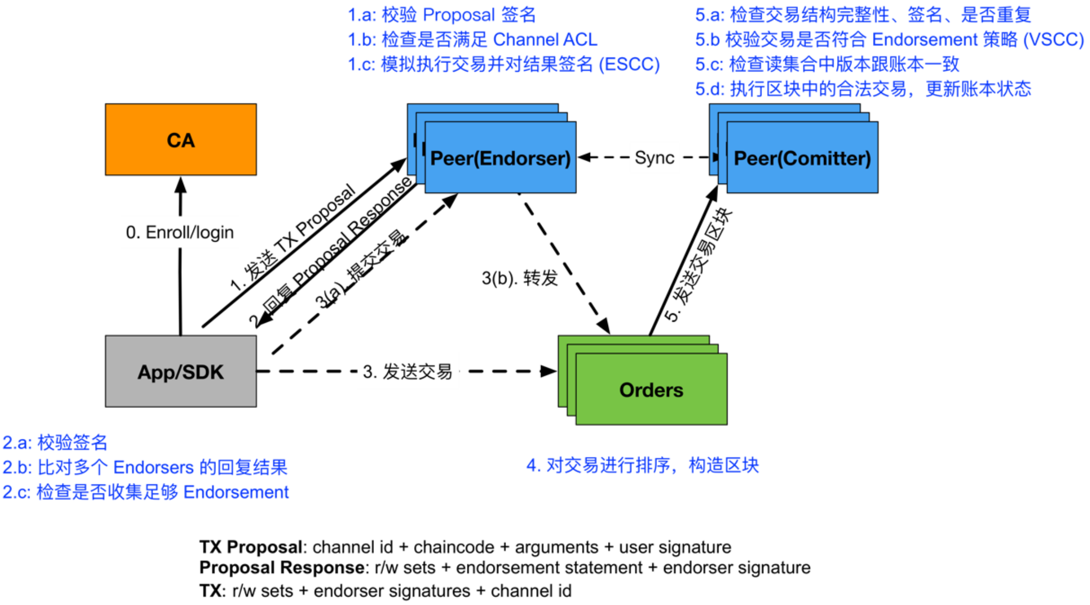
Hyperledger Fabric启用CouchDB为状态数据库

一.概述

1. 数据请求流

超级账本采用背书/共识模型，模拟执行和区块验证是在不同角色的节点中分开执行的。模拟执行是并发的，这样可以提高扩展性和吞吐量：

- 背书节点：模拟执行链码
- Peer节点：验证交易并提交



1.a: 校验 Proposal 签名
1.b: 检查是否满足 Channel ACL
1.c: 模拟执行交易并对结果签名 (ESCC)

5.a: 检查交易结构完整性、签名、是否重复
5.b 校验交易是否符合 Endorsement 策略 (VSCC)
5.c: 检查读集合中版本跟账本一致
5.d: 执行区块中的合法交易，更新账本状态

2.a: 校验签名
2.b: 比对多个 Endorsers 的回复结果
2.c: 检查是否收集足够 Endorsement

4. 对交易进行排序，构造区块

TX Proposal: channel id + chaincode + arguments + user signature
Proposal Response: r/w sets + endorsement statement + endorser signature
TX: r/w sets + endorser signatures + channel id

