蚂蚁链

区块链服务平台
运维指南

产品版本：V3.1.1

文档版本：20250314

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	**单击设置> 网络> 设置网络类型。**
粗体	表示按键、菜单、页面名称等UI元素。	**在结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.控制台运维

本节介绍控制台的系统启停、日志位置及说明、虚拟机排查的方法，还给出了SQLite数据的查看方式、表介绍以及如何操作数据库做运维（例如纳管已有的链，状态修复等）

系统启停

- 启动

  执行如下的指令来启动中途岛midway。

```
cd /home/admin/midway/bin
./start.sh start
```

- 检测

  执行如下的指令，请求中途岛心跳访问，检测运行状态。

```
curl http://localhost:80/healthcheck.html
Midway is Running!
```

- 停止

  执行如下的指令来停止中途岛midway。

```
cd /home/admin/midway/bin
./stop.sh stop
```

日志说明

日志名	说明	位置
midway.log	系统业务日志	/home/admin/midway/log
gc.log	JVM虚拟机日志	/home/admin/midway/log

虚拟机排查

- 内存

```
jmap -heap pid
显示Java堆详细信息

jmap -histo:live pid
显示堆中对象的统计信息

jmap -dump:format=b,file=heapdump.phrof pid
生成堆转储快照dump文件
```

- 线程

```
jstack -l pid
查看线程输出

排查CPU高的问题
top -H -p 17850             #找到load占比大的pid
printf "%x\n" pid           #十六进制转换
jstack pid|grep 45d8 -A 30 #定位线程情况
```

数据查看

中途岛系统的数据库采用SQLite，Linux系列操作系统默认自带，以下是常规的数据查看方法。

- 打开数据库

  通过终端，打开中途岛数据库文件 (/home/admin/midway/data/midway.db)，进入SQLite命令行模式。

```
cd /home/admin/midway/data
sqlite3 midway.db
SQLite version 3.24.0 2018-06-04 19:24:41
Enter ".help" for usage hints.
sqlite>
```

- 数据库表

```
sqlite> .tables
blochain    blockchain_node    block_info     healthcheck_info  operate_plan    system_config    block_task_trace
history_info       operate_step    transaction_info    authority_info  role_authority  role_info  user_info  user_role
```

表名	说明
blockchain	区块链基本信息表，包含链相关元数据
blockchain_node	区块链节点信息表，包括节点相关的元数据
block_info	块信息表，支持浏览器功能。定时拉块写入，保留最新一定数量的记录。
transaction_info	交易信息表，支持浏览器功能。定时拉块写入，保留最新一定数量的记录。
history_info	历史交易统计表，支持浏览器功能。定时拉块写入，保留最新一定数量的记录。
block_task_trace	拉块任务表，支持定时任务异步拉块。
healthcheck_info	心跳服务信息表，配置进行可用性监控的服务信息。
operate_plan	操作计划表，自动化操作记录表（部署，加节点，停启服等）
operate_step	操作步骤表，自动化操作步骤，与操作计划表1对多：1 plan —> N step
system_config	系统配置表，保留全局项数据
authority_info	用户权限信息表，记录多级权限的元数据
role_authority	账户角色表，记录角色与权限的关系
role_info	角色信息表，记录角色信息
user_info	账户信息表，记录账户名、密码等信息
user_role	用户角色信息表，记录账户与角色的关系

- 执行SQL

```
sqlite> select * from system_config;
block_height|230684|链最新块高|20200228135900|20200228135900
trans_count|191|链交易总量|20200229005929|20200229005929
```

2.自愈恢复能力

自愈恢复能力已集成至BaaS私有化标准版，该能力将大大缓解运维的工作压力。

背景介绍

传统方式下的私有化项目，部署至客户现场后，可以通过软件交付自带的管控平台对网络和链节点进行监控以实时获取最新状态。当外界因素（断电机器重启、服务器宕机、节点磁盘写满）导致某些正在运行中的节点运行状态出现异常时，只能通过邮件、钉钉、电话等方式发出告警，并由技术或运维同学进行问题排查和解决。

基于上述的运维方式，我们针对私有化部署场景配套提供了随系统自运行的自愈恢复能力，它能够在管控监控数据刷新的同时针对常见的内置故障类型自发的完成从故障感知、故障诊断、故障恢复一系列闭环操作，从而提高故障的解决效率，降低运维成本。

使用说明

默认情况下，自愈恢复功能不启用。如果需要开启，只需打开配置文件中的开关（enable设置为true）即可。

```
# brain智能诊断及自愈设置
brain:
  enable: false
  chain_black_list:
  treatment_black_list:
  event_check_period: 60
  event_check_count: 3
  event_check_count_expire: 7200
  event_namespace: midway_brain_
  event_suffix: _event
  event_count_suffix: _eventCount
```

以上各配置项含义如下表所示。

配置项	说明
enable	是否开启自愈恢复功能： • false：默认值，自愈恢复功能未开启 • true：自愈恢复功能已开启
chain_black_list	指定哪些链是需要过滤掉启用自愈恢复能力（如果存在多条，中间用逗号分隔）
treatment_black_list	指定废除掉哪些自愈方案（如果存在多条，中间用逗号分隔）
event_check_period	故障事件收敛周期（一个收敛周期内重复出现的同一故障忽略不处理）
event_check_count	同一故障事件处理次数上限（重试超过该次数则针对此次故障不做自愈恢复，需人工介入）
event_check_count_expire	故障诊断次数缓存过期时间
event_namespace	用于标识故障事件缓存key统一前缀
event_suffix	用于标识故障事件缓存key统一后缀
event_count_suffix	用于标识故障事件出现次数缓存key统一后缀

