华东师范大学数据科学与工程学院实验报告

课程名称：区块链与分享型数据库	**年级**：2018	**上机实践成绩**：
指导教师：张召	**姓名**：池欣宁	**学号**：10185501409
上机实践名称：Fabric		**上机实践日期**：2021.5.13
上机实践编号：	**组号**：	**上机实践时间**：10:00-11:30

一、实验目的

完成 Fabric 基本环境的安装与配置.

运行 End2End 网络案例.

初步认识 Fabric 的架构特点.

二、实验任务

(1) Fabric 基本环境部署
(2) End2End 案例运行
(3) 理解 Fabric 的系统架构，案例的启动逻辑与其中的智能合约的逻辑
(4) 手动调用案例中的合约
(5) Java 实现

三、使用环境

Vmware 虚拟机：Fabric 相关实验报告，Ubuntu16.04

macOS BigSur 11.3

四、实验过程

Part1 Fabric 环境搭建与案例运行

HyperLedger Fabric 是一个基于模块化架构的分布式账本解决方案平台，它具有部署灵活、扩展便捷等特性。与其他区块链平台解决方案相比，它建立在可信任的基础之上，更多的适用于联盟链形式，适合企业之间的区块链联盟方向。

Step1. 环境搭建

1. **安装 Git**
2. **安装 Docker**
3. **安装 Docker-compose**
4. **安装 Go 语言**
5. **Fabric 安装**

```
dase@ubuntu:~/go/src/github.com/hyperledger/fabric/examples/ch
aincode/java$ docker images
REPOSITORY                      TAG             IMAGE ID       C
REATED          SIZE
hyperledger/fabric-ccenv        1.4             01caec52b792   3
 weeks ago      1.42GB
hyperledger/fabric-javaenv      1.4             ca55b5fc2746   6
 months ago     524MB
hyperledger/fabric-tools        latest          e642eef94cae   9
 months ago     1.5GB
hyperledger/fabric-orderer      latest          1a326828a41f   9
 months ago     127MB
hyperledger/fabric-peer         latest          b31292eb8166   9
 months ago     135MB
hyperledger/fabric-zookeeper    latest          bbcd552150f4   1
0 months ago    276MB
hyperledger/fabric-kafka        latest          7e0396b6d64e   1
0 months ago    270MB
hyperledger/fabric-baseos       amd64-0.4.21    5272411bf370   1
0 months ago    88.7MB
```

使用 docker images 查看本地镜像，这些镜像是搭建 Fabric 所需架构。

Step2. 案例运行

E2E_cli 的项目架构

```
dase@ubuntu:~/go/src/github.com/hyperledger/fabric/examples/e2
e_cli$ ls
base                        docker-compose-e2e-template.yaml
channel-artifacts           docker-compose-e2e.yaml
configtx.yaml               download-dockerimages.sh
crypto-config               end-to-end.rst
crypto-config.yaml          generateArtifacts.sh
docker-compose-cli.yaml     network_setup.sh
docker-compose-couch.yaml   scripts
```

