CITA-Cloud

Rivtower

2021 年 05 月 13 日

Contents

CITA-Cloud 是一个基于云原生的联盟链框架，用户可以基于该框架快速定制出一个适合具体业务场景的链。其主要特性有：

- **语言无关**。 整个框架采用微服务架构，不同微服务可以采用不同的编程语言。各种语言的开发者都可以方便的参与。也可以在开发过程中选择最适合的语言，方便复用已有的软件栈或者库。
- **组件解耦**。 微服务之间高度解耦，每个微服务可以有多种不同的实现，相互之间可以随意替换。可以形成组件市场。用户根据需要选择适合的组件，组成一条自定义的链。场景变化时，也可以快速切换其他的组件。
- **场景定制**。 如果已有的组件都不能满足用户需求，可以针对场景定制某个组件。同时可以复用其他组件，达到快速定制的目标。
- **兼容**。 得益于灵活的架构，可以复用已有区块链产品的组件，达到兼容其其生态的目的。也可以通过跨链的方式与其他区块链系统协作。
- Github 主页
- 官方网站
- 技术论坛
- rfcs

CHAPTER 1

CITA-Cloud 介绍

1.1 产品定位

参见产品定位白皮书。

1.2 运行环境

CITA-Cloud 默认运行在 k8s 环境中，原因有：

- 方便支持各种基础设施，尤其是各种云计算设施。
- 方便实施自动化运维。
- 方便复用云原生社区已有的技术。

1.3 工程仓库

CITA-Cloud 的代码仓库都在 github 上cita-cloud 组织下。主要代码仓库有：

- rfcs 存放白皮书以及后续的改进建议。
- cita_cloud_proto 存放 CITA-Cloud 微服务间 gRPC 接口定义文件。
- runner_k8s 用于生成在 k8s 上运行所需配置文件的工具。
- docs 存放文档。

其余仓库主要是各个微服务实现。仓库名称以 `cita_cloud_proto` 中定义的微服务名称开头，下划线后接用于标识不同实现的名称。比如 `storage_sqlite`，是基于 `sqlite` 实现的 `storage` 微服务。发布件以 `Docker` 镜像的方式发布在 `DockerHub` 上，参见链接。

CHAPTER 2

快速入门

2.1 环境准备

2.1.1 硬件配置建议

- CPU：2 核或以上
- 内存：4GB 或以上
- 硬盘：30G 或以上

2.1.2 操作系统

常见的 `Linux` 发行版本均可，例如：`CentOS`，`Debian`，`Ubuntu` 等等。

2.1.3 软件依赖

- docker 安装方法参见官方文档。

2.1.4 k8s 集群

k8s 集群的搭建非常复杂，为了快速体验，我们推荐使用 minikube，可以在本地快速搭建一个单节点的 k8s 集群。

安装运行 minikube 参见链接。

国内需要设置一些镜像参数，参考：

```
minikube start --registry-mirror=https://hub-mirror.c.163.com --image-
↪repository=registry.cn-hangzhou.aliyuncs.com/google_containers --vm-driver=docker --
↪alsologtostderr -v=8 --base-image registry.cn-hangzhou.aliyuncs.com/google_
↪containers/kicbase:v0.0.10
```

安装 k8s 集群命令行工具 kubectl，参考官方文档。

2.2 生成链的配置

2.2.1 软件依赖

- python3

2.2.2 生成配置

1. 下载配置生成工具。

```
$ wget https://github.com/cita-cloud/runner_k8s/archive/refs/heads/v5.0.0.zip
$ unzip v5.0.0.zip
$ cd runner_k8s-5.0.0
$ ls
config.xml          create_k8s_config.py  create_syncthing_config.py  requirements.
↪txt
create_account.py   create_pvc.py         README.md                   service-
↪config.toml
```

2. 安装依赖包。

```
pip install -r requirements.txt
```

3. 创建 PVC。

```
$ minikube ssh
docker@minikube:~$ mkdir cita-cloud-datadir
```

(下页继续)

(续上页)

```
docker@minikube:~$ exit
$ ./create_pvc.py local_pvc
$ ls
local-pvc.yaml
$ kubectl apply -f local-pvc.yaml
```

4. 生成配置。

```
./create_k8s_config.py local_cluster --kms_password 123456 --peers_count 3 --pvc_
↪name local-pvc
$ ls
cita-cloud  test-chain.yaml
```

生成的 cita-cloud 目录结构如下：

```
$ tree cita-cloud
cita-cloud
└── test-chain
    ├── node0
    ├── node1
    ├── node2
```

最外层是 `cita-cloud`；第二层是链的名称，这里采用默认值 `test-chain`；最里面是各个节点的文件夹。

`node0`，`node1`, `node2` 是三个节点文件夹，里面有相应节点的配置文件。

`test-chain.yaml` 用于将链部署到 `k8s` 集群，里面声明了相关的 `secret/deployment/service`，文件名跟链的名称保持一致。

针对更复杂的情况，有更多可设置的参数，参见runner_k8s 文档。

2.3 运行链

2.3.1 部署

```
$ scp -i ~/.minikube/machines/minikube/id_rsa -r cita-cloud docker@`minikube ip`:~/
↪cita-cloud-datadir/
$ kubectl apply -f test-chain.yaml
```

2.3.2 查看运行情况

查看节点是否运行正常：

```
$ kubectl get deployments.apps
NAME                        READY   UP-TO-DATE   AVAILABLE   AGE
test-chain-0                1/1     1            1           71s
test-chain-1                1/1     1            1           71s
test-chain-2                1/1     1            1           70s
```