3.跨子网通讯能力

3.1. 概述

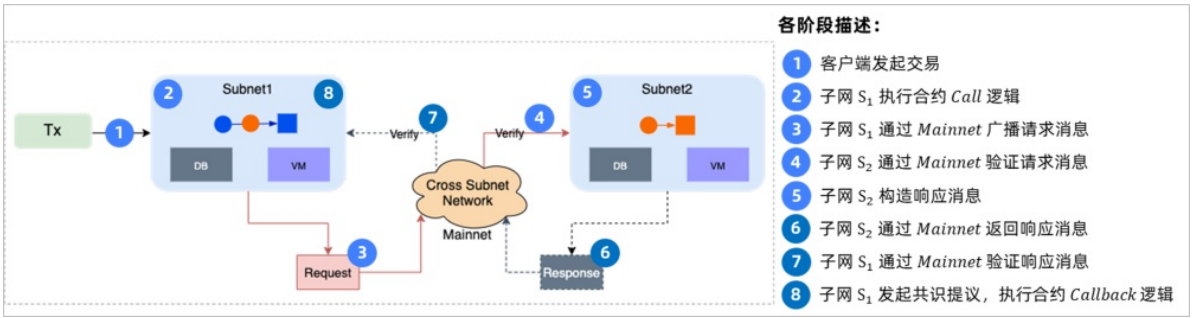
业务背景

目前区块链技术已经迈入到大规模商业应用阶段，大型场景日均交易量过亿，不仅在区块链应用的性能方面提出了新要求，在安全性、稳定性和可用性等各个方面也都提出了更高级别的要求。因为复杂的隐私保护、多方协同等实际需求，一些应用场景面临多元化组网设计问题。蚂蚁链针对实际需求设计了动态组网（Subnet）功能，支持联盟内参与方按需自由配置子网，从单一参与方而言，即可同时参与不同业务；与多种业务混杂于同一主链相比，子网方案有利于业务隔离，实现账本数据、共识流程等隔离。

在此基础上，还支持跨子网数据查询和合约读操作，如 `QueryBlock` 、 `QueryBlockHeader` 、 `QueryTransaction`和`LocalCall` 。

跨子网信息交互模型

跨子网信息交互模型：子网 S_1 上的请求信息经过共识确定，可以安全可靠地广播到子网S_2中，子网S_2可以认证请求信息，并且安全可靠地返回数据信息。



- **跨子网通讯协议：** 设计跨子网的通讯协议，提供业务安全可靠的跨子网通讯机制，保证数据读取的一致性、安全性以及子网间的信任传递机制。
- **基于主网的信任根校验：** MainnetDB主网是特殊的子网，负责子网信息的管理；子网消息传递中携带节点签名，主网负责验证成员在子网的有效性。
- **合约异步编程：** Call和 的合约异步编程，提供业务跨子网通讯时一种可用的合约编写方案Callback 支持。

功能说明

- **子网授权管理**
  子网授权管理，包括**数据查询**、**合约读**的新建（ `AddAcQuery/` ）、查询（ `FindAcQuery/` ）、删除（ `DeleteAcQuery/` ）、列表（ListSubnetAccessControl）等操作，通过调用子网上部署的 `SubnetAccessControl.sol` 合约实现。

  > **说明** 调用方式与其他Subnet的系统合约调用方式一致，需要使用Administrator的账户权限。

- **子网授权子命令**

  子网授权是在多子网基础之上的操作，对于子网间授权，在对应的子网上新增授权规则，即子网授权子命令是在各个子网上发起的操作，授权规则中指定允许的子网 Id 和方法。

- **跨子网服务**

  跨子网数据读服务，支持子网A在子网B上进行**数据查询**、**合约读**操作，一方面需要子网B允许子网A访问的权限，另一方面，需要业务合约按照要求实现对应的调用功能。