2.超级账本存储元素

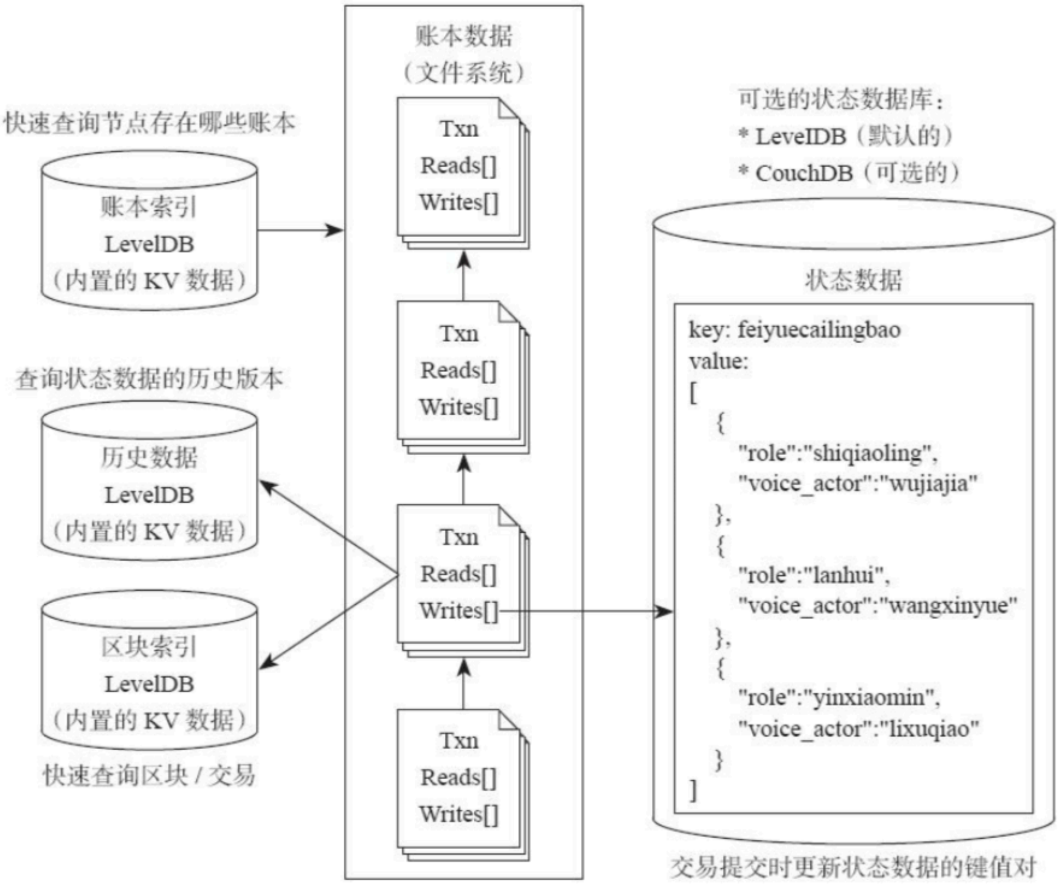
超级账本包含以下元素:

- 账本编号:快速查询存在哪些账本
- 账本数据: 实际的区块数据存储
- 区块索引: 快速查询区块/交易
- 状态数据: 最新的世界状态数据
- 历史数据: 跟踪键的历史

每个Peer节点会维护四个DB，分别为:

- 账本索引库(IdStore)：存储ChainID
- 状态数据库(StateDB): 存储world state
- 历史数据库(HistoryDB): 存储Key的版本变化

- 区块索引库(BlockIndex):存储Block索引



3.状态数据库

状态数据库可选类型包括LevelDB和CouchDB。LevelDB是嵌入在peer进程中的默认键/值状态数据库，CouchDB是一个可选的外部状态数据库。与LevelDB键/值存储一样，CouchDB可以存储任何以chaincode建模的二进制数据（CouchDB附件函数在内部用于非json二进制数据）。但是，当chaincode值（例如，资产）被建模为JSON数据时，作为JSON文档存储，CouchDB支持对chaincode数据进行丰富的查询。

LevelDB和CouchDB都支持核心chaincode操作，例如获取和设置一个键（资产），并根据键进行查询。键可以通过范围查询，可以对组合键进行建模，以支持针对多个参数的等价查询。例如，作为所有者的组合键，资产id可以用于查询某个实体拥有的所有资产。这些基于key的查询可以用于针对账本的只读查询，以及更新总账的事务。

如果将资产建模为JSON并使用CouchDB，那么就可以使用chaincode中的CouchDB JSON查询语言对chaincode数据值执行复杂的富查询，这些类型的查询对于理解账本上的内容很有帮助。对于这些类型的查询，事务协议响应通常对客户端应用程序有用，但通常不会作为事务提交到排序服务。事实上，也无法保证结果集在chaincode执行与富查询提交时间之间的稳定性，因此使用富查询的结果去执行最终的事务更新操作是不合适的，除非可以保证结果集在chaincode执行时间与提交时间之间的稳定性，或者可以处理在后续交易中的潜在变化。例如，如果对Alice所拥有的所有资产执行一个富查询并将其传输给Bob，那么一个新的资产可能会被另一个事务分配给Alice，这是在chaincode执行时间和提交时间之间的另一个事务，可能此过程中会错过这个“虚值”。

CouchDB作为一个独立的数据库进程与peer一起运行，因此在设置、管理和操作方面有额外的考虑。我们可以考虑从默认的嵌入式LevelDB开始，如果需要额外的复杂的富查询，可以转移到CouchDB。将chaincode资产数据建模为JSON是一种很好的做法，这样我们就可以在将来执行需要的复杂的富查询。

二. 启用CouchDB

本文均采用Hyperledger Fabric1.2中fabric-samples中相关组件与资源，在测试环境(fabric-samples/chaincode-docker-devmode)通过Docker启动CouchDB服务