查看日志：

```
$ minikube ssh
docker@minikube:~$ tail -10f cita-cloud-datadir/cita-cloud/test-chain/node0/logs/
↪controller-service.log
2020-08-27T07:42:43.280172163+00:00 INFO controller::chain - 1 blocks finalized
2020-08-27T07:42:43.282871996+00:00 INFO controller::chain - executed block 1397␣
↪hash: 0x 16469..e061
2020-08-27T07:42:43.354375501+00:00 INFO controller::pool - before update len of pool␣
↪0, will update 0 tx
2020-08-27T07:42:43.354447445+00:00 INFO controller::pool - after update len of pool 0
2020-08-27T07:42:43.354467947+00:00 INFO controller::pool - low_bound before update: 0
2020-08-27T07:42:43.354484999+00:00 INFO controller::pool - low_bound after update: 0
2020-08-27T07:42:43.385062636+00:00 INFO controller::controller - get block from␣
↪network
2020-08-27T07:42:43.385154627+00:00 INFO controller::controller - add block
2020-08-27T07:42:43.386148650+00:00 INFO controller::chain - add block 0x 11cf7..cad9
2020-08-27T07:42:46.319058783+00:00 INFO controller - reconfigure consensus!
```

2.4 基本操作

2.4.1 软件依赖

- grpcurl 下载链接

2.4.2 下载 cita_cloud_proto

```
$ wget https://github.com/cita-cloud/cita_cloud_proto/archive/refs/heads/v5.0.0.zip
$ unzip v5.0.0.zip
$ cd cita_cloud_proto-5.0.0
$ ls
build.rs  Cargo.toml  protos  README.md  src
```

2.4.3 查询操作

cita-cloud 的 RPC 接口采用的是 gRPC，因此需要使用 grpcurl 工具才能在命令行访问。

同时需要相应的 protobuf 定义文件。

查看块高:

```
$ ./grpcurl -emit-defaults -plaintext -d '{"flag": false}' \
-proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
-import-path ~/cita_cloud_proto-5.0.0/protos \
`minikube ip`:30004 controller.RPCService/GetBlockNumber
{
"blockNumber": "320"
}
```

查看系统配置:

```
$ ./grpcurl -emit-defaults -plaintext -d '{"flag": false}' \
-proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
-import-path ~/cita_cloud_proto-5.0.0/protos \
`minikube ip`:30004 controller.RPCService/GetSystemConfig
{
  "version": 0,
  "chainId": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE=",
  "admin": "CU1ivX7bcvYbAEKIbHfPDhR/bEI=",
  "blockInterval": 6,
  "validators": [
    "eO1A2A+j/eHvUnvLAny4ycq/i5U=",
    "78dGY0xDCXTQq1dmcRFmUdBdpek=",
    "FqpkYXH8c3JMWST4/kKMLpGBEMk="
  ],
  "versionPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  "chainIdPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  "adminPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  "blockIntervalPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
```

(下页继续)

(续上页)

```
  "validatorsPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
}
```

注意：部分参数实际类型为 bytes，grpcurl 工具对其进行了 Base64 编码，输出显示为字符串。

CHAPTER 3

配置说明

参见runner_k8s 文档。

3.1 链级配置

注意

当你计划使用 CITA-Cloud 设计产品时，[环境准备] 好之后，不要着急启动节点，请仔细阅读本节内容，并选择最适合你产品需求的配置。

链级配置指的是链自身的一些属性、系统初始配置，创世块，节点网络地址等配置，用户在起链前进行初始化配置。

本文档会详细介绍链的各个可配置项，然后通过具体的操作示例，演示如何起链前对链进行初始化配置。

3.1.1 `--chain_name`

指定链的名字。

- 链的名字本身不会保存在链上，但是系统初始配置中的 `chain_id` 是用 `chain_name` 计算得到的。计算方式为 `chain_id = sha3_256(chain_name)`。
- `chain_id` 在生成的 `init_sys_config.toml` 文件中可以查看到。
- 链的名字会作为文件夹的名称，该文件夹里面再按节点序号创建 `node0`，`node1`，`node2` 等节点文件夹，分别存放每个节点的配置文件。

- 如果没有传递 `chain_name` 参数，则默认链的名字为 `test-chain`。

3.1.2 --timestamp

指定起链的时间戳。

- 具体数值是指自 1970-1-1 以来的毫秒数，默认是取当前的时间。
- 这个值在生成的 `genesis.toml` 文件中可以查看到。
- 单集群模式下没有该参数，直接采用默认行为，取当前时间。

3.1.3 --super_admin

指定超级管理员地址。

- 该账户拥有最高权限，用来治理整条链。用户**必须**自己设置超级管理员。
- 单集群模式下没有该参数，自动创建相关的账户。
- 这个值在生成的 `init_sys_config.toml` 文件中可以查看到。

3.1.4 --authorities

指定所有共识节点的账户地址。

- 账户地址用逗号分割，个数需要和创建节点的个数保持一致，顺序也要和节点网络地址等保持一致。
- 安全起见，我们建议的流程是：先由每个共识节点单独生成各自的私钥和地址，私钥请务必由自己妥善保管；地址交由负责起链的超级管理员，生成链的配置。分发配置之后，起链前，由各节点独自将自己的私钥补充进来。
- 单集群模式下没有该参数，自动生成对应节点数量的私钥/地址对。

3.1.5 --nodes

指定所有节点的网络地址，`ip` 或者域名。

- 节点网络地址用逗号分割，个数需要和创建节点的个数保持一致，顺序也要和共识节点的账户地址等保持一致。
- 单集群模式下没有该参数，自动创建 `service` 作为节点网络地址。