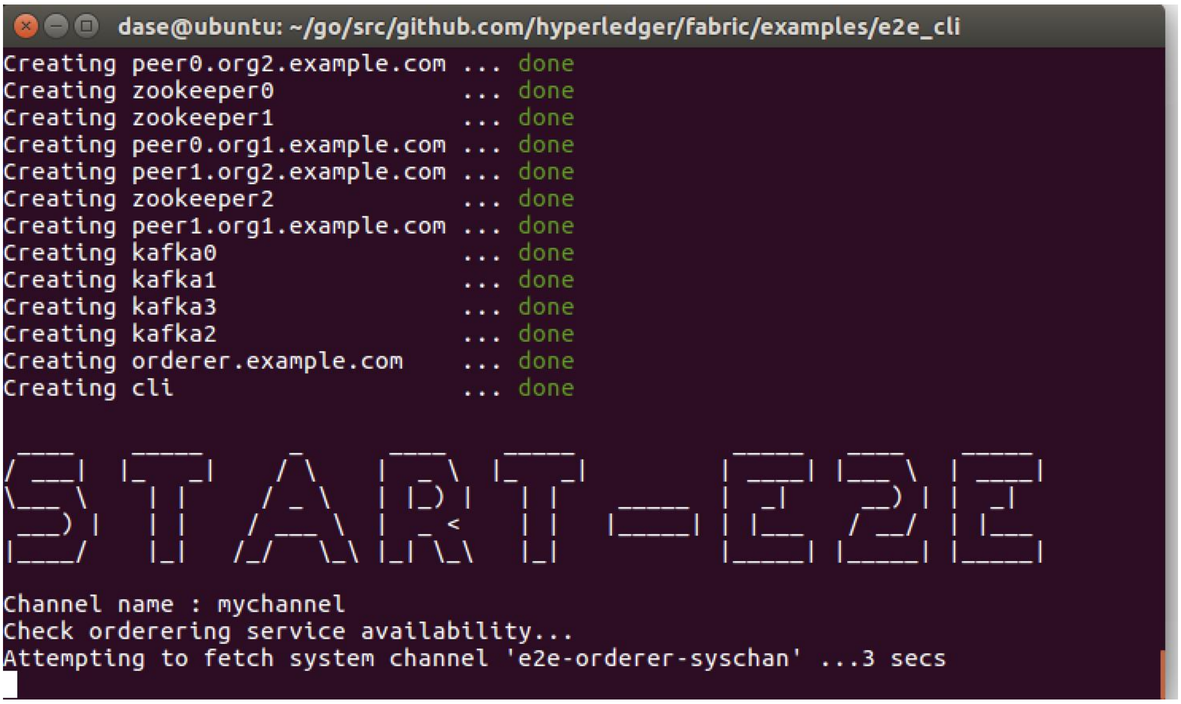
- docker-*.yaml：不同网络拓扑的 docker-compose 配置文件
- crypto-config.yaml：用来给网络节点、CA、用户初始化 PKI 环境（生成的文件存储在 crypto-config 目录下）
- configtx.yaml：生成 channel 和 order 初始化信息（生成文件存储在 channel-artifacts 目录下）
- examples 目录：智能合约代码（chaincode），其余的.sh 文件就是这个例子的执行脚本。

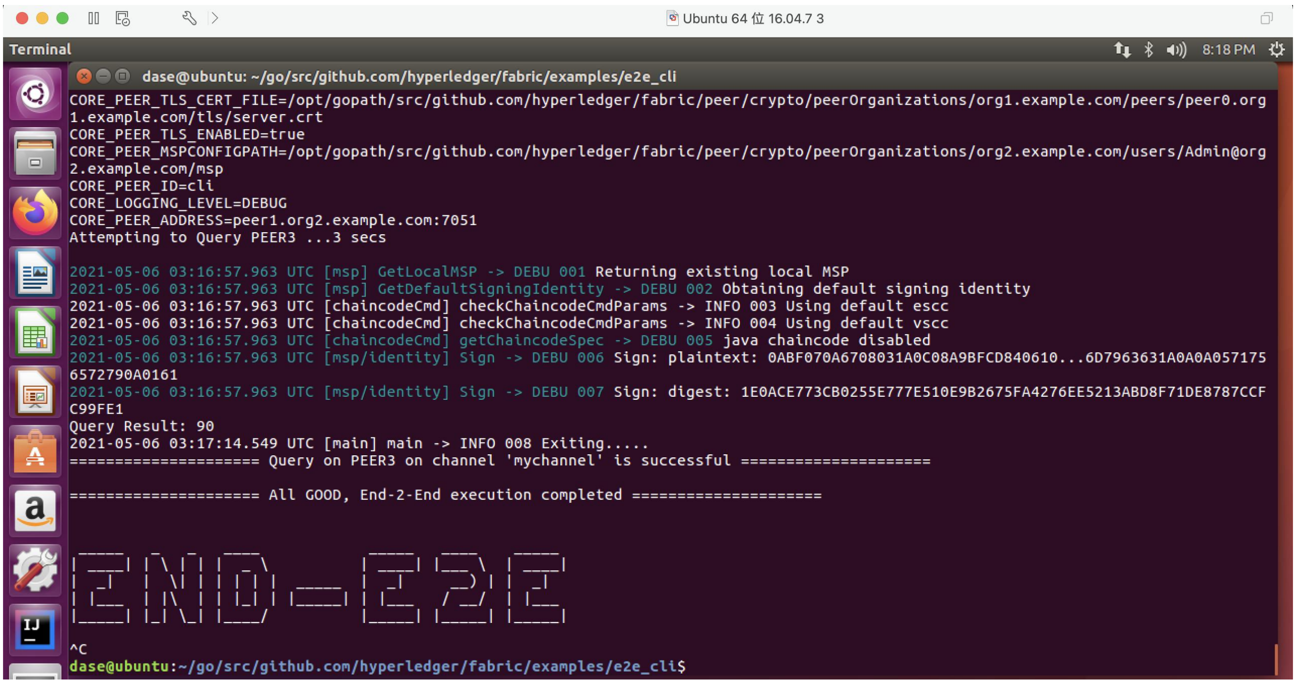
根据 script.sh 可以一步步的看到项目案例运行时 terminal 的输出：