1.配置CouchDB启动信息

> 参考:fabric-samples/first-network/docker-compose-couch.yaml

```
  couchdb0:
    container_name: couchdb0
    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=
    # Comment/Uncomment the port mapping if you want to hide/expose the CouchDB service,
    # for example map it to utilize Fauxton User Interface in dev environments.
    ports:
      - "5984:5984"
    networks:
      - byfn
```

> 修改:fabric-samples/chaincode-docker-devmode/docker-compose-simple.yaml 末尾添加并修改

```
  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=
    # Comment/Uncomment the port mapping if you want to hide/expose the
CouchDB service,
    # for example map it to utilize Fauxton User Interface in dev
environments.
    ports:
      - "5984:5984"
```

2.配置CouchDB连接信息

> 参考fabric-samples/first-network/docker-compose-couch.yaml

```
  peer0.org1.example.com:
    environment:
      - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
      - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb0:5984
      # The CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME and
CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD
      # provide the credentials for ledger to connect to CouchDB.  The
username and password must
      # match the username and password set for the associated CouchDB.
      - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=
      - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=
    depends_on:
      - couchdb0
```

> 修改:fabric-samples/chaincode-docker-devmode/docker-compose-simple.yaml 中peer模块

修改前

```
  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
    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
```

修改后

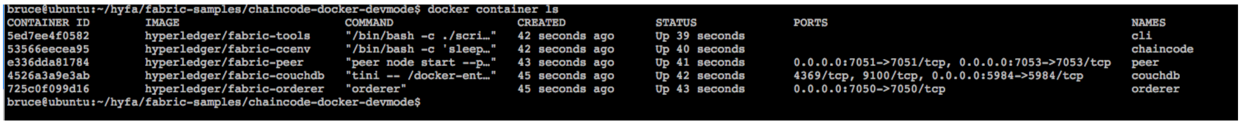
```
peer:
  container_name: peer
  image: hyperledger/fabric-peer
  environment:
    - CORE_PEER_ID=peer
    - CORE_PEER_ADDRESS=peer:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer:7051
    - CORE_PEER_LOCALMSPID=DEFAULT
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_LOGGING_LEVEL=DEBUG
    - 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
```

> 注意JSON文件的格式以及配置信息的一致性，如couchdb名称等

3.启动测试环境

```
# docker-compose  -f docker-compose-simple.yaml  up -d
# docker container ls
```



```
bruce@ubuntu:~/hyfa/fabric-samples/chaincode-docker-devmode$ docker container ls
CONTAINER ID        IMAGE                         COMMAND                  CREATED             STATUS              PORTS                                              NAMES
5ed7ee4f0582        hyperledger/fabric-tools      "/bin/bash -c ./scri…"   42 seconds ago      Up 39 seconds                                                          cli
53566eecea95        hyperledger/fabric-ccenv      "/bin/bash -c 'sleep…"   42 seconds ago      Up 40 seconds                                                          chaincode
e336dda81784        hyperledger/fabric-peer       "peer node start --p…"   43 seconds ago      Up 41 seconds       0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp   peer
4526a3a9e3ab        hyperledger/fabric-couchdb    "tini -- /docker-ent…"   45 seconds ago      Up 42 seconds       4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp       couchdb
725c0f099d16        hyperledger/fabric-orderer    "orderer"                45 seconds ago      Up 43 seconds       0.0.0.0:7050->7050/tcp                           orderer
bruce@ubuntu:~/hyfa/fabric-samples/chaincode-docker-devmode$
```

三.编写链码

1.代码结构

代码包:testdb

代码文件

- domain.go //数据结构代码
- main.go //业务测试代码

2.数据结构

```
package main

type BillStruct struct {
    ObjectType   string `json:"DocType"`      //对象类型定义
    BillInfoID   string `json:"BillInfoID"`   //票据ID
    BillInfoAmt  string `json:"BillInfoAmt"`  //票据金额
    BillInfoType string `json:"BillInfoType"` //票据类型
    BillIsseData string `json:"BillIsseData"` //出票日期
    BillDueDate  string `json:"BillDueDate"`  //到期日期

    HoldrAcct       string `json:"HoldrAcct"`       //持票人名称
    HoldrCmID       string `json:"HoldrCmID"`       //持票人ID
    WaitEndroseAcct string `json:"WaitEndroseAcct"` //待背书人名称
    WaitEndorseCmID string `json:"WaitEndorseCmID"` //待背书人ID
}
```

