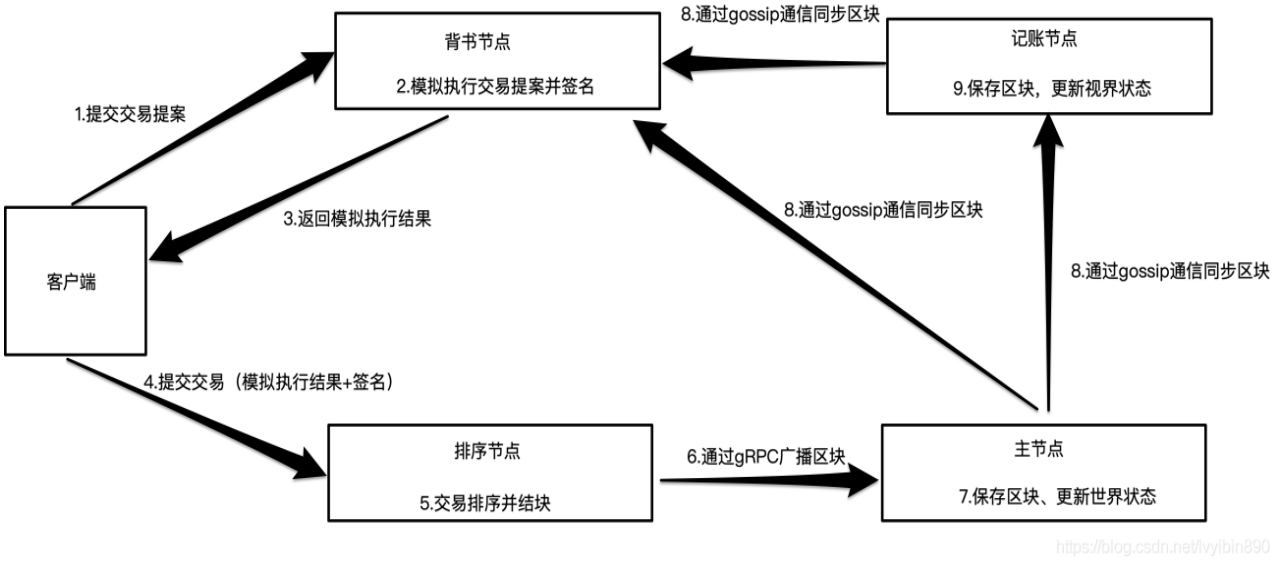
分析 Fabric 源代码

Fabric 交易的整体流程图：



Fabric 交易流程的步骤：

1. 客户端节点构造交提案，选择背书节点提交。
2. 背书节点模式执行交易提案并生成签名。
3. 背书节点将结果返回给客户端。
4. 客户端提交交易。
5. 排序节点对交易排序并做成区块。
6. 排序节点将区块打包并广播区块给组织中的主节点。
7. 主节点保存区块，更新世界状态。
8. 主节点在组织内部通过 gossip 传递消息同步区块。
9. 记账节点保存区块，更新世界状态。

数据结构与分析

交易的结构原型是从 Envelope 的结构体开始，然后封装其他结构体比如 Payload 和 Signature。

```
// Envelope wraps a Payload with a signature so that the message may be authenticated
type Envelope struct {
    // A marshaled Payload
    Payload []byte
    // A signature by the creator specified in the Payload header
    Signature            []byte
    XXX_NoUnkeyedLiteral struct{}
    XXX_unrecognized     []byte
    XXX_sizecache        int32
}
```

从它的命名可以知道，交易就像是一个信封，在排序节点的时候不会去拆开信封来看，作用就只是排序而不关注交易的数据。而当交易完成被打包成区块分发到各个记账节点时，记账节点才会去打开这个信封查看交易数据同时还需要验证才行。Envelope 结构体还有 Payload 和 Signature 字段，交易的关键信息都存储在 Payload 中，Signature 则是存储签名信息。往下看 Payload 的结构体。

Payload 结构体有两个字段，分别是 Header（Payload 的头部字段信息）和 Data（Payload 的交易序列化的信息数据）

```
// Payload is the message contents (and header to allow for signing)
type Payload struct {
    // Header is included to provide identity and prevent replay
    Header *Header
    // Data, the encoding of which is defined by the type in the header
    Data                 []byte
    XXX_NoUnkeyedLiteral struct{}
    XXX_unrecognized     []byte
    XXX_sizecache        int32
}
```

Header 字段的结构体又包含两个字段的序列化对象，有 ChannelHeader 和 SignatureHeader。

```
type Header struct {
  // 顾名思义，交易通道的Header
    ChannelHeader        []byte
  // 顾名思义，交易签名的Header
    SignatureHeader      []byte
    XXX_NoUnkeyedLiteral struct{}
    XXX_unrecognized     []byte
    XXX_sizecache        int32
}
```

SignatureHeader 结构体又有两个字段，是 Creator（消息的创建者，一个序列化的身份信息），Nonce（随机数），这两个字段可以生成一个 TxID。

```
type SignatureHeader struct {
  // Creator 表示消息的创建者，他是一个序列化的身份信息
    Creator []byte
  // Nonce是只使用一次的随机数字，可用于检测replay attacks
  // Creator和Nonce两个字段可以用来生成一个TxID
    Nonce                []byte
    XXX_NoUnkeyedLiteral struct{}
    XXX_unrecognized     []byte
    XXX_sizecache        int32
}
```

ChannelHeader 的结构体有以下的字段，有 Type（Header 的 type），Version（消息协议的版本），Timestamp（消息创建的时间戳），ChannelID（消息绑定的通道 ID），TxID（消息的唯一标识符），Epoch（Header 的纪元）和 Extension（根据 Type 字段可以附加的扩展部分）。

```
type ChannelHeader struct {
  // HeaderType，之前说的Payload中Data字段会与这个Type有关
  // 它包含下面几个类型，具体每个Heaeder类型是做什么的这里暂时不说了
    // const (
    //  HeaderType_MESSAGE              HeaderType = 0
    //  HeaderType_CONFIG               HeaderType = 1
    //  HeaderType_CONFIG_UPDATE        HeaderType = 2
    //  HeaderType_ENDORSER_TRANSACTION HeaderType = 3
    //  HeaderType_ORDERER_TRANSACTION  HeaderType = 4
    //  HeaderType_DELIVER_SEEK_INFO    HeaderType = 5
    //  HeaderType_CHAINCODE_PACKAGE    HeaderType = 6
    //  HeaderType_PEER_ADMIN_OPERATION HeaderType = 8
    //  HeaderType_TOKEN_TRANSACTION    HeaderType = 9
    //)
    Type int32
    // Version表示消息协议的版本
  // 在创世区块中被这么赋值 msgVersion = int32(1)
    Version int32
    // Timestamp表示这条消息创建时的本地时间
    Timestamp *timestamp.Timestamp
    // ChannelId表示这条消息绑定的通道的ID
    ChannelId string
    // TxID表示交易的唯一标识符，唯一ID
    TxId string
    // Epoch直译表示新纪元，epoch表示生成此Header的纪元,他是根据Block height定义的
    Epoch uint64
    // Extension是根据Type字段可以附加的扩展部分
    Extension []byte
  // TlsCertHash表示客户端TLS证书的哈希，如果使用的是mutual TLS
    TlsCertHash          []byte
    XXX_NoUnkeyedLiteral struct{}
    XXX_unrecognized     []byte
    XXX_sizecache        int32
}
```

