简单的资产Chaincode

应用程序是一个基本样本链代码，用于在分类账上创建资产

安装Go及Docker, Docker-compose, 并配置Go相应的环境变量

创建目录

为chaincode应用程序创建一个目录作为其子目录

```
$ mkdir -p $GOPATH/src/test
$ cd $GOPATH/src/test
```

新建文件

新建一个文件,用于编写Go代码

```
$ touch test.go
$ vim test.go
```

编写代码

必须实现 Chaincode接口的 `Init` 和 `Invoke` 函数。因此，须在文件中添加go import语句以获取链代码的依赖。

导入chaincode shim包和 peer protobuf包。然后添加一个结构 `SimpleChaincdoe` 作为Chaincode函数的接收器

```
package main

import (
    "fmt"

    "github.com/hyperledger/fabric/core/chaincode/shim"
    "github.com/hyperledger/fabric/protos/peer"
)

type SimpleChaincdoe struct {
}
```

初始化Chaincode

Init方法

- 获取参数, 使用GetStringArgs 函数检索调用 的参数
- 检查合法性, 检查参数数量是否为2个, 如果不是, 则返回错误信息
- 利用两参数, 调用PutState方法向账本中写入状态, 如果有错误则返回(shim.Error()), 否则返回nil(shim.Success)

```
func (t *SimpleChaincdoe) Init(stub shim.ChaincodeStubInterface)
peer.Response {

}
```

调用实现

Invoke方法

验证函数名称为`set`或`get`，并调用那些链式代码应用程序函数，通过`shim.Success`或`shim.Error`函数返回响应

- 获取函数名与参数
- 对获取到的参数名称进行判断, 如果为set, 则调用set方法, 反之调用get
- set/get函数返回两个值(result，err)
- 如果err不为空则返回错误

- err为空则返回[]byte(result)

```
func (t *SimpleChaincdoe) Invoke(stub shim.ChaincodeStubInterface)
peer.Response {

}
```

实现Chaincode应用

chaincode应用程序实现了两个可以通过`Invoke`函数调用的函数(set/get)

为了访问分类账的状态，利用 chaincode shim API 的 ChaincodeStubInterface.PutState 和ChaincodeStubInterface.GetState函数

set函数, 返回两个值

- 检查参数个数是否为2
- 利用PutState方法将状态写入
- 如果成功,则返回要写入的状态, 失败返回错误: fmt.Errorf("...")

```
func set(stub shim.ChaincodeStubInterface, args []string) (string,
error) {

}
```

get函数, 返回两个值

- 接收参数并判断个数 是否为1个
- 调用GetState方法返回并接收两个返回值(value, err)
- 判断err及value是否为空 return "", fmt.Errorf("......")
- 返回值 return string(value), nil

```
func get(stub shim.ChaincodeStubInterface, args []string) (string,
error) {

}
```

main方法

```
func main() {
    if err := shim.Start(new(SimpleChaincdoe)); err != nil {
            fmt.Printf("Error starting SimpleAsset chaincode: %s",
err)
    }
}
```

构建Chaincode

编译chaincode

```
$ go get -u --tags nopkcs11
github.com/hyperledger/fabric/core/chaincode/shim
$ go build --tags nopkcs11
```

使用开发模式测试

正常情况下chaincode由对等体启动和维护。然而，在“开发模式”下，链码由用户构建并启动

如果没有安装Hyperledger Fabric Samples请先安装

如果没有下载Docker images请先下载

跳转至`fabric-samples`的`chaincode-docker-devmode`目录

```
$ cd ~/hyfa/fabric-samples/chaincode-docker-devmode/
```

使用`docker images`查看Docker镜像信息(显示本地Docker Registry)

```
$ sudo docker images
```

会看到如下输出

```
REPOSITORY                    TAG                 IMAGE ID
   CREATED             SIZE
hyperledger/fabric-tools      latest              b7bfddf508bc
About an hour ago   1.46GB
hyperledger/fabric-tools      x86_64-1.1.0        b7bfddf508bc
About an hour ago   1.46GB
hyperledger/fabric-orderer    latest              ce0c810df36a
About an hour ago   180MB
hyperledger/fabric-orderer    x86_64-1.1.0        ce0c810df36a
About an hour ago   180MB
hyperledger/fabric-peer       latest              b023f9be0771
About an hour ago   187MB
hyperledger/fabric-peer       x86_64-1.1.0        b023f9be0771
About an hour ago   187MB
hyperledger/fabric-javaenv    latest              82098abb1a17
About an hour ago   1.52GB
hyperledger/fabric-javaenv    x86_64-1.1.0        82098abb1a17
About an hour ago   1.52GB
hyperledger/fabric-ccenv      latest              c8b4909d8d46
About an hour ago   1.39GB
hyperledger/fabric-ccenv      x86_64-1.1.0        c8b4909d8d46
About an hour ago   1.39GB
......
```

使用三个终端

终端1 启动网络

启动网络

```
$ sudo docker-compose -f docker-compose-simple.yaml up -d
```

上面的命令以`SingleSampleMSPSolo`orderer配置文件启动网络，并以“dev模式”启动对等体。它还启动了两个额外的容器：一个用于chaincode环境，一个用于与chaincode交互的CLI。创建和加入通道的命令被嵌入到CLI容器中，因此可以立即跳转到链式代码调用

终端2 建立并启动链码

打开第二个终端, 进入到`chaincode-docker-devmode` 目录

```
$ cd ~/hyfa/fabric-samples/chaincode-docker-devmode/
```

进入

```
$ sudo docker exec -it chaincode bash
```

命令提示符变为:

`root@858726aed16e:/opt/gopath/src/chaincode#`

编译

进入test目录编译chaincode

```
root@858726aed16e:/opt/gopath/src/chaincode#  cd test
root@858726aed16e:/opt/gopath/src/chaincode/test#  go build
```

运行chaincode

```
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=test:0 ./test
```

终端3 使用链码

```
$ sudo docker exec -it cli bash
```

安装及实例化

进入CLI容器后执行如下命令安装及实例化chaincode

```
peer chaincode install -p chaincodedev/chaincode/test -n test -v 0
peer chaincode instantiate -n test -v 0 -c '{"Args":["a","10"]}' -C
myc
```

调用

进行调用,将`a`的值更改为`20`

```
peer chaincode invoke -n test -c '{"Args":["set", "a", "20"]}' -C
myc
```

执行成功, 输出如下内容

```
......
...... Chaincode invoke successful. result: status:200 payload:"20"
......
```

查询

查询`a`的值

```
peer chaincode query -n test -c '{"Args":["query","a"]}' -C myc
```

执行成功, 输出: `Query Result: 20`


区块链部落
专注于区块链技术
识别图中二维码关注我们