Hyperledger Fabric 链码开发实战——弹珠资产管理(Marbles)

1、本篇背景

Marbles是IBM开源的区块链项目，基本网络是Hyperledger Fabric，GitHub地址如下：

```
https://github.com/IBM-Blockchain/marbles
```

有对应的中文翻译版，这里就不再详细介绍了。Marbles是一个非常简单的资产转移演示，多个用户可以创建并相互转移弹珠。

这里不作项目介绍和安装内容，本篇内容主要是参考Marbles项目进行链码的开发和测试，所以需要对Marbles有基本的了解。

假设我们从零开始用链码开发简单的弹珠资产转移功能，实现如下几个业务需求：

1. 创建一个弹珠，弹珠字段包括弹珠ID、颜色、尺寸和拥有者；
2. 通过弹珠ID查询对应的弹珠信息；
3. 通过弹珠ID删除对应的弹珠；
4. 改变弹珠的拥有者；
5. 查询指定ID范围的弹珠信息；
6. 查询某个拥有者的所有弹珠信息；
7. 通过弹珠ID查询所有的交易历史信息。

2、链码开发

了解了上面的业务需求后，我们直接进入链码的开发。

先创建一个用于保存本次实验的项目文件夹，这里命名为"marbles_chaincode"，在该目录下创建一个与文件夹同名的后缀名为".go"链码文件。

2.1 创建结构体

先创建"弹珠"以及链码的结构体

```
// 定义"弹珠"链码的结构体
type MarblesChaincode struct {
}

/*
 * 定义"弹珠"结构体
```

```
 * 字段包括ID、颜色、尺寸和拥有者等
 */
type Marble struct {
  ObjectType string `json:"objectType"` // CouchDB的字段
  ID string `json:"id"` // ID（唯一字符串，将用作键）
  Color string `json:"color"` // 颜色（字符串，CSS颜色名称）
  Size int `json:"size"`  //  尺寸（整型，以毫米为单位）
  Owner string `json:"owner"` //  拥有者（字符串）
}
```

2.2 实现Init函数和main函数

这里先使用实现Init函数和main函数，Invoke函数和其他跟业务有关的功能后面一起实现。

```
// 在链码初始化过程中调用Init来初始化任何数据
func (t *MarblesChaincode) Init (stub shim.ChaincodeStubInterface) pb.Respon
se {
  fmt.Println("Marbles Init Success")
  return shim.Success(nil)
}

// 后面实现Invoke函数和其他功能
func (t *MarblesChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Respo
nse {
  fmt.Println("Marbles Invoke")
    // ...
    return shim.Error("没找到对应方法~")
}

// Go语言的入口是main函数
func main() {
  err := shim.Start(new(MarblesChaincode))
  if err != nil {
    fmt.Printf("Error starting Marbles Chaincode - %s", err)
  }
}
```

2.3 初始化弹珠

初始化创建一个弹珠时，需要根据弹珠ID判断是否已经存在账本中，不存在则将通过传入的参数来创建Marble对象，然后转换为Json格式的内容写入到账本中。

```
func (t *MarblesChaincode) initMarble (stub shim.ChaincodeStubInterface, arg
s []string) pb.Response {
  if len(args) != 4 {
```

```
    return shim.Error("Incorrect number of arguments. Expecting 4")
  }

  marbleObjectType := "marble"
  marbleId := args[0]
  // 查询账本中是否已经存在该弹珠ID的信息
  marbleIdAsBytes, err := stub.GetState(marbleId)
  if err != nil {
    return shim.Error(err.Error())
  }

  if marbleIdAsBytes != nil {
    return shim.Error("该Marble已存在~")
  }

  marbleColor := args[1]
  // 字符串转换为整型
  marbleSize, err := strconv.Atoi(args[2])
  if err != nil {
    return shim.Error("size字段为整型~")
  }
  marbleOwner := args[3]
  // 创建Marble对象
  marble := &Marble{marbleObjectType, marbleId, marbleColor, marbleSize, mar
bleOwner}
  // 转换为Json格式的内容
  marbleJsonAsBytes, err := json.Marshal(marble)
  // 写入到账本中
  err = stub.PutState(marbleId, marbleJsonAsBytes)
  if err != nil {
    return shim.Error(err.Error())
  }

  return shim.Success(nil)
}
```