说完 Header 的部分，接下来到 Payload 的 Data 部分，Data 是一个具体的反序列化对象，根据 ChannelHeader 中的 Type 不同而不同，就是不同的交易类型会导致 Data 的元数据不同。当 Payload.Data 赋值的时候，可能是不同的结构体对象序列化之后的字节切片。但是在验证交易需要获取交易信息的时候，又被反序列化成了一个统一的结构对象。

由一个 GetTransaction 来看，可以看到获取是从 Payload.Data 中反序列化成了一个 Transaction 对象，所以可以按照 Transaction 对象来分析

```
// Transaction
tx, err := utils.GetTransaction(payload.Data)
// GetTransaction Get Transaction from bytes
func GetTransaction(txBytes []byte) (*peer.Transaction, error) {
        tx := &peer.Transaction{}
        err := proto.Unmarshal(txBytes, tx)
        return tx, errors.Wrap(err, "error unmarshaling Transaction")
}
```

接下来介绍 Transaction 的结构体，包括一个或多个 TransactionAction，每个 TransactionAction 都将一个提案绑定到多个 actions 里，Transaction 是要被发送到排序节点的最终结果。

- Transaction 是原子性的，要么提交所有 actions，要么都不提交。
- 当 Transaction 包含一个以上的 Headers 是，那么每个 Header.Creator 字段必须意义。
- 一个客户端可以自由发布多个独立提案，每个提案都包含他们的 Header 和 Payload。
- 每个提案都被独立背书，生成一个 action，每个背书者都有签名。
- 任意数量的独立提案可能包含在一个 Transaction 中以确保是原子性的。

```
type Transaction struct {
        // Actions 是 TransactionAction 的数组
    // 为了每个 Transaction 容纳多个 acitons，必须要是数组
        Actions                             []*TransactionAction
        XXX_NoUnkeyedLiteral  struct{}
        XXX_unrecognized            []byte
        XXX_sizecache                 int32
}
```

TransactionAction 结构体，其 Payload 字段是根据 Header 中的 Type 字段来定义的，当 Type 是 ChainCode 时，它就是 ChainCodeActionPayload 的序列化；

```
// TransactionAction 绑定一个提案到它的 aciton.
// header 中的 type 字段决定了应用于账本的 action 的 type
type TransactionAction struct {
        // 交易提案 action 的 Header，Proposal 的 header
        Header []byte
        // 交易提案 action 的主要信息
    // 当 type 是 chaincode 时，它是 ChaincodeActionPayload 的序列化
        Payload                             []byte
        XXX_NoUnkeyedLiteral  struct{}
        XXX_unrecognized            []byte
        XXX_sizecache                 int32
}
```

ChainCodeActionPayload 的结构体可分成两个模块，字段 ChaincodeProposalPayload 类型的序列化信息，包含了原始调用函数的参数信息，还有 Action 列表中的 ChaincodeEndorsedAction 类型。

```
//  ChaincodeActionPayload 是当 Header 的 type 设置为 CHAINCODE 时用于赋值给
TransactionAction 的 Payload 的
type  ChaincodeActionPayload  struct  {
        //  这个字段是  ChaincodeProposalPayload  类型的序列化信息
            ChaincodeProposalPayload  []byte
        //  应用于账本的 actions 的列表
            Action                                *ChaincodeEndorsedAction
            XXX_NoUnkeyedLiteral  struct{}
            XXX_unrecognized          []byte
            XXX_sizecache               int32

    }
```

以下是 ChaincodeProposalPayload 类型的结构体，是提案的主要信息，包含两个字段，字段 Input 包含了链码调用的一些参数，TransientMap 则是包含数据比如密码材料，可以用来实现某种形式的应用级机密性。

```
//  ChaincodeProposalPayload 是提案的具体信息，当 Header 的类型是 CHAINCODE 时
//  它包含了这次调用的参数
type  ChaincodeProposalPayload  struct  {
        //  包含了这次调用的参数
            Input  []byte
        //  TransientMap 包含一些数据(例如密码材料)，可以用来实现某种形式的应用
级机密性
        //  这个字段的内容应该总是从交易中被省略，从分类账中被排除。
            TransientMap                map[string][]byte
            XXX_NoUnkeyedLiteral  struct{}
            XXX_unrecognized          []byte
            XXX_sizecache               int32
    }
```

以下是 ChaincodeEndorsedAction 的结构体，包含了一个指定的提案(Proposal)背书信息，主要有两个字段，为 ProposalResponsePayload 提案的响应信息，是一个序列化，还有 Endorsements 提案的背书，基本上是背书者在 ProposalResponsePayload 上的签名。

```
//  ChaincodeEndorsedAction  包含了一个指定提案的背书信息
type  ChaincodeEndorsedAction  struct  {
        //  提案响应信息
            ProposalResponsePayload  []byte
        //  提案的背书，基本上是背书者在 proposalResponsePayload 上的签名
            Endorsements                 []*Endorsement
            XXX_NoUnkeyedLiteral  struct{}
            XXX_unrecognized          []byte
            XXX_sizecache               int32                }
```

以下是 ProposalResponsePayload 的结构体，包含提案的对应哈希值(ProposalHash)字段，还有根据 Type 字段可以附加的扩展部分——Extension 字段。

```
type ProposalResponsePayload struct {
        // 提案相应的哈希
            ProposalHash []byte
        // 根据 Type 字段可以附加的扩展部分
            Extension                       []byte
            XXX_NoUnkeyedLiteral struct{}
            XXX_unrecognized        []byte
            XXX_sizecache             int32
    }
```

对于 Chaincode，它是 ChaincodeAction 的序列化信息，ChaincodeAction 包含了链码执行生成的 Events、Results（读写集）、Response（响应）、ChaincodeID（链码 ID）、TokenExpectation。

```
// ChaincodeAction 包含了链码执行生成的 events 的 actions
type ChaincodeAction struct {
        // 包含了链码执行的读写集
            Results []byte
        // 包含了链码调用生成的 events
            Events []byte
        // 包含了链码调用生成的 Response
            Response *Response
        // 链码 ID
            ChaincodeId *ChaincodeID
        // 包含了链码调用生成的 token expectation
            TokenExpectation        *token.TokenExpectation
            XXX_NoUnkeyedLiteral struct{}
            XXX_unrecognized        []byte
            XXX_sizecache             int32
    }
```

