环境准备

系统环境准备

需要事先准备如下环境：

- Ubuntu 16.04 / MacOS 10.15
- Golang：1.13.x

启动链

为了简化fabric链的运行，这里是基于fabric-samples v1.4.0里链启动方式来进行部署

1.克隆fabric-samples仓库

```
git clone git@github.com:hyperledger/fabric-samples.git
```

2.切换分支到v1.4.0

```
git checkout v1.4.0
```

3.修改docker-compose-simple.yaml

在chaincode及cli里添加如下environment以及volumes：

```
cd github.com/smart-audit/src/fabric
cp docker-compose-simple.yaml
github.com/hyperledger/fabric-samples/chaincode-docker-
devmode/docker-compose-simple.yaml
```

4.拷贝smart-audit源码至chaincode目录

```
cd github.com/smart-audit/src
cp -r fabric github.com/hyperledger/fabric-samples/chaincode/smart_audit
cp -r contract github.com/hyperledger/fabric-samples/chaincode/smart_audit
cp -r oracles github.com/hyperledger/fabric-samples/chaincode/smart_audit
```

5.启动链

```
cd github.com/hyperledger/fabric-samples/chaincode-docker-devmode
docker-compose -f docker-compose-simple.yaml up
```

合约调用

部署合约

部署时间合约

1.通过如下命令初始化时间服务合约

```
docker exec cli peer chaincode install -p chaincodedev/chaincode/smart_audit/fabric/time -n Time -v 0
```

返回如下信息时初始化成功:

```
Installed remotely response:<status:200 payload:"OK" >
```

2.通过如下命令实例化时间服务合约

```
docker exec cli peer chaincode instantiate -n Time -v 0 -c '{"Args":[]}' -C myc
```

实例化后没有返回错误则部署成功

启动位置合约

1.通过如下命令初始化位置服务合约

```
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/location -n
Location -v 0
```

返回如下信息时初始化成功:

```
Installed remotely response:<status:200 payload:"OK" >
```

2.通过如下命令实例化位置服务合约

```
docker exec cli peer chaincode instantiate -n Location -v 0
-c '{"Args":[]}' -C myc
```

实例化后没有返回错误则部署成功

启动人脸识别合约

1.通过如下命令初始化人脸识别服务合约

```
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/face -n
FaceRecognize -v 0
```

返回如下信息时初始化成功:

```
Installed remotely response:<status:200 payload:"OK" >
```

2.通过如下命令实例化人脸识别服务合约

```
docker exec cli peer chaincode instantiate -n FaceRecognize
-v 0 -c '{"Args":[]}' -C myc
```

实例化后没有返回错误则部署成功

启动物体识别合约

1.通过如下命令初始化物体识别服务合约

```
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/identify -n
ObjectRecognize -v 0
```

返回如下信息时初始化成功:

```
Installed remotely response:<status:200 payload:"OK" >
```

2.通过如下命令实例化物体识别服务合约

```
docker exec cli peer chaincode instantiate -n
ObjectRecognize -v 0 -c '{"Args":[]}' -C myc
```

实例化后没有返回错误则部署成功

启动审计业务合约

1.通过如下命令初始化审计业务合约

```
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/audit -n audit -v
0
```

返回如下信息时初始化成功:

```
Installed remotely response:<status:200 payload:"OK" >
```

2.通过如下命令实例化审计业务合约

```
docker exec cli peer chaincode instantiate -n audit -v 0 -c
'{"Args":["init","maintainer1","maintainer2"]}' -C myc
```

实例化后没有返回错误则部署成功

调用合约

合约维护人员查询

1.通过如下命令查询合约维护人员初始化是否成功：

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["getMaintainers"]}'
```

如果查询成功，则会返回如下响应消息：

```
Chaincode invoke successful. result: status:200 payload:"
{\"result\":[{\"Name\":\"maintainer1\",\"ID\":0},
{\"Name\":\"maintainer2\",\"ID\":1}]}"
```

可以看到返回结果中包含了在部署审计合约时传入的运维人员。

录入审计当事人

1.通过如下命令录入一个审计当事人

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["registerAuditee", "ZhanSan"]}'
```

如果录入成功会返回如下响应消息：

```
Chaincode invoke successful. result: status:200 payload:"0"
```

此处返回的payload里为审计当事人的ID，会从0递增

2.通过如下命令查询我们注册的审计当事人信息

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["getAuditee", "0"]}'
```

如果调用成功会返回如下响应信息：

```
Chaincode invoke successful. result: status:200 payload:"
{\"Name\":\"ZhanSan\",\"ID\":0}"
```

录入规则

1.通过如下命令录入一个审计规则：

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["registerRules", "AND", "Time", "(>= 9) AND (<=
18)", "Location", "IN(39.9 116.3 1000)", "FaceRecognize",
"", "ObjectRecognize", ""]}'
```

如果录入成功则会返回如下响应消息：

```
Chaincode invoke successful. result: status:200 payload:"0"
```