【在后面 java 代码中书写也可以对照这个 script.sh 查看对应的报错信息】

```
echo
echo " ____    _____      _      ____    _____           ____    ___   ____  "
echo "/ ___|  |_   _|    / \    |  _ \  |_   _|         |___ \  / _ \ |___ \ "
echo "\___ \    | |     / _ \   | |_) |   | |   _____     __) || | | |  __) |"
echo " ___) |   | |    / ___ \  |  _ <    | |  |_____|   / __/ | |_| | / __/ "
echo "|____/    |_|   /_/   \_\ |_| \_\   |_|           |_____| \___/ |_____|"
echo
```


echo
echo "==================== All GOOD, End-2-End execution completed ==================== "
echo
echo
echo "
echo "
echo "
echo "
echo "
echo
exit 0


dase@ubuntu: ~/go/src/github.com/hyperledger/fabric/examples/e2e_cli
Creating peer0.org2.example.com ... done
Creating zookeeper0 ... done
Creating zookeeper1 ... done
Creating peer0.org1.example.com ... done
Creating peer1.org2.example.com ... done
Creating zookeeper2 ... done
Creating peer1.org1.example.com ... done
Creating kafka0 ... done
Creating kafka1 ... done
Creating kafka3 ... done
Creating kafka2 ... done
Creating orderer.example.com ... done
Creating cli ... done
Channel name : mychannel
Check orderering service availability...
Attempting to fetch system channel 'e2e-orderer-syschan' ...3 secs

