#修改crypto-config、configtx文件:

1个Orderer、1个Org、5个Peer、1个User

#修改build.sh脚本，重新生成证书:

提前添加bin目录(包含二进制文件)，不然无法执行cryptogen、configtxgen命令；将生成的config、crypto-config目录放到network\_resource里面(和docker-compose.yml指定的路径一致)

#在docker-compose.yml里将ca证书换为新生成的:

crypto-config/peerOrganizations/org1.example.com/ca/...\_sk

#在docker-compose.yml里新增节点配置:

1个ca、1个orderer、5个peer(peer0—peer4)，修改名字和端口号

#build.sh启动容器(1+1+5)

#修改Config.java文件:

将ORG2的配置注释掉；修改ORG1节点信息（注意端口号）

#新建Manager类，合并main函数

将创建通道、新增节点、链码安装和实例化通过函数调用

#docker-compose文件增加couchdb，验证fabcar链码可用couchdb

#完成链码修改（基于marble）

可安装并且实例化，链码名称为identity.go

#java添加数据前端及功能实现

通过按钮获取输入的数据（name,address,type),调用链码的initIdentity函数进行添加

#java查询数据前端及功能实现

通过名称name，查询数据（name,address,type),调用链码的quaryIdentityByName函数进行查询

#################################

执行过程：

1）进入v8/network目录，执行./build.sh建网（可用docker ps查看已生成的容器，包括5个peer,5个couchdb,1个orderer,1个ca）

2）进入v8/java目录，输入mvn install命令（需安装maven），生成target目录，进入后将block-java-sdk-0.0.1-SNAPSHOT-jar-with-dependencies.jar文件复制到v8/network\_resources目录下，可将其重命名为v8.jar

3）进入v8/network\_resources目录，执行java -cp v8.jar org.app.network.Manager命令，随即弹出图形界面，按顺序操作即可（节点数最多为5；链码实例化过程较长）

* 单机部署

# 基本配置：

docker镜像（orderer、peer、ca等） 1.4.0

1个org，1个orderer，1个ca，15个peer，15个couchdb

# 建网流程：

在v11/network/目录下执行./build.sh，成功后docker ps确认打开的容器数量

# java类说明

v11/java/src/main/java/org/app/network/目录下的Manager类：

提供了一个构造器，接收一个jsonObject参数，其中包含两个jsonArray（分别对应entry、peer），每个jsonArray又包含多个jsonObject（对应每个节点），每个jsonObject包含ip、port两个属性

* 调用举例：

import org.app.network.Manager //导入Manager类

public class main{

public static void main(String[] args) throws JSONException {

//自定义一个jsonObject对象（最外层），用于存放entrys、peers

JSONObject jsonObject = new JSONObject();

//自定义一个entryArray对象，用于存放所有的entry

JSONArray entryArray = new JSONArray();

//假设有2个entry，每个节点都是一个jsonObject对象，

//其中包含ip、port两条属性

JSONObject entry0 = new JSONObject(); //entry0

entry0.put("ip", "localhost");

entry0.put("port", "7051");

JSONObject entry1 = new JSONObject(); //entry1

entry1.put("ip", "localhost");

entry1.put("port", "7056");

//把生成的2个entry加入entryArray

entryArray.put(entry0);

entryArray.put(entry1);

//自定义一个peerArray对象，用于存放所有的peer

JSONArray peerArray = new JSONArray();

//假设有3个peer，每个节点都是一个jsonObject对象，

//其中同样包含ip、port两条属性

JSONObject peer0 = new JSONObject(); //peer0

peer0.put("ip", "localhost");

peer0.put("port", "8051");

JSONObject peer1 = new JSONObject(); //peer1

peer1.put("ip", "localhost");

peer1.put("port", "8056");

JSONObject peer2 = new JSONObject(); //peer2

peer2.put("ip", "localhost");

peer2.put("port", "9051");

//把生成的3个peer加入peerArray

peerArray.put(peer0);

peerArray.put(peer1);

peerArray.put(peer2);

//最后把entryArray、peerArray加入最外层的jsonObject

jsonObject.put("entry",entryArray);

jsonObject.put("peer",peerArray);

//通过jsonObject调用函数构造器

Manager m = new Manager(jsonObject);

//执行结束，返回true

boolean b = m.isOver();

}

}

* 最终结果：创建通道—>加入2个entry—>安装/实例化链码—>再加入3个peer（只有entry可实现添加/查询数据功能，peer只是作为commit节点）

v11/java/src/main/java/org/app/chaincode/invocation目录下的InvokeChaincode类：

定义了两个函数。invokeInit()：用于初始化，仅在第一次添加数据时执行； invoke(String fcn, JSONObject jsonObject)：其中fcn对应链码中的initIdentity函数，jsonObject代表添加的对象（包含identifier、syn\_data、id\_type、root\_id）。最后同步到数据库时会自动添加时间戳（invoke\_time）

* 调用举例：

import org.app.chaincode.invocation.InvokeChaincode //导入InvokeChaincode类

public class main{

public static void main(String[] args) throws JSONException {

//自定义一个jsonObject对象，存放需添加的数据

JSONObject jsonObject = new JSONObject();

//其中包含identifier、syn\_data、id\_type、root\_id四条属性

jsonObject.put("identifier","bupt");

jsonObject.put("syn\_data","10.3.8.211");

jsonObject.put("id\_type ","handle");

jsonObject.put("root\_id","01");

InvokeChaincode invokeTest = new InvokeChaincode();

invokeTest. invoke(“initIdentity”, jsonObject) //调用链码，写入数据

}

}

* 最终结果：可以通过couchdb（地址是localhost:5984/\_utils）查到该数据的所有信息，包括添加数据的时间（invoke\_time）

v11/java/src/main/java/org/app/config/目录下的Config类：

用于设置证书、通道配置文件、链码的路径，以及orderer和ca的ip和端口等，一般不用进行更改

# 其他

v11.jar包在v11/network\_resources目录下