Fabric排序服务插件实践

陈楷

2018年8月

01

Fabric排序服务

排序服务对外的接口


```
service AtomicBroadcast {
    rpc Broadcast(stream common.Envelope) returns (stream BroadcastResponse) {}

    rpc Deliver(stream common.Envelope) returns (stream DeliverResponse) {}
}
```

排序服务内部的接口


```
type Consenter interface {
    HandleChain(support ConsenterSupport, metadata *cb.Metadata) (Chain, error)
}

type Chain interface {
    Order(env *cb.Envelope, configSeq uint64) error

    Configure(config *cb.Envelope, configSeq uint64) error

    WaitReady() error

    Errored() <-chan struct{}

    Start()

    Halt()
}
```

排序服务内部的接口

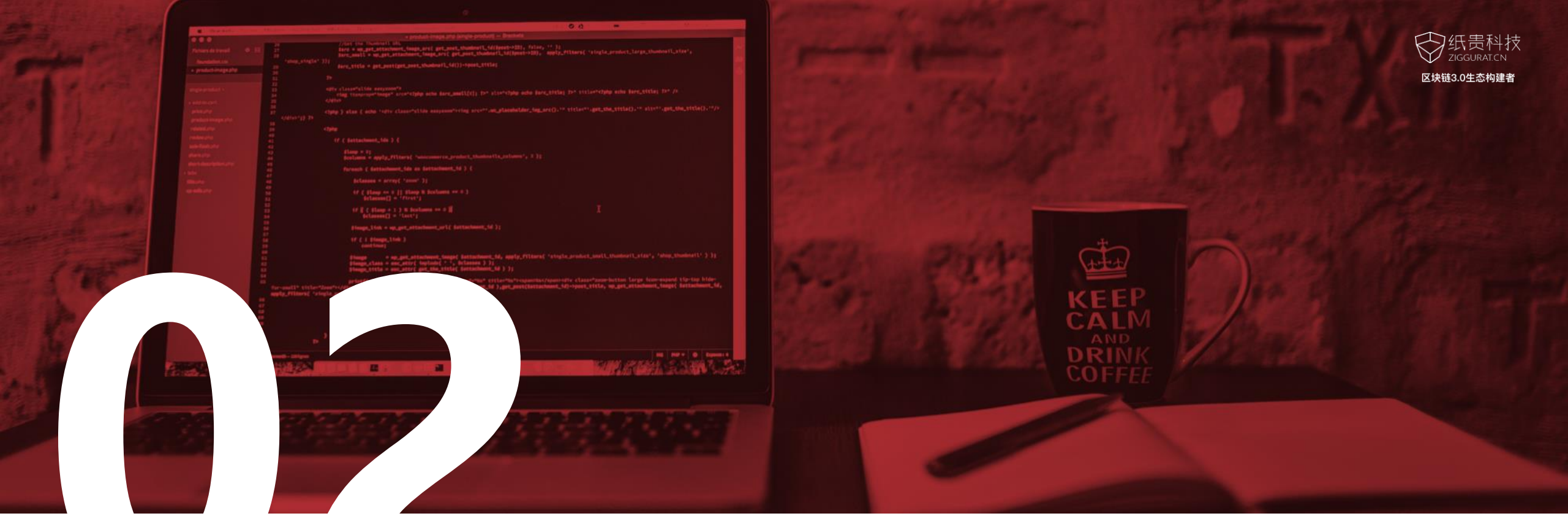

```
type ConsenterSupport interface {
    crypto.LocalSigner
    msgprocessor.Processor

    BlockCutter() blockcutter.Receiver

    SharedConfig() channelconfig.Orderer

    CreateNextBlock(messages []*cb.Envelope) *cb.Block

    WriteBlock(block *cb.Block, encodedMetadataValue []byte)

    WriteConfigBlock(block *cb.Block, encodedMetadataValue []byte)

    Sequence() uint64

    ChainID() string

    Height() uint64
}
```

排序服务处理的消息类型

- 普通交易
- 配置交易
- 定时切块（only in multiple orderers）

排序服务的处理流程

- orderer启动或者重启时创建所有的consenter插件并注册到registry
- 每创建一个channel都会创建一个chain服务，用于处理这条channel的消息
- 消息首先经由broadcast服务接收，然后会根据当前的配置对消息进行验证；如果不满足会直接拒绝，如果满足会将当前消息和当前seq传入chain服务
- chain服务会对消息进行排序，拿到全局排过序的消息后会将seq和当前seq进行对比，如果seq发生过变化，则需要对消息再进行一次验证。验证不通过的会重新提交去重新排序。
- 排好序的消息当满足任一出块条件时，会利用consentersupport来创建新的块，然后写到orderer的账本