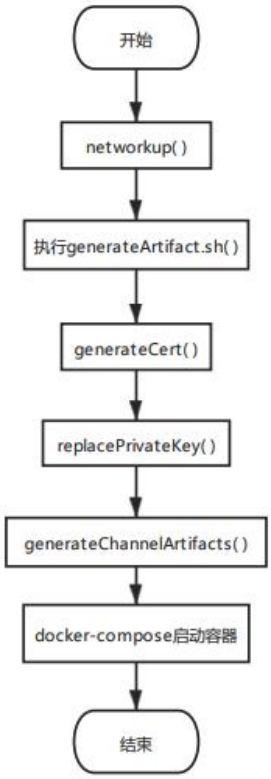

Ubuntu 64 位 16.04.7 3
Terminal
8:18 PM
dase@ubuntu: ~/go/src/github.com/hyperledger/fabric/examples/e2e_cli
CORE_PEER_TLS_CERT_FILE=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/server.crt
CORE_PEER_TLS_ENABLED=true
CORE_PEER_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp
CORE_PEER_ID=cli
CORE_LOGGING_LEVEL=DEBUG
CORE_PEER_ADDRESS=peer1.org2.example.com:7051
Attempting to Query PEER3 ...3 secs
2021-05-06 03:16:57.963 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing local MSP
2021-05-06 03:16:57.963 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining default signing identity
2021-05-06 03:16:57.963 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2021-05-06 03:16:57.963 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2021-05-06 03:16:57.963 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 005 java chaincode disabled
2021-05-06 03:16:57.963 UTC [msp/identity] Sign -> DEBU 006 Sign: plaintext: 0ABF070A6708031A0C08A9BFCD840610...6D7963631A0A0A0571756572790A0161
2021-05-06 03:16:57.963 UTC [msp/identity] Sign -> DEBU 007 Sign: digest: 1E0ACE773CB0255E777E510E9B2675FA4276EE5213ABD8F71DE8787CCFC99FE1
Query Result: 90
2021-05-06 03:17:14.549 UTC [main] main -> INFO 008 Exiting.....
==================== Query on PEER3 on channel 'mychannel' is successful ====================
==================== All GOOD, End-2-End execution completed ====================
^C
dase@ubuntu:~/go/src/github.com/hyperledger/fabric/examples/e2e_cli$

在 E2E_cli 的官方案例中，一次性执行了官方案例是把网络建立，合约初始化，查询操作和交易操作。
这个 Fabric 的结构由 4 个 peerj 节点，order 节点组成，分别用来提出交易和接收交易并检验。kafka 和 zookeeper 集群。
在 script.sh 代码中，整个案例的执行分为是三个部分：

a) Function networkUp(){}根据 crypto-config.yaml 生成不同组织下的节点（CA，peer）用户（admin）的公私钥、数字证书;根据 configtx.yaml 生成创世块、channel 初始化配置、锚节点配置
b) Function networkUp(){}根据 docker-compose 文件生成网络。例子默认是使用 docker-compose-cli.yaml。
c) 根据 script.sh 的代码注释，进行：创建 channel、按配置文件把 peer 加入 channel、安装和初始化 chaincode、Query&Invode 操作。

Part2　End2End 案例详解

Step1.　End2End 案例的启动逻辑



Step2.手动调用案例中的合约

```
root@3ebb5a810fae:/opt/gopath/src/github.com/hyperledger/fabri
c/peer# peer chaincode query -C mychannel -n mycc -c '{"Args":
["query","a"]}'
2021-05-13 02:11:11.004 UTC [msp] GetLocalMSP -> DEBU 001 Retu
rning existing local MSP
2021-05-13 02:11:11.004 UTC [msp] GetDefaultSigningIdentity ->
 DEBU 002 Obtaining default signing identity
2021-05-13 02:11:11.004 UTC [chaincodeCmd] checkChaincodeCmdPa
rams -> INFO 003 Using default escc
2021-05-13 02:11:11.004 UTC [chaincodeCmd] checkChaincodeCmdPa
rams -> INFO 004 Using default vscc
2021-05-13 02:11:11.007 UTC [chaincodeCmd] getChaincodeSpec ->
 DEBU 005 java chaincode disabled
2021-05-13 02:11:11.011 UTC [msp/identity] Sign -> DEBU 006 Si
gn: plaintext: 0ABE070A6608031A0B08BF95F2840610...6D7963631A0A
0A05717565727 90A0161
2021-05-13 02:11:11.011 UTC [msp/identity] Sign -> DEBU 007 Si
gn: digest: F1E9C8D8DA0761924CF6F2579E160A44A88B17B14F1983FF1C
4ED61FFB03DC73
Query Result: 90
2021-05-13 02:11:11.172 UTC [main] main -> INFO 008 Exiting...
..
root@3ebb5a810fae:/opt/gopath/src/github.com/hyperledger/fabri
c/peer#
```