3.测试代码

> 请仔细阅读注释信息，此处不做代码分割描述

```
package main

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

//定义结构体CouchDBChaincode，作为shim.ChaincodeStubInterface实现类对象
type CouchDBChaincode struct {
}

//重写shim.ChaincodeStubInterface接口的Init方法
func (t *CouchDBChaincode) Init(stub shim.ChaincodeStubInterface)
peer.Response {
    return shim.Success(nil)
}

//重写shim.ChaincodeStubInterface接口的Invoke方法
func (t *CouchDBChaincode) Invoke(stub shim.ChaincodeStubInterface)
peer.Response {
    //获取用户意图与参数
    fun, args := stub.GetFunctionAndParameters()
    //根据用户意图判断使用何种实现函数
    if fun == "billInit" {
        return billInit(stub)
    } else if fun == "queryBills" {
        return queryBills(stub, args)
    } else if fun == "queryWaitBills" {
        return queryWaitBills(stub, args)
    }
    //如果用户意图不符合如上，进行错误提示
    return shim.Error("非法操作，指定的函数名无效")
}

//billInit函数：初始化票据数据
func billInit(stub shim.ChaincodeStubInterface) peer.Response {

    /*
定义第一个票据：
持票人名称:AAA
持票人ID:AID
待背书人名称:无
待背书人ID:无
     */

    billA := BillStruct{
        ObjectType:      "billObj",
        BillInfoID:      "POC001",
        BillInfoAmt:     "1000",
        BillInfoType:    "111",
        BillIsseData:    "20180501",
```

```go
        BillDueDate:     "20180508",
        HoldrAcct:       "AAA",
        HoldrCmID:       "AID",
        WaitEndroseAcct: "",
        WaitEndorseCmID: "",
    }
    //通过json.Marshal方法对票据进行序列化操作
    billAByte, _ := json.Marshal(billA)
    //通过stub.PutState方法存储序列化后的字节数组
    err := stub.PutState(billA.BillInfoID, billAByte)
    if err != nil {
        return shim.Error("初始化第一个票据失败:" + err.Error())
    }

    billB := BillStruct{
        ObjectType:      "billObj",
        BillInfoID:      "POC002",
        BillInfoAmt:     "1000",
        BillInfoType:    "111",
        BillIsseData:    "20180501",
        BillDueDate:     "20180508",
        HoldrAcct:       "AAA",
        HoldrCmID:       "AID",
        WaitEndroseAcct: "BBB",
        WaitEndorseCmID: "BID",
    }
    billBByte, _ := json.Marshal(billB)
    err = stub.PutState(billB.BillInfoID, billBByte)
    if err != nil {
        return shim.Error("初始化第二个票据失败:" + err.Error())
    }

    billC := BillStruct{
        ObjectType:      "billObj",
        BillInfoID:      "POC003",
        BillInfoAmt:     "1000",
        BillInfoType:    "111",
        BillIsseData:    "20180501",
        BillDueDate:     "20180508",
        HoldrAcct:       "BBB",
        HoldrCmID:       "BID",
        WaitEndroseAcct: "CCC",
        WaitEndorseCmID: "CID",
    }

    billCByte, _ := json.Marshal(billC)
    err = stub.PutState(billC.BillInfoID, billCByte)
    if err != nil {
        return shim.Error("初始化第三个票据失败:" + err.Error())
    }

    billD := BillStruct{
        ObjectType:      "billObj",
        BillInfoID:      "POC004",
        BillInfoAmt:     "1000",
        BillInfoType:    "111",
        BillIsseData:    "20180501",
        BillDueDate:     "20180508",
        HoldrAcct:       "CCC",
        HoldrCmID:       "CID",
        WaitEndroseAcct: "BBB",
        WaitEndorseCmID: "BID",
    }

    billDByte, _ := json.Marshal(billD)
    err = stub.PutState(billD.BillInfoID, billDByte)
    if err != nil {
        return shim.Error("初始化第四个票据失败:" + err.Error())
    }

    return shim.Success([]byte("所有票据初始化成功"))

}

//queryBills函数:批量查询指定用户的持票列表
func queryBills(stub shim.ChaincodeStubInterface, args []string)
peer.Response {
    //判断是否有参数传入
    if len(args) != 1 {
        return shim.Error("必须指定持票人的证件号码")
    }
    //将第一个参数作为用户ID
    holdrCmID := args[0]

    /*将CouchDB查询字符串拼接成一个JSON串，格式如下:
        {
        "selector": {
            "docType": "billObj",
            "HoldrCmID": "%s"
        }
    }
    */
    queryString := fmt.Sprintf("{\"selector\":
{\"DocType\":\"billObj\",\"HoldrCmID\":\"%s\"}}", holdrCmID)
    //通过自定义的getBillByQueryString函数进行数据查询操作
    result, err := getBillByQueryString(stub, queryString)
    if err != nil {
        return shim.Error("根据持票人的证件号码批量查询持票人持有票据列表时发生错误"
+ err.Error())
    }
    return shim.Success(result)

}

//queryWaitBills函数:批量查询指定用户的待背书票据列表
func queryWaitBills(stub shim.ChaincodeStubInterface, args []string)
peer.Response {
    if len(args) != 1 {
        return shim.Error("必须指定待背书人的证件号码")
    }
    waitEndorseCmID := args[0]

    queryString := fmt.Sprintf("{\"selector\":
{\"docType\":\"billObj\",\"WaitEndorseCmID\":\"%s\"}}", waitEndorseCmID)
    result, err := getBillByQueryString(stub, queryString)

    if err != nil {
        return shim.Error("根据待背书人的证件号码批量查询待背书票据列表时发生错误" +
err.Error())
    }
    return shim.Success(result)
}

//自定义函数:getBillByQueryString:根据指定的查询字符串(CouchDB查询语句)查询数据
func getBillByQueryString(stub shim.ChaincodeStubInterface, queryString
string) ([]byte, error) {
    //通过stub.GetQueryResult方法获取迭代器iterator
    iterator, err := stub.GetQueryResult(queryString)
    if err != nil {
        return nil, err
    }
    //延迟关闭迭代器iterator
    defer iterator.Close()
    //定义字节缓冲变量
    var buffer bytes.Buffer
    //定义分割符
    var isSplit bool
    //对迭代器进行遍历操作
    for iterator.HasNext() {
        //通过迭代器的Next()方法获取下一个对象的Key与Value值(*queryresult.KV)
        result, err := iterator.Next()
        if err != nil {
            return nil, err
        }

        if isSplit {
            buffer.WriteString(";")
        }
        //定义格式
        // key:result.key result.Value
        buffer.WriteString("key:")
        buffer.WriteString(result.Key)
        buffer.WriteString(",value:")
        buffer.WriteString(string(result.Value))
        //获取到第一个值后，将isSplit设置为true，用于跟第二个值进行分割
        isSplit = true

    }
    //返回buffer对象的字节类型
    return buffer.Bytes(), nil
}

func main() {
    //启动链码CouchDBChaincode
    err := shim.Start(new(CouchDBChaincode))
    //如有报错，提示报错信息
    if err != nil {
        fmt.Errorf(err.Error())
    }

}
```