3.1.6 --lbs_tokens

负载均衡服务的 token 列表。

- 使用云厂商的负载均衡服务将集群内服务端口暴露到集群之外，需要事先申请 token。
- token 用逗号分割，个数需要和创建节点的个数保持一致，顺序也要和节点网络地址等保持一致。
- 单集群模式下没有该参数，因为单集群模式下不使用负载均衡服务，而是创建 NodePort 类型的 service。

3.1.7 --node_ports

指定每个节点的起始端口。

- 每个链节点都有多个服务需要暴露到集群之外，用户只需要指定一个起始端口，多个服务自动往后排列。每个节点的端口信息如下：

```
1. 网络 40000
2. syncthing 22000
3. rpc 50004
4. executor_call 50002
5. monitor_process 9256
6. monitor_exporter 9349
7. executor_chaincode 7052
8. executor_eventhub 7053
9. debug 9999

比如，预留端口为 30000~30008，则要使用的参数为 30000。
注意：根据配置参数的不同，其中部分端口背后的服务可能不会启动，但是端口依然会保留。
例如，预留端口为 30000~30008，不开启 monitor 的时候，30005 和 30006 端口将不会使用，但是
debug 对应的依然是 30008，而不会往前顺延。
```

- 多个节点的起始端口用逗号分割，个数需要和创建节点的个数保持一致，顺序也要和节点网络地址等保持一致。
- 单集群模式下对应的参数为--node_port，只需要传递一个起始端口，所有的节点的要暴露的端口会依次往下排。

3.1.8 --sync_device_ids

指定每个节点用于同步的 device id。

- 通过 create_syncthing_config.py 创建，在创建 device id 的同时，会创建 cert.pem 和 key.pem 两个密钥文件。与共识节点的账户类似，生成配置阶段只需提供 device id，分发配置之后，起链前，由各节点独自将自己的密钥补充进来。
- 多个节点的 device id 用逗号分割，个数需要和创建节点的个数保持一致，顺序也要和节点网络地址等保持一致。
- 单集群模式下没有该参数，自动为每个节点创建 device id。

--kms_passwords

指定每个节点的 kms 服务的密码。

- 每个节点在创建自己的节点账户时就需要设置密码，生成配置时也需要提供。
- 多个节点的密码用逗号分割，个数需要和创建节点的个数保持一致，顺序也要和节点网络地址等保持一致。
- 单集群模式下对应的参数为--kms_password，只需要传递一个密码，所有的节点共用一个密码。

3.1.9 --pvc_names

指定每个节点使用的持久化存储的 persistentVolumeClaim name。

- 公有云环境请咨询相应的云服务商。
- 自建环境可以使用 create_pvc.py 创建，目前支持 local path 和 nfs。
- 多个节点的 pvc name 用逗号分割，个数需要和创建节点的个数保持一致，顺序也要和节点网络地址等保持一致。
- 单集群模式下对应的参数为--pvc_name，只需要传递一个 pvc name，所有的节点共用同一个持久化存储。

3.1.10 --need_debug

是否需要增加调试用的容器。

- 如果设置为 true, 则会在每个节点的 Pod 内增加一个包含丰富调试工具的容器，方便定位问题。
- 默认为 false。

3.1.11 --enable_tls

是否打开通信加密功能。

- 默认为 true。

3.1.12 --service_config

选择链使用的各个微服务的具体实现。

cita-cloud 分为六个微服务：network, consensus, executor, storage, controller, kms。

每个微服务都可能有多个不同的实现，service-config.toml 用来配置每个微服务分别选择哪些实现，以及相应的启动命令。

目前微服务的实现有：

1. network。目前有 network_p2p 和 network_direct 两个实现。分别实现 p2p 和直连两种网络链接方式。
2. consensus。目前有 consensus_raft 和 consensus_bft 两个实现。
3. executor。目前有 executor_poc 和 executor_evm 两个实现。前者是一个空的实现，什么都不做，后者集成了以太坊的 EVM。
4. storage。目前有 storage_rocksdb，storage_sqlite 两个实现。
5. controller。目前只有 controller 这一个实现。
6. kms。目前有 kms_eth 和 kms_sm 两个实现，分别兼容以太坊和国密。

注意：六个微服务缺一不可；每个微服务只能选择一个实现，不能多选。

3.2 微服务配置

3.2.1 network_p2p

命令行参数：

```
USAGE:
    network run [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
```

(下页继续)

(续上页)

```
    -p, --port <grpc-port>       Sets grpc port of this service [default: 50000]
    -k, --key_file <key-file>    Sets path of network key file [default: network-key]
```

配置文件：

1. 日志配置文件 network-log4rs.yaml。日志配置参见log4rs 文档。
2. 节点私钥 network-key。用于通信加密，文件内容为 0x 开头的十六进制字符串。
3. 服务配置文件 network-config.toml。

服务配置文件例子：

```
$ cat network-config.toml
enable_tls = true
port = 40000
[[peers]]
ip = "test-chain-1"
port = 40000

[[peers]]
ip = "test-chain-2"
port = 40000
```

- enable_tls 是否打开通信加密功能。
- peers：配置对端节点信息。在网络服务启动时会首先连接 peers 中的节点。

3.2.2 network_direct

命令行参数：

```
USAGE:
    network run [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -p, --port <grpc-port>       Sets grpc port of this service [default: 50000]
    -k, --key_file <key-file>    Sets path of network key file. It's not used for␣
↪network_direct.
                                 Leave it here for compatibility [default: network-
↪key]
```

配置文件：

1. 日志配置文件 network-log4rs.yaml。日志配置参见log4rs 文档。
2. 服务配置文件 network-config.toml。同 network_p2p。