02

Tendermint 介绍

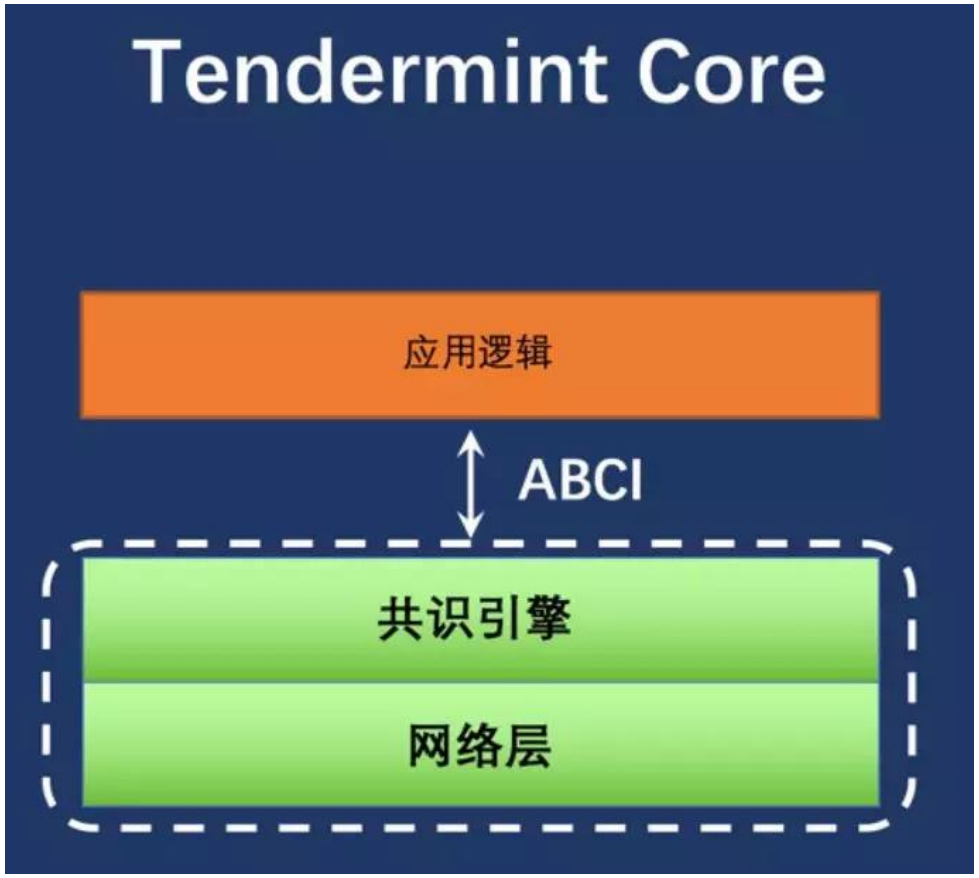
Tendermint 介绍

Tendermint： 将共识引擎和 P2P 网络层，与区块链应用状态层解耦。将区块链所要完成的应用逻辑抽象为 interface，即基于socket 的 ABCI。