3.2. 使用说明

操作流程

跨子网通讯的流程包括：

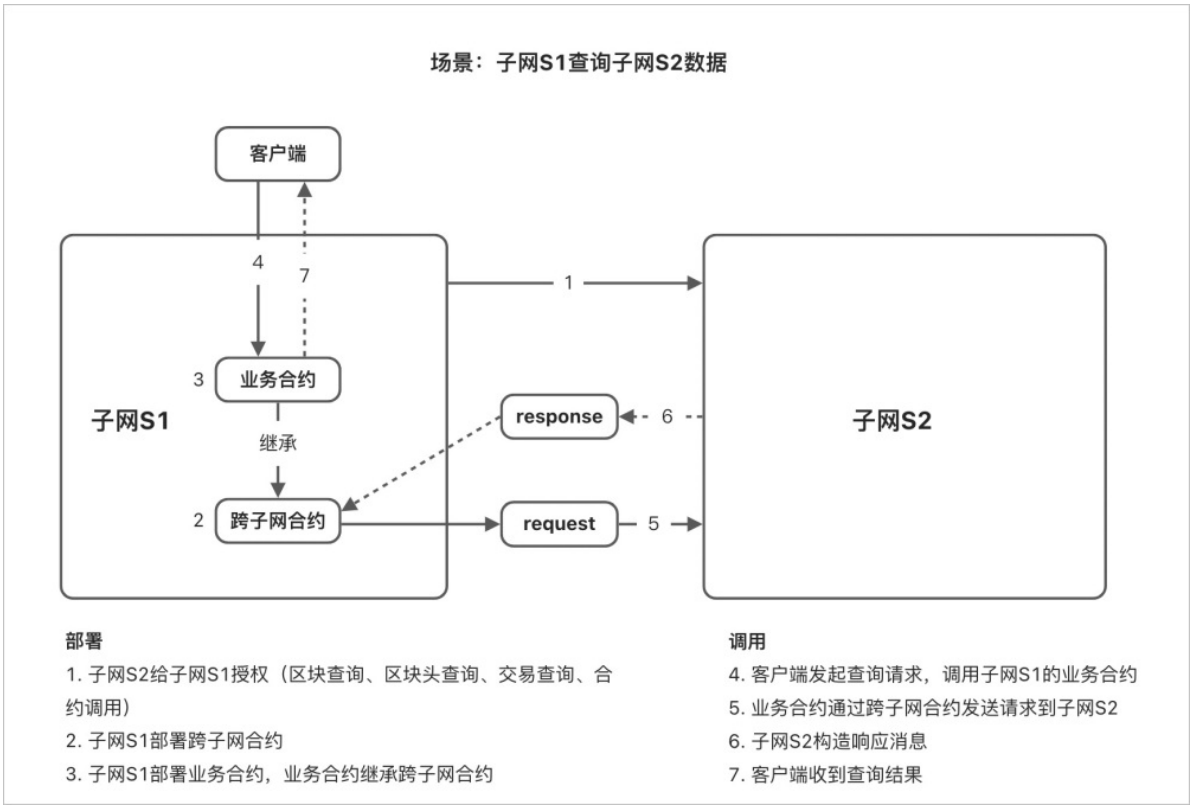
1. 子网创建
2. 子网间授权
3. 跨子网合约部署
4. 业务合约编写
5. 业务合约部署
6. 业务合约调用

跨子网通讯的工程开发建议使用**蚂蚁链管控平台**创建子网，**mychain_tool**工具完成子网间授权和跨子网合约部署，**蚂蚁链SDK**实现业务合约的部署和调用。

> 说明
> - 蚂蚁链管控平台创建子网的操作可参见《蚂蚁链BaaS私有化标准版用户指南》中创建区块链 > 创建子链章节。
> - mychain_tool的使用请参见mychain_tool章节。
> - 蚂蚁链SDK的使用请参见《蚂蚁链BaaS私有化标准版开发指南》。

跨子网通讯流程图

以子网S1查询子网S2数据为例，跨子网通讯流程图如下。



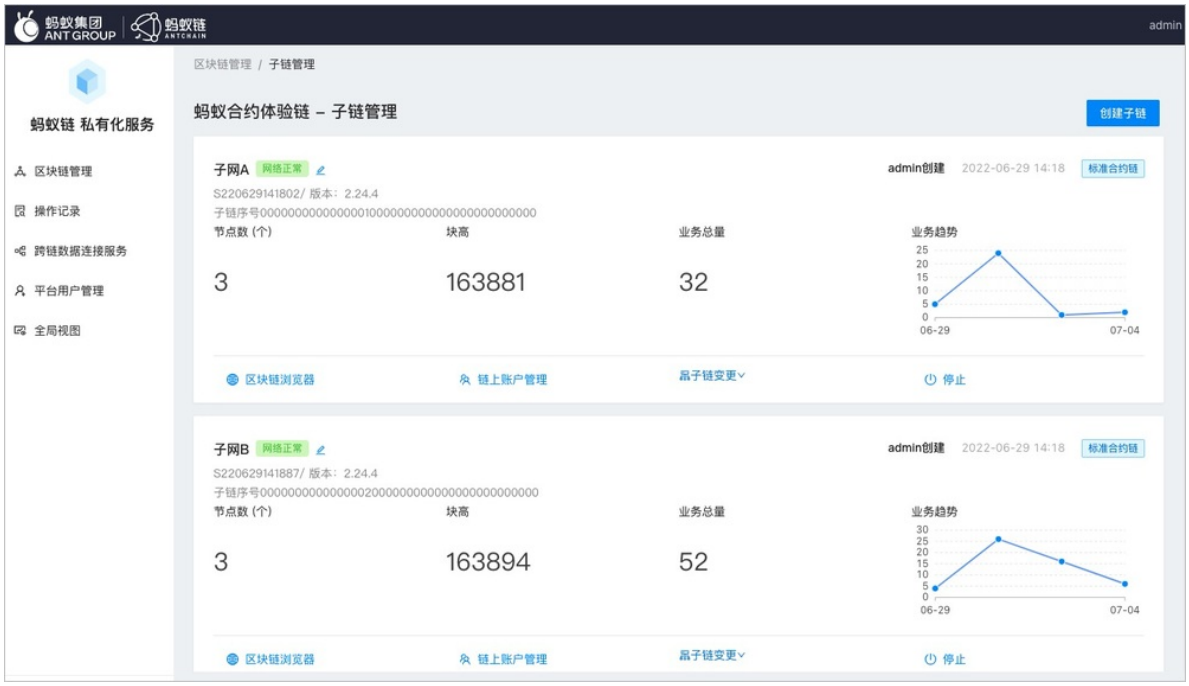
3.3. 跨子网操作流程

3.3.1. 子网创建

子网创建可以通过mychain_tool或蚂蚁链管控平台两种方式进行创建。

方式一：蚂蚁链中途岛管控台（推荐）

具体操作步骤请参见《蚂蚁链BaaS私有化标准版用户指南》中创建区块链 > 创建子链章节。



方式二：mychain_tool

执行如下命令分别创建子网A、子网B：

```
./mychain_tool subnet -m add -c ../conf/genesis_subnet.json
```

