

区块链零基础入门 Tendermint，**你的**第一个 区块链应用程序



齐划一



qi@huayi.email



山东大学



内容提要

- 区块链的基本概念
- Tendermint 的应用程序接口
- 动手编写你的第一个区块链应用程序
- 接下来学什么？

1

什么是区块链



区块链

多节点运行的、事务型的、公开的数据库



事务型的数据库

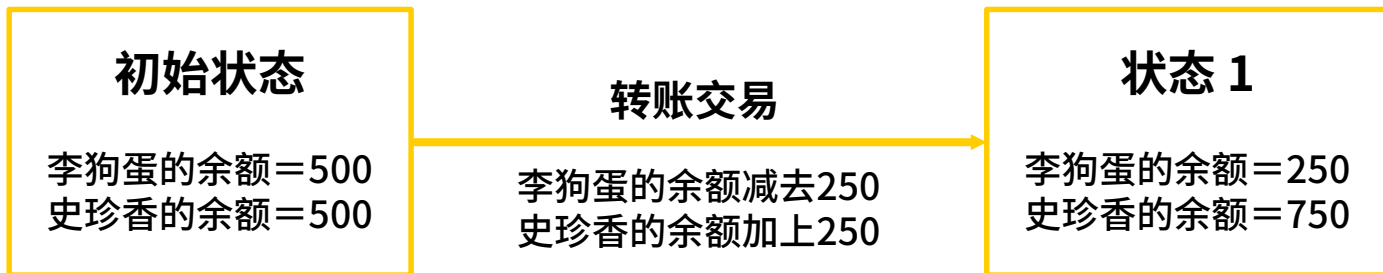


下一个状态 = 当前状态 \otimes 下一个交易



事务型的数据库

例：账户余额模型。典型应用：以太坊。





如何证明发起转账交易的是我本人？



数字签名



如何证明发起转账交易的是我本人？

- 密码  数据库和交易内容均为公开可见的。
- 数字签名 

每个用户拥有一组密钥，分为公钥和私钥。

- 私钥（仅自己可见）：解密数据、对指定内容生成数字签名
- 公钥（所有人可见）：加密数据、验证数字签名的正确性



数字签名

交易内容	发起者	签名
李狗蛋向史珍香转账250	李狗蛋	数字签名



哈希值

数字签名 = 哈希值 \otimes 私钥



数字签名

交易内容	发起者	签名
李狗蛋向史珍香转账250	李狗蛋	数字签名



哈希值

数字签名 = 哈希值 \otimes 私钥



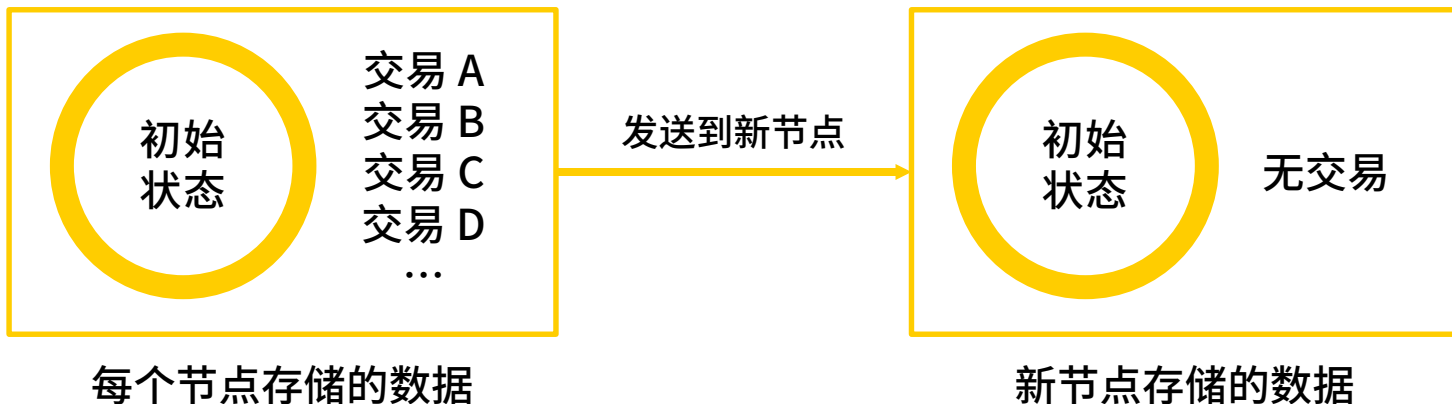
如何防止“重放攻击”？

留作习题。



多节点运行的数据库

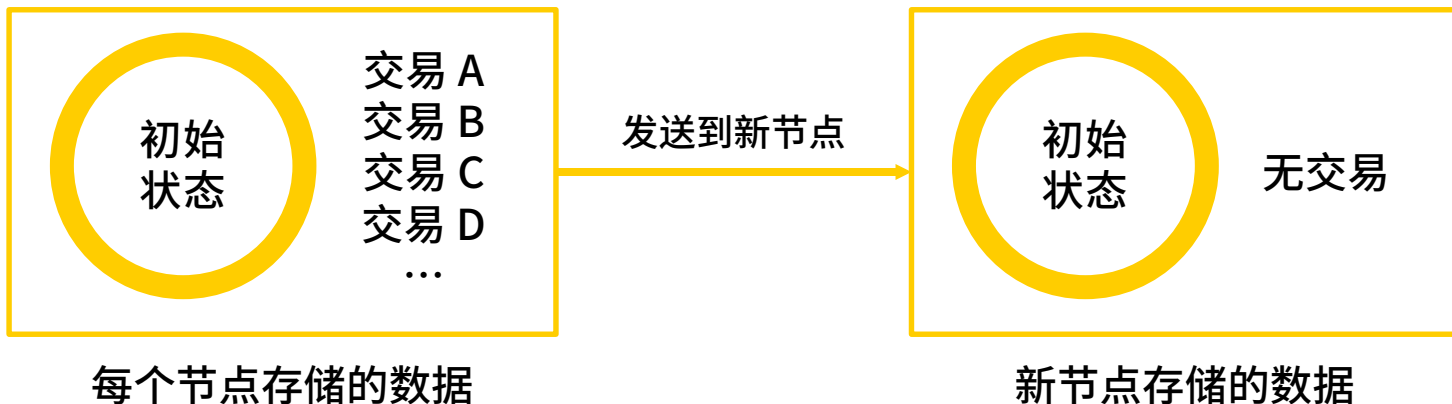
假设各节点已经通过 P2P 技术连接。





