Libra开发入门

dragon
longzhi@secondstate.io
www.secondsate.io

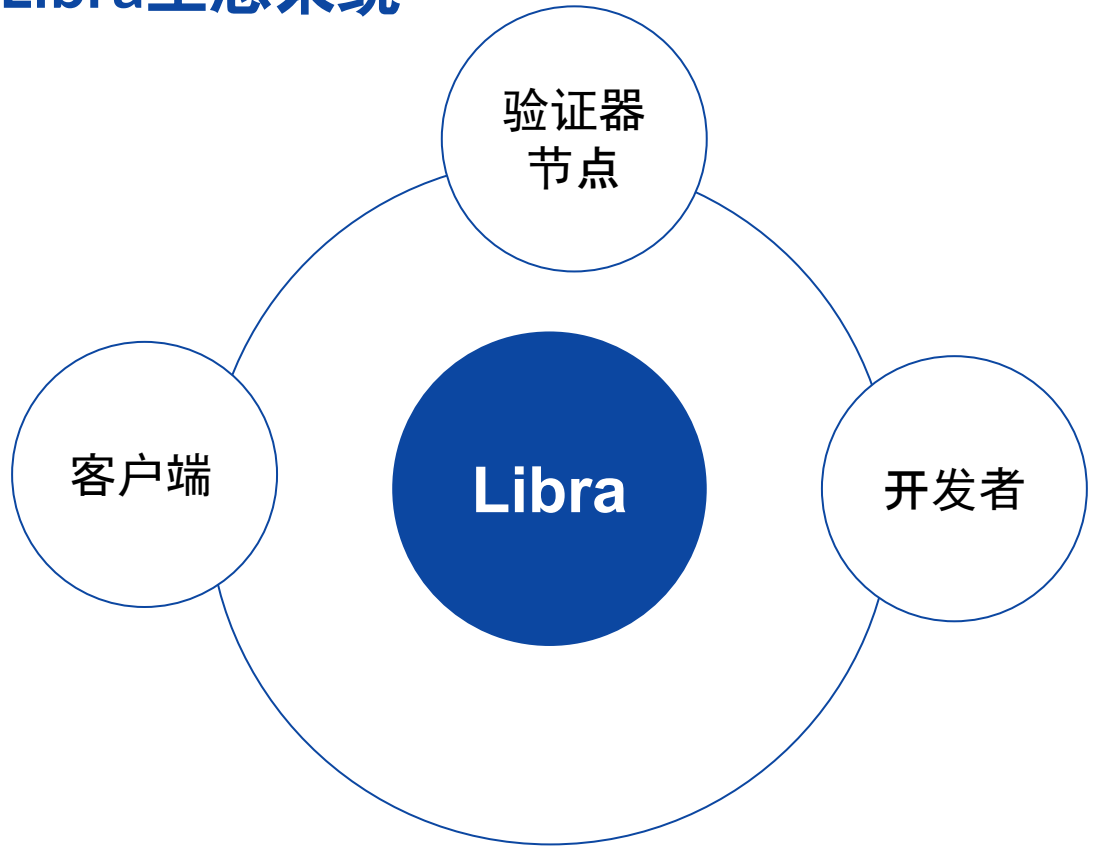
“

Libra 区块链的目标是成为金融服务的坚实基础，以满足全球数十亿人日常财务需求的全新的全球货币。Libra 区块链是从零开始构建，其优先考虑扩展性、安全性、存储、吞吐量效率及未来的适应性