使用 Go 编写的 e2e 案例的逻辑理解，结合 script.sh
首先是 Init 函数 Init 结合 script.sh 脚本，其实是将写好的实例化逻辑存入 stub 对象。Fabric 仅关心存什么数据，所以在 init 里面有一个 stub 存入。·

```
// stub 包含了外部调用传递的参数和额外的内容。
func (t *SimpleChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response
{}
// Initialize the chaincode
// 拿到了账户 a 和它相应的资产
// 主要功能逻辑 拿 A 和 A 的资产(4 行），然后存 stub,return 即可完成。
A = args[0]
Aval, err = strconv.Atoi(args[1])
B = args[2]
Bval, err = strconv.Atoi(args[3])
// 把参数存到它传给你的 stub 的参数。
// Write the state to the ledger
// stub.PutState 把参数存过来
// 写好实例化逻辑存入 stub 对象
err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
```

```
func (t *SimpleChaincode) Invoke(stub shim.ChaincodeStubInterface)
pb.Response {}
//这个 Invoke 主要实现参数判断，不能把所有功能放进去毕竟。
```

Invoke 里面暴露了三个接口，分别是 invoke/delete/query
这三个函数的功能也是需要我们对应着 go 的功能进行实现的。
在 go 语言的 e2e 实例中：invoke 函数的核心部分分析如下（除去异常处理部分）

总结下来，无非就是，args 解析用户名，解析余额，转账，存回 stub 状态中。

```
//拿到账户名 step1
A = args[0]
B = args[1]
// 拿 A，B 的余额 step2
Avalbytes, err := stub.GetState(A)
Aval, _ = strconv.Atoi(string(Avalbytes))
Bvalbytes, err := stub.GetState(B)
Bval, _ = strconv.Atoi(string(Bvalbytes))
//step3
Aval = Aval - X
Bval = Bval + X
fmt.Printf("Aval = %d, Bval = %d\n", Aval, Bval)
// Write the state back to the ledger
// step4 存回去
err = stub.PutState(A, []byte(strconv.Itoa(Aval)))
err = stub.PutState(B, []byte(strconv.Itoa(Bval)))
```

根据 goE2E 部分的代码核心，我们可以看到就是上面的核心代码实现了下面的转账命令：
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。

```
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/tlsca
certs/tlsca.example.com-cert.pem -C mychannel -n mycc -c '{"Args":["invoke","a","b","50"]}'
```

delete 函数的核心如下：获取待删除账户，删掉，一共只有两步。

```
A := args[0]
err := stub.DelState(A)
```

query 函数的核心如下：获取待查询账户，写 json 类型的 Response 进行返回。

```
// step0
A = args[0]
Avalbytes, err := stub.GetState(A)
jsonResp := "{\"Name\":\"" + A + "\",\"Amount\":\"" + string(Avalbytes) +
"\"}"
fmt.Printf("Query Response:%s\n", jsonResp)
```

Part3　使用 Java 编写链码

要求：

该链码要实现的逻辑与上一份文件中提供的代码文件 chaincode_example02_e2e.go 完全一致，请注意，是完全一致，所以你可以参考 chaincode_example02_e2e.go 中的代码逻辑，进行 java 版本的编写。

测试:

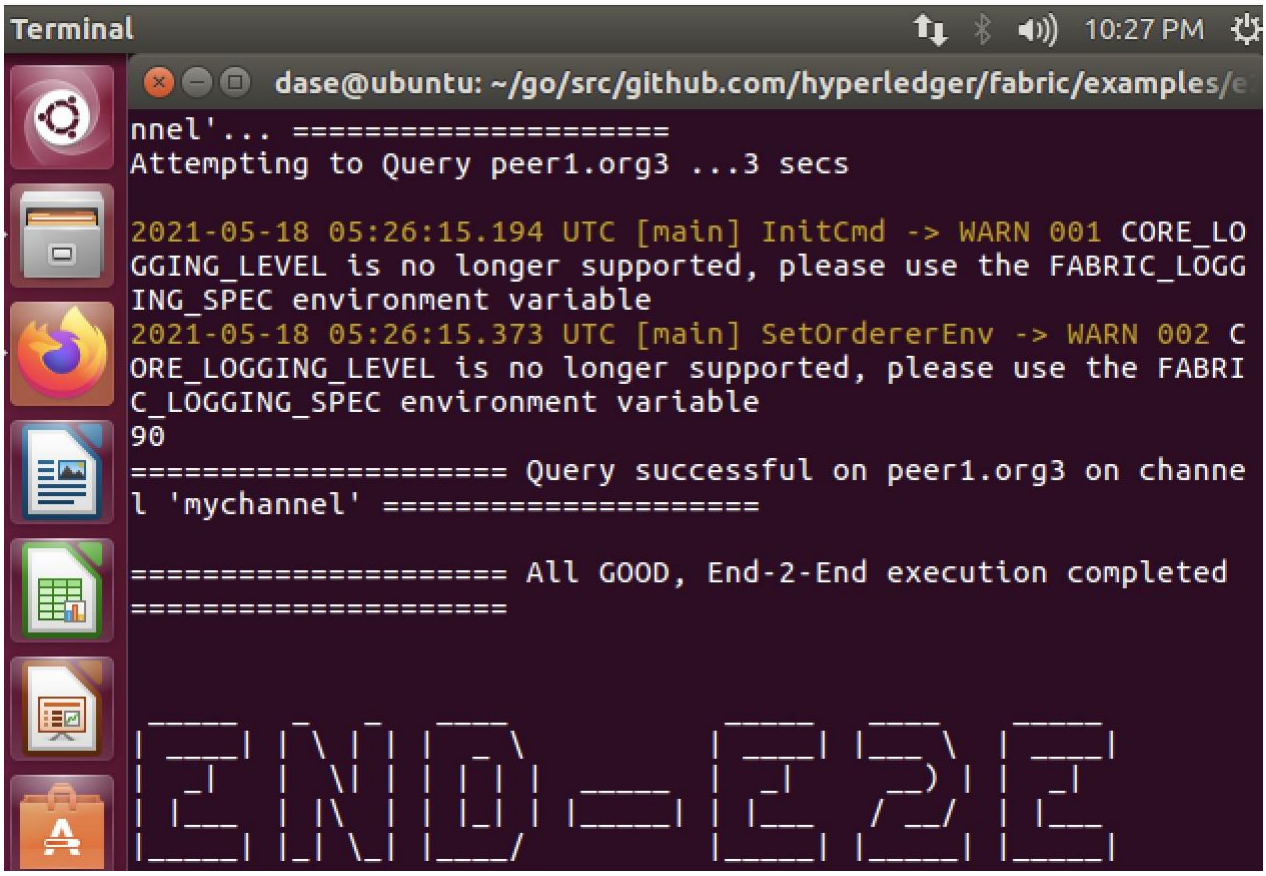
直接将代码放入 e2e 案例中运行，看运行结果是否与 Go 版本的链码相同。

环境：

1.3 版本的 fabric

最后，进入到 e2e 文件夹中，执行./network_setup.sh up，进行案例的运行。

最后我进行完 java 部分代码的编写以后，得到的结果截图如上一次实验中的相同：



既然要求中跟 java 代码完全一致，所以在实验报告里不再麻烦的复制一遍完整代码了，只在下面记录一下写代码时遇到的比较棘手的问题。

1. 将 ErrorException 都放在 java 的 Try Catch 里面：

   在 go 中有很多 error exception，在 java 中可以适当的将这些包裹在 try catch 代码块中。

2. stub.getState(A)返回的是 Byte[]数组，　stub.getStringState(A)才返回的是字符串。（Invoke 函数的逻辑中）

3. 将 Go 中的 print 都写成，_logger.info(String.format("Aval = %d, Bval = %d\n", Aval,Bval));