多节点运行的数据库

假设各节点已经通过 P2P 技术连接。



我为什么要相信你发送的数据？

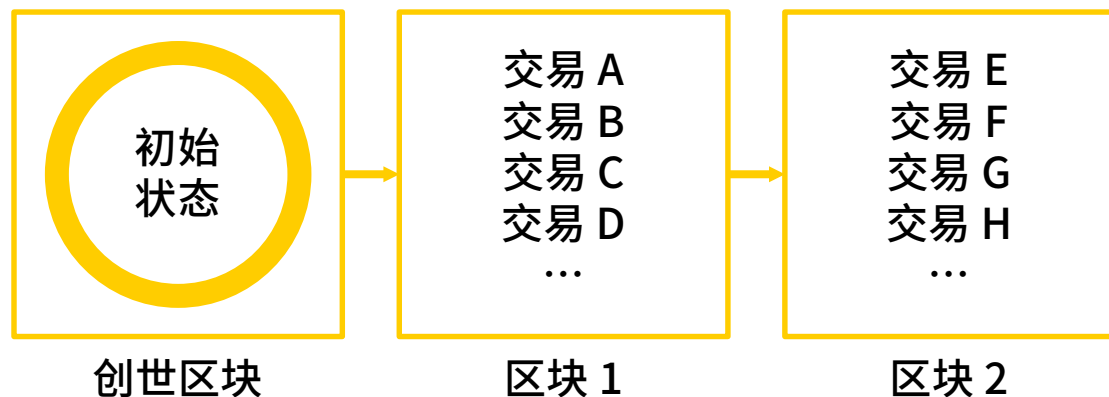


无法达成共识！



多节点运行的数据库

引入“区块”的概念。



规定，要想成为某区块的下一区块，必须符合特定条件，否则无效。



多节点运行的数据库

比特币（公有链）

节点遍布全球，人人可加入。

① 相邻区块之间满足以下关系：

“特定内容+？”的哈希 < 特定值

其中，特定内容和特定值可从当前区块获得，下一区块需要提供一个解“？”；

② 若两节点的数据不一致，区块数量多的有效。

工作量证明

Tendermint（私有链）

节点数量较少，不可随意加入。

每次产生新区块时：

- ① 各节点轮流当提议者；
- ② 提议者生成新区块；
- ③ 各权威节点投票，表决通过。

权益证明



总结

区块链：

- 事务型数据库；
- 每个交易产生一个新的状态；
- 交易经过精心设计，能够避免各种安全问题；
- 多个交易组成一个区块，区块的产生要满足一定的条件；
- 每个节点均保存了数据库的全部状态，即多个区块。
创世区块包含了初始状态，其余的区块包含了多个交易；
- 每个节点保存的数据是一致的。



使用 Tendermint 开发

Tendermint 帮你做到：

- ✓ ● 事务型数据库；
- ✓ ● 每个交易产生一个新的状态；
- ✓ ● 交易经过精心设计，能够避免各种安全问题；
- ✓ ● 多个交易组成一个区块，区块的产生要满足一定的条件；
- ✓ ● 每个节点均保存了数据库的全部状态，即多个区块。