3.2.3 consensus_raft

命令行参数：

```
USAGE:
    consensus run [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -p, --port <grpc-port>    Sets grpc port of this service [default: 50003]
```

配置文件：

1. 日志配置文件 consensus-log4rs.yaml。日志配置参见log4rs 文档。
2. 节点账户地址 node_address。文件内容为 0x 开头的十六进制字符串。
3. 服务配置文件 consensus-config.toml。

服务配置文件例子：

```
network_port = 50000
controller_port = 50004
node_id = 0
```

- network_port 网络微服务的 gRPC 端口。
- controller_port 控制微服务的 gRPC 端口。
- node_id 废弃参数。

3.2.4 consensus_bft

命令行参数：

```
USAGE:
    consensus run [OPTIONS]

FLAGS:
```

(下页继续)

(续上页)

```
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -p, --port <grpc-port>    Sets grpc port of this service [default: 50001]
```

配置文件:

1. 日志配置文件 consensus-log4rs.yaml。日志配置参见log4rs 文档。
2. 节点账户地址 node_address。文件内容为 0x 开头的十六进制字符串。
3. 节点私钥文件 node_key。文件内容为 0x 开头的十六进制字符串。
4. 服务配置文件 consensus-config.toml。同 consensus_raft。

3.2.5 executor_poc

命令行参数:

```
USAGE:
    executor run [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -p, --port <grpc-port>    Sets grpc port of this service [default: 50002]
```

配置文件:

1. 日志配置文件 executor-log4rs.yaml。日志配置参见log4rs 文档。

3.2.6 executor_evm

命令行参数:

```
USAGE:
    executor run [FLAGS] [OPTIONS]

FLAGS:
    -c, --compatibility    Sets eth compatibility, default false
    -h, --help             Prints help information
    -V, --version          Prints version information
```

(下页继续)

(续上页)

```
OPTIONS:
    -p, --port <grpc-port>    Set executor port, default 50002
```

- -c 参数表示是否与以太坊兼容（CITA 默认与以太坊在块的时间戳精度上不兼容，CITA 为毫秒，以太坊为秒）。

配置文件：

1. 日志配置文件 executor-log4rs.yaml。日志配置参见log4rs 文档。

3.2.7 storage_rocksdb

命令行参数：

```
USAGE:
    storage run [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -d, --db <db-path>        Sets db path [default: chain_data]
    -p, --port <grpc-port>    Sets grpc port of this service [default: 50003]
```

配置文件：

1. 日志配置文件 storage-log4rs.yaml。日志配置参见log4rs 文档。

3.2.8 storage_sqlite

命令行参数：

```
USAGE:
    storage run [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -d, --db <db-path>        Sets db path [default: storage.db]
    -p, --port <grpc-port>    Sets grpc port of this service [default: 50003]
```

配置文件：

1. 日志配置文件 storage-log4rs.yaml。日志配置参见log4rs 文档。

3.2.9 controller

命令行参数：

```
USAGE:
    controller run [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -p, --port <grpc-port>    Sets grpc port of this service [default: 50004]
```

配置文件：

1. 日志配置文件 controller-log4rs.yaml。日志配置参见log4rs 文档。
2. key_id，内容为一个十进制数字，表示节点账户在 kms 数据库中的序号。
3. 服务配置文件 controller-config.toml。

服务配置文件例子：

```
network_port = 50000
consensus_port = 50001
storage_port = 50003
kms_port = 50005
executor_port = 50002
block_delay_number = 0
```

- network_port 网络微服务的 gRPC 端口。
- consensus_port 共识微服务的 gRPC 端口。
- storage_port 存储微服务的 gRPC 端口。
- kms_port kms 微服务的 gRPC 端口。
- executor_port 执行器微服务的 gRPC 端口。
- block_delay_number 区块从共识完成到最终确认，延迟的区块数量。

3.2.10 kms_sm

命令行参数：

```
USAGE:
    kms run [OPTIONS]

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

OPTIONS:
    -d, --db <db-path>          Sets path of db file [default: kms.db]
    -p, --port <grpc-port>      Sets grpc port of this service [default: 50005]
    -k, --key <key-file>        Sets path of key_file
```

- `-k` 参数传递一个文件，内容为 `kms` 服务的密码。如果不传递该参数，则会进入交互模式，需要用户在命令行上输入密码。

配置文件：

1. 日志配置文件 `kms-log4rs.yaml`。日志配置参见log4rs 文档。

3.2.11 kms_eth

跟 `kms_sm` 相同。

3.3 密码学算法配置

目前 `CITA-Cloud` 支持两种密码学算法的组合（分别为签名算法和散列算法）：

1. [secp256k1] 与 [kecak]
2. [sm2] 与 [sm3]

用户可以通过在 `service-config.toml` 选择不同的 `kms` 微服务具体实现来选择不同的密码学组合。

1. 对应 `kms_eth`。
2. 对应 `kms_sm`。

CHAPTER 4

部署指南

4.1 单集群

链的所有节点都在同一个 k8s 集群中。

4.1.1 NFS

持久化存储使用 NFS。

硬件配置:

- 一个 k8s 集群。
- 一台专用的 NFS 服务器。

部署步骤:

1. 在 NFS 服务器上安装 nfs 服务。参考链接。
2. 生成 nfs-pvc。

```
$ ./create_pvc.py nfs_pvc --nfs_server 172.17.0.2 --nfs_path /data/nfs
$ ls
nfs-pvc.yaml
$ kubectl apply -f nfs-pvc.yaml
```

