链码开发

现在几乎可以使用区块链系统。 但还没有设置任何链式代码（智能合约）

票据结构

定义票据结构及票据状态文件: `structure.go`

```
$ mkdir chaincode
$ vim chaincode/structure.go
```

```
package main

// 票据状态
const (
    BillInfo_State_NewPublish    = "NewPublish"       // 新发布票据
    BillInfo_State_EndorseWaitSign = "EndorseWaitSign"  // 等待签收票
据
    BillInfo_State_EndorseSigned = "EndorseSigned"       // 票据签收成
功
    BillInfo_State_EndorseReject = "EndorseReject"       // 拒绝签收票
据
)
```

```
const (
    Bill_Prefix  = "Bill_"              // 票据key前缀
    IndexName  = "holderName~billNo"    // search的映射名
)

//const HolderIdDayTimeBillTypeBillNoIndexName = "holderId~dayTime-
billType-billNo"

// 票据数据结构
type Bill struct {
    BillInfoID          string      `json:"BillInfoID"`          //
票据号码
    BillInfoAmt         string      `json:"BillInfoAmt"`         //
票据金额
    BillInfoType        string      `json:"BillInfoType"`        //
票据类型

    BillInfoIsseDate    string      `json:"BillInfoIsseDate"`    //
票据出票日期
    billInfoDueDate     string      `json:"billInfoDueDate"`     //
票据到期日期

    DrwrCmID            string      `json:"DrwrCmID"`            //
出票人证件号码
    DrwrAcct            string      `json:"DrwrAcct"`            //
出票人名称

    AccptrCmID          string      `json:"AccptrCmID"`          //
承兑人证件号码
    AccptrAcct          string      `json:"AccptrAcct"`          //
承兑人名称

    PyeeCmID            string      `json:"PyeeCmID"`            //
收款人证件号码
    PyeeAcct            string      `json:"PyeeAcct"`            //
收款人名称
```

```
    HoldrCmID           string      `json:"HodrCmID"`        // 当前持
票人证件号码
    HoldrAcct           string      `json:"HodrAcct`         // 当前持
票人名称

    WaitEndorseCmID string      `json:"WaitEndorseCmID"`     // 待背书
人证件号码
    WaitEndorseAcct string      `json:"WaitEndorseAcct"`     // 待背书
人名称

    RejectEndorseCmID   string      `json:"RejectEndorseCmID"`  //
拒绝背书人证件号码
    RejectEndorseAcct   string      `json:"RejectEndorseAcct"`  //
拒绝背书人名称

    State               string      `json:"State"`              //
票据状态
    History             []HistoryItem   `json:"History"`        //
票据背书历史
}

// 票据历史信息
type HistoryItem struct {
    TxId        string      `json:"TxId"`
    Bill        Bill        `json:"bill"`
}
```

定义一个通用接口的工具文件 `utils.go`

```
$ vim chaincode/utils.go
```

```
package main

import (
    "encoding/json"
    "fmt"
```

```
)

// chaincode Response结构
type chaincodeRet struct {
    Code    int         // 0代表成功，否则为1
    Dec     string      // 描述
}

// 根据返回码和描述信息返回序列化后的字节数组
func GetRetByte(code int, dec string) ([]byte)  {
    b, err := getRet(code, dec)
    if err != nil {
        return []byte{}
    }
    return b
}

// 根据返回码和描述信息返回序列化后的字符串
func GetRetString(code int, dec string) (string)  {
    b, err := getRet(code, dec)
    if err != nil {
        return ""
    }
    return string(b[:])
}

// 根据返回码和描述信息进行序列化
func getRet(code int, dec string) ([]byte, error)  {
    var c chaincodeRet
    c.Code = code
    c.Dec = dec

    b, err := json.Marshal(c)
    if err != nil {
        return []byte{0x00}, fmt.Errorf("序列化失败")
    }
    return b, err
}
```

链码主文件

添加链码实现主文件: `main.go` （智能合约的主要入口点）

```
$ vim chaincode/main.go
```

```
package main

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

type BillChainCode struct {

}

func (t *BillChainCode) Init(stub shim.ChaincodeStubInterface)
peer.Response {
    return shim.Success(nil)
}

func (t *BillChainCode) Invoke(stub shim.ChaincodeStubInterface)
peer.Response {
    function, args := stub.GetFunctionAndParameters()

    if function == "issue" {
        return t.issue(stub, args)
    }else if function == "queryMyBills" {
        return t.queryMyBills(stub, args)
    }else if function == "queryBillByNo" {
        return t.queryBillByNo(stub, args)
```

```
    }else  if function == "queryMyWaitBills" {
        return t.queryMyWaitBills(stub, args)
    }else if function == "endorse" {
        return t.endorse(stub, args)
    }else if function == "accept" {
        return t.accept(stub, args)
    }else if function == "reject" {
        return t.reject(stub, args)
    }

    return shim.Error("指定的函数名称错误")
}

func main()  {
    err := shim.Start(new(BillChainCode))
    if err != nil {
        fmt.Println("启动链码错误: ", err)
    }
}
```