四.安装链码

1.上传链码

> 上传链码包testdb至:fabric-samples/chaincode中

```
# ls /home/bruce/hyfa/fabric-samples/chaincode/testdb/
 domain.go  main.go
```

2.编译链码

```
# cd  /home/bruce/hyfa/fabric-samples/chaincode/testdb/
# go build
# ls
domain.go  main.go  testdb
```

3.启动链码

> 进入chaincode容器进行操作

```
# docker container exec -it chaincode bash #进入chaincode容器进行操作
# cd testdb/
# CORE_PEER_ADDRESS=peer:7052 CORE_CHAINCODE_ID_NAME=testCouchDB:1.0
./testdb


2018-08-05 10:33:37.063 UTC [shim] SetupChaincodeLogging -> INFO 001
Chaincode log level not provided; defaulting to: INFO
2018-08-05 10:33:37.063 UTC [shim] SetupChaincodeLogging -> INFO 002
Chaincode (build level: ) starting up ...
```

4.安装与实例化链码

> 进入cli容器进行操作

```
# docker container exec -it cli bash
# peer chaincode install -n testCouchDB -v 1.0 -p
chaincodedev/chaincode/testdb
# peer chaincode instantiate -n testCouchDB -v 1.0 -C myc -c '{"Args":
["init"]}'
如有更新请用如下命令进行操作
# peer chaincode install -n testCouchDB -v 1.1 -p
chaincodedev/chaincode/testdb
# peer chaincode upgrade -n testCouchDB -v 1.1 -C myc -c '{"Args":
["init"]}'
```

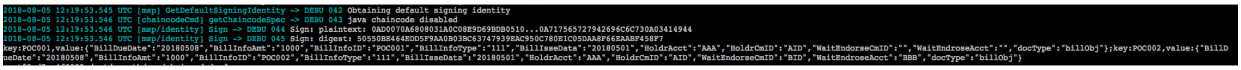
五.测试链码

1.初始化票据

```
# peer chaincode  invoke  -n testCouchDB -C myc -c '{"Args":["billInit"]}'
```