3. 生成链的配置。

```
$ ./create_k8s_config.py local_cluster --kms_password 123456 --peers_count 3 --
↪pvc_name nfs-pvc
$ ls
cita-cloud  test-chain.yaml
```

4. 拷贝配置文件到 NFS 服务器。

```
scp -r ./cita-cloud 172.17.0.2:/data/nfs/
```

5. 部署链。

```
kubectl apply -f test-chain.yaml
```

4.2 多集群

链的节点分布在多个 k8s 集群中。

4.3 对接云服务商

4.3.1 阿里云

4.3.2 华为云

CHAPTER 5

升级说明

CHAPTER 6

节点管理

6.1 节点分类

节点分为两大类：

- 共识节点：共识节点具有出块和投票权限，交易由共识节点排序并打包成块，共识完成后即被确认为合法区块。
- 普通节点：普通节点没有出块和投票权限，其他方面和共识节点相同。可以同步和验证链上所有的原始数据，接受交易数据并向其他节点广播。

公有链没有节点准入机制，意味着任何节点都可以接入链并同步其全部的数据，在满足一定的条件下都可以参加共识。而 `cita-cloud` 对于共识节点和普通节点都进行了准入管理。对于身份验证失败的节点，即使该节点能够在网络层与其他节点连通，这些节点也会拒绝与之建立通讯会话，如此可避免信息泄漏。

6.2 共识节点管理

参见治理-共识节点管理

6.3 普通节点管理

CHAPTER 7

隐私保护

CHAPTER 8

治理

本章节的操作需要超级管理员权限，且都非常危险，请谨慎操作。

8.1 安装工具

目前工具还比较原始，且没有提供编译好的二进制，需要用户自行下载源码编译安装。

参见仓库地址。

8.2 超级管理员

超级管理员可以指定下一任超级管理员。

操作示例：

在链的配置过程中，会生成存放超级管理员私钥的数据库文件 `kms.db` 和账户在数据库中的序号 `key_id`。

使用跟链匹配的 `kms` 组件和前述 `kms.db` 来启动 `kms` 服务。

通过`-i` 传递超级管理员账户的 `key_id`，`-a` 参数传递新的 `super admin` 账户地址。

```
$ ./update_admin update -a 0x380fc667f4ccd2e91366331f76d4030d27427185 -i 1 -c␣
↪`minikube ip`:30004
2021-04-28T16:06:16.609927914+08:00 INFO update_admin - waiting for a while...
2021-04-28T16:06:25.677037921+08:00 INFO update_admin - OK!
```

查询 SystemConfig 确认 admin 变成了新的账户地址。

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 \
    controller.RPCService/GetSystemConfig
{
  ...
  "admin": "OA/GZ/TM0ukTZjMfdtQDDSdCcYU=",
  ...
  "adminPreHash": "2bq+B1JoDUUFggNM7tjhFI/Y6MQQIGwS4EkaoiG9a6E=",
  ...
}
$ echo OA/GZ/TM0ukTZjMfdtQDDSdCcYU= | base64 -d | hexdump -C
00000000  38 0f c6 67 f4 cc d2 e9  13 66 33 1f 76 d4 03 0d  |8..g.....f3.v...|
00000010  27 42 71 85                                       |'Bq.|
00000014
```

8.3 共识节点管理

设置新的共识节点列表。

操作示例：

在链的配置过程中，会生成存放超级管理员私钥的数据库文件 kms.db 和账户在数据库中的序号 key_id。

使用跟链匹配的 kms 组件和前述 kms.db 来启动 kms 服务。

通过-i 传递超级管理员账户的 key_id，-v 参数传递新的共识节点列表，用逗号隔开的一组账户地址。

首先查看当前的验证人列表:

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 \
    controller.RPCService/GetSystemConfig
{
  ...
  "validators": [
    "G9qFsSUTToPVoFgkwkoWhXU02X4=",
    "jQbZ2EUmojWMbMnu7t8m+ImhtTw=",
    "cp68Y+9FPCRkRqU/hAuwAuy0KjA=",
    "EknAFRpaXJeTdza63Qe6eIA4Vvw="
```

(下页继续)

(续上页)

```
  ],
  ...
  "validatorsPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  ...
}
```

设置新的共识节点列表为 0x9b87c3739321b4753aee1ef012d4ec5d7b26dd4d, 0x2a783705d63870f83d22520d93bee754e32f04c5,0xffe7019074ddc13d3eab1454fa3445458dc0f0d0。

```
$ ./update_validators update -i 1 -c `minikube ip`:30004 -v␣
↪0x9b87c3739321b4753aee1ef012d4ec5d7b26dd4d,
↪0x2a783705d63870f83d22520d93bee754e32f04c5,
↪0xffe7019074ddc13d3eab1454fa3445458dc0f0d0
2021-04-29T12:01:00.012615335+08:00 INFO update_validators - waiting for a while...
2021-04-29T12:01:15.036363788+08:00 INFO update_validators - OK!
```

查询 SystemConfig 确认 validators 变成了新的一组账户地址。

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 \
    controller.RPCService/GetSystemConfig
{
  ...
  "validators": [
    "m4fDc5MhtHU67h7wEtTsXXsm3U0=",
    "Kng3BdY4cPg9IlINk77nVOMvBMU=",
    "/+cBkHTdwT0+qxRU+jRFRY3A8NA="
  ],
  ...
  "validatorsPreHash": "odfZw+wlze5TaSAdETqrdTClDIeLUz0ahoPn6ewLLN8=",
  ...
}

