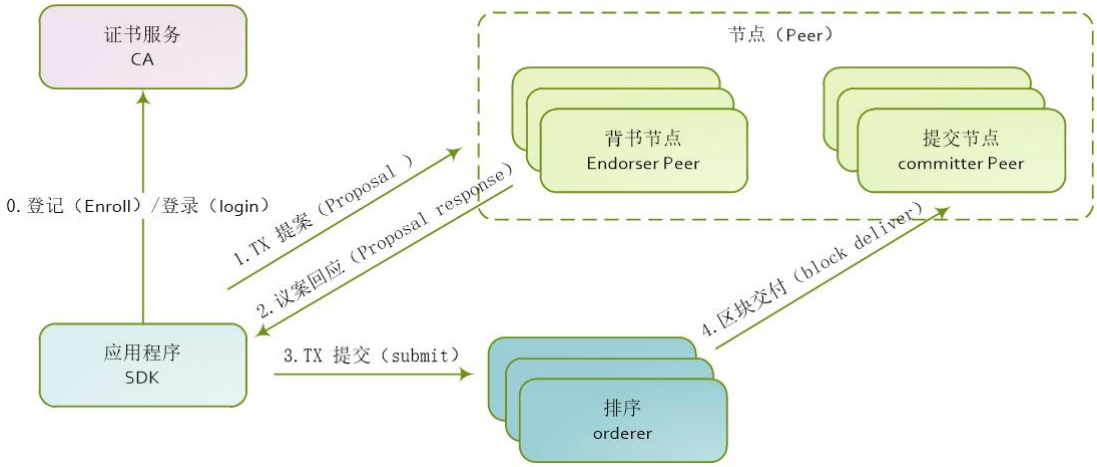
作业3：说明Fabric交易从产生到记录的过程

3180106071 刘轩铭

目标：分析Fabric中交易从产生到记入账本每个环节及相关的数据结构

在作业中，为了便于查询相关资料进行学习，我选择fabric1.0版本进行分析，并通过github的fabric仓库下载到fabric1.0版本的go源代码，对其进行分析。

整体交易过程



在Fabric中，本由一个节点处理的过程，在逻辑上被分解为不同的角色，每个角色承担不同的功能；节点（Peer）分解为背书节点（Endorser peer）和提交节点（Committer peer），为了达到处理的顺序性，还提炼出排序（Orderer）节点。

根据上课学习的内容，我们知道交易过程分为如下几个过程：

- 应用程序客户端通过SDK调用证书服务（CA）服务，进行注册和登记，并获取身份证书；
- 应用程序客户端通过SDK向区块链网络发起一个交易提案（Proposal），交易提案把带有本次交易要调用的合约标识、合约方法和参数信息以及客户端签名等信息发送给背书（Endorser）节点。
- 背书（Endorser）节点收到交易提案（Proposal）后，验证签名并确定提交者是否有权执行操作，同时根据背书策略模拟执行智能合约，并将结果及其各自的CA证书签名发还给应用程序客户端。
- 应用程序客户端收到背书（Endorser）节点返回的信息后，判断提案结果是否一致，以及是否参照指定的背书策略执行，如果没有足够的背书，则中止处理；否则，应用程序客户端把数据打包到一起组成一个交易并签名，发送给Orderers。
- Orderers对接收到的交易进行共识排序，然后按照区块生成策略，将一批交易打包到一起，生成新的区块，发送给提交（Committer）节点；
- 提交（Committer）节点收到区块后，会对区块中的每笔交易进行校验，检查交易依赖的输入输出是否符合当前区块链的状态，完成后将区块追加到本地的区块链，并修改世界状态。

具体交易过程分析

Client发起交易

- 节点首先对状态进行初始化，这其中包括对于背书节点、Fabric本身、orderer节点等的初始化

  ```
  peer chaincode invoke ...
  ```