此处返回的payload里为审计规则的ID，会从0递增

2.通过如下命令查询我们注册的审计规则

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["getRules", "0"]}'
```

如果调用成功则会返回如下响应消息：

```
Chaincode invoke successful. result: status:200 payload:"
{\"Operator\":\"AND\",\"Rules\":
{\"FaceRecognize\":0,\"Location\":0,\"ObjectRecognize\":0,\
"Time\":0},\"ID\":0}"
```

录入项目

1.通过如下命令录入一个项目：

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["registerProject", "POS Audit", "This is a bank
project, used by bank employees to check if they did check
the POS related bussiness themselfs within the specified
time and location","0", "0"]}'
```

如果录入成功则会返回如下响应消息：

```
Chaincode invoke successful. result: status:200 payload:"0"
```

此处返回的payload里为项目的ID，会从0递增

2.通过如下命令查询我们注册的项目信息：

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["getProject", "0"]}'
```

如果查询成功则会返回如下响应消息：

```
Chaincode invoke successful. result: status:200 payload:"
{\"Name\":\"POS Audit\",\"ID\":0,\"Description\":\"This is
a bank project, used by bank employees to check if they did
check the POS related bussiness themselfs within the
specified time and location\",\"AuditeeRulesMap\":{\"
{\\\"Name\\\":\\\"ZhanSan\\\",\\\"ID\\\":0}\":\"
{\\\"Operator\\\":\\\"AND\\\",\\\"Rules\\\":
{\\\"FaceRecognize\\\":0,\\\"Location\\\":0,\\\"ObjectRecog
nize\\\":0,\\\"Time\\\":0},\\\"ID\\\":0}\"}}"
```

新增审计事件

1.通过如下命令新增一个审计事件

注意此处的时间验证因为是根据当前时间验证需要将Time后的时间切到现在的日期

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["addEvent", "0", "0", "1589532423", "Time",
"2020-06-01T15:04:05.000Z", "Location", "39.901 116.299",
"FaceRecognize", "/9j/4SMF...", "ObjectRecognize",
"iVBORw0..."]}'
```

如果新增成功则会返回如下响应消息：

```
Chaincode invoke successful. result: status:200
payload:"OK"
```

2.通过如下命令可以查询所有满足条件的审计事件：

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["queryEvents", "0", "0"]}'
```

如果查询成功则会返回如下响应消息：

```
Chaincode invoke successful. result: status:200 payload:"
{\"result\":[{\"ID\":
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0],\"Auditee\":{\"Name\":\"\",\"ID\":0},\"Project\":
{\"Name\":\"\",\"ID\":0,\"Description\":\"\",\"AuditeeRules
Map\":null},\"Rule\":{\"Operator\":\"\",\"Rules\":
{},\"ID\":0},\"Timestamp\":1589532423,\"Params\":
[\"Time\",\"2020-06-
01T15:04:05.000Z\",\"Location\",\"39.901
116.299\",\"FaceRecognize\",\"/9j/4SMF...\",\"ObjectRecogni
ze\",\"iVBORw0...\"],\"Index\":0}]}"
```

快速启动所有合约

在启动链后，可以用以下命令直接复制在terminal里执行，就能完成合约的部署。

```
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/time -n Time -v 0
docker exec cli peer chaincode instantiate -n Time -v 0 -c
'{"Args":[]}' -C myc
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/face -n
FaceRecognize -v 0
docker exec cli peer chaincode instantiate -n FaceRecognize
-v 0 -c '{"Args":[]}' -C myc
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/identify -n
ObjectRecognize -v 0
docker exec cli peer chaincode instantiate -n
ObjectRecognize -v 0 -c '{"Args":[]}' -C myc
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/location -n
Location -v 0
docker exec cli peer chaincode instantiate -n Location -v 0
-c '{"Args":[]}' -C myc
docker exec cli peer chaincode install -p
chaincodedev/chaincode/smart_audit/fabric/audit -n audit -v
0
docker exec cli peer chaincode instantiate -n audit -v 0 -c
'{"Args":["init","maintainer1","maintainer2"]}' -C myc
```

快速录入数据

在启动链后，可以用以下命令直接复制在terminal里执行，就能简单的数据录入。

```
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["registerAuditee", "ZhanSan"]}'
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["registerRules", "AND", "Time", "(>= 9) AND (<=
18)", "Location", "IN(39.9 116.3 1000)", "FaceRecognize",
"", "ObjectRecognize", ""]}'
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["registerProject", "POS Audit", "This is
location","0", "0"]}'
docker exec cli peer chaincode invoke -n audit -C myc -c
'{"Args": ["addEvent", "0", "0", "1589532423", "Time",
"2020-05-26T15:04:05.000Z", "Location", "39.901 116.299",
"FaceRecognize", "/9j/4SMF...", "ObjectRecognize",
"iVBORw0..."]}'
```

参考链接

1. Fabric测试环境部署
2. 创建合约