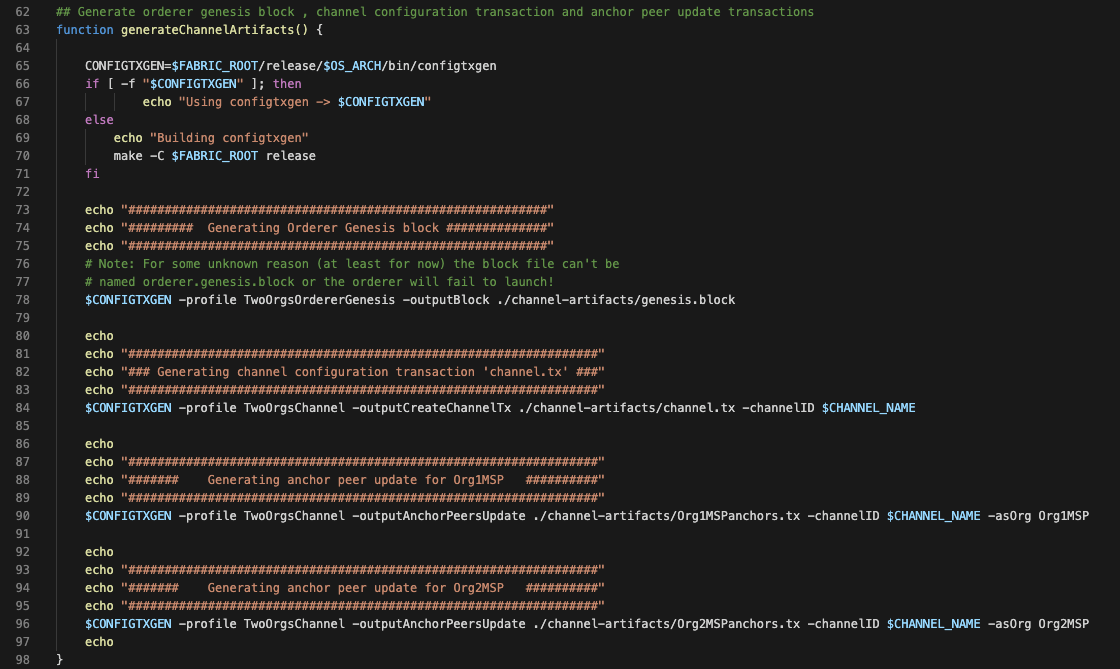


图 9-7 generateChannelArtifacts()函数

最后，generateArtifacts.sh 文件执行完毕。执行返回到network\_setup.sh文件执行：

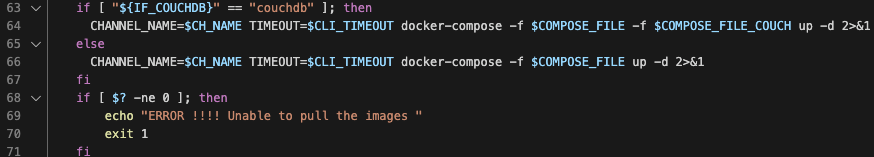
****

图 9-8 network\_setup.sh后续

**【实验过程】**

**1. 手动调用案例中的合约**

这里，我们尝试手动调用该案例中部署的智能合约。

成功运行End2End案例后，重新打开一个窗口，输入如下命令：

|  |
| --- |
| docker exec -it cli bash |

此时会进入cli客户端，如下所示：

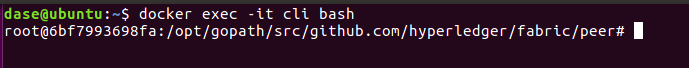


图 9-9 cli客户端

然后输入如下命令，查询一下账户a的资产：

|  |
| --- |
| peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}' |

输出如下，可以看到账户a的资产为90：

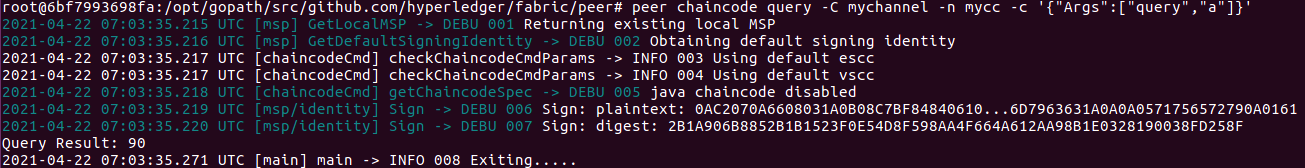


图 9-10 账户a的资产

同样的，也查询一下账户b的资产：

|  |
| --- |
| peer chaincode query -C mychannel -n mycc -c '{"Args":["query","b"]}' |

