链码的概念与使用

概念:

Chaincode: 链上代码, 简称链码, 一般是指开发人员编写的应用代码

链码被部署在Fabric网络节点上,

只能被部署在Peer节点上

运行在隔离沙盒(当前为Docker容器)中,

并通过gRPC协议与相应的Peer节点进行交互, 以操作分布式账本中的数据

在Fabric网络环境中各节点进行通信时

protobuf: 将字符转换为字节(对数据进行序列化与反序列化)

分类

链码一般分为:

- 用户链码
- 系统链码

用户链码

由应用开发人员使用Go(Java/JS)语言编写基于区块链分布式账本的状态及处理逻辑

运行在链码容器中, 通过Fabric提供的接口与账本平台进行交互

系统链码

负责Fabric节点自身的处理逻辑, 包括系统配置、背书、校验等工作

系统链码仅支持Go语言, 在Peer节点启动时会自动完成注册和部署

系统链码共有五种类型:

配置系统链码(CSCC)

CSCC: Configuration System Chaincode

负责账本和链的配置管理

背书管理系统链码(ESCC)

ESCC: Endorsement System Chaincode

负责背书(签名)过程, 并可以支持对背书策略进行管理

> 对传入的链码提案的模拟运行结果进行签名, 之后创建响应消息返回给客户端

生命周期系统链码(LSCC)

LSCC: Lifecycle System Chaincode

负责对用户链码的生命周期进行管理

链码生命周期包括安装、部署、升级、权限管理、获取信息等环节.

查询系统链码(QSCC)

QSCC: Query System Chaincode

负责提供账本和链的信息查询功能

验证系统链码(VSCC)

VSCC: Verification System Chaincode

交易提交前根据背书策略进行检查

验证过程:

1. 首先解析出交易结构, 并对交易结构格式进行校验
2. 检查交易的读集合中元素版本跟本地账本中版本一致
3. 检查带有合法的背书信息(主要是检查签名信息)
4. 通过则返回正确, 否则返回错误消息

链码生命周期

管理Chaincode的生命周期四个命令：

安装, 实例化, 升级, 打包, 签名

`package` ， `install` ， `instantiate` ， `upgrade`

未来还会支持`stop`和`start`命令, 来禁用和重新启用链代码

链代码成功安装和实例化后，链代码处于活动状态（正在运行），可通过`invoke`命令调用处理事务

链代码可以在安装后随时升级

安装链码

install命令将链码的源码和环境等内容封装为一个链码安装打包文件(Chaincode Install Package, CIP), 并传输到背书节点.

背书节点解析后一般会保存在$CORE_PEER_FILESYSTEMPATH/chaincodes目录下

安装链码只需要与Peer交互

进入到sacc目录

```
$ cd ~/hyfa/fabric-samples/chaincode/sacc/
```

构建Chaincode

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

$ go build --tags nopkcs11
```

命令解释

go get: 根据要求和实际情况从互联网上下载或更新指定的代码包及其依赖包，并进行编译和安装

-u: 利用网络来更新已有代码包及其依赖包。默认情况下，该命令只会从网络上下载本地不存在的代码包，而不会更新已有的代码包

go build: 加上可编译的go源文件可以得到一个可执行文件

如果在执行 `go build --tags nopkcsll` 命令时出现如下错误

```
$GOPATH/src/github.com/hyperledger/fabric/vendor/github.com/miekg/pk
cs11/pkcs11.go:26:18: fatal error: ltdl.h: No such file or directory
compilation terminated.

解决方式: 安装 libltdl-dev
$ sudo apt install libltdl-dev
```

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

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

> 该目录下有 `myc.tx` 文件

启动网络

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

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CLI

chaincode容器的作用是为发简化的方式建立并启动链码

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

> 命令提示符变为:
>
> `root@858726aed16e:/opt/gopath/src/chaincode#`

编译

进入sacc目录编译chaincode

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

运行chaincode

```
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:0 ./sacc
```

mycc: 链码名称

0: 链码初始版本号

终端3 使用链码

安装及实例化

```
$ cd ~/hyfa/fabric-samples/chaincode-docker-devmode/
$ sudo docker exec -it cli bash
```

进入CLI容器后执行如下命令安装及实例化chaincode

> **容器中已经有 `myc.block` 、 `myc.tx` 两个文件， 可以直接使用**

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

执行成功, 输出: `Query Result: 20`

打包链码及签名

通过将链码相关数据进行封装, 可以实现对其进行打包和签名操作

打包

```
peer chaincode package -n mycc -p chaincodedev/chaincode/sacc -v 0 -
s -S -i "AND('OrgA.admin')" ccpack.out
```

> -s: 创建角色支持的CC部署规范包, 而不是原始的CC部署规范
>
> -S: 如果创建CC部署规范方案角色支持,也与本地MSP签名
>
> -i: 指定实例化策略

打包后的文件, 可以直接用于install操作, 如:

`peer chaincode install ccpack.out`

签名

对一个打包文件进行签名操作(添加当前MSP签名到签名列表中)

```
peer chaincode signpackage ccpack.out signedccpack.out
```

升级链码

退出终端3, 停止终端2的服务

终端2中重新运行chaincode

```
$ sudo docker exec -it chaincode bash

cd sacc
go build
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:1 ./sacc
```

注意版本为 `1.0`, 旧版本为 `0`

终端3

打开终端3, 进入CLI容器

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

安装及升级

```
peer chaincode install -p chaincodedev/chaincode/sacc -n mycc -v 1

peer chaincode upgrade -n mycc -v 1 -c '{"Args":["a", "100"]}' -C
myc
```

注意版本号必须一致

在对某链码代码升级前, 推荐先将所有该链码的容器停止, 并从Peer上备份并移除旧链码部署文件. 之后先在个别Peer节点上部署新链码, 对原有数据进行测试, 成功后再在其它节点上进行升级操作

查询

查询`a`的值

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

执行成功, 输出: `Query Result:100`


区块链部落
专注于区块链技术
识别图中二维码关注我们