需要三个终端

终端一：启动网络

首先进入开发者模式目录：

/home/li/go/src/github.com/hyperledger/fabric-samples/chaincode-docker-devmode

启动网络：

docker-compose -f docker-compose-simple.yaml up

可以看到下面四个容器都是UP状态 docker ps



如果报错

之前遇到问题：cli一直被容器所占用

原因：没有关闭之前运行的网络，

docker-compose -f docker-compose-simple.yaml down

第二个终端：编译链码

先进入链码容器

sudo docker exec -it chaincode bash

进入chaincode容器

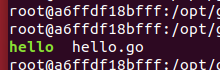
编译想要测试的chaincode:

这里以hello为例 在chaincode 建一个hello文件夹，里面建个hello.go，复制下面的代码

**package** main  
  
**import** (  
 **"fmt"  
  
 "github.com/hyperledger/fabric/core/chaincode/shim"  
 "github.com/hyperledger/fabric/protos/peer"**)  
  
**type** HelloChaincode **struct** {  
}  
  
  
**func** (t \*HelloChaincode) Init(stub shim.ChaincodeStubInterface) peer.Response {  
 fmt.Println(**"开始实例化链码........."**)  
 \_, args := stub.GetFunctionAndParameters()  
 **if** len(args) != 2 {  
 **return** shim.Error(**"指定了错误的参数个数"**)  
 }  
 fmt.Println(**"保存数据......."**)  
  
  
 err := stub.PutState(args[0], []byte(args[1]))  
 **if** err != nil {  
 **return** shim.Error(fmt.Sprintf(**"保存数据时出错........"**))  
 }  
 **return** shim.Success(nil)  
}  
  
  
**func** (t \*HelloChaincode) Invoke(stub shim.ChaincodeStubInterface) peer.Response {  
 *// Extract the function and args from the transaction proposal* fun, args := stub.GetFunctionAndParameters()  
  
 **if** fun == **"query"** {  
 **return** query(stub, args)  
 }  
  
  
 **return** shim.Error(**"非法操作，指定功能不能实现"**)  
}  
  
**func** query(stub shim.ChaincodeStubInterface, args []string) peer.Response{  
 **if** len(args) != 1{  
 **return** shim.Error(**"指定的参数错误，必须且只能指定相应的Key"**)  
 }  
 result,err:=stub.GetState(args[0])  
 **if** err!=nil{  
 **return** shim.Error(**"根据指定的"**+args[0]+**"查询数据时发生错误"**)  
 }  
  
  
 **if** result==nil{  
 **return** shim.Error(**"根据指定的"**+args[0]+**"没有查询到相应的数据"**)  
 }  
  
 **return** shim.Success(result)  
}  
  
  
**func** main() {  
 **if** err := shim.Start(new(HelloChaincode)); err != nil {  
 fmt.Printf(**"链码启动失败: %s"**, err)  
 }  
}

cd hello(自己写的hello代码)

go build



会产生可执行文件

启动这个可执行文件：

CORE\_PEER\_ADDRESS=peer:7052 CORE\_CHAINCODE\_ID\_NAME=hellocc:0 ./hello

暂停在那

第三个终端：在cli中测试链码

进入cli容器：

docker exec -it cli bash

安装和实例化链码（实例化设置了a的初始值10）：

安装

peer chaincode install -p chaincodedev/chaincode/hello -n hellocc -v 0

1563192061(1)

实例化

peer chaincode instantiate -n hellocc -v 0 -c '{"Args":["init","hello","world"]}' -C myc

（调用函数不同

peer chaincode invoke -n testcc -c '{"Args":["set","a","90"]}' -C myc

这里不需要

）

查询

peer chaincode query -n hellocc -c '{"Args":["query","hello"]}' -C myc

输出world

退出第三个终端使用

Exit

退出第二个终端

键盘输入ctrl+c

再exit

退出第一个终端

键盘输入ctrl+c

再输入docker-compose -f docker-compose-simple.yaml down

关闭网络

下面万能

docker stop $(docker ps -aq)

docker rm $(docker ps -aq)