最后是 Endorsement 结构体，主要包含了一些背书者的身份信息以及签名，有 Endorse（背书者）字段和 Signature（签名）字段。

```
type Endorsement struct {
        // 背书者本身的身份信息
        Endorser []byte
    // 背书签名
        Signature                       []byte
        XXX_NoUnkeyedLiteral struct{}
        XXX_unrecognized        []byte
        XXX_sizecache             int32
}
```

背书节点的交易背书主要流程（过程）：

1. 检查 Proposal
2. 为交易创建模拟器，调用模拟器进行模拟执行交易，生成执行结果
3. 背书模块对执行结果和 Proposal 身份信息背书，生成背书响应发送给客户端（Client）。

Peer 接收 Proposal 并处理流程：

```
func (e *Endorser) ProcessProposal(ctx context.Context, signedProp *pb.SignedProposal) (*pb.ProposalResponse, error) {
	...
	// 0 -- check and validate
	// 这里有相当多的工作量
	vr, err := e.preProcess(signedProp)
	if err != nil {
		resp := vr.resp
		return resp, err
	}
	prop, hdrExt, chainID, txid := vr.prop, vr.hdrExt, vr.chainID, vr.txid
	// 获取指定账本模拟器
	// obtaining once the tx simulator for this proposal. This will be nil
	// for chainless proposals
	// Also obtain a history query executor for history queries, since tx simulator does not cover history
	var txsim ledger.TxSimulator
	var historyQueryExecutor ledger.HistoryQueryExecutor
	if acquireTxSimulator(chainID, vr.hdrExt.ChaincodeId) {
		if txsim, err = e.s.GetTxSimulator(chainID, txid); err != nil {
			return &pb.ProposalResponse{Response: &pb.Response{Status: 500, Message: err.Error()}}, nil
		}
		...
		defer txsim.Done()

		if historyQueryExecutor, err = e.s.GetHistoryQueryExecutor(chainID); err != nil {
			return &pb.ProposalResponse{Response: &pb.Response{Status: 500, Message: err.Error()}}, nil
		}
	}

	txParams := &ccprovider.TransactionParams{
		ChannelID:                  chainID,
		TxID:                       txid,
		SignedProp:                 signedProp,
```

```
                Proposal:                       prop,
                TXSimulator:                    txsim,
                HistoryQueryExecutor:  historyQueryExecutor,
        }
        //  this  could  be  a  request  to  a  chainless  SysCC

        //  模拟执行交易，失败则返回背书失败的响应
        //  1  --  simulate
        cd,  res,  simulationResult,  ccevent,  err  :=  e.SimulateProposal(txPa
rams,  hdrExt.ChaincodeId)
        if  err  !=  nil  {
                return  &pb.ProposalResponse{Response:  &pb.Response{Status:  50
0,  Message:  err.Error()}},  nil
        }
        if  res  !=  nil  {
                if  res.Status  >=  shim.ERROR  {
                        endorserLogger.Errorf("[%s][%s]  simulateProposal()  res
ulted  in  chaincode  %s  response  status  %d  for  txid:  %s",  chainID,  shor
ttxid(txid),  hdrExt.ChaincodeId,  res.Status,  txid)
                        var  cceventBytes  []byte
                        if  ccevent  !=  nil  {
                                cceventBytes,  err  =  putils.GetBytesChaincodeE
vent(ccevent)
                                if  err  !=  nil  {
                                        return  nil,  errors.Wrap(err,  "failed
  to  marshal  event  bytes")
                                }
                        }
                        pResp,  err  :=  putils.CreateProposalResponseFailure(pr
op.Header,  prop.Payload,  res,  simulationResult,  cceventBytes,  hdrExt.Chainc
odeId,  hdrExt.PayloadVisibility)
                        if  err  !=  nil  {
                                return  &pb.ProposalResponse{Response:  &pb.Resp
onse{Status:  500,  Message:  err.Error()}},  nil
                        }

                        return  pResp,  nil
                }
        }

        //  对模拟执行的结果进行签名背书
        //  2  --  endorse  and  get  a  marshalled  ProposalResponse  message
        var  pResp  *pb.ProposalResponse

        //  TODO  till  we  implement  global  ESCC,  CSCC  for  system  chainco
des
        //  chainless  proposals  (such  as  CSCC)  don't  have  to  be  endorse
d
```

```
        if chainID == "" {
                pResp = &pb.ProposalResponse{Response: res}
        } else {
                // Note: To endorseProposal(), we pass the released txsi
m. Hence, an error would occur if we try to use this txsim
                pResp, err = e.endorseProposal(ctx, chainID, txid, signedP
rop, prop, res, simulationResult, ccevent, hdrExt.PayloadVisibility, hdrEx
t.ChaincodeId, txsim, cd)

                ...
        }
        // Set the proposal response payload - it
        // contains the "return value" from the
        // chaincode invocation
        pResp.Response = res
        // total failed proposals = ProposalsReceived-SuccessfulProposals
        e.Metrics.SuccessfulProposals.Add(1)
        success = true
        return pResp, nil
}
```