$ echo m4fDc5MhtHU67h7wEtTsXXsm3U0= | base64 -d | hexdump -C
00000000  9b 87 c3 73 93 21 b4 75  3a ee 1e f0 12 d4 ec 5d  |...s.!.u:......]|
00000010  7b 26 dd 4d                                       |{&.M|
00000014
$ echo Kng3BdY4cPg9IlINk77nVOMvBMU= | base64 -d | hexdump -C
00000000  2a 78 37 05 d6 38 70 f8  3d 22 52 0d 93 be e7 54  |*x7..8p.="R....T|
00000010  e3 2f 04 c5                                       |./..|
00000014
$ echo /+cBkHTdwT0+qxRU+jRFRY3A8NA= | base64 -d | hexdump -C
```

(下页继续)

(续上页)

```
00000000  ff e7 01 90 74 dd c1 3d  3e ab 14 54 fa 34 45 45  |....t..=>..T.4EE|
00000010  8d c0 f0 d0                                       |....|
00000014
```

共识节点列表变更之后，原有共识节点的共识会停掉，链可能会停止出块。

直到使用新的共节点账户的共识节点启动并完成同步之后才会继续共识并出块。

8.4 修改出块间隔

设置新的出块间隔。

操作示例：

在链的配置过程中，会生成存放超级管理员私钥的数据库文件 `kms.db` 和账户在数据库中的序号 `key_id`。

使用跟链匹配的 `kms` 组件和前述 `kms.db` 来启动 `kms` 服务。

通过`-i` 传递超级管理员账户的 `key_id`，`-a` 参数传递新的 `super admin` 账户地址。

首先查看当前的出块间隔：

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 \
    controller.RPCService/GetSystemConfig
{
  ...
  "blockInterval": 6,
  ...
  "blockIntervalPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  ...
}
```

修改出块间隔为 3 秒：

```
$ ./set_block_interval set -i 1 -c `minikube ip`:30004 -b 3
2021-04-29T15:12:19.397725063+08:00 INFO set_block_interval - waiting for a while...
2021-04-29T15:12:28.412780267+08:00 INFO set_block_interval - OK!
```

查询 `SystemConfig` 确认 `blockInterval` 变成了新的值。

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
```

(下页继续)

(续上页)

```
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 \
    controller.RPCService/GetSystemConfig
{
  ...
  "blockInterval": 3,
  ...
  "blockIntervalPreHash": "OT4KJp/CcUb+d3KQJK8VYHZvVyL0RaGUR4mWE96s5+U=",
  ...
}
```

8.5 紧急制动

超级管理员在极端情况下的维护手段，开启紧急制动模式后，链上只接收超级管理员发送的交易，其他交易全部拒绝。

主要使用场景为进行一些升级，维护等操作时，不希望有其他交易干扰。

8.5.1 开启

在链的配置过程中，会生成存放超级管理员私钥的数据库文件 `kms.db` 和账户在数据库中的序号 `key_id`。

使用跟链匹配的 `kms` 组件和前述 `kms.db` 来启动 `kms` 服务。

通过`-i` 传递超级管理员账户的 `key_id`。

```
$ ./emergency_brake enable -i 1
...
2021-04-27T10:58:38.048269659+08:00 INFO emergency_brake - waiting for a while...
2021-04-27T10:58:44.061257726+08:00 INFO emergency_brake - OK!
```

查询 `SystemConfig` 确认 `emergencyBrake` 变成了 `True`。

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 \
    controller.RPCService/GetSystemConfig
{
  ...
  "emergencyBrake": true,
  ...
```

(下页继续)

(续上页)

```
  "emergencyBrakePreHash": "W1qqQrQxL7IEoVbSeeD5iAXmKpjN+Sk8TV82H878PYg="
}
```

此时，发送普通交易将会返回 `forbidden` 错误信息。

8.5.2 关闭

准备条件以及参数跟开启时相同。

```
$ ./emergency_brake disable -i 1
...
2021-04-27T10:58:38.048269659+08:00 INFO emergency_brake - waiting for a while...
2021-04-27T10:58:44.061257726+08:00 INFO emergency_brake - OK!
```

查询 `SystemConfig` 确认 `emergencyBrake` 变成了 `False`。

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 \
    controller.RPCService/GetSystemConfig
{
  ...
  "emergencyBrake": false,
  ...
  "emergencyBrakePreHash": "F0nyI2Tqr/yVbw5b6mAwc38mRp3zAotrq+Em+MO+6d8="
}
```

此时，就可以正常发送普通交易了。

CHAPTER 9

数据管理

9.1 数据同步

9.2 数据迁移

9.3 数据裁剪

CHAPTER 10

运维

10.1 日志管理

10.2 异常排查

10.3 服务监控

CHAPTER 11

架构设计

11.1 整体设计原则

CITA-Cloud 总体设计上依然采用微服务架构，划分为 Controller，Network，Consensus，Storage，Executor，KMS 六个微服务。微服务之间相互解耦，达到不同实现可以灵活替换，自由组合的目的。

解耦设计的细节参见底层链技术白皮书。

在此基础上，为了能够快速构建起完整的，成熟的生态。在之前解耦的基础上，让解耦出的每一个微服务都能独立完成某项功能，每个微服务的接口能够自洽，方便直接复用已有的库或者软件。

11.2 协议详解

11.2.1 Network

Network 微服务，主要提供网络部分的功能，分为收/发两大部分功能。收的部分采用了控制反转，收到的网络报文根据报文头中的字段分发到不同的 Grpc 地址，因此只提供了一个注册接口 (RegisterNetworkMsgHandler); 发的部分提供了单播 (SendMsg) 和广播 (Broadcast) 两个接口。此外就是一个查询网络连接状态的接口 (GetNetworkStatus)。

实现上就是已有网络库的简单封装。

其中一个实现network_direct直接使用系统网络库，内部实现为节点的全互联。另外一个实现network_p2p，使用了已有的一个 p2p 库。

11.2.2 Storage

Storage 微服务，主要提供 KV 存储相关的功能，涵盖了常用的增删改查功能。接口上有：Store(其语义是 updata，同时包含增和改的功能)，Load，Delete。

针对区块链业务，预先定义了不同的 region：