> 说明 具体可参见《蚂蚁链BaaS私有化标准版运维指南》的mychain_tool > subnet子命令。

3.3.2. 子网间授权

子网授权管理

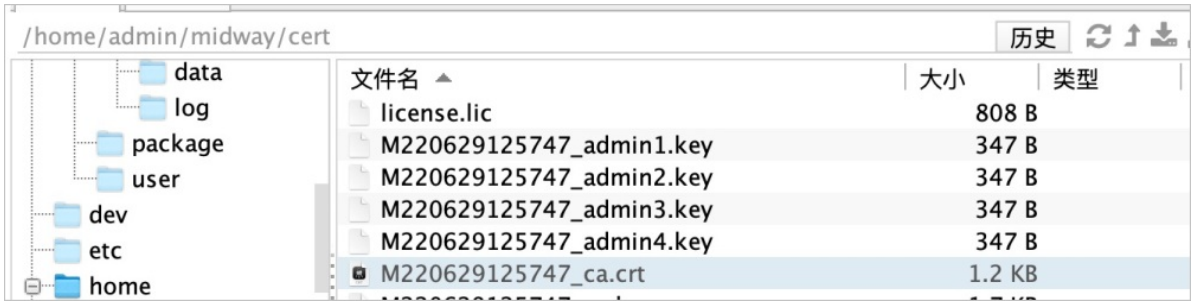
子网授权管理，包括**数据查询**、**合约读**的新建（`AddAcQuery/`）、查询（`FindAcQuery/`）、删除（`DeleteAcQuery/`）、列表（`ListSubnetAccessControl`）等操作，通过调用子网上部署的`SubnetAccessControl.sol`合约实现。子网授权管理需要使用Administrator的账户权限，权限配置流程如下。

管理员的配置信息在../conf/node_config.json（相对于mychain_tool的路径，mychain_tool工具的位置在/data1/mychain_deploy/bin），修改配置文件中的*user_keys*、*ca_path*、*cert_path*、*key_path*。

```
{
 "node_config" : {
 "identity" : "Administrator",
 "key_path" : "../../user",
 "digest_type" : "sha256",
 "user_keys" : [
 {
 "key_name" : "Administrator1.key",
 "key_passwd" : "123abc"
 },
 {
 "key_name" : "Administrator2.key",
 "key_passwd" : "123abc"
 },
 {
 "key_name" : "Administrator3.key",
 "key_passwd" : "123abc"
 }
 ],
 "node" : {
 "endpoints" : [
 "127.0.0.1:18130"
 ],
 "ca_path" : "../../ca/ca.crt",
 "cert_path" : "../../ca/client.crt",
 "key_path" : "../../ca/client.key",
 "passwd" : "123abc",
 "enable_anchor" : false
 }
 }
}
```

其中，*user_keys*和*ca.crt*在midway的目录/home/admin/midway/cert下，*client.crt*和*client.key* 在目录/data1/mychain_deploy/certs下。

passwd默认**123abc**。



子网授权子命令

子网授权是在多子网基础之上的操作，对于子网间授权，在对应的子网上新增授权规则，即子网授权子命令是在各个子网上发起的操作，授权规则中指定允许的子网 Id 和方法，如命令形式：在子网A上允许子网B进行 `QueryBlock` 的操作。

- 授权示例如下：

```
# 通过下标方式，或者子网标识方式，都可以
./mychain_tool subnet_access_control -t addq -m QueryBlock -i 2 -g 1

./mychain_tool subnet_access_control -t addq -m QueryBlock -i
0000000000000002000000000000000000000000 -g     0000000000000001000000000000000000000000
```

- 查询授权列表：

```
# 查询当前子网授权列表
> ./mychain_tool subnet_access_control -t list -g 1
get input subnet index 1
Execute SysContractCommand on subnet : 0000000000000001000000000000000000000000

start to execute the action:

call contract transaction:
[{"hash":"a499b2695095655382138811l7b725f8bc70571e0bf3c58a0de3d5c62242a156","type":"11","timestamp":"1656577143708","no
:"6091653609518628006","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21f
8b64bde0491f9347cbc73854576bba59bcf9dd8cf03e39edb44ace130","value":"0","gas":"10000000","group_id":"000000000000000010000
0000000000000000","version":"2","data":"dd9feb5b","extensions":"","signature":
["ee9ff76c3001d4d6a147ff5fb5e4d64549f97eefbd4397e0e8602d36099edfb21d04b4e499c4d537bd9b9370461f23d0893bb0abea49d3907677f
1cb72a501","94a13f462247d486d389ec2cd0aa2514b48775a95bf5a0f6b5879a25fcc960e4613732dd288dcbe5fcd7360dadcd16707a26cebe3c4
085904b5654d4367200","2293c36fedfb129ab93d71cf8b5b3d72d7e721ae2cdfba22693dbecb02bf15eb4c24b6a4ed93b9a5fbb737ffdcc51bb4b
49688abeb31e95954825b5be08501","c778fbd331e00efde005116fdbdd0ee08a7130beb4b9478aa52583602f869dd87aa434bf6af13a461181b3f
70701116d5eedef92c61a37cd3d59fcdb78e801"]}]

Call contract transaction receipt:
[{"result":"0","gas_used":"41508","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288b64bde0491f9347cbc73854576bba
f9dd8cf03e39edb44ace130","topics":
["call_contract"],"log_data":""}],"output":"0000000000000000000000000000000000000000000000000000000000000020000000000000
0000000000000000000000000000000000000000000000867b202261635f717565727922203a205b207b20227065726d69747465645f6d796368616
5f696422203a2022303030303030303030303030303030303230303030303030303030303030303030303030303030303030222c20226d6574686f64222
0225175657279426c6f636b22207d5d2c202261635f63616c6c22203a205b20205d207d0000000000000000000000000000000000000000000000000
"}]

# 对 output 进行解码，如下：当前列表为空
{ "ac_query" : [ { "permitted_mychain_id" : "0000000000000020000000000000000000000000", "method" : "QueryBlock" }], "
ac_call" : [ ] }
```

