# 环境准备

## Fabric网络环境

使用1.1.0版本fabric-samples的环境运行

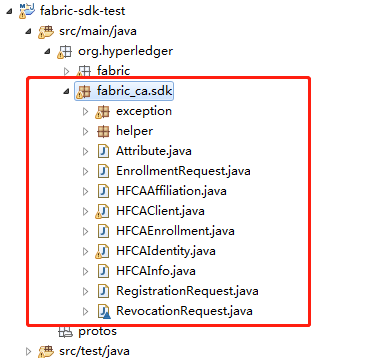
git checkout -b release-1.1 origin/release-1.1

## Fabric-ca服务器

Fabric-ca的环境请参照《Fabric-ca集成》中的环境搭建，当启动CA服务器则表示CA服务器搭建成功。作者CA服务器使用了Mysql数据库，可参考《Fabric-ca服务器配置数据库和集群》中配置数据库的方式启动。

## Java运行环境

Java运行环境请参考《Fabric-java-sdk例子》中的环境搭建，git拉取的代码中有Fabric-ca模块如下图，这个便是我们Fabric-ca需要的java代码。



# 使用Cryptogen工具生成必要的文件

在运行fabric-samples的机器上，安装1.1.0版本的Cryptogen和configtxgen工具，也就是在《搭建第一个fabric网络》中使用的bin文件夹，因为这里使用的1.1.0版本的fabric，所以下载对应版本工具。之后进入first-network文件夹执行以下命令生成证书文件。

../bin/cryptogen generate --config=./crypto-config.yaml

# 使通过Fabric-ca服务器生成前提证书

## CA添加所需组织

fabric-ca-client affiliation add com

fabric-ca-client affiliation add com.example

fabric-ca-client affiliation add com.example.org1

fabric-ca-client affiliation add com.example.org2

## CA生成各个组织的根证书和管理员证书

请参照《Fabric-ca集成》中：

1. 生成example.com的证书。
2. 生成org1.example.com和org2.example.com的根证书

生成根证书：

mkdir -p /root/fabric-ca-files/org1.example.com/msp

fabric-ca-client getcacert -M /root/fabric-ca-files/org1.example.com/msp

## CA生成orderer和peer节点证书

请参照《Fabric-ca集成》中：生成orderer的证书、生成peer的证书。生成orderer和peer节点证书。

# 合并工具和CA服务器生成的证书

请参照《Fabric-ca集成》中：

1. 替换ordererOrganizations\example.com中的文件
2. 替换peerOrganizations\org1.example.com中的文件
3. 替换peerOrganizations\org2.example.com中的文件

进行替换文件。

# 编写Java代码

## 初始化配置信息

/\*Fabric-ca服务器ip地址和端口号 \*/

**private** **static** **final** String ***ip*** = "http://192.168.15.142:7054";

/\*Fabric-ca服务器名称 \*/

**private** **static** **final** String ***CA\_NAME*** = "client";

/\*本地生成证书根目录 \*/

**private** **static** **final** String ***MSP\_PATH*** = "D:\\msp\\";

/\*Fabric-ca操作对象 \*/

**private** HFCAClient client;

通过构造函数，初始化Fabric-ca客户端操作对象

**public** UserEnroll() **throws** Exception{

//通过CA服务器地址和端口、CA服务器名称，获得CA客户端实体类

client = HFCAClient.*createNewInstance*(***CA\_NAME***, ***ip***, **null**);

//调用client的登录方法，参数为登录用户的用户名和用户密码

CryptoSuite cryptoSuite = CryptoSuite.Factory.*getCryptoSuite*();

client.setCryptoSuite(cryptoSuite);

}

## 连接CA服务器查看CA服务器基本信息

//新建对象，初始化操作对象

UserEnroll u = **new** UserEnroll();

//获取CA服务器的信息

HFCAInfo HFCAInfo = u.client.info();

HFCAInfo对象属性有：

//CA服务器名称

**private** **final** String caName;

//CA服务器证书链

**private** **final** String caChain;

//CA服务器版本

**private** **final** String version;

## 登录已在CA服务器注册的用户

//新建对象，初始化操作对象

UserEnroll u = **new** UserEnroll();

//用户登录

u.client.enroll("admin", "adminpw");

## 注册新用户

//新建对象，初始化操作对象

UserEnroll u = **new** UserEnroll();

//初始化执行注册用户

CaUser serverUser = **new** CaUser();

serverUser.setName("User8@org1.example.com");

Enrollment enrollment = u.client.enroll(serverUser.getName(),"password");

//执行注册用户设置登录凭证

serverUser.setEnrollment(enrollment);

//初始化注册用户

CaUser registerUser = **new** CaUser("User@org1.example.com","password","com.example.org1");

//注册新用户的信息：用户名称与用户所在组织名

//用户名不能有重复，组织名必须是执行注册的人拥有操作权限的组织列表中存在的

RegistrationRequest regreq = **new** RegistrationRequest(registerUser.getName(), registerUser.getAffiliation());

regreq.setSecret(registerUser.getPassword());