2.4 通过弹珠ID查询对应的弹珠信息

通过shim包的 `GetState` API方法来获取账本中的弹珠信息

```
// 通过弹珠ID查询对应的弹珠信息
func (t *MarblesChaincode) getMarbleInfoByID (stub shim.ChaincodeStubInterfa
ce, args []string) pb.Response {
  if len(args) != 1 {
    return shim.Error("Incorrect number of arguments. Expecting 1")
  }
```

```
  marbleId := args[0]
  // 查询账本中是否已经存在该弹珠ID的信息
  marbleIdAsBytes, err := stub.GetState(marbleId)

  if err != nil {
    return shim.Error(err.Error())
  } else if marbleIdAsBytes == nil {
    return shim.Error("该Marble不存在~")
  }

  return shim.Success(marbleIdAsBytes)
}
```

2.5 通过弹珠ID删除对应的弹珠

先通过 `GetState` 判断账本中是否存在对应弹珠ID的信息，存在则通过 `DelState` 删除账本记录。

```
// 通过弹珠ID删除对应的弹珠
func (t *MarblesChaincode) deleteMarbleByID (stub shim.ChaincodeStubInterfac
e, args []string) pb.Response {
  if len(args) != 1 {
    return shim.Error("Incorrect number of arguments. Expecting 1")
  }

  marbleId := args[0]
  // 查询账本中是否已经存在该弹珠ID的信息
  marbleIdAsBytes, err := stub.GetState(marbleId)

  if err != nil {
    return shim.Error(err.Error())
  } else if marbleIdAsBytes == nil {
    return shim.Error("该Marble不存在~")
  }

  // 删除账本中的记录
  err = stub.DelState(marbleId)
  if err != nil {
    return shim.Error(err.Error())
  }

  return shim.Success(nil)
}
```

2.6 改变弹珠的拥有者

先通过 `GetState` 判断账本中是否存在对应弹珠ID的信息，存在则读取账本中Json格式的弹珠信息转换为 `Marble` 对象，修改 `Marble` 对象中的拥有者 `Owner` 字段，最后再转换为Json格式的内容通过 `PutState` 写入到账本中。

```
// 改变弹珠的拥有者
func (t *MarblesChaincode) changeMarbleOwner (stub shim.ChaincodeStubInterfa
ce, args []string) pb.Response {
  if len(args) != 2 {
        return shim.Error("Incorrect number of arguments. Expecting 2")
    }

  marbleId := args[0]
  newOwner := args[1]

  // 查询账本中是否已经存在该弹珠ID的信息
  marbleIdAsBytes, err := stub.GetState(marbleId)

  if err != nil {
    return shim.Error(err.Error())
  } else if marbleIdAsBytes == nil {
    return shim.Error("该Marble不存在~")
  }

  marble := Marble{}
  err = json.Unmarshal(marbleIdAsBytes, &marble)
  if err != nil {
    return shim.Error(err.Error())
  }
  marble.Owner = newOwner
  marbleJsonAsBytes, err := json.Marshal(marble)
  if err != nil {
    return shim.Error(err.Error())
  }

  // 写入到账本中
  err = stub.PutState(marbleId, marbleJsonAsBytes)
  if err != nil {
    return shim.Error(err.Error())
  }

  return shim.Success(nil)
}
```

2.7 查询指定ID范围的弹珠信息

通过 `GetStateByRange` 方法查询指定范围内的键值（即所有弹珠信息），然后遍历所有键值

（即所有弹珠信息），转换为Json格式的字符串数组并返回。

```
// 查询指定ID范围的弹珠信息
func (t *MarblesChaincode) getMarbleByRange (stub shim.ChaincodeStubInterfac
e, args []string) pb.Response {
  if len(args) != 2 {
        return shim.Error("Incorrect number of arguments. Expecting 2")
    }

  startKey := args[0]
  endKey := args[1]

  // 查询指定范围内的键值
  resultsIterator, err := stub.GetStateByRange(startKey, endKey)
  if err != nil {
    return shim.Error(err.Error())
  }
  defer resultsIterator.Close()

  var buffer bytes.Buffer
  buffer.WriteString("[")

  bArrayMemberAlreadyWritten := false
  // 遍历弹珠信息，转换为Json格式的字符串数组并返回
  for resultsIterator.HasNext() {
    queryResult, err := resultsIterator.Next()
    if err != nil {
          return shim.Error(err.Error())
        }

    if bArrayMemberAlreadyWritten == true {
      buffer.WriteString(",")
    }

    buffer.WriteString("{\"Key\":")
        buffer.WriteString("\"")
        buffer.WriteString(queryResult.Key)
        buffer.WriteString("\"")

    buffer.WriteString(", \"Record\":")
        // Record is a JSON object, so we write as-is
        buffer.WriteString(string(queryResult.Value))
        buffer.WriteString("}")
        bArrayMemberAlreadyWritten = true
  }
  buffer.WriteString("]")
  fmt.Printf("- getMarblesByRange queryResult:\n%s\n", buffer.String())
```

```
  return shim.Success(buffer.Bytes())
}
```