创世区块包含了初始状态，其余的区块包含了多个交易；
- ✓ ● 每个节点保存的数据是一致的。



2

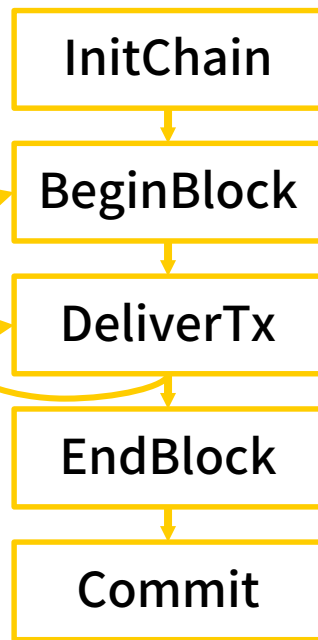
Tendermint 应用程序接口



Tendermint ABCI

Tendermint 提供了“应用程序接口”（ABCI），见[这里](#)。ABCI 定义了一些方法，我们需要实现。

- InitChain —— 根据创世区块的数据，初始化区块链节点。
- BeginBlock —— 新的区块开始时被调用。
- CheckTx —— 检查交易是否合规（注意，不可靠）。
- DeliverTx —— 执行交易，但不得更改区块链节点数据。
- EndBlock —— 区块结束时被调用。
- Commit —— 根据之前执行的交易更新区块链节点数据。
- Info —— 返回区块链节点的基本信息，如区块高度等。

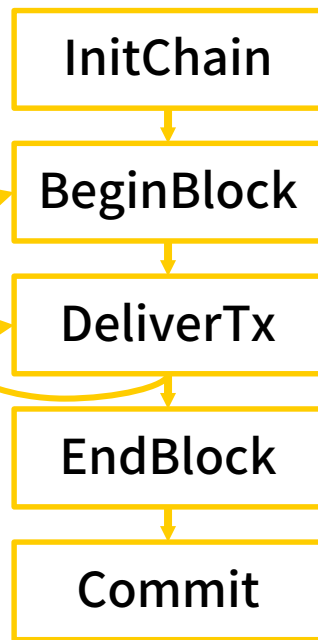




例：账户模型

全局变量：一个账户→余额的字典类型，名为 balance。

- InitChain：初始化 balance，设置账户的初始余额。
- BeginBlock：将 balance 复制一份，名为 new_balance。
- CheckTx：检查交易的签名是否为付款人的有效签名、余额是否充足等。
- DeliverTx：重复 CheckTx 的检查过程。再根据交易的具体内容修改 new_balance 中两个账户对应的余额。
- EndBlock：什么都不做。
- Commit：用 new_balance 覆盖 balance。