//设置注册用户的属性，根据注册用户情况设置

Attribute attrAf = **new** Attribute("hf.AffiliationMgr", "true");

regreq.addAttribute(attrAf );

Attribute attrRevoker = **new** Attribute("hf.Revoker", "true");

regreq.addAttribute(attrRevoker);

u.client.register(regreq, serverUser);

CaUser类信息如下，需要实现org.hyperledger.fabric.sdk.User的接口。

//用户名

**public** String name;

//用户密码

**public** String password;

//用户类型

**public** String type;

//用户所属组织

**public** String affiliation;

因为在注册方法里只使用了跟用户登录凭证有关的方法，所以对于User接的方法，其他的都不需要具体实现，只需要修改getEnrollment()方法

@Override

**public** Enrollment getEnrollment() {

// **TODO** Auto-generated method stub

**return** enrollment;

}

## 获取用户的证书文件

//新建对象，初始化操作对象

UserEnroll u = **new** UserEnroll();

CaUser serverUser = **new** CaUser();

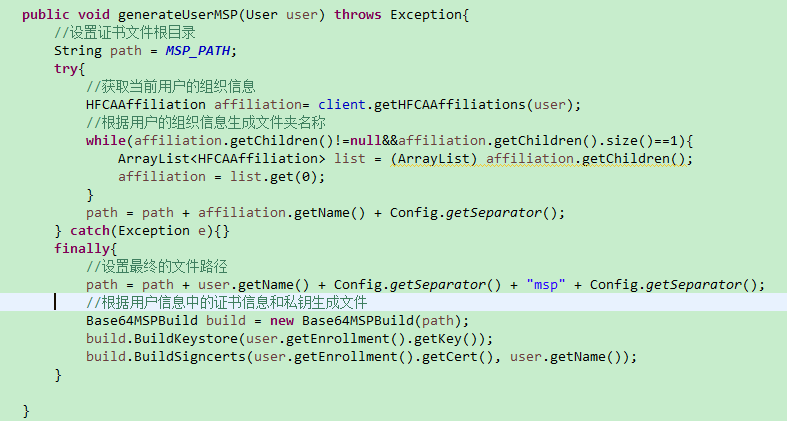
//用户登录

Enrollment enrollment = u.client.enroll("User@org1.example.com","password");

//用户设置登录凭证

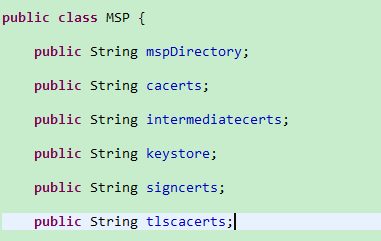
serverUser.setEnrollment(enrollment);

u.generateUserMSP(serverUser);









# 使用代码生成的证书进行交易

## 生成组织管理员和普通话用户证书

注册org1组织管理员Admin@org1.example.com，设置属性如下：

attributes:

- name: hf.Registrar.Roles

value: client,orderer,peer,user

- name: hf.Registrar.DelegateRoles

value: client,orderer,peer,user

- name: hf.GenCRL

value: true

- name: hf.Revoker

value: true

- name: hf.AffiliationMgr

value: true

- name: hf.IntermediateCA

value: true

- name: role

value: admin

ecert: true

注册or1普通用户User@org1.example.com，可以使用默认属性，即不设置属性。

## 替换管理员证书和用户用户证书

先将org1内msp\admincerts内的管理员证书替换成上一步代码生成的Admin@org1.example.com的证书：Admin@org1.example.com-cer.pem

替换的地方：

crypto-config\peerOrganizations\org1.example.com\msp\admincerts

crypto-config\peerOrganizations\org1.example.com\peers\peer0.org1.example.com\msp\admincerts

crypto-config\peerOrganizations\org1.example.com\peers\peer1.org1.example.com\msp\admincerts

crypto-config\peerOrganizations\org1.example.com\users\Admin@org1.example.com\msp\admincerts

crypto-config\peerOrganizations\org1.example.com\users\User1@org1.example.com\msp\admincerts

然后将Admin@org1.example.com和User1@org1.example.com里的证书和私钥换成java代码生成的对应文件。

Cacerts：CA服务器根证书

Keystore：java代码生成的私钥

Signcerts：java代码生成的用户证书

## 使用替换后的证书文件生成交易配置文件

在运行fabric-samples的机器上，进入目录执行以下命令生成配置文件：

export FABRIC\_CFG\_PATH=$PWD

../bin/configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./channel-artifacts/genesis.block

export CHANNEL\_NAME=mychannel

../bin/configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID $CHANNEL\_NAME

../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org1MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org1MSP

../bin/configtxgen -profile TwoOrgsChannel -outputAnchorPeersUpdate ./channel-artifacts/Org2MSPanchors.tx -channelID $CHANNEL\_NAME -asOrg Org2MSP

## 用替换后的证书文件去执行交易

参考《搭建第一个fabric网络》，执行启动网络，创建channel，安装执行chaincode等操作。

切换用户User执行以下命令：

CORE\_PEER\_MSPCONFIGPATH=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.example.com/users/User@org1.example.com/msp

如果一系列操作执行成功，说明证书生成成功。