2.8 查询某个拥有者的所有弹珠信息

通过 `GetQueryResult` 方法对（支持富查询功能的）状态数据库（State DB）进行富查询(rich query)，目前仅有CouchDB类型的状态数据库支持富查询。

```
// 查询某个拥有者的所有弹珠信息
func (t *MarblesChaincode) getMarbleByOwner(stub shim.ChaincodeStubInterface
, args []string) pb.Response {
  marbleOwner := args[0]
  queryStr := fmt.Sprintf("{\"selector\":{\"owner\":\"%s\"}}",marbleOwner)

  resultsIterator, err := stub.GetQueryResult(queryStr)
  if err != nil {
            return shim.Error(err.Error())
    }

  defer resultsIterator.Close()

  var buffer bytes.Buffer
  buffer.WriteString("[")

  bArrayMemberAlreadyWritten := false
  for resultsIterator.HasNext() {
    queryResult, err := resultsIterator.Next()
    if err != nil {
            return shim.Error(err.Error())
        }

    if bArrayMemberAlreadyWritten == true {
      buffer.WriteString(",")
    }

    buffer.WriteString("{\"Key\":")
        buffer.WriteString("\"")
        buffer.WriteString(queryResult.Key)
        buffer.WriteString("\"")

    buffer.WriteString(", \"Record\":")
        buffer.WriteString(string(queryResult.Value))
        buffer.WriteString("}")
        bArrayMemberAlreadyWritten = true
  }
  buffer.WriteString("]")
```

```
  fmt.Printf("- getMarbleByOwner queryResult:\n%s\n", buffer.String())
  return shim.Success(buffer.Bytes())
}
```

2.9 通过弹珠ID查询所有的交易历史信息

通过 `GetHistoryForKey` 方法查询某个键（即对应弹珠ID）的所有历史值。

```
// 通过弹珠ID查询所有的交易历史信息
func (t *MarblesChaincode) getMarbleHistoryForKey (stub shim.ChaincodeStubInterface, args []string) pb.Response {
  if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }

  marbleId := args[0]

  // 返回某个键的所有历史值
  resultsIterator, err := stub.GetHistoryForKey(marbleId)
  if err != nil {
    return shim.Error(err.Error())
  }

  defer resultsIterator.Close()

  var buffer bytes.Buffer
  buffer.WriteString("[")

  bArrayMemberAlreadyWritten := false
  for resultsIterator.HasNext() {
    queryResult, err := resultsIterator.Next()
    if err != nil {
          return shim.Error(err.Error())
        }

    if bArrayMemberAlreadyWritten == true {
      buffer.WriteString(",")
    }

    buffer.WriteString("{\"TxId\":")
        buffer.WriteString("\"")
        buffer.WriteString(queryResult.TxId)
        buffer.WriteString("\"")

    buffer.WriteString(", \"Timestamp\":")
    buffer.WriteString("\"")
        buffer.WriteString(time.Unix(queryResult.Timestamp.Seconds, int64(qu
```

```
eryResult.Timestamp.Nanos)).String())
        buffer.WriteString("\"")

    buffer.WriteString("{\"Value\":")
        buffer.WriteString("\"")
        buffer.WriteString(string(queryResult.Value))
        buffer.WriteString("\"")

    buffer.WriteString("{\"IsDelete\":")
        buffer.WriteString("\"")
        buffer.WriteString(strconv.FormatBool(queryResult.IsDelete))
        buffer.WriteString("\"")

        bArrayMemberAlreadyWritten = true
  }
  buffer.WriteString("]")
  fmt.Printf("- getMarblesByRange queryResult:\n%s\n", buffer.String())
  return shim.Success(buffer.Bytes())
}
```