- Libra白皮书

SECOND STATE

Libra生态系统



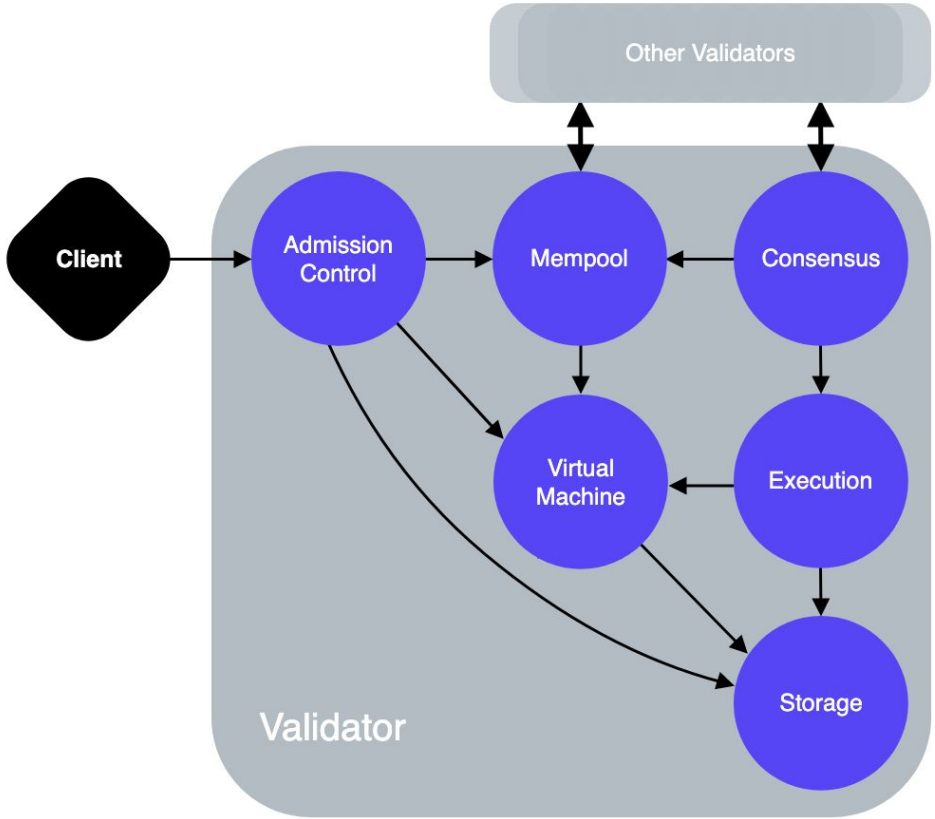
SECOND STATE

客户端

1. 负责与Libra区块链进行交互
2. 由最终用户运行
3. 允许用户构建、签名并发送交易给节点
4. 允许用户向节点发出查询，请求交易或账户状态

验证器节点

验证器节点 是Libra生态系统中的实体，它们共同决定将哪些交易添加到Libra区块链中。验证器使用共识协议 以便它们可以容忍恶意验证器的存在。验证器节点维护区块链上所有交易的历史记录。在内部，验证器节点需要保持当前状态以执行交易并计算下一个状态。



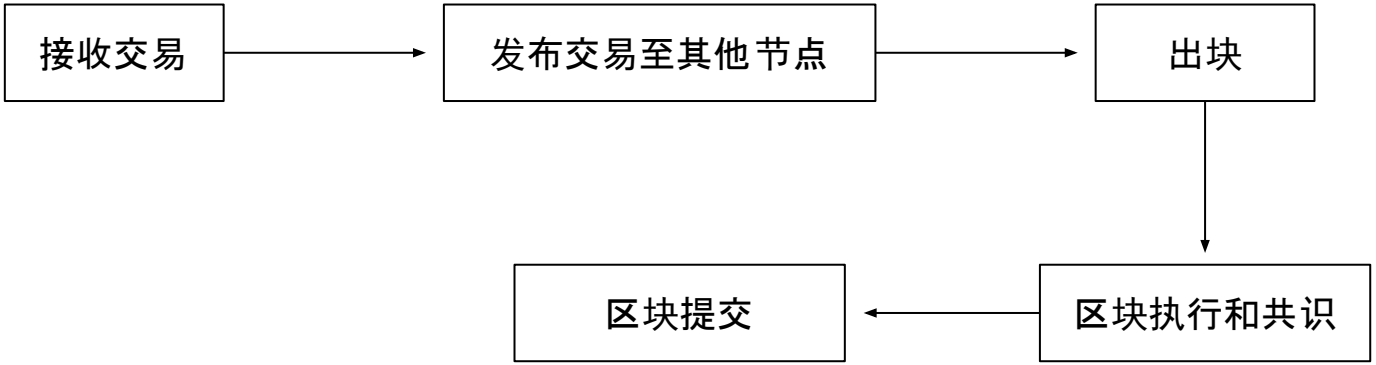
SECOND STATE

Libra核心概念

1. 交易:转账或其他操作，会改变状态
2. 分布式账本状态:所有账户的总状态
3. 账户:模块+资源
4. 验证器节点:共识

SECOND STATE

交易的生命周期



Move语言

- Move是一种新的编程语言，为 Libra 区块链提供安全、可编程的基础。
- Libra区块链中的帐户作为容器，包含了任意数量的Move资源和Move模块。
- 提交给 Libra 区块链的每个交易都使用 Move 编写的交易脚本来实现其逻辑。
- 交易脚本可以通过调用模块声明的过程（procedures）来更新区块链的全局状态。