```
enum Regions {
    GLOBAL = 0;
    TRANSACTIONS = 1;
    HEADERS = 2;
    BODIES = 3;
    BLOCK_HASH = 4;
    PROOF = 5;
    RESULT = 6;
    TRANSACTION_HASH2BLOCK_HEIGHT = 7;
    BLOCK_HASH2BLOCK_HEIGHT = 8;  // In SQL db, reuse 4
    TRANSACTION_INDEX = 9;
    BUTTON = 10;
}
```

用户可以根据自己的需要调整 region 列表。

具体实现上，就是已有存储系统的简单封装。

当前实现有基于 sqlite 的storage_sqlite，基于 rocksdb 的storage_rocksdb，以及基于 tikv 的storage_tikv。

region 的实现，基于 sqlite 时是用不同的 table 来实现；基于 tikv 时是直接把 region 序列化之后拼接在 key 的前面；基于 rocksdb 时是用不同的 ColumnFamily 来实现。

11.2.3 KMS

KMS 微服务，主要提供私钥加密存储，以及相关的密码学服务。接口有：GenerateKeyPair，HashData，VerifyDataHash，SignMessage，RecoverSignature，形成——生成密钥对/哈希以及相关的验证/签名以及相关的验证——这么一个自洽的功能集合。此外还有一个查询接口 GetCryptoInfo。可以认为是一个软件加密机加上一个密码学工具箱。

目前的实现kms_eth和kms_sm，分别实现了 secp256k1+keccak 和 sm2+sm3 两种密码学算法组合。

11.2.4 Executor

Executor 微服务，主要提供根据交易改变链上状态以及查询链上状态的功能。接口有：Exec 和 Call，分别对应执行和查询。

至于执行的细节，比如采用何种 VM；状态有哪些内容；状态如何组织和保存，都由具体实现来决定。

后续会给出一些兼容已有链的特定实现，比如executor_chaincode就是兼容 Fabric 的实现。executor_evm则是兼容以太坊的实现。

也可以针对一些特定应用场景，提供特定的 VM 和智能合约编程语言。比如可信计算，隐私计算，数据格式转换等。

甚至可以不提供智能合约，直接针对具体应用实现，类似于原生合约。

11.2.5 Consensus

Consensus 微服务，主要提供让提案在多个共识参与方之间达成一致的功能。接口包括：

1. 实现在 controller 中的：GetProposal 本节点提交的提案；CheckProposal 检查别的节点提交的提案；CommitBlock 确认一个提案。
2. 实现在 Consensus 中的：CheckBlock 检查同步自别的节点的已经确认的提案；Reconfigure 处理系统配置变更。

单独这个微服务的功能，可以认为是一个分歧解决机。

实现上，目前尝试了 PoX 类型的consensus_proof_of_sleep，基于 raft 的consensus_raft，以及基于 bft 的consensus_bft。

11.2.6 Controller

Controller 微服务在整个区块链中处于核心的位置，主导所有主要的流程，并给上层用户提供 RPC 接口。

接口除了前述的针对 Consensus 微服务的接口，就是针对上层用户的 RPC 接口。其中最重要的是 SendRawTransaction 发送交易接口，剩下的都是一些信息查询接口。

单独就这个微服务来说，可以认为是一个提案管理系统。用户通过发送交易接口，提交原始交易数据，Controller 管理这些原始交易数据。通过计算原始交易数据的哈希，组装 CompactBlock，以及再次哈希，形成 Consensus 需要的提案，管理这些提案。这里所说的管理，包括持久化，同步，以及验证其合法性。

Blockchain.proto 文件中定义了一套交易和块的数据结构，但是前面所述的从原始交易数据如何产生最终 Consensus 需要的提案，并且这个过程还是要可验证的，这些都由具体实现决定。未来我们会提供一个框架，方便用户自定义整个流程，甚至是自定义交易和块等核心数据结构。

实现上，目前只有一个controller_poc。主要模块还是以自己实现为主，只有同步模块使用了 `Syncthing`。实现上的技术细节参见项目 Wiki。整个实现的代码量还是比较大，后续会考虑进一步细化设计，让尽量多的模块能够复用已有的软件或者库。

CHAPTER 12

RPC 列表

12.1 GetPeerCount

当前节点连接数。

- 参数:

无。

- 返回值

uint64 - 本节点连接节点个数。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetPeerCount
{
  "peerCount": "2"
}
```

12.2 GetBlockNumber

返回当前块高度。

- 参数

bool - true 表示获取 Pending 状态的块高；false 表示获取 latest 状态的块高。

- 返回值

uint64 - 当前块高度。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"flag": false}' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetBlockNumber
{
  "blockNumber": "320"
}
```

12.3 GetTransaction

根据交易哈希查询交易。

- 参数

bytes - 交易哈希值。

- 返回值

RawTransaction - 交易结构体。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"hash":
↪"n7T5mKGARwowz5DFLV5tpanV+XCxnQ4P4YAuSjDTmdA="}' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetTransaction
{
  "normalTx": {
    "transaction": {
      "version": 0,
      "to": "AQEBAQEBAQEBAQEBAQEBAQEBAQEB",
      "nonce": "test",
```

(下页继续)