  通过该命令调用链码，也就是发起一个提案

- 之后陆续调用chaincode.Cmd（`src/github.com/hyperledger/fabric/peer/chaincode/chaincode.go:49`)和invokeCmd（`src/github.com/hyperledger/fabric/peer/chaincode/invoke.go`函数，并执行函数chaincodeInvoke（`src/github.com/hyperledger/fabric/peer/chaincode/invoke.go`)完成参数的配置和调用的发起。

```
func chaincodeInvoke(cmd *cobra.Command, args []string, cf
*ChaincodeCmdFactory) error {
    var err error
    if cf == nil {
        cf, err = InitCmdFactory(true, true)
        if err != nil {
            return err
        }
    }
    defer cf.BroadcastClient.Close()

    return chaincodeInvokeOrQuery(cmd, args, true, cf)
}
```

- 可以看到，在chaincodeInvoke函数中，构造了在chaincodeInvokeOrQuery对象，在这个构造函数中，又一次调用函数ChaincodeInvokeOrQuery执行交易发起逻辑（`src/github.com/hyperledger/fabric/peer/chaincode/common.go:`）
- 在ChaincodeInvokeOrQuery函数中，通过对象方法func (EndorserClient) ProcessProposal，发送交易提案对象SignedProposal到peer服务端，获取结果对象ProposalResponse

```
creator, err := signer.Serialize()
var prop *pb.Proposal
prop, _, err =
putils.CreateProposalFromCIS(pcommon.HeaderType_ENDORSER_TRANSACTION,
cID, invocation, creator)
var signedProp *pb.SignedProposal
signedProp, err = putils.GetSignedProposal(prop, signer) //Proposal签名

var proposalResp *pb.ProposalResponse
proposalResp, err = endorserClient.ProcessProposal(context.Background(),
signedProp)
//代码在peer/chaincode/common.go
```

背书节点对交易做背书

- 首先背书节点完成一系列初始化和注册工作，创建客户端和服务端。
- 正式开始背书：

当接收到proposal请求时, 即客户端触发对象方法 func (EndorserClient) ProcessProposal时，服务端对应进入对象方法func (e *Endorser) ProcessProposal。
(`src/github.com/hyperledger/fabric/core/endorser/endorser.go`)

- 第一步，校验SignedProposal合法性，并获取ChannelHeader和SignatureHeader

```
//校验SignedProposal合法性
prop, hdr, hdrExt, err := validation.ValidateProposalMessage(signedProp)
//获取ChannelHeader
chdr, err := putils.UnmarshalChannelHeader(hdr.ChannelHeader)
//获取SignatureHeader
shdr, err := putils.GetSignatureHeader(hdr.SignatureHeader)
//代码在core/endorser/endorser.go
```

- 第二步，校验是否系统链码且提案不可调用，获取chainID和TxId，获取Ledger并校验txid是否已存在，非系统链码校验提案权限（是否符合通道策略）。
- 第三步，获取账本的交易模拟器和历史记录查询器，并模拟提案执行。这一步主要通过e.simulateProposal(ctx, chainID, txid, signedProp, prop, hdrExt.ChaincodeId, txsim)完成。
- 最后为提案背书，构造ProposalResponse并返回给Endorser客户端：

```
var pResp *pb.ProposalResponse
//为提案背书，即调取escc系统链码
pResp, err = e.endorseProposal(ctx, chainID, txid, signedProp, prop,
res, simulationResult, ccevent, hdrExt.PayloadVisibility,
hdrExt.ChaincodeId, txsim, cd)
pResp.Response.Payload = res.Payload
return pResp, nil
//代码在core/endorser/endorser.go
```

- 校验结果对象ProposalResponse后，通过函数putils.CreateSignedT将提案结果封装为已签名交易，装入common.Envelope
(`src/github.com/hyperledger/fabric/peer/chaincode/common.go:488`)
- 最后回到chaincodeInvokeOrQuery函数中，将获取的结果对象ProposalResponse做校验，最终打印结果到控制台 (`src/github.com/hyperledger/fabric/peer/chaincode/common.go`)

排序节点对交易进行处理

- 在背书之后，将已签名交易common.Envelope，通过对象方法func (BroadcastClient) Send，发送到orderer节点。成功后返回ProposalResponse。

```
if invoke {
    env, err := putils.CreateSignedTx(prop, signer, proposalResp) //创建签名交易
    err = bc.Send(env) //广播交易
}
//代码在peer/chaincode/common.go
```