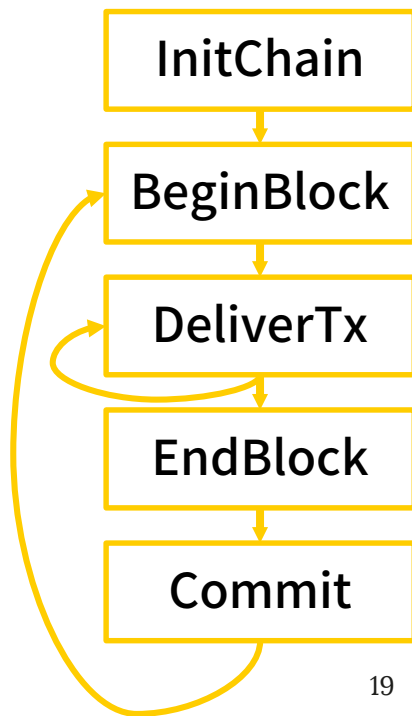


习题：实名投票

创世区块中已经给定被选举人（包含编号）的集合、选举人（包含公钥、投票权重）的集合。每个选举人仅可投票一次，仅能向一人投票，投票后不能反悔。实现该 ABCI。

- InitChain
- BeginBlock
- CheckTx
- DeliverTx
- EndBlock

- Commit
- Info：除返回区块链基本信息外，还要返回当前的最高票得主集合。





更多的应用

要做到更炫酷的事情，首先要有想法，其次要有合适的密码学工具。

- 匿名钱包：ZeroCash。使用了零知识证明。
- 公证：保存文件的哈希值和当前时间。
- 供应链：保存商品每一个生产环节的信息。由于数据量过大，使用默克尔树，在链下保存数据，链上保存默克尔根。
- 可信随机数：根据上一区块的数据，哈希得到新的随机数种子。再通过伪随机数算法得到可信的随机数。
- 域名系统：保存域名数据库，包括过期时间、所有者的公钥。HTTPS 证书的签发不再需要 CA，只需要所有者签名即可。



注意

每个交易的执行动作必须可预测，不能引入随机性。例如：

- 编程语言的未定义行为，例如 Go 语言的 map 类型的循环顺序。
- 存在多种合理的结果，例如转换为 JSON 的结果是不确定的。**[注]**
- 使用了多线程、锁等特性，执行结果很可能是不确定的。
- 引入了时间、不确定的随机数种子等。
- 数据是从网络上获取的。

网游开发、区块链开发的共同点：一切行为可预测。



注意

每个交易的执行动作必须可预测，不能引入随机性。例如：

- 编程语言的未定义行为，例如 Go 语言的 map 类型的循环顺序。
- 存在多种合理的结果，例如转换为 JSON 的结果是不确定的。**[注]**
- 使用了多线程、锁等特性，执行结果很可能是不确定的。
- 引入了时间、不确定的随机数种子等。
- 数据是从网络上获取的。

网游开发、区块链开发的共同点：一切行为可预测。

[注] 山东大学校园网登录过程中假设了 `JSON.stringify()` 的结果是唯一的，而事实上不是如此。这会导致小众的浏览器无法登录。📺

3

第一个 Tendermint 应用程序



编译 Tendermint

- 在虚拟机或实体机上安装一个 Linux 发行版。例如，Arch Linux。
- 安装 Go 语言编译器（1.14 或更新版）、基础编译工具（make 等）。
- 克隆 Tendermint 的源码。
`git clone https://github.com/tendermint/tendermint.git`
- 编译 Tendermint。
`make build`
- 运行 Tendermint。
`cd build`
`./tendermint <参数>`



编译 Tendermint

- 在虚拟机或实体机上安装一个 Linux 发行版。例如，Arch Linux。
- 安装 Go 语言编译器（1.14 或更新版）、基础编译工具（make 等）。
- 克隆 Tendermint 的源码。
`git clone https://github.com/tendermint/tendermint.git`
- 编译 Tendermint。
`make build`
- 运行 Tendermint。
`cd build`
`./tendermint <参数>`

中国大陆用户

- 使用 Gitee 克隆 GitHub 上的代码。
- 使用 goproxy.cn 设置 Go Proxy。



编写 ABCI 应用程序

- 通过 JetBrains Toolbox 安装 JetBrains GoLand 以编辑 Go 语言代码。使用学生邮箱获取免费的教育版使用资格。
- 打开 Tendermint 源代码。找到 go.mod 文件，将前几行修改为如下内容。

```
module github.com/tendermint/tendermint  
go 1.14  
replace github.com/tendermint/tendermint => ./
```

```
require (  
后面的不做修改
```