4. query 函数的返回值：

我在进行 E2E 的测试的时候，因为没有 javaIDEA 本地的测试，只能通过 Terminal 打印的信息进行测试，然后就会报错如下：

```
CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
Attempting to Query PEER0 ...3 secs
Attempting to Query PEER0 ...38 secs
Attempting to Query PEER0 ...41 secs
Attempting to Query PEER0 ...44 secs
Attempting to Query PEER0 ...48 secs
Attempting to Query PEER0 ...51 secs
Attempting to Query PEER0 ...54 secs
Attempting to Query PEER0 ...57 secs
Attempting to Query PEER0 ...60 secs

2021-05-18 04:37:21.855 UTC [main] InitCmd -> WARN 001 CORE_LO
GGING_LEVEL is no longer supported, please use the FABRIC_LOGG
ING_SPEC environment variable
2021-05-18 04:37:21.860 UTC [main] SetOrdererEnv -> WARN 002 C
ORE_LOGGING_LEVEL is no longer supported, please use the FABRI
C_LOGGING_SPEC environment variable
100
!!!!!!!!!!!!!!! Query result on PEER0 is INVALID !!!!!!!!!!!!!
!!!
================== ERROR !!! FAILED to execute End-2-End Scena
rio ==================
```

然后为了寻找到底在哪里出了 ERROR,就需要回到 script.sh 脚本中进行查看，毕竟 DEBUG 需要根据不同的报错信息，进行相应的修改。这里是因为 query 失败导致过了 60secs 的 TIMEOUT.

```
# continue to poll
# we either get a successful response, or reach TIMEOUT
while test "$(($(date +%s)-starttime))" -lt "$TIMEOUT" -a $rc -ne 0
do
   sleep 3
   echo "Attempting to Query PEER$PEER ...$(($(date +%s)-starttime)) secs"
   peer chaincode query -C $CHANNEL_NAME -n mycc -c '{"Args":["query","a"]}' >&log.txt
   test $? -eq 0 && VALUE=$(cat log.txt | awk '/Query Result/ {print $NF}')
   test "$VALUE" = "$2" && let rc=0
done
echo
cat log.txt
if test $rc -eq 0 ; then
  echo "===================== Query on PEER$PEER on channel '$CHANNEL_NAME' is successful ===================== "
else
  echo "!!!!!!!!!!!!!!! Query result on PEER$PEER is INVALID !!!!!!!!!!!!!!!!"
      echo "================== ERROR !!! FAILED to execute End-2-End Scenario =================="
  echo
  exit 1
fi
```

最后求助了助教学长和同学，才发现是 Query 函数的返回值处出了问题。

需要在 newSuccessResponse 的返回值中，返回一个 AvalBytes.以对应 go 代码中的

```
return shim.Success(Avalbytes)
```

return newSuccessResponse(Avalbytes, ByteString.copyFrom(Avalbytes, UTF_8).toByteArray());

需要注意的是，直接在 Java 里返回 Avalbytes 是不行的，因为在 Java 中，我们的 Avalbytes 是 String 类型，需要将其转化为 Aval 数组。

五、总结

1. 这次实验收获颇丰，在帮同学调试代码的时候遇到了各种各样形形色色的错误，打了无数次 up 和 down 在最后看到案例成功运行的时候还是收获了很多成就感的。

虽然 go 的逻辑很简单，但是 java 在实现的过程中也有各种各样的坑，已经在过程中写了，在此处不再赘述。

此外就是注意编码的一些细节，如下图比如在这里如果我换成 Java 的变量命名规则可能会更好。

```
Name\":\"" + A + "\",\"Amount\":\"" + AvalBytes + "\"}";
Query Response:"+jsonResp);                    找不同！！！
nse(Avalbytes, ByteString.copyFrom(Avalbytes, UTF_8).toByteArray());
```

2. 在实验八的指导中和网上查阅 FabricE2E 案例中都有提到环境的安装部署是一个比较繁琐的过程，其中可能出现各种各样的错误需要尝试，配置文件设置错误、路径设置错误、网络错误等，导致安装失败。可以在有时间的时候进行这一步的踩坑拓展，可以提高理解和对系统的熟悉度。