preProcess 检查 Proposal 和获取信息

```
// preProcess checks the tx proposal headers, uniqueness and ACL
// 检查 proposal、ACL
func (e *Endorser) preProcess(signedProp *pb.SignedProposal) (*validateResu
lt, error) {
        // 包含 proposal、header、chainID、txid 等信息
        vr := &validateResult{}
        // at first, we check whether the message is valid
        // 检查 proposal，并获取各种需要的信息
        prop, hdr, hdrExt, err := validation.ValidateProposalMessage(signed
Prop)
        if err != nil {
                e.Metrics.ProposalValidationFailed.Add(1)
                vr.resp = &pb.ProposalResponse{Response: &pb.Response{Status:
 500, Message: err.Error()}}
                return vr, err
        }
        // 获取 Header 中的 2 个 Header
        chdr, err := putils.UnmarshalChannelHeader(hdr.ChannelHeader)
        if err != nil {
                vr.resp = &pb.ProposalResponse{Response: &pb.Response{Status:
 500, Message: err.Error()}}
                return vr, err
        }
        shdr, err := putils.GetSignatureHeader(hdr.SignatureHeader)
        if err != nil {
```

```
                vr.resp = &pb.ProposalResponse{Response: &pb.Response{Status:
  500, Message: err.Error()}}
                return vr, err
        }
        // 检查是否调用了不可外部（用户）的系统链码
        // 先找到链码实例，然后调用链码的方法判断本身是否可调用
        // block invocations to security-sensitive system chaincodes
        if e.s.IsSysCCAndNotInvokableExternal(hdrExt.ChaincodeId.Name) {
                endorserLogger.Errorf("Error: an attempt was made by %#v
to invoke system chaincode %s", shdr.Creator, hdrExt.ChaincodeId.Name)
                err = errors.Errorf("chaincode %s cannot be invoked throu
gh a proposal", hdrExt.ChaincodeId.Name)
                vr.resp = &pb.ProposalResponse{Response: &pb.Response{Status:
  500, Message: err.Error()}}
                return vr, err
        }
        chainID := chdr.ChannelId
        txid := chdr.TxId
        endorserLogger.Debugf("[%s][%s] processing txid: %s", chainID, shor
ttxid(txid), txid)
        if chainID != "" {
                // labels that provide context for failure metrics
                meterLabels := []string{
                        "channel", chainID,
                        "chaincode", hdrExt.ChaincodeId.Name + ":" + hdrExt
.ChaincodeId.Version,
                }

                // 检查交易是否已上链
                // Here we handle uniqueness check and ACLs for proposa
ls targeting a chain
                // Notice that ValidateProposalMessage has already verifie
d that TxID is computed properly
                if _, err = e.s.GetTransactionByID(chainID, txid); err ==
  nil {
                        // increment failure due to duplicate transactions
. Useful for catching replay attacks in
                        // addition to benign retries
                        e.Metrics.DuplicateTxsFailure.With(meterLabels...).Add(1
)
                        err = errors.Errorf("duplicate transaction found [%
s]. Creator [%x]", txid, shdr.Creator)
                        vr.resp = &pb.ProposalResponse{Response: &pb.Response
{Status: 500, Message: err.Error()}}
                        return vr, err
                }

                // 用户链码检查 ACL
```

```
                // check ACL only for application chaincodes; ACLs
                // for system chaincodes are checked elsewhere
                if !e.s.IsSysCC(hdrExt.ChaincodeId.Name) {
                        // check that the proposal complies with the Cha
nnel's writers
                        if err = e.s.CheckACL(signedProp, chdr, shdr, hdrE
xt); err != nil {
                                e.Metrics.ProposalACLCheckFailed.With(meterLabel
s...).Add(1)
                                vr.resp = &pb.ProposalResponse{Response: &pb.
Response{Status: 500, Message: err.Error()}}
                                return vr, err
                        }
                }
        } else {
                // chainless proposals do not/cannot affect ledger and c
annot be submitted as transactions
                // ignore uniqueness checks; also, chainless proposals ar
e not validated using the policies
                // of the chain since by definition there is no chain;
  they are validated against the local
                // MSP of the peer instead by the call to ValidateProp
osalMessage above
        }
        // 保存提取的各信息
        vr.prop, vr.hdrExt, vr.chainID, vr.txid = prop, hdrExt, chainID,
  txid
        return vr, nil
}
// ValidateProposalMessage checks the validity of a SignedProposal messa
ge
// this function returns Header and ChaincodeHeaderExtension messages si
nce they
// have been unmarshalled and validated
func ValidateProposalMessage(signedProp *pb.SignedProposal) (*pb.Proposal, *
common.Header, *pb.ChaincodeHeaderExtension, error) {
        if signedProp == nil {
                return nil, nil, nil, errors.New("nil arguments")
        }
        putilsLogger.Debugf("ValidateProposalMessage starts for signed propo
sal %p", signedProp)
        // extract the Proposal message from signedProp
        prop, err := utils.GetProposal(signedProp.ProposalBytes)
        if err != nil {
                return nil, nil, nil, err
        }
        // 1) look at the ProposalHeader
        hdr, err := utils.GetHeader(prop.Header)
```

```
        if err != nil {
                return nil, nil, nil, err
        }
        // validate the header
        chdr, shdr, err := validateCommonHeader(hdr)
        if err != nil {
                return nil, nil, nil, err
        }
        // 从SignatureHeader交易客户端的签名
        // validate the signature
        err = checkSignatureFromCreator(shdr.Creator, signedProp.Signature,
signedProp.ProposalBytes, chdr.ChannelId)
        if err != nil {
                // log the exact message on the peer but return a gen
eric error message to
                // avoid malicious users scanning for channels
                putilsLogger.Warningf("channel [%s]: %s", chdr.ChannelId, er
r)
                sId := &msp.SerializedIdentity{}
                err := proto.Unmarshal(shdr.Creator, sId)
                if err != nil {
                        // log the error here as well but still only r
eturn the generic error
                        err = errors.Wrap(err, "could not deserialize a S
erializedIdentity")
                        putilsLogger.Warningf("channel [%s]: %s", chdr.Channe
lId, err)
                }
                return nil, nil, nil, errors.Errorf("access denied: channe
l [%s] creator org [%s]", chdr.ChannelId, sId.Mspid)
        }
        // 检查txid的计算是否符合规则
        // Verify that the transaction ID has been computed properly.
        // This check is needed to ensure that the lookup into the l
edger
        // for the same TxID catches duplicates.
        err = utils.CheckTxID(
                chdr.TxId,
                shdr.Nonce,
                shdr.Creator)
        if err != nil {
                return nil, nil, nil, err
        }
        // 依据不同的proposal类型对proposal分别进行检查
        // continue the validation in a way that depends on the type
  specified in the header
        switch common.HeaderType(chdr.Type) {
        case common.HeaderType_CONFIG:
```

```
                //which the types are different the validation is the same
                //viz, validate a proposal to a chaincode. If we need other
                //special validation for confguration, we would have to implement
                //special validation
                fallthrough
        case common.HeaderType_ENDORSER_TRANSACTION:
                //  主要是提取 ChaincodeHeaderExtension
                // validation of the proposal message knowing it's of type CHAINCODE
                chaincodeHdrExt, err := validateChaincodeProposalMessage(prop, hdr)
                if err != nil {
                        return nil, nil, nil, err
                }
                return prop, hdr, chaincodeHdrExt, err
        default:
                //NOTE : we proably need a case
                return nil, nil, nil, errors.Errorf("unsupported proposal type %d", common.HeaderType(chdr.Type))
        }
}
```