SECOND STATE

Move语言特点

1. 静态类型：低级错误可以在编译的时候发现，而不是拖到运行期才爆出bug。

2. 中间表示层(IR)：轻量语法层，由字节码验证器进行验证

3. 资源：只能被消耗，不能被复制，只能在账户或者Move对象之间流动。相比其他语言的Value方式，可随意拷贝，容易出现漏洞。Libra上的数字资产是一种资源（不能复制，不能凭空消失）

4. 借鉴了Rust的move语义：读取变量时，必须指定取值方式，要么是copy，要么是move

Move中的几个重要概念

交易脚本

包含在交易中；

不属于任何模块；

一次性执行；

可以调用其他模块的过程和资源；

模块

类似以太坊智能合 约或者Java中类的概念；

可以定义资源和过程（函数）；

非受限类型（Unrestricted Type），包括原生类型和非资源类的结构体（struct）

过程有public和private，public可被其他模块调用；

用户在自己的账户下发布模块

资源

变量的读取方式：move or copy?

用copy方式取值，相当于克隆，原变量值不变仍可继续使用；而用move方式取值，原变量的引用转移给了新变量，原变量失效了。

```
// 两个参数
public main(payee: address, amount: u64) {
// 从sender余额扣除amount个Coin
 let coin: 0x0.Currency.Coin = 0x0.Currency.withdraw_from_sender(copy(amount));

// 将coin累加到payee的Coin余额中
 0x0.Currency.deposit(copy(payee), move(coin));
}
```

SECOND STATE

以太坊转账

接收方凭空多出币！

```
pragma solidity >=0.5.0 <0.7.0;
contract Coin {
   mapping (address => uint) public balances;
   event Sent(address from, address to, uint amount);

   function send(address receiver, uint amount) public {
       require(amount <= balances[msg.sender], "Insufficient balance.");
       balances[msg.sender] -= amount;
       balances[receiver] += amount;
       balances[receiver] += amount; // 又是猫干的!
       emit Sent(msg.sender, receiver, amount);
    }
    // ..........
}
```

SECOND STATE

Move转账

字节码验证器报错！

```
public main(payee: address, amount: u64) {
  let coin: 0x0.Currency.Coin = 0x0.Currency.withdraw_from_sender(copy(amount));
  0x0.Currency.deposit(copy(payee), move(coin));
  0x0.Currency.deposit(copy(payee), move(coin)); // 猫干的!
}
```

Move的不足之处

Libra 区块链如果要支持新的 Move 程序，必须要暂停整条链，并且所有三分之二的验证人节点进行软件升级，才能够支持同样的 Move 程序，这在本质上意味着，每次要添加新的 Move 程序到区块链，都要进行硬分叉，期间伴随着区块链服务暂停。

而智能合约的一个决定性的特点就是它有能力按照要求在区块链在并不需要暂停服务的情况下，通过共识，部署并且执行新代码

SECOND STATE

Libra CLI的使用

SECOND STATE

开发环境

- MacOS or Linux
- Git
- Homebrew on MacOS
- yum or apt-get on Linux

SECOND STATE

准备

```
git clone https://github.com/libra/libra.git && cd libra

#git checkout testnet

./scripts/dev_setup.sh

source $HOME/.cargo/env

cargo build

mkdir test
```

SECOND STATE

第一笔转账交易

```
cargo run -p libra_swarm -- -s
```

```
// 创建Alice账号
```

```
libra% account create
```

```
// 创建Bob账号
```

```
libra% account list
```

```
// 给Alice铸币1000 Libra coin
```

```
libra% account mint 0 1000
```

```
# 查询Alice账户余额
```

```
libra% query balance 0
```

SECOND STATE

第一笔转账交易

```
transfer | transferb | t |
tb<sender_account_address>|<sender_account_ref_id>
<receiver_account_address>|<receiver_account_ref_id>
<number_of_coins> [gas_unit_price_in_micro_libras
(default=0)] [max_gas_amount_in_micro_libras (default
100000)] Suffix 'b' is for blocking

libra% transfer 0 1 10
```

SECOND STATE

查询交易后序列号

```
libra% query sequence 0
```

```
libra% query sequence 1
```

SECOND STATE

查询交易后余额

```
libra% query balance 0 1
```

```
libra% query balance 0 2
```

SECOND STATE

交易脚本

SECOND STATE

一个简单的交易脚本

peer_to_peer_transfer.mvir

```
import 0x0.LibraAccount;
main (payee: address, amount: u64) {
  LibraAccount.pay_from_sender(move(payee), move(amount));
  return;
}
```