> **说明** `-g 1` 代表查询在某个子网执行命令，可通过在线工具解码 output。

- 更多查询结果展示如下：

```
{
 "ac_query": [{
 "permitted_mychain_id": "0000000000000020000000000000000000000000",
 "method": "QueryBlock"
 }, {
 "permitted_mychain_id": "0000000000000020000000000000000000000000",
 "method": "QueryBlockHeader"
 }, {
 "permitted_mychain_id": "0000000000000020000000000000000000000000",
 "method": "QueryTransaction"
 }],
 "ac_call": [{
 "permitted_mychain_id": "0000000000000020000000000000000000000000",
 "method": "LocalCall",
 "contract_id": "27321e07197e4d90d196e8ddec937344e6c7803f1d203d2e564eb3185e8b1ce1",
 "contract_function": "set(uint256)"
 }]
}
```

相关字段的具体含义如下：

- ac_query：授权数据查询
- ac_call：授权合约LocalCall
- permitted_mychain_id：本子网允许访问的其他子网
- contract_id：合约地址
- contract_function：合约方法

3.3.3. 跨子网合约部署

子网S与其他子网通讯，需要在子网S上部署跨子网服务合约。合约部署可以使用mychain_tool工具。

Solidity 合约形式

平台支持的跨子网Soliidty合约地址：CrossSubnetServiceEvm的hash值：

- （sha256）普通链： `0x64c34185322ab6c40640d526359c08ea1d19a0dc74906453a40ecfdbd30c72c6`
- （sm3）国密链： `0xccd6af7efc394a42f9a47693fcc51cdf184af6facad888d6c8e84937256ae7f1`

合约部署

```
./mychain_tool system_manage -a --name CrossSubnetServiceEvm -f
../conf/compiled_system_contracts/contract_cross_subnet_service.evm -g 0
```

合约升级

```
./mychain_tool system_manage -u --name CrossSubnetServiceEvm -f
../conf/compiled_system_contracts/contract_cross_subnet_service.bin-runtime -g 0
```

Wasm 合约形式

平台支持的跨子网WASM合约地址：CrossSubnetServiceWasm的hash值：

- （sha256）普通链：`0x8619e5a1ad2e5163c9c78f08ba28cd8481b68b93cdda6b7e63fc4b6a1f47ae40`
- （sm3）国密链：`0x949955db255d0751ed63d5efe2df67cee2e10e0ba0f2ea5f76c063ddd6c8cbc8`

合约部署

```
./mychain_tool system_manage -a --name CrossSubnetServiceWasm -f
../conf/compiled_system_contracts/contract_cross_subnet_service.wasc -g 0
```

合约升级

```
./mychain_tool system_manage -u --name CrossSubnetServiceWasm -f
../conf/compiled_system_contracts/contract_cross_subnet_service.wasc -g 0
```

> **说明** 上述是主网上部署，如果是对应子网，需要加上 `-g（子网下标、标识）`，如 `-g 1`。使用 `./mychain_tool contract -c CrossSubnetServiceEvm -g 1` 可以查询在对应子网上是否已经部署了这个合约。