输出如下，可以看到账户b的资产为210：

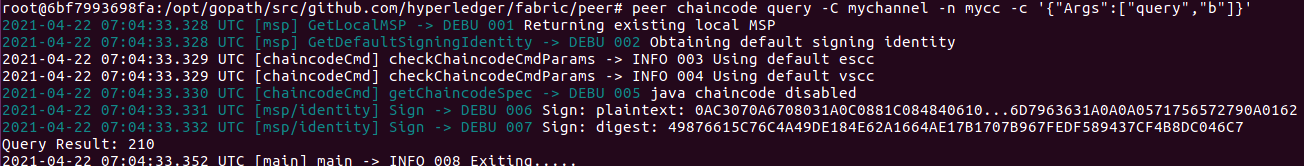


图 9-11 账户b的资产

然后我们来做一下转账操作，账户a给账户b转账50:

|  |
| --- |
| peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n mycc -c '{"Args":["invoke","a","b","50"]}' |

看到执行成功的提示后，你可以再分别查询一下两个账户的余额，这里不再赘述。

**2. 理解Fabric中智能合约的逻辑**

通过上述手动调用智能合约的过程，我们可以得知，End2End案例不仅仅搭建了一个fabric网络，还部署了一个智能合约，并进行了合约实例化操作以及调用操作（其实从一开始的余额查询就可以看出来，账户a余额90，账户b余额210，显然一开始在运行End2End案例过程中，存在转账操作）。