2.查询指定用户所持票据

```
# peer chaincode  query  -n testCouchDB -C myc -c '{"Args":
["queryBills","AID"]}'
```



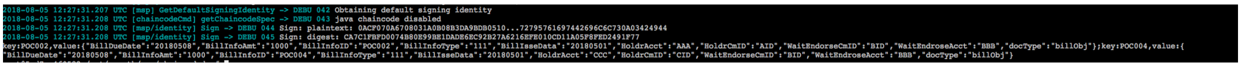
```
2018-08-05 12:19:53.545 UTC [msp] GetDefaultSigningIdentity -> DEBU 042 Obtaining default signing identity
2018-08-05 12:19:53.546 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 043 java chaincode disabled
2018-08-05 12:19:53.546 UTC [msp/identity] Sign -> DEBU 044 Sign: plaintext: 0AD0070A6808031A0C08E9D69BDB0510...0A717565727942696C6C730A03414944
2018-08-05 12:19:53.546 UTC [msp/identity] Sign -> DEBU 045 Sign: digest: 50550BE464EDD5F9AA0B03BC63747939EAC950C780E1C05DAA8F66EAABF458F7
key:POC001,value:{"BillDueDate":"20180508","BillInfoAmt":"1000","BillInfoID":"POC001","BillInfoType":"111","BillIsseData":"20180501","HoldrAcct":"AAA","HoldrCmID":"AID","WaitEndorseCmID":"","WaitEndroseAcct":"","docType":"billObj"};key:POC002,value:{"BillD
ueDate":"20180508","BillInfoAmt":"1000","BillInfoID":"POC002","BillInfoType":"111","BillIsseData":"20180501","HoldrAcct":"AAA","HoldrCmID":"AID","WaitEndorseCmID":"BID","WaitEndroseAcct":"BBB","docType":"billObj"}
```

```
key: POC001, value: {
    "BillDueDate": "20180508",
    "BillInfoAmt": "1000",
    "BillInfoID": "POC001",
    "BillInfoType": "111",
    "BillIsseData": "20180501",
    "HoldrAcct": "AAA",
    "HoldrCmID": "AID",
    "WaitEndorseCmID": "",
    "WaitEndroseAcct": "",
    "docType": "billObj"
};
key: POC002, value: {
    "BillDueDate": "20180508",
    "BillInfoAmt": "1000",
    "BillInfoID": "POC002",
    "BillInfoType": "111",
    "BillIsseData": "20180501",
    "HoldrAcct": "AAA",
    "HoldrCmID": "AID",
    "WaitEndorseCmID": "BID",
    "WaitEndroseAcct": "BBB",
    "docType": "billObj"
}
```

> 查询结果可以看到我们定义的分隔符;

3.查询指定用户待背书票据

```
# peer chaincode  query  -n testCouchDB -C myc -c '{"Args":
["queryWaitBills","BID"]}'
```



```
2018-08-05 12:27:31.207 UTC [msp] GetDefaultSigningIdentity -> DEBU 042 Obtaining default signing identity
2018-08-05 12:27:31.208 UTC [chaincodeCmd] getChaincodeSpec -> DEBU 043 java chaincode disabled
2018-08-05 12:27:31.208 UTC [msp/identity] Sign -> DEBU 044 Sign: plaintext: 0ACF070A6708031A0B08B3DA9BDB0510...72795761697442696C6C730A03424944
2018-08-05 12:27:31.208 UTC [msp/identity] Sign -> DEBU 045 Sign: digest: CA7C1FBFD0074B80E99BE1DADE6EC92B27A6216EFE010CD11A05F8FED2491F77
key:POC002,value:{"BillDueDate":"20180508","BillInfoAmt":"1000","BillInfoID":"POC002","BillInfoType":"111","BillIsseData":"20180501","HoldrAcct":"AAA","HoldrCmID":"AID","WaitEndorseCmID":"BID","WaitEndroseAcct":"BBB","docType":"billObj"};key:POC004,value:{
"BillDueDate":"20180508","BillInfoAmt":"1000","BillInfoID":"POC004","BillInfoType":"111","BillIsseData":"20180501","HoldrAcct":"CCC","HoldrCmID":"CID","WaitEndorseCmID":"BID","WaitEndroseAcct":"BBB","docType":"billObj"}
```

```
key: POC002, value: {
    "BillDueDate": "20180508",
    "BillInfoAmt": "1000",
    "BillInfoID": "POC002",
    "BillInfoType": "111",
    "BillIsseData": "20180501",
    "HoldrAcct": "AAA",
    "HoldrCmID": "AID",
    "WaitEndorseCmID": "BID",
    "WaitEndroseAcct": "BBB",
    "docType": "billObj"
};
key: POC004, value: {
    "BillDueDate": "20180508",
    "BillInfoAmt": "1000",
    "BillInfoID": "POC004",
    "BillInfoType": "111",
    "BillIsseData": "20180501",
    "HoldrAcct": "CCC",
    "HoldrCmID": "CID",
    "WaitEndorseCmID": "BID",
    "WaitEndroseAcct": "BBB",
    "docType": "billObj"
}
```