(续上页)

```
      "quota": "300000",
      "validUntilBlock": "249",
      "data": "QKarrbmLQYf9k279T9lf0NsMpdDNO3Q7N29j6sofdSM=",
      "value": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=",
      "chainId": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE="
    },
    "transactionHash": "kN+0q0M5vGiwOFBVE7FgCEQThkQ+BlaPFpOmrgwPpQs=",
    "witness": {
      "signature": "sQ0XtpH3n0On2RDQZxzz/
↪pLAIon0O8+iMqnkz1pCqbzftwka0rcVlcmq+w8iKGnFi8WiBjDQLjScwokdpb+Q9q5v9f2ltUTNtXO1M6Gn9uZDD+/
↪thY57zec+rNDOpyD2yh4eeh8J/EXcad8SS98PTzx8MrNbBYH2x50bPDcNu4I=",
      "sender": "Hx2Xykv9907bllhHBXjAsAr+dbY="
    }
  }
}
```

12.4 GetSystemConfig

查询链上系统配置数据。

- 参数

无。

- 返回值

`SystemConfig` - 系统配置结构体。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetSystemConfig
{
  "version": 0,
  "chainId": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAE=",
  "admin": "CU1ivX7bcvYbAEKIbHfPDhR/bEI=",
  "blockInterval": 6,
  "validators": [
    "eO1A2A+j/eHvUnvLAny4ycq/i5U=",
    "78dGY0xDCXTQq1dmcRFmUdBdpek=",
    "FqpkYXH8c3JMWST4/kKMLpGBEMk="
  ],
```

(下页继续)

(续上页)

```
  "versionPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  "chainIdPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  "adminPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  "blockIntervalPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
  "validatorsPreHash": "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
}
```

12.5 GetVersion

获取当前软件的版本号。

- 参数

无。

- 返回值

`String` - 软件版本号。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetVersion
{
  "version": "4.0.0"
}
```

12.6 GetBlockHash

获取指定块高的块哈希值。

- 参数

`uint64` - 块高度。

- 返回值

`bytes` - 块哈希值。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"block_number": 10}' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
```

(下页继续)

(续上页)

```
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetBlockHash
{
  "hash": "zt8g46kl/bZraBM2nb+6Jao7bu0IAUb83U+IPsgs3Hk="
}
```

12.7 GetTransactionBlockNumber

获取指定交易所在的块高度。

- 参数

bytes - 交易哈希值。

- 返回值

uint64 - 块高度。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"hash": "fJkDwCaMi7mOnHTVQ/
↪IcGNr83aoUxnj5kAkDDhRkya0="}' \
   -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
   -import-path ~/cita_cloud_proto-5.0.0/protos \
   `minikube ip`:30004 controller.RPCService/GetTransactionBlockNumber
{
  "blockNumber": "22"
}
```

12.8 GetTransactionIndex

获取指定交易在块中的序号。

- 参数

bytes - 交易哈希值。

- 返回值

uint64 - 序号。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"hash": "fJkDwCaMi7mOnHTVQ/
↪IcGNr83aoUxnj5kAkDDhRkya0="}' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetTransactionIndex
{
  "txIndex": "2"
}
```

12.9 GetBlockByHash

根据块哈希值查询块。

- 参数

bytes - 块哈希值。

- 返回值

CompactBlock - 块结构体。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"hash":
↪"CeTdeke5A8WeTexqWBxl5elFb8vaHFkLYcTkPo5JAMs="}' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetBlockByHash
{
  "version": 0,
  "header": {
    "prevhash": "6r28BwIkV6qTbLD9//WKcGWK3zZNbarD4cVIzSy+M2E=",
    "timestamp": "1616058361343",
    "height": "10",
    "transactionsRoot": "GrIdg1XPoX+OYRlIMegajyK+yMco/vt0ftA161CCqis=",
    "proposer": "eO1A2A+j/eHvUnvLAny4ycq/i5U="
  },
  "body": {
    "txHashes": [

    ]
  }
}
```

12.10 GetBlockByNumber

根据块高度查询块。

- 参数

`uint64` - 块高度。

- 返回值

`CompactBlock` - 块结构体。

- 示例

```
$ ./grpcurl -emit-defaults -plaintext -d '{"block_number": 10}' \
    -proto ~/cita_cloud_proto-5.0.0/protos/controller.proto \
    -import-path ~/cita_cloud_proto-5.0.0/protos \
    `minikube ip`:30004 controller.RPCService/GetBlockByNumber
{
  "version": 0,
  "header": {
    "prevhash": "6r28BwIkV6qTbLD9//WKcGWK3zZNbarD4cVIzSy+M2E=",
    "timestamp": "1616058361343",
    "height": "10",
    "transactionsRoot": "GrIdg1XPoX+OYRlIMegajyK+yMco/vt0ftA161CCqis=",
    "proposer": "eO1A2A+j/eHvUnvLAny4ycq/i5U="
  },
  "body": {
    "txHashes": [

    ]
  }
}
```

12.11 SendRawTransaction

发送交易。

- 参数

`RawTransaction` - 交易结构体。

- 返回值

`bytes` - 交易哈希值。

CHAPTER 13

版本

13.1 v3.0.0

版本发布说明

13.2 v4.0.0

版本发布说明

13.3 v5.0.0

版本发布说明

CHAPTER 14

中间件

14.1 分布式身份

参见分布式数字身份产业联盟。

14.2 数据存证

参见RivLink。

14.3 物联网设备

参见RivIoT。

14.4 数据协作

参见RivFlow。

14.5 跨链连接

14.6 隐私计算

CHAPTER 15

常见问题

CHAPTER 16

词汇表