背书节点模拟执行交易

获取模拟器

```
// Support contains functions that the endorser requires to execute its tasks
type Support interface {
        crypto.SignerSupport
        // IsSysCCAndNotInvokableExternal returns true if the supplied chaincode is
        // ia system chaincode and it NOT invokable
        IsSysCCAndNotInvokableExternal(name string) bool

        // GetTxSimulator returns the transaction simulator for the specified ledger
        // a client may obtain more than one such simulator; they are made unique
        // by way of the supplied txid
        GetTxSimulator(ledgername string, txid string) (ledger.TxSimulator, error)

}
```

```
// GetTxSimulator returns the transaction simulator for the specified ledger
// a client may obtain more than one such simulator; they are made unique
// by way of the supplied txid
func (s *SupportImpl) GetTxSimulator(ledgername string, txid string) (ledger.TxSimulator, error) {
	// 使用账本和 txid 创建模拟器，每个交易有单独的模拟器
	lgr := s.Peer.GetLedger(ledgername)
	if lgr == nil {
		return nil, errors.Errorf("Channel does not exist: %s", ledgername)
	}
	return lgr.NewTxSimulator(txid)
}
// NewTxSimulator returns new `ledger.TxSimulator`
func (l *kvLedger) NewTxSimulator(txid string) (ledger.TxSimulator, error) {
	return l.txtmgmt.NewTxSimulator(txid)
}
// NewTxSimulator implements method in interface `txmgmt.TxMgr`
func (txmgr *LockBasedTxMgr) NewTxSimulator(txid string) (ledger.TxSimulator, error) {
	logger.Debugf("constructing new tx simulator")
	s, err := newLockBasedTxSimulator(txmgr, txid)
	if err != nil {
		return nil, err
	}
	txmgr.commitRWLock.RLock()
	return s, nil
}
// 就 2 项重要的：查询执行器、读写集构建器
// LockBasedTxSimulator is a transaction simulator used in `LockBasedTxMgr`
type lockBasedTxSimulator struct {
	lockBasedQueryExecutor
	rwsetBuilder                *rwsetutil.RWSetBuilder
	writePerformed              bool
	pvtdataQueriesPerformed     bool
	simulationResultsComputed   bool
	paginatedQueriesPerformed   bool
}
func newLockBasedTxSimulator(txmgr *LockBasedTxMgr, txid string) (*lockBasedTxSimulator, error) {
	// 创建读写集构建器，能帮助构建读写集
	rwsetBuilder := rwsetutil.NewRWSetBuilder()
	helper := newQueryHelper(txmgr, rwsetBuilder)
	logger.Debugf("constructing new tx simulator txid = [%s]", txid)
```

```
        return &lockBasedTxSimulator{lockBasedQueryExecutor{helper, txid}, rw
setBuilder, false, false, false, false}, nil
}
// LockBasedQueryExecutor is a query executor used in `LockBasedTxMgr`
// "只读"，不包含写相关的操作
type lockBasedQueryExecutor struct {
        helper *queryHelper
        txid      string
}
```

执行 Endorser 的部分

```
if acquireTxSimulator(chainID, vr.hdrExt.ChaincodeId) {
        if txsim, err = e.s.GetTxSimulator(chainID, txid); err != nil {
                return &pb.ProposalResponse{Response: &pb.Response{Status: 50
0, Message: err.Error()}}, nil
        }
        defer txsim.Done()

        // 历史查询器
        if historyQueryExecutor, err = e.s.GetHistoryQueryExecutor(chainID);
  err != nil {
                return &pb.ProposalResponse{Response: &pb.Response{Status: 50
0, Message: err.Error()}}, nil
        }
}
txParams := &ccprovider.TransactionParams{
        ChannelID:                    chainID,
        TxID:                             txid,
        SignedProp:                   signedProp,
        Proposal:                      prop,
        TXSimulator:                 txsim,        // 模拟器在此
        HistoryQueryExecutor: historyQueryExecutor,
}
// this could be a request to a chainless SysCC
// TODO: if the proposal has an extension, it will be of type Chain
codeAction;
//              if it's present it means that no simulation is to be
performed because
//              we're trying to emulate a submitting peer. On the other
  hand, we need
//              to validate the supplied action before endorsing it
// 模拟执行交易，失败则返回背书失败的响应
// 1 -- simulate
cd, res, simulationResult, ccevent, err := e.SimulateProposal(txParams, h
drExt.ChaincodeId)
```

```
// SimulateProposal simulates the proposal by calling the chaincode
func (e *Endorser) SimulateProposal(txParams *ccprovider.TransactionParams,
  cid *pb.ChaincodeID) (ccprovider.ChaincodeDefinition, *pb.Response, []byte
, *pb.ChaincodeEvent, error) {
        endorserLogger.Debugf("[%s][%s] Entry chaincode: %s", txParams.Chann
elID, shorttxid(txParams.TxID), cid)
        defer endorserLogger.Debugf("[%s][%s] Exit", txParams.ChannelID, sho
rttxid(txParams.TxID))
        // we do expect the payload to be a ChaincodeInvocationSpec
        // if we are supporting other payloads in future, this be gla
ringly point
        // as something that should change
        // 根据 Proposal 生成 Invoke 需要的信息
        cis, err := putils.GetChaincodeInvocationSpec(txParams.Proposal)
        if err != nil {
                return nil, nil, nil, nil, err
        }
        // 链码的元数据
        var cdLedger ccprovider.ChaincodeDefinition
        var version string
        // 设置 version
        if !e.s.IsSysCC(cid.Name) {
                // 根据要调用的链码名称，从 lscc 获取链码的元数据
                cdLedger, err = e.s.GetChaincodeDefinition(cid.Name, txParam
s.TXSimulator)
                if err != nil {
                        return nil, nil, nil, nil, errors.WithMessage(err,
  fmt.Sprintf("make sure the chaincode %s has been successfully instanti
ated and try again", cid.Name))
                }
                version = cdLedger.CCVersion()

                // 实际被打桩了，无实现
                err = e.s.CheckInstantiationPolicy(cid.Name, version, cdLedg
er)
                if err != nil {
                        return nil, nil, nil, nil, err
                }
        } else {
                // scc 版本是固定的"latest"
                version = util.GetSysCCVersion()
        }
        // ---3. execute the proposal and get simulation results
        var simResult *ledger.TxSimulationResults
        var pubSimResBytes []byte
```

```
        var res *pb.Response
        var ccevent *pb.ChaincodeEvent
        // 模拟执行，执行结果保存在模拟器
        res, ccevent, err = e.callChaincode(txParams, version, cis.Chainco
deSpec.Input, cid)
        if err != nil {
                endorserLogger.Errorf("[%s][%s] failed to invoke chaincode
%s, error: %+v", txParams.ChannelID, shorttxid(txParams.TxID), cid, err)
                return nil, nil, nil, nil, err
        }

        if txParams.TXSimulator != nil {
                // 通过模拟器获取模拟执行结果，包含公开和私密数据 2 份读写集
                if simResult, err = txParams.TXSimulator.GetTxSimulationResu
lts(); err != nil {
                        txParams.TXSimulator.Done()
                        return nil, nil, nil, nil, err
                }
                // 存在私密数据
                if simResult.PvtSimulationResults != nil {
                        if cid.Name == "lscc" {
                                // TODO: remove once we can store collect
ion configuration outside of LSCC
                                txParams.TXSimulator.Done()
                                return nil, nil, nil, nil, errors.New("Priv
ate data is forbidden to be used in instantiate")
                        }
                        // 获取要通过 Gossip 传播的私密数据
                        pvtDataWithConfig, err := e.AssemblePvtRWSet(simResul
t.PvtSimulationResults, txParams.TXSimulator)
                        // To read collection config need to read collec
tion updates before
                        // releasing the lock, hence txParams.TXSimulator.D
one()  moved down here
                        txParams.TXSimulator.Done()

                        if err != nil {
                                return nil, nil, nil, nil, errors.WithMessa
ge(err, "failed to obtain collections config")
                        }
                        endorsedAt, err := e.s.GetLedgerHeight(txParams.Chann
elID)
                        if err != nil {
                                return nil, nil, nil, nil, errors.WithMessa
ge(err, fmt.Sprint("failed to obtain ledger height for channel", txParam
s.ChannelID))
                        }
```

```
                        // Add ledger height at which transaction was en
dorsed,
                        // `endorsedAt` is obtained from the block storag
e and at times this could be 'endorsement Height + 1'.
                        // However, since we use this height only to se
lect the configuration (3rd parameter in distributePrivateData) and
                        // manage transient store purge for orphaned priv
ate writesets (4th parameter in distributePrivateData), this works for
now.
                        // Ideally, ledger should add support in the sim
ulator as a first class function `GetHeight()`.
                        pvtDataWithConfig.EndorsedAt = endorsedAt
                        // 把私密数据同通道 id、交易 id 和区块高度发出去，代表私
密数据所属的区块和交易
                        if err := e.distributePrivateData(txParams.ChannelID,
 txParams.TxID, pvtDataWithConfig, endorsedAt); err != nil {
                                return nil, nil, nil, nil, err
                        }
                }
                // 交易模拟完成，释放模拟器占用的资源
                txParams.TXSimulator.Done()
                // 获取模拟执行的公开结果
                if pubSimResBytes, err = simResult.GetPubSimulationBytes();
 err != nil {
                        return nil, nil, nil, nil, err
                }
        }
        // 返回链码元数据、模拟执行结果、交易执行产生的事件
        return cdLedger, res, pubSimResBytes, ccevent, nil
}
```