另外关于LevelDB,CouchDB还是MongoDB，今后可能随着Hyperledger Fabric的版本变化而采取不同的数据库类型，我们拭目以待，现在唯一能做的，就是在已有的资源下面用Hyperledger Fabric为业务场景创造最大的业务价值。

```
package maintype BillStruct struct {
    ObjectType   string `json:"DocType"`      //对象类型定义
    BillInfoID   string `json:"BillInfoID"`   //票据ID
    BillInfoAmt  string `json:"BillInfoAmt"`  //票据金额
    BillInfoType string `json:"BillInfoType"` //票据类型
    BillIsseData string `json:"BillIsseData"` //出票日期
    BillDueDate  string `json:"BillDueDate"`  //到期日期

    HoldrAcct       string `json:"HoldrAcct"`       //持票人名称
    HoldrCmID       string `json:"HoldrCmID"`       //持票人ID
    WaitEndroseAcct string `json:"WaitEndroseAcct"` //待背书人名称
    WaitEndorseCmID string `json:"WaitEndorseCmID"` //待背书人ID}
3.测试代码
请仔细阅读注释信息，此处不做代码分割描述

package mainimport (    "github.com/hyperledger/fabric/core/chaincode/shim"
    "fmt"
    "github.com/hyperledger/fabric/protos/peer"
    "encoding/json"
    "bytes")//定义结构体CouchDBChaincode，作为shim.ChaincodeStubInterface实现类
对象type CouchDBChaincode struct {
}//重写shim.ChaincodeStubInterface接口的Init方法func (t *CouchDBChaincode)
Init(stub shim.ChaincodeStubInterface) peer.Response {    return
shim.Success(nil)
```

```
}//重写shim.ChaincodeStubInterface接口的Invoke方法func (t *CouchDBChaincode)
Invoke(stub shim.ChaincodeStubInterface) peer.Response {    //获取用户意图与参
数
    fun, args := stub.GetFunctionAndParameters()    //根据用户意图判断使用何种实
现函数
    if fun == "billInit" {        return billInit(stub)
    } else if fun == "queryBills" {        return queryBills(stub, args)
    } else if fun == "queryWaitBills" {        return queryWaitBills(stub,
args)
    }    //如果用户意图不符合如上，进行错误提示
    return shim.Error("非法操作，指定的函数名无效")
}//billInit函数：初始化票据数据func billInit(stub shim.ChaincodeStubInterface)
peer.Response {    /*
定义第一个票据：
持票人名称:AAA
持票人ID:AID
待背书人名称:无
待背书人ID:无
     */

    billA := BillStruct{
        ObjectType:      "billObj",
        BillInfoID:      "POC001",
        BillInfoAmt:     "1000",
        BillInfoType:    "111",
        BillIsseData:    "20180501",
        BillDueDate:     "20180508",
        HoldrAcct:       "AAA",
        HoldrCmID:       "AID",
        WaitEndroseAcct: "",
        WaitEndorseCmID: "",
    }    //通过json.Marshal方法对票据进行序列化操作
    billAByte, _ := json.Marshal(billA)    //通过stub.PutState方法存储序列化后
的字节数组
    err := stub.PutState(billA.BillInfoID, billAByte)    if err != nil {
     return shim.Error("初始化第一个票据失败:" + err.Error())
    }

    billB := BillStruct{
        ObjectType:      "billObj",
        BillInfoID:      "POC002",
        BillInfoAmt:     "1000",
        BillInfoType:    "111",
        BillIsseData:    "20180501",
        BillDueDate:     "20180508",
        HoldrAcct:       "AAA",
        HoldrCmID:       "AID",
        WaitEndroseAcct: "BBB",
        WaitEndorseCmID: "BID",
    }
    billBByte, _ := json.Marshal(billB)
    err = stub.PutState(billB.BillInfoID, billBByte)    if err != nil {
   return shim.Error("初始化第二个票据失败:" + err.Error())
    }

    billC := BillStruct{
        ObjectType:      "billObj",
        BillInfoID:      "POC003",
        BillInfoAmt:     "1000",
        BillInfoType:    "111",
        BillIsseData:    "20180501",
        BillDueDate:     "20180508",
        HoldrAcct:       "BBB",