进入e2e案例的scripts文件夹中：

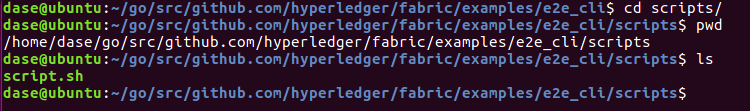


图 9-12 script文件夹

可以看到其中的脚本文件，该文件中的内容验证了上述猜测。

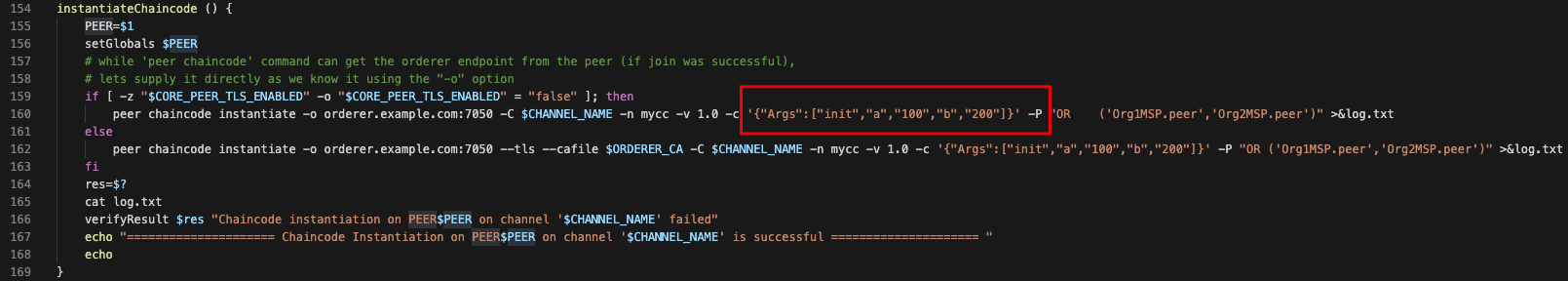


图 9-13 script.sh文件

可以看到，instantiateChaincode方法，该方法对智能合约进行了初始化操作，从其中的初始化参数可以看到，账户a资产100，账户b资产200。

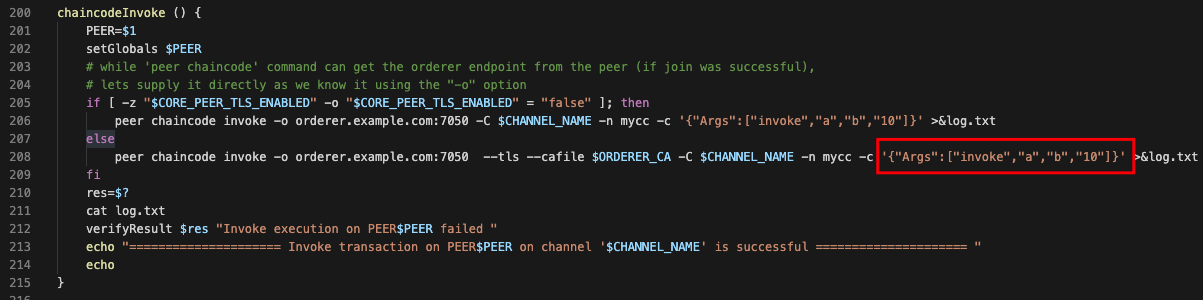


图 9-14 初始化参数设置

后续的chaincodeInvoke方法，账户a给账户b转账10。

那么，这个脚本所操作的智能合约在哪？调用的时候到底发生了啥？

刚才我们在scripts文件夹下，通过如下操作，找到相应智能合约所在的位置：

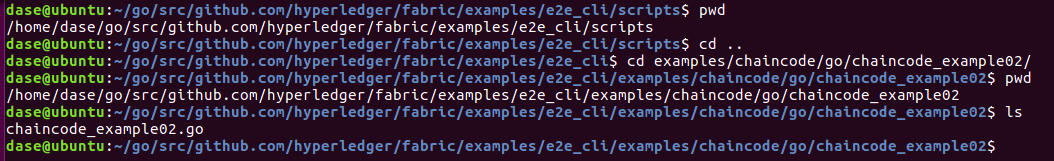


图 9-15 智能合约所在位置

可以看到，e2e案例使用的智能合约是用Go写的（另外也支持Java），这里，我们大致理解一下这个合约即可，后续的合约会使用Java进行编写。

在script.sh脚本中，通过具体的实例化智能合约命令，可以观察到其中的-c 参数指定了智能合约初始化时传入的参数内容，需要对比智能合约中的初始化方法来判断在该操作中发生了什么。具体的合约方法如下：

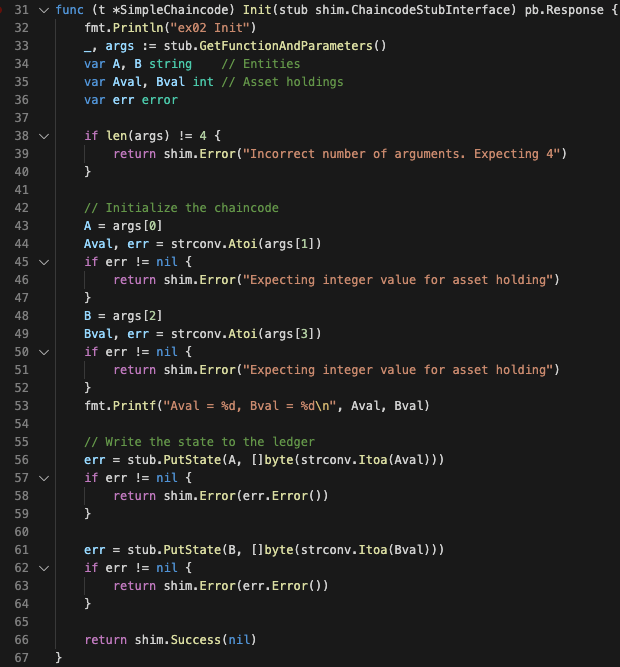


图 9-16 合约方法

阅读该方法，可以看出该方法会接收四个参数，而在-c '{"Args":["init","a","100","b","200"]}'命令中传入了五个参数，第一个参数为方法名，后四个参数为该方法将要接收的参数内容。所以，在 init 方法中接收的四个参数分别是“a”、“100”、“b”和“200”。通过代码不难观察出初始化方法的目的是创建一个 key 为 a 的账户并给该账户一个值为 100 的资产，同创建一个 key 为 b 的账户并也给该账户一个值为 200 的资产。这里就和上述实验内容串起来了。