3.3.4. 业务合约编写

Solidity 合约形式

Solidity 合约开发流程：

- 目录：integrate_test/data_prod/common/biz_stable.evm
- 源文件：https://btn-test.oss-cn-hangzhou.aliyuncs.com/subnet/biz_stable.evm
- md5sum：34f23518d4bc4a9091a663c99b4fcf05

```
pragma solidity ^0.4.2;

import "./strings.sol";
import "./utils.sol";
import './CrossSubnetInterface.sol';

// 实现一个 demo 合约
contract BizStableContract {
 using strings for *;
 using utils for *;

 // 跨子网合约
 identity ibc_base_address;
 // 业务 ID 与跨子网访问请求 ID 的关联关系
 mapping(bytes32 => bytes32) requests;
 mapping(bytes32 => string) request_datas;
 mapping(bytes32 => bool) request_flag;

 // Query Block/State/Tx
 struct BizRequest {
 bytes32 biz_id;
 bytes32 request_id;
 }

 BizRequest[] bizRequests;

 event LOG(bytes32 biz_id, bytes32 request_id, string response_data);

 // 权限控制
 modifier onlyIbcBase() {
 require(msg.sender == ibc_base_address, "PERMISSION_ERROR: Permission denied!");
 _;
 }
```

```
 // 配置跨子网合约 ID
 function setIbcBaseAddress(identity _addr) public {
 ibc_base_address = _addr;
 }

 // 调用跨子网合约的跨子网访问接口访问目标区块链指定一笔交易
 function crossSubnetRequest(bytes32 biz_id, string src_subnet_id, string dest_subnet_id, string method, string args) p
ublic returns (bytes32) {

 // 1. 跨合约调用，需要通过合约接口定义及跨子网合约 ID 生成一个合约对象
 CrossSubnetInterface ibc = CrossSubnetInterface(ibc_base_address);

 // 2. 拼装跨子网访问命令，拼装区块链头查询命令


 // 3. 发送跨子网访问请求
 bytes32 request_id = ibc.crossSubnetRequest(biz_id, src_subnet_id, dest_subnet_id, method, args, true, this, 0);

 // 4. 记录跨子网合约返回的 request id
 requests[biz_id] = request_id;

 BizRequest memory bizRequest = BizRequest(biz_id, request_id);
 bizRequests.push(bizRequest);

 // 5. 请求阶段结束，等待回调
 request_flag[request_id] = false;
 return request_id;
 }

 // 业务合约用于接收跨子网合约的跨子网访问请求结果回调
 function crossSubnetCallbackResponse(bytes32 _request_id, bytes32 _biz_id, uint32 _error_code, string _resp_body,
identity _call_identity) onlyIbcBase external returns (bool) {
 // 业务处理返回跨子网
 emit LOG(_biz_id, _request_id, _resp_body);

 request_datas[_request_id] = _resp_body;
 request_flag[_request_id] = true;
 return true;
 }

 // 查询 request_id 对应的 response_data
 function getRequest(bytes32 _request_id) public returns (string) {
 require(request_flag[_request_id], "not exist");

 return request_datas[_request_id];
 }

}
```

Wasm 合约形式

Wasm 合约开发流程：

- 目录：integrate_test/data_prod/resource/vm/wasm/subnet/BizStableContract.cpp
- 源文件：https://btn-test.oss-cn-hangzhou.aliyuncs.com/s()ubnet/biz_stable.wasc
- md5sum：c59e56e7a2e84686410dffdef487a940

```
#include <mychainlib/contract.h>
#include "./schema/biz_ant_generated.h"

using namespace mychain;
using namespace BizStates;

CONTRACT_VERSION(24);
class BizStableContract:public Contract {

public:

 // 跨子网数据访问接口的 API 定义
 const std::string CROSS_SUBNET_REQUEST_API = "crossSubnetRequest";

 StatesMPtr m_states;

 BizStableContract(){
```

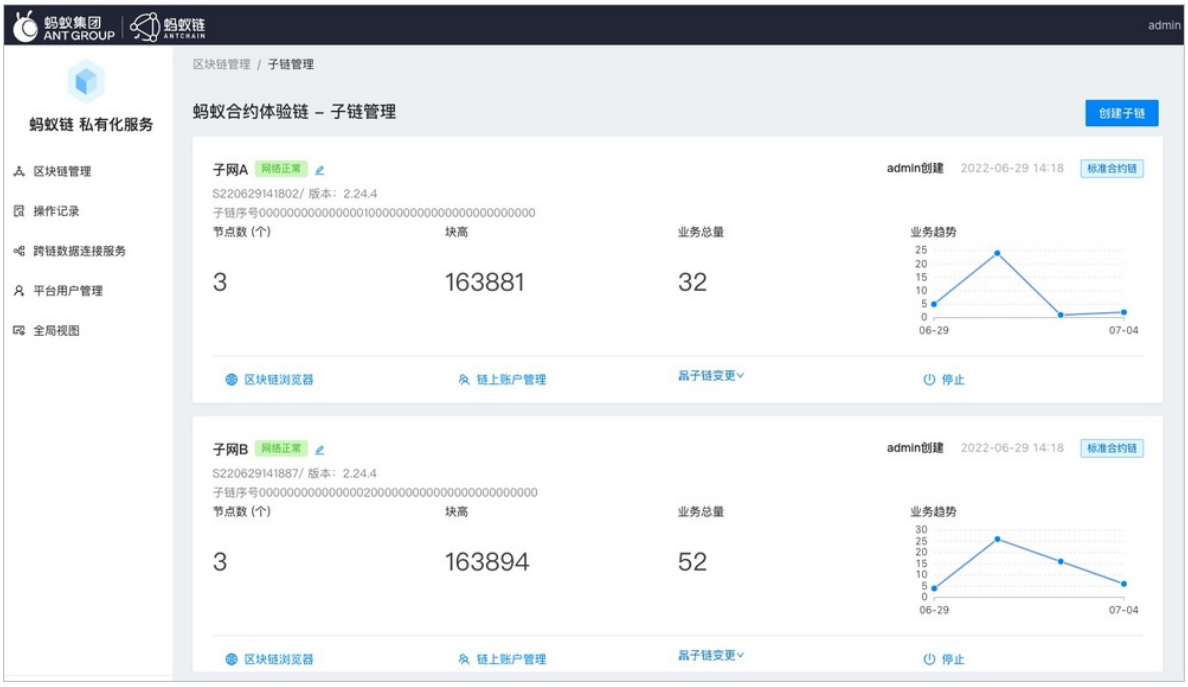
```
m_states = GetStatesM();
}

INTERFACE void Init() {
InitRoot(); //初始化schema持久化存储环境
m_states = GetStatesM();
}

/**
* 调用跨子网服务合约的数据访问接口
**/
INTERFACE std::string crossSubnetRequest(const std::string& ibc_base_address, const std::string& biz_id, const
std::string& src_subnet_id, const std::string& dest_subnet_id, const std::string& method, const std::string& args){

// 1. 调用跨子网合约接口，发送请求
Identity address(ibc_base_address);
auto ret = CallContract<std::string>(address, CROSS_SUBNET_REQUEST_API, 0, 0,
biz_id,
src_subnet_id,
dest_subnet_id,
method,
args,
true,
GetSelf().to_hex(),
0);

// 2. 调用结果校验
Require(ret.code == 0, "call cross_subnet_contract fail " + address.to_hex() + " : " + ret.msg);

// 3. 处理请求信息
// 一般保存请求 ID，即 ret.result 字段

// 4. 请求阶段结束，等待回调
std::string request_id = ret.result;
std::string key = Hex2Bin(request_id);
ExistDOMPtr exist;
if (m_states->get_request_flag()->has_element(key)) {
println("have\n%s", key.c_str());
exist = m_states->get_request_flag()->get_element(key);
} else {
println("not have\n%s", key.c_str());
exist = m_states->get_request_flag()->add_element(key);
}
exist->set_is_exist(false);
return request_id;
}

/**
* 业务合约用于接收跨子网服务合约的数据访问请求结果回调
**/
INTERFACE void crossSubnetCallbackResponse (const std::string& _request_id,
const std::string& _biz_id,
const uint32_t& _error_code,
const std::string& _resp_body,
const std::string& _call_identity){

// 业务自定义的逻辑，处理返回数据
// ...
println("show crossSubnetCallbackResponse back:%s", _request_id.c_str());
println("show resp_body:%s", _resp_body.c_str());
IdentityDOMPtr id = m_states->get_request_datas()->add_element(_request_id);
id->set_identity(_resp_body);
ExistDOMPtr exist;
if (m_states->get_request_flag()->has_element(_request_id)) {
println("crossSubnetCallbackResponse have\n");
exist = m_states->get_request_flag()->get_element(_request_id);
} else {
println("crossSubnetCallbackResponse not have\n");
exist = m_states->get_request_flag()->add_element(_request_id);
}
exist->set_is_exist(true);
return;
}
```

