node测试

关闭网络

```
$ sudo ./byfn.sh -m down
```

进入到fabric-samples/fabcar目录中

```
$ cd ../fabcar/
$ ls
```

> 如果是完整的环境,目录中应该包含如下文件:
>
> `enrollAdmin.js invoke.js package.json query.js registerUser.js startFabric.sh`

移除所有处于活动中的Docker容器

```
$ sudo docker rm -f $(docker ps -aq)
```

docker rm : 删除当前指定的Docker容器

docker rm -f $(docker ps -aq)

-f: 强制删除

查看Docker

```
$ sudo docker ps
```

清除所有的网络缓存:

```
$ sudo docker network prune
```

删除链码图像(链码镜像)

删除fabcar智能合约的底层链码图像。如果您是第一次运行此项目可以不执行(系统上不会有此链接代码图像)

```
$ sudo docker rmi dev-peer0.org1.example.com-fabcar-1.0-
5c906e402ed29f20260ae42283216aa75549c571e2e380f3615826365d8269ba
```

删除指定的Docker镜像文件

docker rmi image_id

安装客户端

安装应用程序的Fabric依赖关系

```
$ npm install
```

> **下载最好使用稳定的VPN**

> npm install: 根据 package.json 读取依赖的信息并安装
>
> **如果未安装Node则先按如下步骤安装Node及npm**
>
> ### 安装nvm

```
$ sudo apt update
$ curl -o-
https://raw.githubusercontent.com/creationix/nvm/v0.33.10/instal
l.sh | bash

$ export NVM_DIR="$HOME/.nvm"
$ [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

安装Node

```
$ nvm install v8.11.1
```

检查Node版本

```
$ node -v
```

输出: `v8.11.1`

检查npm版本

```
$ npm -v
```

输出: `5.6.0`

启动网络

```
$ sudo ./startFabric.sh
```

该命令将启动各种Fabric实体，并启动用Golang编写的链式代码的智能合约容器

如出现以下错误

```
ERROR: manifest for hyperledger/fabric-ca:latest not found
```

则说明环境中缺少 fabric-ca 镜像

> 下载镜像
> $ sudo docker pull hyperledger/fabric-ca:x86_64-1.1.0-preview
>
> 将其标记为最新
> $ sudo docker tag hyperledger/fabric-ca:x86_64-1.1.0-preview hyperledger/fabric-ca:latest
>
> 检查
> $ sudo docker images
>
> 重新启动
> $ sudo ./startFabric.sh

可选执行(流式处理CA日志, 打开新的终端并执行如下命令)

```
$ sudo docker logs -f ca.example.com
```

注册管理员用户

```
$ node enrollAdmin.js
```

命令执行后输出如下内容:

```
Successfully enrolled admin user "admin"
Assigned the admin user to the fabric client ::
{"name":"admin","mspid":"Org1MSP","roles":null,"affiliation":"","enr
ollmentSecret":"","enrollment":
{"signingIdentity":"dc412dcc161b5732737e98e77fda03433b55408d79b10195
f0ff150fc995924a","identity":{"certificate":"-----BEGIN CERTIFICATE-
----
\nMIICATCCAaigAwIBAgIUMjxyVKytJHiYigb+usxuVlmeI8kwCgYIKoZIzj0EAwIw\n
czELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbGlmb3JuaWExFjAUBgNVBAcTDVNh\nbi
BGcmFuY2lzY28xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVBAMT\nE2Nh
Lm9yZzEuZXhhbXBsZS5jb20wHhcNMTgwNDI2MDcyNzAwWhcNMTkwNDI2MDcz\nMjAwWj
AhMQ8wDQYDVQQLEwZjbGllbnQxDjAMBgNVBAMTBWFkbWluMFkwEwYHKoZI\nzj0CAQYI
KoZIzj0DAQcDQgAEszinoLQrvnKVYl9FUT8ebxT2jIz5lKCk5olL1cox\n/JchmLPG8E
w1roM2TgG64rvT1nrl1EvMwmD8oEOMgmGqwKNsMGowDgYDVR0PAQH/\nBAQDAgeAMAwG
A1UdEwEB/wQCMAAwHQYDVR0OBBYEFNG9kJBZBDSOwFxVHTDByNOl\nMvDSMCsGA1UdIw
QkMCKAIEI5qg3NdtruuLoM2nAYUdFFBNMarRst3dusalc2Xkl8\nMAoGCCqGSM49BAMC
A0cAMEQCIA1Ugh8NW3tS0GkuUrURdwQrSnFkdWTQhJ1/GvRd\nJbeTAiBGdDpHu/6mZG
8dpguA0EaqSHrWJBQra4Vj1Fm9F1+zNg==\n-----END CERTIFICATE-----\n"}}}
```

成功执行后会调用证书签名请求（CSR），并最终将eCert和密钥材料输出到此文件夹中`./hfc-key-store`，应用程序将在创建用户或加载身份对象时查找此位置

注册 user1用户