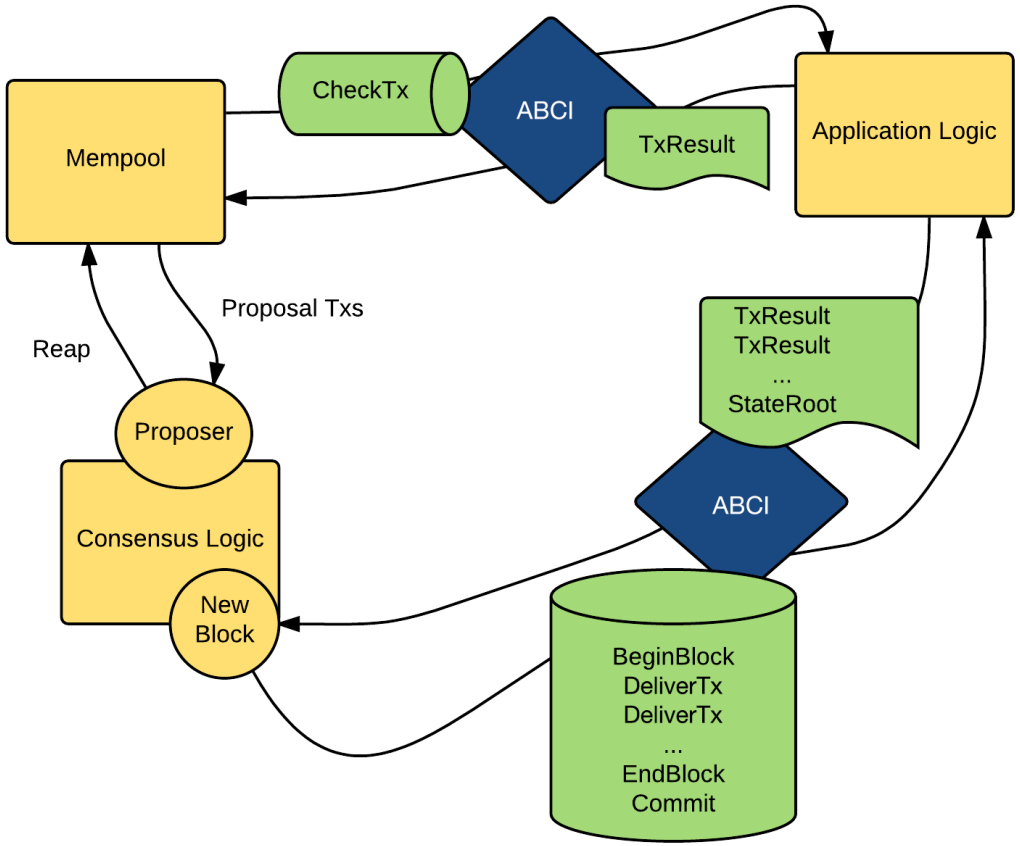
Tendermint Core: 共识引擎

- 在节点间共享交易和区块
- 维护一个不可篡改的交易顺序（区块链）

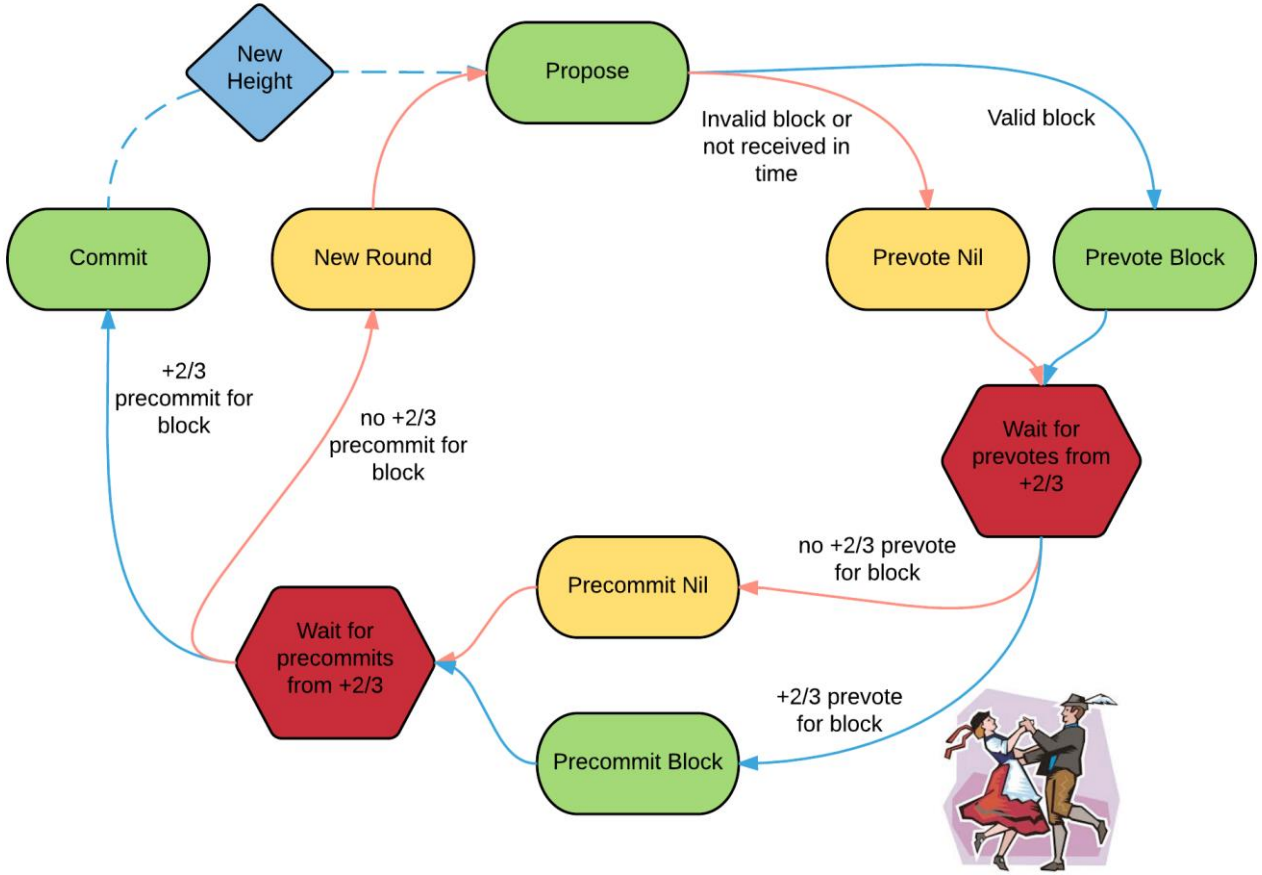
ABCI: Application BlockChain Interface



Tendermint 工作原理



ABCI message flow



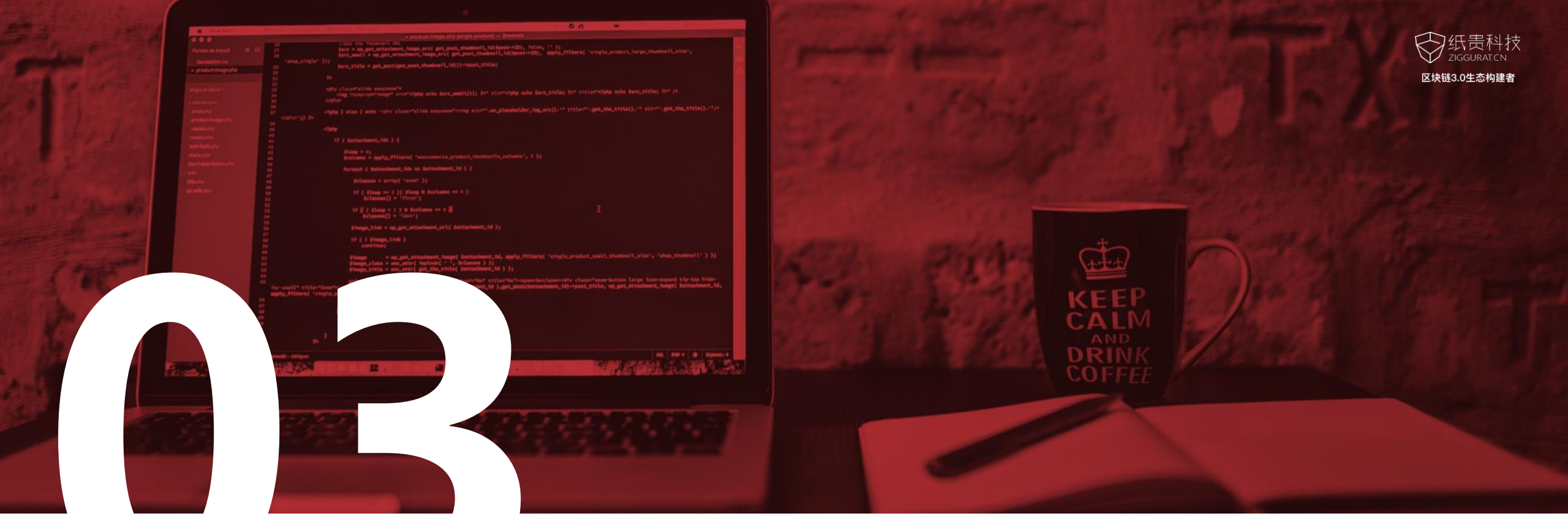
Consensus Overview

Tendermint 工作原理

- Propose阶段，本轮的proposer会通过gossip协议发送proposal给其他节点，接受到proposal的节点同样也会通过gossip协议转发proposal给其他节点。
- Prevote阶段，所有的节点会独立生成自己的prevote，包括然后通过gossip协议发送给其他节点。
- Precommit阶段，同样所有的节点会独立生成自己的precommit，当接受到超过2/3的接受本轮proposal的prevote时，将通过gossip协议将precommit发送给其他节点，否则不会发送precommit。当在指定时间内收到超过2/3的precommit后，进入Commit阶段，否则会重新回到Propose阶段，进入下一轮的投票。
- Commit阶段，需要同时满足两个条件，第一是节点需要已经接收到当前待提交的block；第二是收到至少2/3的precommit，就可以提交block到账本中。