- 当orderer.server接收到请求时，即客户端调用对象方法func (BroadcastClient) Send时，将会调用对象方法func (oc *OrdererClient) Broadcast
(`src/github.com/hyperledger/fabric/peer/common/ordererclient.go`)，从而进一步调用func (AtomicBroadcastClient) Broadcast， 使得请求落到对象方法func

(AtomicBroadcastServer) Broadcast，对应实现类即orderer.server， 方法实现位于（`src/github.com/hyperledger/fabric/orderer/common/server/server.go:134`）

- 请求进一步由对象方法func (bh *handlerImpl) Handle 处理（`src/github.com/hyperledger/fabric/orderer/common/broadcast/broadcast.go`）
  - 调用对象方法func (r *Registrar) BroadcastChannelSupport， 解析消息并获取相应辅助类。如获取通道header信息，消息类型，并获取对应的接口Consenter排序算法对象， 排序算法接口被封装在接口ChannelSupport，ChainSupport中（`src/github.com/hyperledger/fabric/orderer/common/broadcast/broadcast.go`）
  - 对应常规链码invoke，而非安装链码等Config修改类型的提案， 将调用对象方法func (s *StandardChannel) ProcessNormalMsg，对消息进行验证（`src/github.com/hyperledger/fabric/orderer/common/broadcast/broadcast.go`）。
  - 使用对象方法type (Consenter) Order，对消息对象Envelope进行排序。使用solo和kafka、raft三种实现（在1.0版本没有Raft实现，在最新的版本中实现了Raft方法）。
  - 若排序正常，则调用对象方法func (x *atomicBroadcastBroadcastServer) Send，响应客户端返回成功（`src/github.com/hyperledger/fabric/orderer/common/broadcast/broadcast.go`）

将已背书交易写入账本

当orderer完成区块生成后，peer获取区块并写入本地账本

关键数据结构

Proposal

即向Endorser发起的提案。

Proposal相关结构体定义

SignedProposal定义

```
type SignedProposal struct {
    ProposalBytes []byte //Proposal序列化，即type Proposal struct
    Signature []byte //signer.Sign(ProposalBytes)
}
//代码在protos/peer/proposal.pb.go
```

该结构体对应于完成背书后返回的签名提案。

Proposal定义

```
type Proposal struct {
    Header []byte //Header序列化，即type Header struct
    Payload []byte //ChaincodeProposalPayload序列化，即type ChaincodeProposalPayload struct
    Extension []byte //扩展
}
//代码在protos/peer/proposal.pb.go
```

该结构其实是一个签名头，代表着最初的提案

ProposalResponse结构体定义

ProposalResponse定义

```
type ProposalResponse struct {
    Version int32
    Timestamp *google_protobuf1.Timestamp
    Response *Response //type Response struct，peer.Response{Status: 200, Message: "OK"}}
    Payload []byte
    Endorsement *Endorsement //type Endorsement struct
}
//代码在protos/peer/proposal_response.pb.go
```

对应背书检查完成后，返回发起交易的节点一个Pb类型的实例

Response定义

```
type Response struct { //peer.Response{Status: 200, Message: "OK"}}
    Status int32
    Message string
    Payload []byte
}
//代码在protos/peer/proposal_response.pb.go
```

Endorsement定义

```
type Endorsement struct {
    Endorser []byte //bccspmsp.signer
    Signature []byte
}
//代码在protos/peer/proposal_response.pb.go
```

Envelope

Envelope直译为信封，封装Payload和Signature。

```
type Envelope struct { //用签名包装Payload，以便对信息做身份验证
    Payload []byte //Payload序列化
    Signature []byte //Payload header中指定的创建者签名
}
//代码在protos/common/common.pb.go
```

Transaction

Transaction结构体：

```
type Transaction struct {
    Actions []*TransactionAction //Payload.Data是个TransactionAction数组，容纳每个交易
}
//代码在protos/peer/transaction.pb.go
```

TransactionAction结构体：

```
type TransactionAction struct {
    Header []byte
    Payload []byte
}
//代码在protos/peer/transaction.pb.go
```