callChaincode 调用 chaincode 模块模拟执行交易，获取交易执行的公开和私密数据读写集，以及交易执行产生的事件，并把结果返回给上层进行背书。在前面的流程中，还未区分系统链码 SCC 和用户链码 UCC，SCC 和 UCC 会通过 Execute 函数被传递给 chaincode 模块执行交易。如果是调用 lscc 部署或升级 UCC，则会调用 ExecuteLegacyInit 执行链码容器的初始化。最后返回链码模拟执行结果和事件。

```
// call specified chaincode (system or user)
func (e *Endorser) callChaincode(txParams *ccprovider.TransactionParams, ve
rsion string, input *pb.ChaincodeInput, cid *pb.ChaincodeID) (*pb.Response
, *pb.ChaincodeEvent, error) {
        ...
        // scc 也在这执行
        // is this a system chaincode
        res, ccevent, err = e.s.Execute(txParams, txParams.ChannelID, cid.
Name, version, txParams.TxID, txParams.SignedProp, txParams.Proposal, input
)
        if err != nil {
```

```
                return nil, nil, err
        }
        ...
        // 如果是调用 lscc 部署或升级链码，会走这段流程
        if cid.Name == "lscc" && len(input.Args) >= 3 && (string(input.
Args[0]) == "deploy" || string(input.Args[0]) == "upgrade") {
                userCDS, err := putils.GetChaincodeDeploymentSpec(input.Args[
2], e.PlatformRegistry)
                ...
                // 进行链码容器初始化，最后会调用链码的 Init 的函数
                _, _, err = e.s.ExecuteLegacyInit(txParams, txParams.Channe
lID, cds.ChaincodeSpec.ChaincodeId.Name, cds.ChaincodeSpec.ChaincodeId.Version
, txParams.TxID, txParams.SignedProp, txParams.Proposal, cds)
                ...
        }
        // ----- END -------
        return res, ccevent, err
}
```

Execute 函数调用

```
// Execute a proposal and return the chaincode response
func (s *SupportImpl) Execute(txParams *ccprovider.TransactionParams, cid,
  name, version, txid string, signedProp *pb.SignedProposal, prop *pb.Pro
posal, input *pb.ChaincodeInput) (*pb.Response, *pb.ChaincodeEvent, error)
  {
        cccid := &ccprovider.CCContext{
                Name:        name,
                Version:  version,
        }
        // decorate the chaincode input
        decorators := library.InitRegistry(library.Config{}).Lookup(library.De
coration).([]decoration.Decorator)
        input.Decorations = make(map[string][]byte)
        input = decoration.Apply(prop, input, decorators...)
        txParams.ProposalDecorations = input.Decorations
        return s.ChaincodeSupport.Execute(txParams, cccid, input)
}
```

Chaincode 部分

```
func (cs *ChaincodeSupport) Execute(txParams *ccprovider.TransactionParams,
  cccid *ccprovider.CCContext, input *pb.ChaincodeInput) (*pb.Response, *pb
.ChaincodeEvent, error) {
        // Invoke 得到 ChaincodeMessage
        resp, err := cs.Invoke(txParams, cccid, input)
        // 根据 ChaincodeMessage 得到 Response 和事件
```

```
        return processChaincodeExecutionResult(txParams.TxID, cccid.Name, res
p, err)
}

func (cs *ChaincodeSupport) Invoke(txParams *ccprovider.TransactionParams,
cccid *ccprovider.CCContext, input *pb.ChaincodeInput) (*pb.ChaincodeMessage
, error) {
        h, err := cs.Launch(txParams.ChannelID, cccid.Name, cccid.Version,
  txParams.TXSimulator)
        if err != nil {
                return nil, err
        }
        // 执行调用链码的交易（和链码之间的消息为
ChaincodeMessage_TRANSACTION）
        cctype := pb.ChaincodeMessage_TRANSACTION
        return cs.execute(cctype, txParams, cccid, input, h)
}
```

获取链码执行环境，调用 Launch 函数可获取链码执行环境，即用户链码容器。如果已实例化的链码在当前背书节点上，链码容器未启动，则启动链码容器，Launch 会返回一个跟链码容器交互的 Handler。

```
// Launch starts executing chaincode if it is not already running. Th
is method
// blocks until the peer side handler gets into ready state or encou
nters a fatal
// error. If the chaincode is already running, it simply returns.
func (cs *ChaincodeSupport) Launch(chainID, chaincodeName, chaincodeVersion
  string, qe ledger.QueryExecutor) (*Handler, error) {
        cname := chaincodeName + ":" + chaincodeVersion
        if h := cs.HandlerRegistry.Handler(cname); h != nil {
                return h, nil
        }
        // 启动链码容器 ...
        h := cs.HandlerRegistry.Handler(cname)
        if h == nil {
                return nil, errors.Wrapf(err, "[channel %s] claimed to st
art chaincode container for %s but could not find handler", chainID,
cname)
        }
        return h, nil
}
```

模拟执行交易

调用 execute 封装出执行交易的消息，然后使用 Handler 执行交易。

```
// execute executes a transaction and waits for it to complete until a timeout value.
func (cs *ChaincodeSupport) execute(cctyp pb.ChaincodeMessage_Type, txParams *ccprovider.TransactionParams, cccid *ccprovider.CCContext, input *pb.ChaincodeInput, h *Handler) (*pb.ChaincodeMessage, error) {
        input.Decorations = txParams.ProposalDecorations
        // 创建消息
        ccMsg, err := createCCMessage(cctyp, txParams.ChannelID, txParams.TxID, input)
        if err != nil {
                return nil, errors.WithMessage(err, "failed to create chaincode message")
        }
        // 执行交易
        ccresp, err := h.Execute(txParams, cccid, ccMsg, cs.ExecuteTimeout)
        if err != nil {
                return nil, errors.WithMessage(err, fmt.Sprintf("error sending"))
        }
        return ccresp, nil
}
```