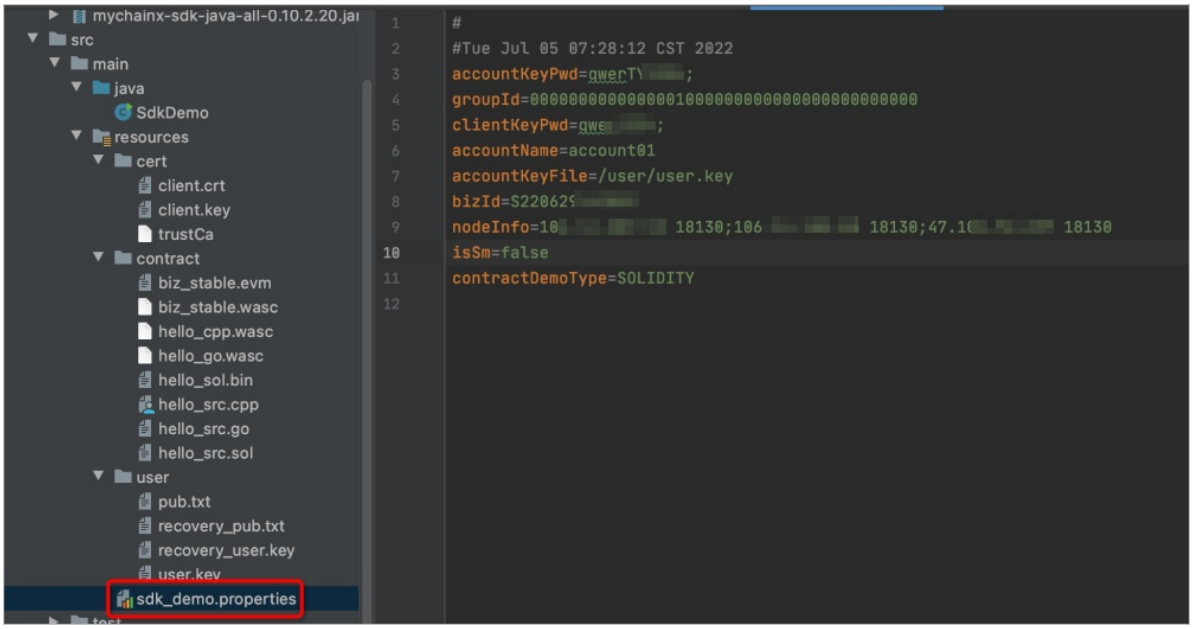
```
// 查询 request_id 对应的 response_data
INTERFACE std::string getRequest(const std::string& _request_id) {
std::string key = Hex2Bin(_request_id);
println("getRequest key:%s\n%s", _request_id.c_str(), key.c_str());
Require(m_states->get_request_flag()->has_element(key), "not exist");
IdentityDOMPtr id = m_states->get_request_datas()->get_element(key);
std::string data = id->get_identity();
printf("data:%s\n", data.c_str());
return data;
}

};
INTERFACE_EXPORT(BizStableContract,
(Init)(crossSubnetRequest)(crossSubnetCallbackResponse)(getRequest))
```