2.10 实现Invoke函数

```
// 在链码每个交易中，Invoke函数会被调用。
func (t *MarblesChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Respo
nse {
  fmt.Println("Marbles Invoke")
  function, args := stub.GetFunctionAndParameters()
  if function == "initMarble" {
    return t.initMarble(stub, args)
  } else if function == "getMarbleInfoByID" {
    return t.getMarbleInfoByID(stub, args)
  } else if function == "deleteMarbleByID" {
    return t.deleteMarbleByID(stub, args)
  } else if function == "changeMarbleOwner" {
    return t.changeMarbleOwner(stub, args)
  } else if function == "getMarbleByRange" {
    return t.getMarbleByRange(stub, args)
  } else if function == "getMarbleByOwner" {
    return t.getMarbleByOwner(stub, args)
  } else if function == "getMarbleHistoryForKey" {
    return t.getMarbleHistoryForKey(stub, args)
  }
  return shim.Error("没找到对应方法~")
}
```

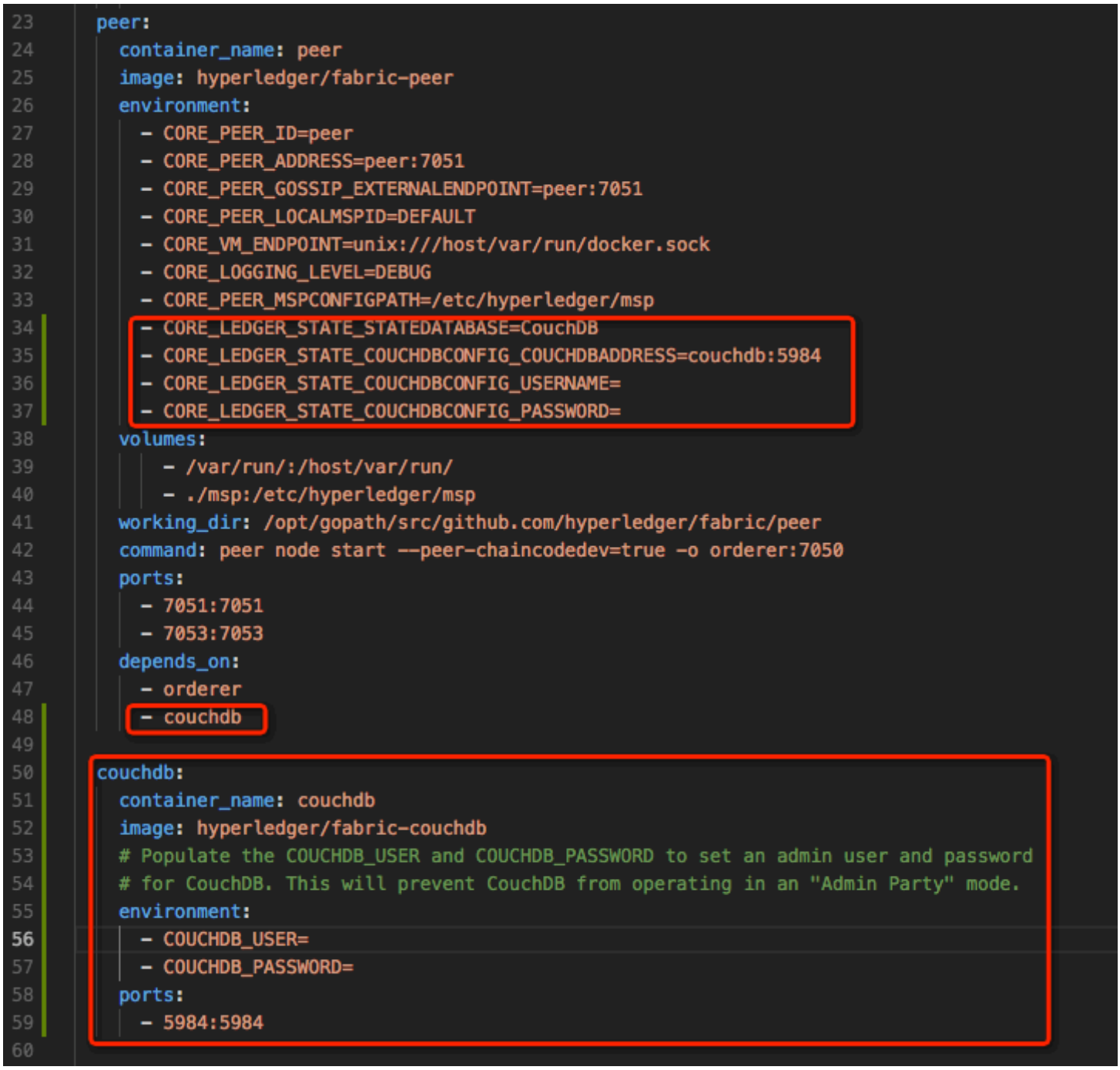
3、富查询配置CouchDB

富查询需要配置CouchDB，进入到 `fabric-samples/chaincode-docker-devmode` 目录，找到 `docker-compose-simple.yaml` 配置文件进行修改，修改格式如下：

```
services:
    peer:
        environment:
            ...
            - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp
            ###以下是添加的内容
            - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp
            - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
            - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb:5984
            - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
            - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=

        depends_on:
            - orderer
            ###以下是添加的内容