Tendermint abci 程序接口


```
func (BaseApplication) Info(req RequestInfo) ResponseInfo {
	return ResponseInfo{}
}

func (BaseApplication) DeliverTx(tx []byte) ResponseDeliverTx {
	return ResponseDeliverTx{Code: CodeTypeOK}
}

func (BaseApplication) CheckTx(tx []byte) ResponseCheckTx {
	return ResponseCheckTx{Code: CodeTypeOK}
}

func (BaseApplication) Commit() ResponseCommit {
	return ResponseCommit{}
}

func (BaseApplication) BeginBlock(req RequestBeginBlock) ResponseBeginBlock {
	return ResponseBeginBlock{}
}

func (BaseApplication) EndBlock(req RequestEndBlock) ResponseEndBlock {
	return ResponseEndBlock{}
}
```



03

实现Tendermint排序插件

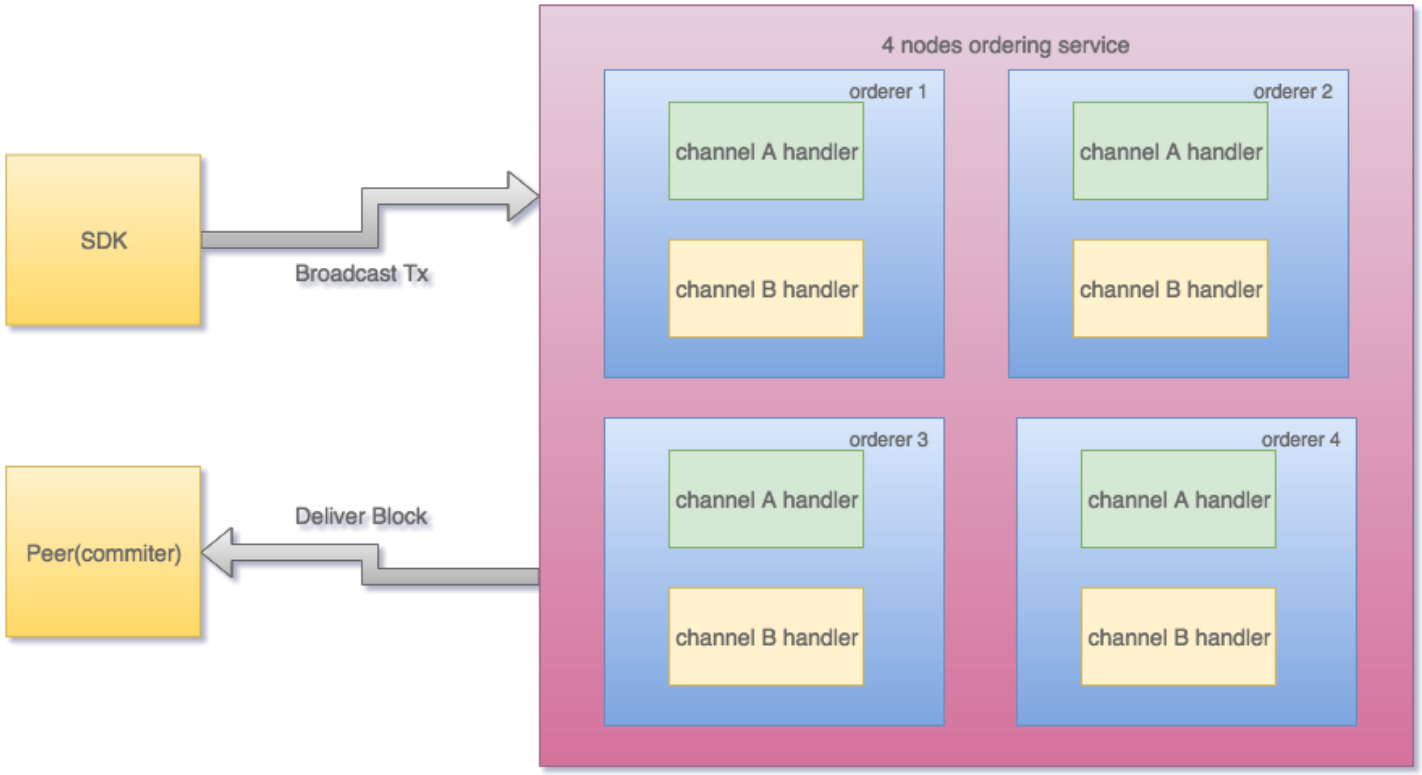
设计考虑

- 一个tendermint服务对应一个通道还是多个通道?
- 直接用tendermint的blockchain作为orderer的账本?
- 怎么处理fabric配置交易?

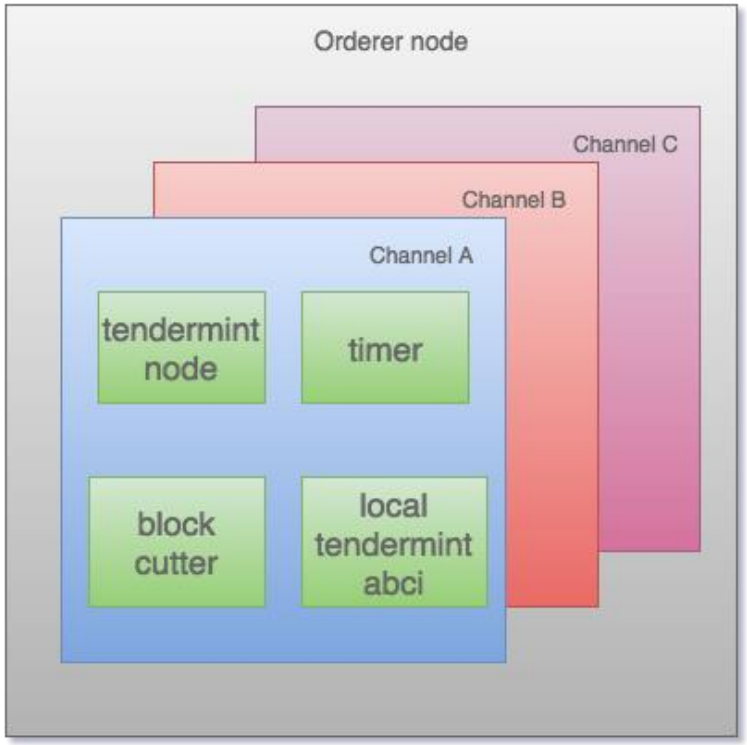
目前的选择

- 一个tendermint服务对应一个channel
- 将orderer的账本数据作为tendermint abci程序的状态数据库，创建一个tendermint abci app来处理orderer的账本
- 沿用之前的策略，每个配置交易还是单独创建一个区块。