接下来需要验证该智能合约中的 invoke 和 query 方法。先看 query 方法，如下所示：

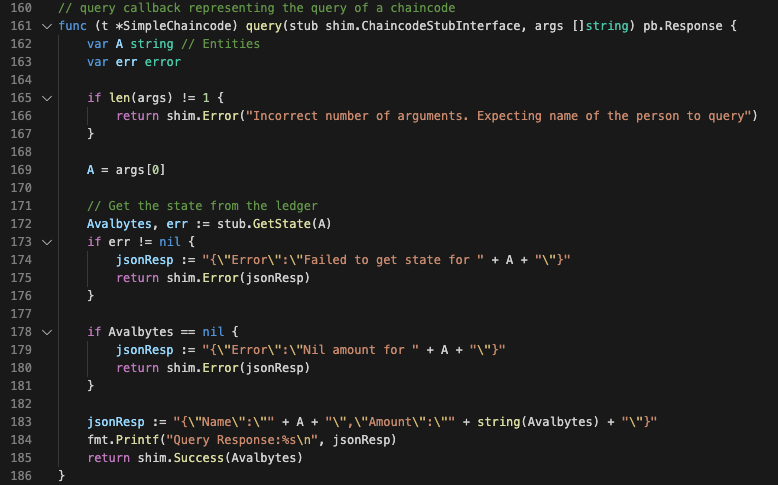


图 9-17 query 方法

该方法需要接收且仅能接受一个参数，所接收的参数为智能合约中所创建账户的 key 值返回内容为该账户下的资产，因此我们通过执行如下命令对a账户的资产进行查询：

|  |
| --- |
| peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}' |

接下来进入 invoke 方法，具体代码示例如下：



图 9-18 invoke 方法

仔细阅读该方法，可以看到该方法需要传入三个参数，第一个和第二个参数分别是不同的账户名，第三个参数是资产值。方法的核心是第一个账户将自己资产中第三个参数值的资产转移到第二个账户名下。如刚才我们执行的转账命令：

|  |
| --- |
| peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n mycc -c '{"Args":["invoke","a","b","50"]}' |

至此，本次实验的内容就结束了。主要是基于e2e案例，带大家理解他的运行逻辑，然后手动调用该案例中的智能合约（ChainCode），最后学习了一下该案例中的示例合约。后面的实验，我们将会使用Java来实现智能合约，然后进行部署和调用。

**【实验小结】**

本实验在实验8的基础上，详细介绍了End2End实例的系统结构、End2End实例的源文档目录结构和文件调用关系以及End2End实例启动逻辑。还介绍了peer命令常用参数的含义，帮助读者理解手动调用智能合约的指令含义。通过阅读End2End案例中的智能合约，读者将了解用户交易的逻辑以及用Go编写智能合约的方法。

读者在完成本实验的基础上可以深入了解End2End实例的依赖包，如Peer、Kafka、Zookeeper等，也可以根据兴趣了解更多Fabric中的实例。

**【习题】**

1. 解释下面命令中每个参数的含义

|  |
| --- |
| peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}' |

2. 解释下面命令中每个参数的含义

|  |
| --- |
| peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem -C mychannel -n mycc -c '{"Args":["invoke","a","b","50"]}' |

**【参考文献】**

[1] <https://hyperledger-fabric.readthedocs.io/en/release-2.2/>（Fabric）

[2] <https://www.cnblogs.com/llongst/p/9519446.html>（End2End）

[3] 陈树宝，郑少华，佟艳娟著.Hyperledger Fabric核心技术[M].北京：电子工业出版社.2019.