编写 ABCI 应用程序

- 通过 JetBrains Toolbox 安装 JetBrains GoLand 以编辑 Go 语言代码。使用学生邮箱获取免费的教育版使用资格。
- 打开 Tendermint 源代码。
- 找到 `abci/types/application.go` 文件，其中的 Application 即为需要实现的接口。如下所示。

```

type Application interface {
    // Info/Query Connection
    Info(RequestInfo) ResponseInfo           // Return application info
    SetOption(RequestSetOption) ResponseSetOption // Set application option
    Query(RequestQuery) ResponseQuery         // Query for state

    // Mempool Connection
    CheckTx(RequestCheckTx) ResponseCheckTx // Validate a tx for the mempool

    // Consensus Connection
    InitChain(RequestInitChain) ResponseInitChain // Initialize blockchain w validation
    BeginBlock(RequestBeginBlock) ResponseBeginBlock // Signals the beginning of a block
    DeliverTx(RequestDeliverTx) ResponseDeliverTx // Deliver a tx for full processing
    EndBlock(RequestEndBlock) ResponseEndBlock // Signals the end of a block,
    Commit() ResponseCommit // Commit the state and return the app
}

```



编写 ABCI 应用程序

- 通过 JetBrains Toolbox 安装 JetBrains GoLand 以编辑 Go 语言代码。使用学生邮箱获取免费的教育版使用资格。
- 打开 Tendermint 源代码。
- 找到 `abci/types/application.go` 文件，其中的 Application 即为需要实现的接口。如上所示。
- 找到 `abci/example/kvstore/kvstore.go` 文件，其中的 Application 实现了上述接口。如下所示。

```
var _ types.Application = (*Application)(nil)
```

```
type Application struct {  
    types.BaseApplication
```

```
    state      State
```

```
    RetainBlocks int64 // blocks to retain after commit (via ResponseCommit.RetainH
```

```
}
```

```
func (app *Application) Info(req types.RequestInfo) (resInfo types.ResponseInfo)
```

```
func (app *Application) DeliverTx(req types.RequestDeliverTx) types.ResponseDeliverTx
```

```
func (app *Application) CheckTx(req types.RequestCheckTx) types.ResponseCheckTx
```

```
func (app *Application) Commit() types.ResponseCommit
```

```
func (app *Application) Query(reqQuery types.RequestQuery) (resQuery types.ResponseQuery)
```



编写 ABCI 应用程序

- 通过 JetBrains Toolbox 安装 JetBrains GoLand 以编辑 Go 语言代码。使用学生邮箱获取免费的教育版使用资格。
- 打开 Tendermint 源代码。
- 找到 `abci/types/application.go` 文件，其中的 Application 即为需要实现的接口。如上所示。
- 找到 `abci/example/kvstore/kvstore.go` 文件，其中的 Application 实现了上述接口。如上所示。
- 找到 `proxy/client.go` 文件，其中的 DefaultClientCreator 根据命令行参数返回不同的 Application，如下所示。

```
func DefaultClientCreator(addr, transport, dbDir string) ClientCreator {  
    switch addr {  
    case "counter":  
        return NewLocalClientCreator(counter.NewApplication(false))  
    case "counter_serial":  
        return NewLocalClientCreator(counter.NewApplication(true))  
    case "kvstore":  
        return NewLocalClientCreator(kvstore.NewApplication())  
    case "persistent_kvstore":  
        return NewLocalClientCreator(kvstore.NewPersistentKVStoreApplication(dbDir))  
    case "noop":  
        return NewLocalClientCreator(types.NewBaseApplication())  
    default:  
        mustConnect := false // loop retrying  
        return NewRemoteClientCreator(addr, transport, mustConnect)  
    }  
}
```




编写 ABCI 应用程序

- 仿照 `abci/example/kvstore/kvstore.go` 文件，编写自己的 ABCI 应用程序。
- 修改 `proxy/client.go` 文件，将自己的 ABCI 应用程序添加到这里，或者，直接让 `DefaultClientCreator` 返回自己的 `Application`。如下所示。
- 重新编译 Tendermint。

```
func DefaultClientCreator(addr, transport, dbDir string) ClientCreator {  
    return NewLocalClientCreator(kvstore.NewApplication())  
}
```