Tendermint 排序服务组件

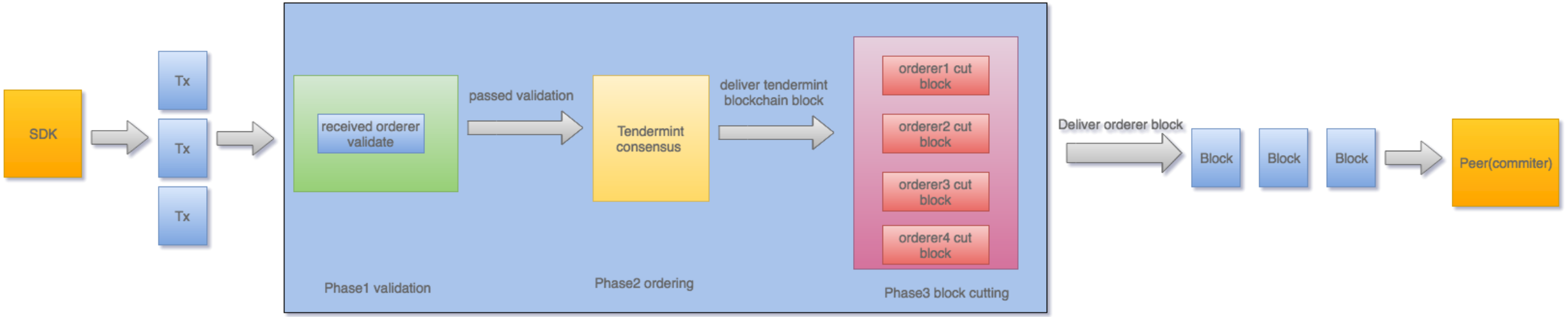


Ordering Service



A Tendermint Orderer node

Tendermint 排序服务流程



Tendermint-based Ordering Service

Tendermint 排序插件配置


```
ChannelTendermintGenesis:
        testchainid:
            # GenesisTime: tendermint consensus network created by
            GenesisTime: 0001-01-01T00:00:00Z

            # Validators: tendermint consensus network validators
            Validators:
                # one validator sample
                -  &validator
                  # PubKey: validator public key
                  PubKey:
034CA96173D8E5DB241EEC871BB28FCE10609AB4791EDCEA9598D9C6E2FDA414
                  # P2PLaddr: validator p2p listen address
                  P2PLaddr: tcp://0.0.0.0:46656
                  # RpcLaddr: validator rpc listen address
                  RpcLaddr: tcp://0.0.0.0:46657
```

Tendermint 排序插件protobuf定义


```
message TendermintOffset {
    int64 height = 1;
    int64 index = 2;
}

message TendermintMessage {
    oneof Type {
        TendermintMessageRegular regular = 1;
        TendermintMessageTimeToCut time_to_cut = 2;
    }
}

message TendermintMessageRegular {
    enum Class {
        NORMAL = 0;
        CONFIG = 1;
    }
    bytes payload = 1;
    uint64 config_seq = 2;
    Class class = 3;
}

message TendermintMessageTimeToCut {
    uint64 block_number = 1;
}

message TendermintMetadata {
    TendermintOffset last_offset_persisted = 1;
```

Tendermint 排序插件Consenter接口实现


```
func (consenter *consenter) HandleChain(support consensus.ConsenterSupport,
metadata *cb.Metadata, header *cb.BlockHeader) (consensus.Chain, error) {
    lastOffsetPersisted := getOffsets(metadata.Value, support.ChainID())
    lastBlockDataHash := header.DataHash
    return newChain(consenter.rootDir, support, lastOffsetPersisted, lastBlockDataHash)
}
```

Tendermint 排序插件Chain接口实现


```
func newChain(rootDir string, support consensus.ConsenterSupport, lastOffsetPersisted *ab.TendermintOffset, lastBlockDataHash
[]byte) (*chain, error) {
	lastCutBlockNumber := getLastCutBlockNumber(support.Height())
	logger.Infof("[channel: %s] Starting chain with last persisted offset %d and last recorded block %d",
		support.ChainID(), lastOffsetPersisted, lastCutBlockNumber)

	chainSupport := newChainSupport(support, lastOffsetPersisted, lastCutBlockNumber)
	ordererApp := newOrdererApp(chainSupport)

	tmnConf, privValidator, validators := getConfig(path.Join(rootDir, support.ChainID()), support)
	genesisDocProvider := newGenesisDocProvider(support, validators,
util.ComputeSHA256([]byte(strconv.Itoa(int(lastCutBlockNumber)))))

	tmLogger, err := tmflags.ParseLogLevel(tmnConf.LogLevel, tmLogger, tmcfg.DefaultLogLevel())
	tmLogger = tmLogger.With("module", "main")

	node, err := tmnode.NewNode(tmnConf,
		privValidator,
		tmproxy.NewLocalClientCreator(ordererApp),
		genesisDocProvider,
		tmnode.DefaultDBProvider,
		tmLogger)
	if err != nil {
		return nil, err
	}

	localClient := tmclient.NewLocal(node)
	return &chain{chainSupport: chainSupport, node: node,
		localClient: localClient,
		haltChan:    make(chan struct{}),
		startChan:   make(chan struct{})}, nil
}
```

Tendermint 排序插件tendermint abci接口实现


```
type OrdererApp struct {
    types.BaseApplication
    chain          *chainSupport
    previousOffset *ab.TendermintOffset
    currentOffset  *ab.TendermintOffset
}
```

Tendermint 排序插件社区回馈

https://jira.hyperledger.org/browse/FAB-8643

https://gerrit.hyperledger.org/r/#/c/24737/

THANKS!

纸贵科技
ZIGGURAT.CN
区块链3.0生态构建者