链码开发

链码源文件的必要结构

```
package main

// 引入必要的包
import (
  "github.com/hyperledger/fabric/core/chaincode/shim"
  "github.com/hyperledger/fabric/protos/peer"
  "fmt"

)
```

实例1

> 链码实现的 Hello World
>
> 链码在实例化时向账本保存一个初始数据，key 为 Hello， value 为 World，然后用户发出查询请求，可以根据 key 查询到相应的 value

链码开发

1. 创建文件夹

   ```
   cd hyperledger/fabric-samples/chaincode
   mkdir hello
   cd hello
   ```

2. 创建并编辑链码文件

   ```
   sudo vim hello.go
   ```

3. 导入链码依赖包

   ```
   package main

   import (
     "github.com/hyperledger/fabric/core/chaincode/shim"
     "github.com/hyperledger/fabric/protos/peer"
     "fmt"
   )
   ```

4. 编写主函数

```
func main() {
  err := shim.Start(new(HelloChaincode))
  if err != nil {
     fmt.Printf("链码启动失败 : %v", err)
  }
}
```

5. 自定义结构体

```
type HelloChaincode struct { }
```

6. 实现 Chaincode 接口（接口必须重写 **Init** 与 **Invoke** 两个函数）

   Init 函数

```
//实例化/升级链码时被自动调用
//-c '{"Args":["Hello","World"]}'
func (t *HelloChaincode) Init(stub shim.ChaincodeStubInterface)
peer.Response {
    fmt.Println(" 开始实例化链码......")
    //获取参数
    //args := stub.GetStringArgs()
    _, args := stub.GetFunctionAndParameters()
    //判断参数长度是否为2个
    if len(args) != 2 {
        return shim.Error(" 指定了错误的参数个数 ")
    }
    fmt.Println(" 保存数据......")

    //通过调用 PutState 函数将数据保存在账本中
    err := stub.PutState(args[0], []byte(args[1]))
    if err != nil {
        return shim.Error(" 保存数据时发生错误......")
    }
    fmt.Println(" 实例化链码成功 ")
    return shim.Success(nil)
}
```

   Invoke 函数

```
//对账本数据进行操作时被自动调用 （query, invoke）
func (t *HelloChaincode) Invoke(stub shim.ChaincodeStubInterface)
peer.Response {
    //获取调用链码时传递的参数内容（包括要调用的函数名及参数）
    fun, args := stub.GetFunctionAndParameters()
    //客户意图
    if fun == "query"{
        return query(stub, args)
    }
    return shim.Error(" 非法操作，指定功能不能实现 ")
}
```

   实现查询函数

```
func query(stub shim.ChaincodeStubInterface, args []string) peer.Response {
    //检查传递的参数个数是否为 1
    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 ")
    }
    //根据指定的 Key 调用 GetState 方法查询数据
    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 ")
    }
    //返回查询结果
    return shim.Success(result)
}
```

链码测试

1. 启动网络

   进入 fabric-samples/chaincode-docker-devmode/ 目录

   ```
   cd ~/go/src/github.com/hyperledger/fabric-samples/chaincode-docker-devmode/
   ```

   启动网络

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

   > 若出现错误应该是之前 docker-compose 未关闭，环境不干净
   >
   > ```
   > docker-compose down -v
   > docker rm $(docker ps -aq)
   > docker rmi $(docker images dev-* -q)
   > docker system prune --volumes
   > ```
   >
   > 若出现
   >
   > ERROR: “...” You have to remove (or rename) that container to be able to reuse that name.
   >
   > 同样是之前开启了同名的 docker
   >
   > ```
   > docker ps -qa | xargs docker rm
   > ```
   >
   > 然后重新启动网络
   >
   > ```
   > docker-compose -f docker-compose-simple.yaml up
   > ```

2. 启动链码

   1. 打开一个新的终端 2 窗口，进入链码 chaincode 容器

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

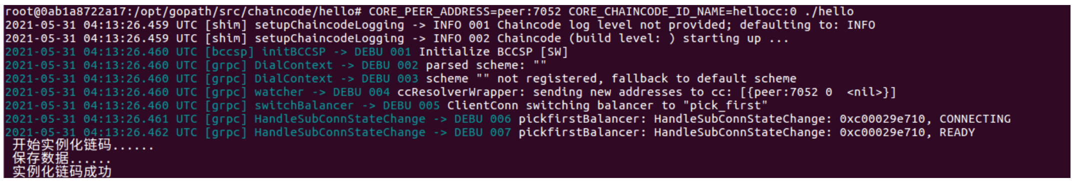
   2. 编译链码

```
cd hello
go build
```

3. 启动链码

```
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=hellocc:0 ./hello
```

命令执行后输出如下信息：



```
root@0ab1a8722a17:/opt/gopath/src/chaincode/hello# CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=hellocc:0 ./hello
2021-05-31 04:13:26.459 UTC [shim] setupChaincodeLogging -> INFO 001 Chaincode log level not provided; defaulting to: INFO
2021-05-31 04:13:26.459 UTC [shim] setupChaincodeLogging -> INFO 002 Chaincode (build level: ) starting up ...
2021-05-31 04:13:26.460 UTC [bccsp] initBCCSP -> DEBU 001 Initialize BCCSP [SW]
2021-05-31 04:13:26.460 UTC [grpc] DialContext -> DEBU 002 parsed scheme: ""
2021-05-31 04:13:26.460 UTC [grpc] DialContext -> DEBU 003 scheme "" not registered, fallback to default scheme
2021-05-31 04:13:26.460 UTC [grpc] watcher -> DEBU 004 ccResolverWrapper: sending new addresses to cc: [{peer:7052 0  <nil>}]
2021-05-31 04:13:26.460 UTC [grpc] switchBalancer -> DEBU 005 ClientConn switching balancer to "pick_first"
2021-05-31 04:13:26.461 UTC [grpc] HandleSubConnStateChange -> DEBU 006 pickfirstBalancer: HandleSubConnStateChange: 0xc00029e710, CONNECTING
2021-05-31 04:13:26.462 UTC [grpc] HandleSubConnStateChange -> DEBU 007 pickfirstBalancer: HandleSubConnStateChange: 0xc00029e710, READY
 开始实例化链码......
 保存数据......
 实例化链码成功
```

```
[shim] SetupChaincodeLogging -> INFO 001 Chaincode log level not
provided;
    defaulting to: INFO
[shim] SetupChaincodeLogging -> INFO 002 Chaincode (build level: )
starting up ...
```

3. 测试链码
    1. 打开一个新的终端 3 窗口，进入 CLI 容器

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

    2. 安装链码

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

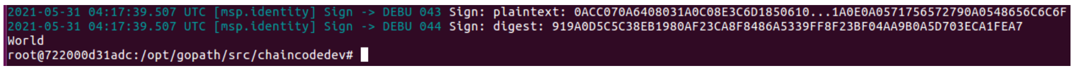
    3. 实例化链码

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

    4. 调用链码

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

    返回查询结果：World



```
2021-05-31 04:17:39.507 UTC [msp.identity] Sign -> DEBU 043 Sign: plaintext: 0ACC070A6408031A0C08E3C6D1850610...1A0E0A0571756572790A0548656C6C6F
2021-05-31 04:17:39.507 UTC [msp.identity] Sign -> DEBU 044 Sign: digest: 919A0D5C5C38EB1980AF23CA8F8486A5339FF8F23BF04AA9B0A5D703ECA1FEA7
World
root@722000d31adc:/opt/gopath/src/chaincodedev# 
```