            - couchdb

    couchdb:
        container_name: couchdb
        image: hyperledger/fabric-couchdb
        # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin use
r and password
        # for CouchDB. This will prevent CouchDB from operating in an "Admin
 Party" mode.
        environment:
            - COUCHDB_USER=
            - COUCHDB_PASSWORD=
        ports:
            - 5984:5984
```

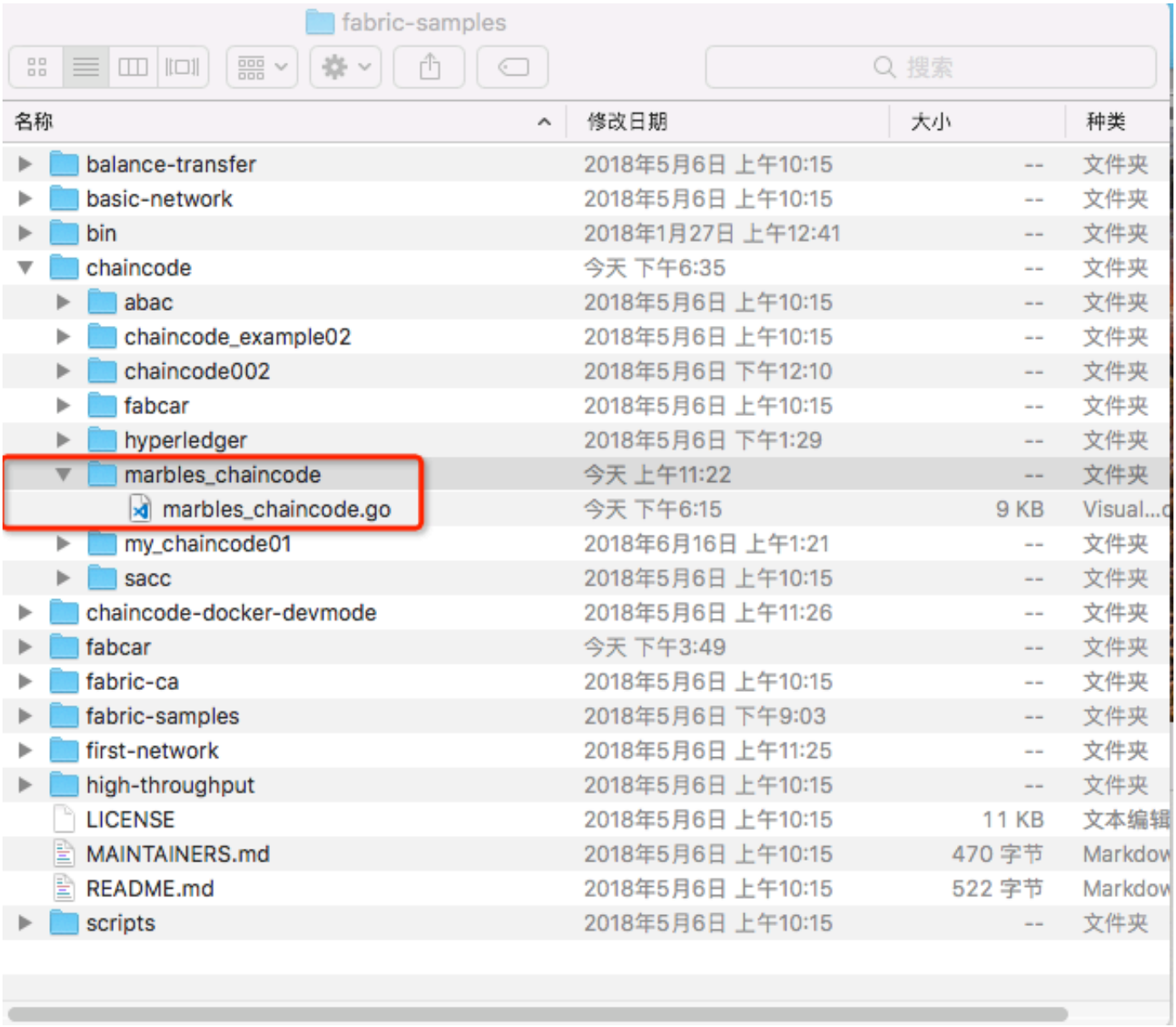


```
peer:
  container_name: peer
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ID=peer
    - CORE_PEER_ADDRESS=peer:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer:7051
    - CORE_PEER_LOCALMSPID=DEFAULT
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_LOGGING_LEVEL=DEBUG
    - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp
    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb:5984
    - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
    - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=
  volumes:
      - /var/run/:/host/var/run/
      - ./msp:/etc/hyperledger/msp
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
  command: peer node start --peer-chaincodedev=true -o orderer:7050
  ports:
    - 7051:7051
    - 7053:7053
  depends_on:
    - orderer
    - couchdb

couchdb:
  container_name: couchdb
  image: hyperledger/fabric-couchdb
  # Populate the COUCHDB_USER and COUCHDB_PASSWORD to set an admin user and password
  # for CouchDB. This will prevent CouchDB from operating in an "Admin Party" mode.
  environment:
    - COUCHDB_USER=
    - COUCHDB_PASSWORD=
  ports:
    - 5984:5984
```