        HoldrCmID:       "BID",
        WaitEndroseAcct: "CCC",
        WaitEndorseCmID: "CID",
    }

    billCByte, _ := json.Marshal(billC)
    err = stub.PutState(billC.BillInfoID, billCByte)    if err != nil {
   return shim.Error("初始化第三个票据失败:" + err.Error())
    }

    billD := BillStruct{
        ObjectType:      "billObj",
        BillInfoID:      "POC004",
        BillInfoAmt:     "1000",
        BillInfoType:    "111",
        BillIsseData:    "20180501",
        BillDueDate:     "20180508",
        HoldrAcct:       "CCC",
        HoldrCmID:       "CID",
        WaitEndroseAcct: "BBB",
        WaitEndorseCmID: "BID",
    }

    billDByte, _ := json.Marshal(billD)
    err = stub.PutState(billD.BillInfoID, billDByte)    if err != nil {
   return shim.Error("初始化第四个票据失败:" + err.Error())
    }    return shim.Success([]byte("所有票据初始化成功"))

}//queryBills函数:批量查询指定用户的持票列表func queryBills(stub
shim.ChaincodeStubInterface, args []string) peer.Response {    //判断是否有参
数传入
    if len(args) != 1 {        return shim.Error("必须指定持票人的证件号码")
    }    //将第一个参数作为用户ID
    holdrCmID := args[0]    /*将CouchDB查询字符串拼接成一个JSON串，格式如下：
        {
```

```
    "selector": {
      "docType": "billObj",
      "HoldrCmID": "%s"
    }
  }
  */
  queryString := fmt.Sprintf("{\"selector\":
{\"DocType\":\"billObj\",\"HoldrCmID\":\"%s\"}}", holdrCmID)    //通过自定义
的getBillByQueryString函数进行数据查询操作
  result, err := getBillByQueryString(stub, queryString)    if err != nil
{    return shim.Error("根据持票人的证件号码批量查询持票人持有票据列表时发生错误"
+ err.Error())
  }    return shim.Success(result)

}//queryWaitBills函数:批量查询指定用户的待背书票据列表func queryWaitBills(stub
shim.ChaincodeStubInterface, args []string) peer.Response {    if len(args)
!= 1 {        return shim.Error("必须指定待背书人的证件号码")
  }
  waitEndorseCmID := args[0]

  queryString := fmt.Sprintf("{\"selector\":
{\"docType\":\"billObj\",\"WaitEndorseCmID\":\"%s\"}}", waitEndorseCmID)
  result, err := getBillByQueryString(stub, queryString)    if err != nil
{    return shim.Error("根据待背书人的证件号码批量查询待背书票据列表时发生错误"
+ err.Error())
  }    return shim.Success(result)
}//自定义函数:getBillByQueryString:根据指定的查询字符串(CouchDB查询语句)查询数据
func getBillByQueryString(stub shim.ChaincodeStubInterface, queryString
string) ([]byte, error) {    //通过stub.GetQueryResult方法获取迭代器iterator
  iterator, err := stub.GetQueryResult(queryString)    if err != nil {
  return nil, err
  }    //延迟关闭迭代器iterator
  defer iterator.Close()    //定义字节缓冲变量
  var buffer bytes.Buffer    //定义分割符
  var isSplit bool
  //对迭代器进行遍历操作
  for iterator.HasNext() {       //通过迭代器的Next()方法获取下一个对象的Key与
Value值(*queryresult.KV)
    result, err := iterator.Next()       if err != nil {
 return nil, err
    }       if isSplit {
      buffer.WriteString(";")
    }       //定义格式
    // key:result.key result.Value
    buffer.WriteString("key:")
    buffer.WriteString(result.Key)
    buffer.WriteString(",value:")
    buffer.WriteString(string(result.Value))       //获取到第一个值后，将
isSplit设置为true，用于跟第二个值进行分割
```

```
        isSplit = true

    }    //返回buffer对象的字节类型
    return buffer.Bytes(), nil}func main() {    //启动链码CouchDBChaincode
    err := shim.Start(new(CouchDBChaincode))    //如有报错，提示报错信息
    if err != nil {
        fmt.Errorf(err.Error())
    }

}
```