```
$ node registerUser.js
```

user1

该命令执行后调用CSR并将密钥和eCert输出到`./hfc-key-store`子目录中

执行命令后输出如下:

```
Successfully loaded admin from persistence
Successfully registered user1 - secret:mrOjTeyeUmWY
Successfully enrolled member user "user1"
User1 was successfully registered and enrolled and is ready to
intreact with the fabric network
```

查询分类帐

```
$ node query.js
```

命令执行后输出如下

```
Successfully loaded user1 from persistence
Query has completed, checking results
Response is  [{"Key":"CAR0", "Record"
{"colour":"blue","make":"Toyota","model":"Prius","owner":"Tomoko"}},
{"Key":"CAR1", "Record":
{"colour":"red","make":"Ford","model":"Mustang","owner":"Brad"}},
{"Key":"CAR2", "Record":
{"colour":"green","make":"Hyundai","model":"Tucson","owner":"Jin
Soo"}},{"Key":"CAR3", "Record":
{"colour":"yellow","make":"Volkswagen","model":"Passat","owner":"Max
"}},{"Key":"CAR4", "Record":
{"colour":"black","make":"Tesla","model":"S","owner":"Adriana"}},
{"Key":"CAR5", "Record":
{"colour":"purple","make":"Peugeot","model":"205","owner":"Michel"}}
,{"Key":"CAR6", "Record":
{"colour":"white","make":"Chery","model":"S22L","owner":"Aarav"}},
{"Key":"CAR7", "Record":
{"colour":"violet","make":"Fiat","model":"Punto","owner":"Pari"}},
{"Key":"CAR8", "Record":
{"colour":"indigo","make":"Tata","model":"Nano","owner":"Valeria"}},
{"Key":"CAR9", "Record":
{"colour":"brown","make":"Holden","model":"Barina","owner":"Shotaro"
}}]
```

由Adriana拥有的黑色特斯拉Model S，由Brad拥有的红色Ford Mustang，由Pari拥有的紫色Fiat Punto等等。

分类账是基于K-V的，在上面的信息中，Key为CAR0至CAR9

打开query.js

```
$ cat query.js
```

发现应用程序的初始部分定义了某些变量，例如通道名称，证书存储位置和网络端点

```
var channel = fabric_client.newChannel('mychannel');
var peer = fabric_client.newPeer('grpc://localhost:7051');
channel.addPeer(peer);

var member_user = null;
var store_path = path.join(__dirname, 'hfc-key-store');
console.log('Store path:'+store_path);
var tx_id = null;
```

query.js文件有如下代码使用第二个身份`user1`作为此应用程序的签署实体。指定`user1`为签名者

`fabric_client.getUserContext('user1', true);`

这是实现查询功能的语句块： 'queryCar' 'CAR4'

```
// queryCar chaincode function - requires 1 argument, ex: args:
['CAR4'],
// queryAllCars chaincode function - requires no arguments , ex:
args: [''],
const request = {
  //targets : --- letting this default to the peers assigned to the
channel
  chaincodeId: 'fabcar',
  fcn: 'queryAllCars',
  args: ['']
};
```

当应用程序运行时，它会调用对等`fabcar`体上的链式代码，运行其中的`queryAllCars`函数,且不传递任何参数.

使用编辑器打开query.js

```
$ vim query.js
```

修改其查询块内容,更改`queryAllCars`为`queryCar`并将`CAR4`作为特定Key为参数传递来执行此操作

```
const request = {
  //targets : --- letting this default to the peers assigned to the
channel
  chaincodeId: 'fabcar',
  fcn: 'queryCar',
  args: ['CAR4']
};
```

保存退出后运行:

```
$ node query.js
```

执行后返回如下

```
Successfully loaded user1 from persistence
Query has completed, checking results
Response is
{"colour":"black","make":"Tesla","model":"S","owner":"Adriana"}
```

使用该`queryCar`功能，我们可以查询任何关键字（例如`CAR0`）并获取与该车相对应的任何品牌，型号，颜色和所有者

更新分类帐

修改`invoke.js`,找到`var request`中的`fcn`与`args`,添加一条新的数据

```
$ vim invoke.js
```

```
    var request = {
                //targets: let default to the peer assigned to the
client
                chaincodeId: 'fabcar',
                fcn: 'createCar',
                args: ['CAR10','Chevy','Volt','Red','Nick'],
                chainId: 'mychannel',
                txId: tx_id
        };
```

保存退出后执行

```
$ node invoke.js
```

执行成功,输出如下

```
Successfully loaded user1 from persistence
Assigning transaction_id:
801d0636b9aa94cc7782af21ec2a10ebb12f929bd722afcee1f2b7b923485c82
Transaction proposal was good
Successfully sent Proposal and received ProposalResponse: Status -
200, message - "OK"
The transaction has been committed on peer localhost:7053
Send transaction promise and event listener promise have completed
Successfully sent transaction to the orderer.
Successfully committed the change to the ledger by the peer
```

返回`query.js`并将参数由`CAR4`更改为`CAR10`

```
$ vim query.js
```

```
const request = {
  //targets : --- letting this default to the peers assigned to the
channel
  chaincodeId: 'fabcar',
  fcn: 'queryCar',
  args: ['CAR10']
};
```

查询:

```
node query.js
```

输出内容如下:

```
Successfully loaded user1 from persistence
Query has completed, checking results
Response is
{"colour":"Red","make":"Chevy","model":"Volt","owner":"Nick"}
```

修改 `invoke.js`, 修改CAR10的拥有者为 Dave

```
$ vim invoke.js
```

```
var request = {
  //targets: let default to the peer assigned to the client
  chaincodeId: 'fabcar',
  fcn: 'changeCarOwner',
  args: ['CAR10','Dave'],
  chainId: 'mychannel',
  txId: tx_id
};
```

保存退出并执行

```
$ node invoke.js
$ node query.js
```

运行输出结果:

```
Successfully loaded user1 from persistence
Query has completed, checking results
Response is
{"colour":"Red","make":"Chevy","model":"Volt","owner":"Dave"}
```

问题:

在执行 node invoke.js 命令后出现如下错误

```
Store path:$HOME/hyfa/fabric-samples/fabcar/hfc-key-store
Successfully loaded user1 from persistence
Assigning transaction_id:
f80947242014765a46a17d797b45c8ed9a5db5cc936a57c731219d9e25646051
Transaction proposal was good
Successfully sent Proposal and received ProposalResponse: Status -
200, message - "OK"
Failed to invoke successfully :: TypeError: Cannot read property
'getConnectivityState' of undefined
$HOME/hyfa/fabric-samples/fabcar/node_modules/fabric-
client/lib/EventHub.js:308
            if(self._stream) state =
self._stream.call.channel_.getConnectivityState();
                                                                    ^

TypeError: Cannot read property 'getConnectivityState' of undefined
    at ClientDuplexStream.<anonymous> ($HOME/hyfa/fabric-
samples/fabcar/node_modules/fabric-client/lib/EventHub.js:308:56)
    at emitOne (events.js:116:13)
    at ClientDuplexStream.emit (events.js:211:7)
    at addChunk (_stream_readable.js:263:12)
```

```
    at readableAddChunk (_stream_readable.js:250:11)
    at ClientDuplexStream.Readable.push (_stream_readable.js:208:10)
    at Object.onReceiveMessage ($HOME/hyfa/fabric-
samples/fabcar/node_modules/grpc/src/client_interceptors.js:1302:19)
    at InterceptingListener.recvMessageWithContext
($HOME/hyfa/fabric-
samples/fabcar/node_modules/grpc/src/client_interceptors.js:629:19)
    at $HOME/hyfa/fabric-
samples/fabcar/node_modules/grpc/src/client_interceptors.js:728:14
```

此问题可以无视, 不会影响到后继查询命令 `node query.js` 的执行