链码开发

票据相关请求处理

创建一个名为`chaincode`的新目录, 目录中的链码将处理来自应用程序的请求

定义票据链码实现接口文件: `billCC.go`

```
$ vim chaincode/billCC.go
```

```
package main

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

// 根据指定的票据号码获取对应的票据信息
func (t *BillChainCode) getBill(stub shim.ChaincodeStubInterface,
bill_No string) (Bill, bool)  {

    // 1. 将定义的前缀与票据号码连接形成一个Key
```

```
    // 2. 以Key为参数查询票据信息

    // 3. 反序列化查询到的票据

    // 4. 返回

}

// 保存指定的票据信息
func (t *BillChainCode) putBill(stub shim.ChaincodeStubInterface,
bill Bill) ([]byte, bool)  {
    // 1. 将参数bill序列化

    // 2. 以定义的前缀与票据号码连接形成一个Key将bill保存在账本中

    // 3. 返回

}

// 发布票据
// args: 0 - {bill object}
func (t *BillChainCode) issue(stub shim.ChaincodeStubInterface,
args[] string) peer.Response  {
    // 1. 检查参数个数是否为一个

    // 2. 反序列化args[0]参数为bill

    // 3. 查重(根据票据号码查询是否存在. 如果根据票据号码查询到信息，证明
已存在)

    // 4. 更改票据状态(票据状态设为新发布). 并保存票据:

    // 5. 保存以当前持票人ID与票据号码构造的复合键，以便持票人批量查询.
value为空即可

    // 6. 返回

}
```

```
// 查询当前用户的票据(根据当前持票人证件号码，批量查询票据)
// args: 0 - holderCmId(当前持票人证件号码)
func (t *BillChainCode) queryMyBills(stub
shim.ChaincodeStubInterface, args []string) peer.Response {
    // 1. 检查参数个数是否为一个

    // 根据指定的组合键查询分类账上的状态
    // -- 在发布票据方法中实现了将复合键(持票人ID与票据号码构造的复合键)保
存在分类账中
    // 2. 根据当前持票人证件号码从search中查询所持有的票号

    // 3. 迭代处理

    // 4. 序列化票据数组

    // 5. 返回查询结果

}

// 根据票据号码获取票据状态及该票据的背书历史
// args: 0 - bill_No
func (t *BillChainCode) queryBillByNo(stub
shim.ChaincodeStubInterface, args []string) peer.Response  {
    // 1. 检查参数个数是否为一个

    // 2. 根据票据号码获取对应的票据状态

    // 3. 获取票据背书变更历史

    // 4. 迭代处理

    // 5. 将背书历史做为票据的属性返回

    // 6. 序列化bill

    // 7. 返回结果
```

```
}

// 查询当前用户的待背书票据(根据待背书人证件号码)
// args: 0 - endorserCmId
func (t *BillChainCode) queryMyWaitBills(stub
shim.ChaincodeStubInterface, args []string) peer.Response {
    // 1. 检查参数个数是否为一个

    // 2. 根据待背书人证件号码从search中查询待背书的票据号码
    // -- 票据背书请求方法中实现了将待背书人ID与票据号码构造的复合键信息的
保存

    // 3. 迭代处理

    // 4. 序列化待背书票据数组

    // 5. 返回结果

}
```