Handler 的执行交易过程

由于链码容器在执行交易的时候会和 Peer 之间进行多次通信，进行数据的读写，上下文可以让数据读写获取到正确的信息，因此先创建交易执行的上下文 Context，再由 Handler 把消息发送到链码容器，等待链码容器发来包含执行结果的消息。

```
func (h *Handler) Execute(txParams *ccprovider.TransactionParams, cccid *ccprovider.CCContext, msg *pb.ChaincodeMessage, timeout time.Duration) (*pb.ChaincodeMessage, error) {
        chaincodeLogger.Debugf("Entry")
        defer chaincodeLogger.Debugf("Exit")
        // 私密数据
        txParams.CollectionStore = h.getCollectionStore(msg.ChannelId)
        // 是否是执行链码初始化
        txParams.IsInitTransaction = (msg.Type == pb.ChaincodeMessage_INIT)
        // 创建交易 context
        txctx, err := h.TXContexts.Create(txParams)
        if err != nil {
                return nil, err
        }
        // 退出时（执行交易完毕），释放交易上下文资源
        defer h.TXContexts.Delete(msg.ChannelId, msg.Txid)
```

```
        //  proposal 保存到 msg
        if  err  :=  h.setChaincodeProposal(txParams.SignedProp,  txParams.Propo
sal,  msg);  err  !=  nil  {
                return  nil,  err
        }
        //  向链码容器发送 msg
        h.serialSendAsync(msg)
        //  等待链码容器响应，或者超时
        var  ccresp  *pb.ChaincodeMessage
        select  {
        case  ccresp  =  <-txctx.ResponseNotifier:
                //  response  is  sent  to  user  or  calling  chaincode.  Chain
codeMessage_ERROR
                //  are  typically  treated  as  error
        case  <-time.After(timeout):
                err  =  errors.New("timeout  expired  while  executing  transact
ion")
                ccName  :=  cccid.Name  +  ":"  +  cccid.Version
                h.Metrics.ExecuteTimeouts.With(
                        "chaincode",  ccName,
                ).Add(1)
        }
        return  ccresp,  err
}
```

处理链码容器模拟响应

链码容器执行的响应会往上传递，直到 ChaincodeSupport.Execute 函数，它调用 processChaincodeExecutionResult 函数响应把链码容器返回的响应转化为交易模拟执行的 Response，而 Response 最后会返回到 Endorser。

```
func  processChaincodeExecutionResult(txid,  ccName  string,  resp  *pb.Chaincod
eMessage,  err  error)  (*pb.Response,  *pb.ChaincodeEvent,  error)  {
        if  err  !=  nil  {
                return  nil,  nil,  errors.Wrapf(err,  "failed  to  execute  tra
nsaction  %s",  txid)
        }
        if  resp  ==  nil  {
                return  nil,  nil,  errors.Errorf("nil  response  from  transact
ion  %s",  txid)
        }
        if  resp.ChaincodeEvent  !=  nil  {
                resp.ChaincodeEvent.ChaincodeId  =  ccName
                resp.ChaincodeEvent.TxId  =  txid
        }
        switch  resp.Type  {
        //  交易执行成功则提取 Payload 中保存的 Response
        case  pb.ChaincodeMessage_COMPLETED:
```

```
                res := &pb.Response{}
                err := proto.Unmarshal(resp.Payload, res)
                if err != nil {
                        return nil, nil, errors.Wrapf(err, "failed to unma
rshal response for transaction %s", txid)
                }
                return res, resp.ChaincodeEvent, nil
        //    失败，则提取 Payload 中保存的错误信息
        case pb.ChaincodeMessage_ERROR:
                return nil, resp.ChaincodeEvent, errors.Errorf("transaction
returned with failure: %s", resp.Payload)
        default:
                return nil, nil, errors.Errorf("unexpected response type %
d for transaction %s", resp.Type, txid)
        }
}
```

释放模拟器资源

在 Endorser.SimulateProposal 中，它获取了交易模拟执行器 TXSimulator，这里可是有很多的资源，如果不及时释放，在高 TPS 下，Peer 压力大，资源泄漏，性能降低等问题会暴露出来。txParams.TXSimulator.Done()用于释放资源，主要是释放查询操作相关的资源。

ESCC 处理模拟执行结果

Endorser 调用 ESCC 对结果进行背书，最终生成 ProposalResponse

```
func (e *Endorser) ProcessProposal(ctx context.Context, signedProp *pb.Sig
nedProposal) (*pb.ProposalResponse, error) {
        // Pre-process, simulate
        if chainID == "" {
                pResp = &pb.ProposalResponse{Response: res}
        } else {
                // Note: To endorseProposal(), we pass the released txsi
m. Hence, an error would occur if we try to use this txsim
                pResp, err = e.endorseProposal(ctx, chainID, txid, signedP
rop, prop, res, simulationResult, ccevent, hdrExt.PayloadVisibility, hdrEx
t.ChaincodeId, txsim, cd)
                // ...
        }
        // ...
}
```

endorseProposal

背书链码实现了可插拔，可以使用不同的 ESCC，系统链码和用户链码的背书过程是不同的。

```
// endorse the proposal by calling the ESCC
func (e *Endorser) endorseProposal(_ context.Context, chainID string, txi
d string, signedProp *pb.SignedProposal, proposal *pb.Proposal, response
*pb.Response, simRes []byte, event *pb.ChaincodeEvent, visibility []byte,
  ccid *pb.ChaincodeID, txsim ledger.TxSimulator, cd ccprovider.ChaincodeDe
finition) (*pb.ProposalResponse, error) {
        endorserLogger.Debugf("[%s][%s] Entry chaincode: %s", chainID, shor
ttxid(txid), ccid)
        defer endorserLogger.Debugf("[%s][%s] Exit", chainID, shorttxid(txid
))
        // 系统链码和用户链码使用不同的 ESCC
        isSysCC := cd == nil
        // 1) extract the name of the escc that is requested to endo
rse this chaincode
        var escc string
        // ie, "lscc" or system chaincodes
        if isSysCC {
                escc = "escc"
        } else {
                escc = cd.Endorsement()
        }
        endorserLogger.Debugf("[%s][%s] escc for chaincode %s is %s", cha
inID, shorttxid(txid), ccid, escc)
        // marshalling event bytes
        var err error
        var eventBytes []byte
        if event != nil {
                eventBytes, err = putils.GetBytesChaincodeEvent(event)
                if err != nil {
                        return nil, errors.Wrap(err, "failed to marshal ev
ent bytes")
                }
        }
        // set version of executing chaincode
        if isSysCC {
                // if we want to allow mixed fabric levels we should
                // set syscc version to ""
                ccid.Version = util.GetSysCCVersion()
        } else {
                ccid.Version = cd.CCVersion()
        }
        // 创建背书上下文信息
        ctx := Context{
                PluginName:          escc, // 插件名称
                Channel:               chainID,
```

```
                SignedProposal:  signedProp,
                ChaincodeID:        ccid,
                Event:                  eventBytes,
                SimRes:                simRes,
                Response:             response,
                Visibility:          visibility,
                Proposal:             proposal,
                TxID:                    txid,
        }
        //  调用插件背书
        return  e.s.EndorseWithPlugin(ctx)
}
```