4、编译和调用链码

确保您搭建并配置好了Hyperledger Fabric的开发环境，我们把上面创建的链码文件夹复制到 `fabric-samples/chaincode` 目录下。



同时，开启三个终端，确保终端进入到"fabric-samples/chaincode-docker-devmode"目录下。

4.1 终端1 - 开启网络

```
###删除所有活跃的容器###
docker rm -f $(docker ps -aq)
###清理网络缓存###
docker network prune
###开启网络###
docker-compose -f docker-compose-simple.yaml up
```

4.1 终端2 - 编译和运行链码

```
###进入Docker容器cli###
docker exec -it chaincode bash
###进入到链码对应目录###
cd marbles_chaincode
```

```
###编译链码###
go build
###启动Peer节点运行链码###
CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=mycc:0 ./marbles_chaincod
e
###如果失败，把"7052"改为"7051"试试看
```

4.3 终端3 - 安装和调用链码

1、启动Docker CLI容器：

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

2、安装和实例化链码：

```
peer chaincode install -p chaincodedev/chaincode/marbles_chaincode -n mycc -
v 0

peer chaincode instantiate -n mycc -v 0 -c '{"Args":[]}' -C myc
```

3、初始化弹珠：

分别创建三个弹珠

```
peer chaincode invoke -n mycc -c '{"Args":["initMarble","1","red","20","Jack
"]}' -C myc
peer chaincode invoke -n mycc -c '{"Args":["initMarble","2","green","66","We
nzil"]}' -C myc
peer chaincode invoke -n mycc -c '{"Args":["initMarble","3","blue","88","Li
Lei"]}' -C myc
```

4、通过弹珠ID查询对应的弹珠信息：

```
peer chaincode invoke -n mycc -c '{"Args":["getMarbleInfoByID","1"]}' -C myc
```

5、改变弹珠的拥有者：

```
peer chaincode invoke -n mycc -c '{"Args":["changeMarbleOwner","1","Han Meim
ei"]}' -C myc
```

6、查询改变拥有者后的弹珠信息：

```
peer chaincode invoke -n mycc -c '{"Args":["getMarbleInfoByID","1"]}' -C myc
```

7、通过弹珠ID删除对应的弹珠：

```
peer chaincode invoke -n mycc -c '{"Args":["deleteMarbleByID","1"]}' -C myc
```

8、查询指定ID范围的弹珠信息：

```
peer chaincode invoke -n mycc -c '{"Args":["getMarbleByRange","2", "3"]}' -C
 myc
```

9、查询某个拥有者的所有弹珠信息：

```
peer chaincode invoke -n mycc -c '{"Args":["getMarbleByOwner","Wenzil"]}' -C
 myc
```

10、通过弹珠ID查询所有的交易历史信息：

```
peer chaincode invoke -n mycc -c '{"Args":["getMarbleHistoryForKey","1"]}' -
C myc
```