SECOND STATE

编译交易脚本

```
cd libra

cp
./language/stdlib/transaction_scripts/peer_to_peer_trans
fer.mvir ./test

libra% dev compile 0 ./test/peer_to_peer_transfer.mvir
false ./test/peer_to_peer_transfer.out
```

SECOND STATE

执行交易脚本

```
execute | e
<sender_account_address>|<sender_account_ref_id>
<compiled_module_path> [parameters]

    Execute custom move script

libra% dev execute 0 ./test/peer_to_peer_transfer.out
0x1234 1000000
```

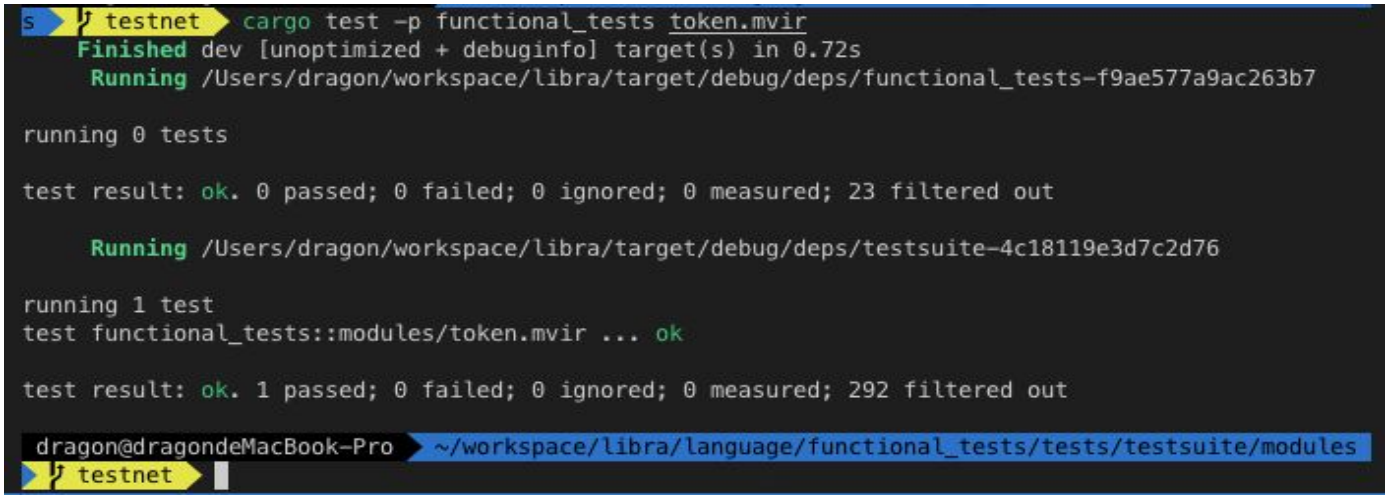
SECOND STATE

模块

一个类ERC20模块

```
cd language/functional_tests/tests/testsuite/modules
```

```
wget
https://github.com/second-state/libra-research/tree/master/examples/ERC20Token
```



SECOND STATE

编译&部署模块

```
cd test
```

```
libra% dev compile 0 ./test/token.mvir true
./test/token.program
```

```
libra% dev publish 0 ./test/token.out
```

SECOND STATE

开启部署modules的权限

Libra 在编译时期 (compilation-time) 时从配置文件里面读取是否可以设定部署 modules 的权限。因此，为把 modules 部署到 local testnet，我们需要在编译前修改这项设定。

在 "config/data/configs/node.config.toml" 这个档案，只需将 "[publishing_options]" 中的 "type=Locked" 改为 "type=Open" 即可。更改完以后，一定要重新编译使设定生效。

SECOND STATE

编译交易脚本

```
libra% dev c 0 false ./test/init.mvir ./test/init.out

libra% dev c 0 false ./test/mint.mvir ./test/mint.out

libra% dev c 0 false ./test/transfer.mvir ./test/transfer.out
```

SECOND STATE

执行交易脚本

```
libra% dev execute 0 ./test/init.out
```

```
libra% dev execute 0 ./test/mint.out 0x1234 1000000
```

```
libra% dev execute 0 ./test/transfer.out 0x1234 1000000
```

SECOND STATE

部署与使用Token的顺序

1、先将 `token.mvir` (含有 Token、TokenCapability 的 module) 部署到 Libra

2、要想使用该 token 账户，必须先调用 `init.mvir` 将 Token.T 发布到账户的 resource 中；

3、所有者可通过 `mint.mvir` 给其他拥有 `resource Token.T` 的账户增发 token；

4、两个拥有 `resource Token.T` 的 account 可以通过 `transfer.mvir` 进行 token 转移。

SECOND
STATE

Thanks !