背书插件实现以下的接口

```
//  Plugin  endorses  a  proposal  response
type  Plugin  interface  {
        //  Endorse  signs  the  given  payload(ProposalResponsePayload  bytes),
  and  optionally  mutates  it.
        //  Returns:
        //  The  Endorsement:  A  signature  over  the  payload,  and  an  ident
ity  that  is  used  to  verify  the  signature
        //  The  payload  that  was  given  as  input  (could  be  modified  wit
hin  this  function)
        //  Or  error  on  failure
        Endorse(payload  []byte,  sp  *peer.SignedProposal)  (*peer.Endorsement,
  []byte,  error)

        //  Init  injects  dependencies  into  the  instance  of  the  Plugin
        Init(dependencies  ...Dependency)  error
}
```

使用背书插件，需要获取插件实例，然后组装响应 Payload，包含了交易执行的多种结果，然后对 Payload 以及签名的 Proposal 背书

```
//  EndorseWithPlugin  endorses  the  response  with  a  plugin
func  (pe  *PluginEndorser)  EndorseWithPlugin(ctx  Context)  (*pb.ProposalRespo
nse,  error)  {
        endorserLogger.Debug("Entering  endorsement  for",  ctx)
        if  ctx.Response  ==  nil  {
                return  nil,  errors.New("response  is  nil")
        }
        if  ctx.Response.Status  >=  shim.ERRORTHRESHOLD  {
                return  &pb.ProposalResponse{Response:  ctx.Response},  nil
        }
        //  获取插件
        plugin,  err  :=  pe.getOrCreatePlugin(PluginName(ctx.PluginName),  ctx.
Channel)
```

```
        if err != nil {
                endorserLogger.Warning("Endorsement with plugin for", ctx,
" failed:", err)
                return nil, errors.Errorf("plugin with name %s could not
 be used: %v", ctx.PluginName, err)
        }
        // 把模拟执行的信息组成生成背书响应 Payload
        prpBytes, err := proposalResponsePayloadFromContext(ctx)
        if err != nil {
                endorserLogger.Warning("Endorsement with plugin for", ctx,
" failed:", err)
                return nil, errors.Wrap(err, "failed assembling proposal r
esponse payload")
        }
        // 对 Payload 和签名的 Proposal 进行背书
        endorsement, prpBytes, err := plugin.Endorse(prpBytes, ctx.SignedPr
oposal)
        if err != nil {
                endorserLogger.Warning("Endorsement with plugin for", ctx,
" failed:", err)
                return nil, errors.WithStack(err)
        }
        resp := &pb.ProposalResponse{
                Version:     1,
                Endorsement: endorsement,
                Payload:     prpBytes,
                Response:    ctx.Response,
        }
        endorserLogger.Debug("Exiting", ctx)
        return resp, nil
}
```

系统提供的默认的背书插件如下，对交易执行结果和 Proposal 签名人信息进行签名。

```
// Endorse signs the given payload(ProposalResponsePayload bytes), and o
ptionally mutates it.
// Returns:
// The Endorsement: A signature over the payload, and an identity tha
t is used to verify the signature
// The payload that was given as input (could be modified within thi
s function)
// Or error on failure
func (e *DefaultEndorsement) Endorse(prpBytes []byte, sp *peer.SignedPropo
sal) (*peer.Endorsement, []byte, error) {
        // 提取 Proposal 的签名人
        signer, err := e.SigningIdentityForRequest(sp)
        if err != nil {
```

```
                return nil, nil, errors.New(fmt.Sprintf("failed fetching si
gning identity: %v", err))
        }
        // 得到签名人身份
        // serialize the signing identity
        identityBytes, err := signer.Serialize()
        if err != nil {
                return nil, nil, errors.New(fmt.Sprintf("could not serializ
e the signing identity: %v", err))
        }
        // 对 Payload 和身份进行签名
        // sign the concatenation of the proposal response and the ser
ialized endorser identity with this endorser's key
        signature, err := signer.Sign(append(prpBytes, identityBytes...))
        if err != nil {
                return nil, nil, errors.New(fmt.Sprintf("could not sign th
e proposal response payload: %v", err))
        }
        endorsement := &peer.Endorsement{Signature: signature, Endorser: id
entityBytes}
        return endorsement, prpBytes, nil
}
```

发送 Response

最后调用 ProcessProposal 把 ProposalResponse 作为返回值，剩下的就交给 gRPC，发送给请求放了。

交易提交的过程

Peer 启动后在后台执行 gossip 服务。过程主要分三个阶段来实现。

1. 提交前准备

   主要完成对区块中交易格式的检查、获取关联该区块但缺失的私密数据、最后构建 blockAndPvtData 结构。格式检查包括检查交易格式、账本是否存在、是否满足 VSCC 和 Policy 等。获取缺失的私密数据根据已有的私密数据计算区块中交易关联的读写集信息，若有缺失则尝试从其他节点获取。最后构建 blockAndPvtData 结构，用于后续的提交工作，需要包括相关的区块和私密数据。

2. 提交过程

   过程包括预处理、验证交易、更新本地区块链结构、更新本地数据库结构。

   - 预处理模块负责构造一个有效的内部区块结构包括处理 Endorser 交易（保留有效的 Endorser 交易）、处理配置交易（获取配置更新的模拟结果，放入写读集）、校验写集合。

- 验证交易是对区块中交易进行 MVCC 检查并校验私密读写集（再次检查哈希值是否匹配），更新区块元数据中的交易有效标记列表。MVCC 检查需要逐个验证块中的 Endorser 交易，需要满足一些条件才被认为有效：公共读集合中 key 版本在该交易前未变、RangeQuery 的结果未变、私密读集合中 key 版本未变。
- 更新本地区块链结构：将区块写入本地 Chunk 文件、更新索引数据库（区块号、哈希值、文件指针、交易偏移、区块元数据）、更新所提交的区块号到私密数据库。
- 更新本地数据库结构：删除过期私密数据、更新私密数据的生命周期记录数据库、更新本地公共状态数据库和私密状态数据库、若启用历史数据库则更新数据。

3. 提交后处理

提交后的处理阶段比较简易，只需清理本地的临时状态数据库和更新账本高度信息。清理工作包括区块关联的临时私密数据和旧区块关联的临时私密数据。

参考资料：

https://blog.csdn.net/lvyibin890/article/details/106236077

https://blog.csdn.net/m0_43499523/article/details/104882106

大部分内容都是看参考资料写的，代码和图片也是从网页上截取的。

姓名：姚熙源

学号：3190300677