编写 ABCI 应用程序

- 由于 Go 语言要求每个项目都要有独特的包名，自 Go 1.14 起，包名应该由域名开头，通常代表某个 Git 仓库。对于私有仓库，在 go.mod 文件中通过 replace 指令进行重定向（之前已提到）。
- 我们的 ABCI 应用程序是在 Tendermint 的基础上直接开发的，因此应该将源代码中的包名从 `github.com/tendermint/tendermint` 替换为自己的包名。几乎每个 Go 程序的 fork 都要这样做。
- 是否可以不修改 Tendermint 源代码，直接开发自己的 ABCI 应用程序？也可以，但 ABCI 应用程序将成为独立的进程，与 Tendermint 进程通过 RPC 进行交互，略为不便，在此不做介绍。



创建创世区块

默认情况下，区块链节点的数据均保存在 `~/.tendermint` 目录下。指定环境变量 `TMHOME` 可改为其他目录。以下用 `$TMHOME` 代指。

- 生成新的创世区块数据。这相当于创建了一条新的私有链。
`./tendermint init`
执行后，`$TMHOME/config/genesis.json` 即为创世区块文件。
- 对该文件进行适当的修改后，分发给每个区块链节点。简单起见，可暂时只在本机上进行调试。



节点初始化和运行

- 保证 `$TMHOME/config/genesis.json` 文件存在。初始化节点。
`./tendermint init`
注意，该命令与创建创世区块的命令相同，根据 `genesis.json` 文件是否存在，行为不同。
- 运行节点。根据之前 `DefaultClientCreator` 的修改情况，可能需要传递 `--proxy_app` 参数，也可能不需要。
`./tendermint node --proxy_app=kvstore`
此处的 `kvstore` 将作为 `DefaultClientCreator` 的 `addr` 参数，根据该函数的逻辑，返回了 `kvstore` 的 `Application`。



节点操作

- 通过发送 HTTP 请求来操作区块链节点。参数可通过 Get、Post 请求、WebSocket 等方式传递。这里以最简单的 Get 请求为例。
- 查看运行状态
`curl -s localhost:26657/status`
- 发送交易
`curl -s 'localhost:26657/broadcast_tx_commit?tx="abcd"'`
tx 的具体格式由你的 ABCI 应用程序决定。
- 更多的操作详见 Tendermint 文档。



节点操作

- `rpc/client/http/http.go` 提供了 HTTP 用于在编程时操作节点。这比使用 Get 请求更优雅。示例如下所示。

```
client, err = rpcClient.New("http://127.0.0.1:26657", "/websocket")
if err != nil { panic(err) }
var txBytes []byte
根据你的应用程序逻辑，提供正确的 txBytes
ret, err := client.BroadcastTxSync(txBytes)
if err != nil { panic(err) }
```

4

接下来学什么？



接下来学什么？

- 动手编写，实现一个完整的、有意义的 Tendermint 项目，例如，编写一个公证用途的区块链。在单节点上成功运行这条区块链。
- 如何在多个节点上运行这条区块链？这会涉及到以下问题：创世区块中要额外指定什么内容？什么是验证节点和非验证节点？节点之间如何互相发现、连接？
- 为这条区块链编写简单的 UI。用户不会通过 curl 或者编程语言去操作区块链节点，他们需要易用的、简明的 UI。
- 你的区块链应用足够安全吗？假设你是敌手，是否有可能找到 ABCI 逻辑中的漏洞？例如，对于钱包区块链，只验证了交易签名是否有效，忘记验证签名者是否真的是付款者。



接下来学什么？

- 区块链的所有节点均存储了链上全部内容，因此，无论是出于效率还是隐私考虑，并不能将一切数据都放到链上。利用合适的密码学工具，在不影响业务流程和安全性的前提下，保护秘密数据的隐私，将大容量的数据移至链下。
- 以太坊提供了智能合约机制。部署一条私有的以太坊链，尝试编写智能合约。比较一下，以太坊智能合约和 Tendermint 各有什么特点？



Thanks!

Any **questions** ?



齐划一



qi@huayi.email



山东大学