3.3.5. 业务合约部署

SDK下载

参见**《蚂蚁链BaaS私有化标准版用户指南》**中**下载SDK代码包**章节获取子链SDK代码包，并修改代码中的相关参数配置以接入区块链。



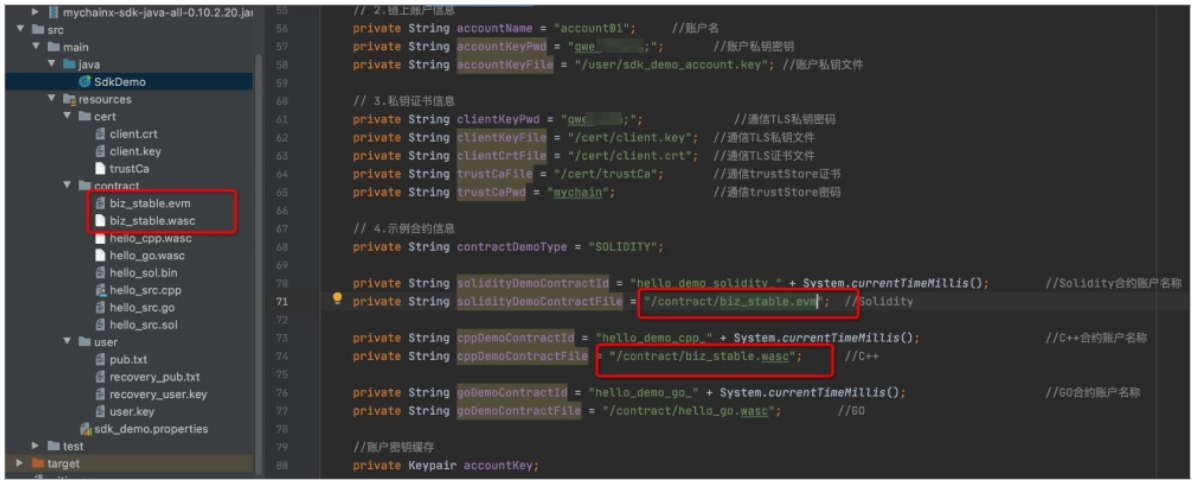


合约部署

工程中放入业务合约，按照正常合约的流程进行部署、调用。

下载Demo包，并完成以下步骤。

1. 引入业务合约（solidity合约和wasm合约）。



2. 部署业务合约。
   - 部署Solidity合约

```
private void deploySolidityContract() throws Exception {
        Identity account = generateIdentity(this.accountName);
        Identity contract = generateIdentity(getDemoContractId(contractDemoType));

        EVMParameter contractParameters = new EVMParameter();

        byte[] contractCode =
Hex.decode(IOUtil.inputStreamToByte(SdkDemo.class.getClass().getResourceAsStream(solidityDemoContractFile)));

        DeployContractRequest request = new DeployContractRequest(account, contract, contractCode, VMTypeEnum.EVM,
                contractParameters, BigInteger.ZERO);

        request.setGroupId(Fixed20ByteArray.valueOf(this.groupId));
        signRequest(request);

        TransactionReceiptResponse result = sdk.getContractService().deployContract(request);

        if (!result.isSuccess()
                || result.getTransactionReceipt().getResult() != 0) {
            exit("deploySolidityContract", getErrorMsg((int) result.getTransactionReceipt().getResult()));
        } else {
            System.out.println("deploy solidity contract success.");
        }
    }
```

- 部署WASM合约

```
private void deployCppContract() throws Exception {
        Identity account = generateIdentity(this.accountName);
        Identity contract = generateIdentity(getDemoContractId(contractDemoType));

        WASMParameter contractParameters = new WASMParameter();

        byte[] contractCode =
IOUtil.inputStreamToByte(SdkDemo.class.getClass().getResourceAsStream(cppDemoContractFile));

        DeployContractRequest request = new DeployContractRequest(account, contract, contractCode, VMTypeEnum.WASM,
                contractParameters, BigInteger.ZERO);

        request.setGroupId(Fixed20ByteArray.valueOf(this.groupId));
        signRequest(request);

        TransactionReceiptResponse result = sdk.getContractService().deployContract(request);

        if (!result.isSuccess()
                || result.getTransactionReceipt().getResult() != 0) {
            exit("deployCppContract", getErrorMsg((int) result.getTransactionReceipt().getResult()));
        } else {
            System.out.println("deploy cpp contract success.");
        }
    }
```

3. 设置跨子网业务合约地址。

   案例中业务合约地址为：`0x64c34185322ab6c40640d526359c08ea1d19a0dc74906453a40ecfdbd30c72c6`。

```
private void callContractSetIbcBaseAddress() {
 Identity account = generateIdentity(this.accountName);
 Identity contract = generateIdentity(getDemoContractId(contractDemoType));

 Identity identity = new Identity("0x64c34185322ab6c40640d526359c08ea1d19a0dc74906453a40ecfdbd30c72c6");
 EVMParameter parameters = new EVMParameter("setIbcBaseAddress(identity)");
 parameters.addIdentity(identity);

 CallContractRequest request = new CallContractRequest(account, contract, parameters, BigInteger.ZERO,
VMTypeEnum.EVM);
 request.setGroupId(Fixed20ByteArray.valueOf(this.groupId));

 CallContractResponse response = sdk.getContractService().callContract(request);
 if (!response.isSuccess()) {
 exit("callSolidityContract", getErrorMsg((int) response.getTransactionReceipt().getResult()));
 } else {
 System.out.println("callSolidityContract success. :" + response.toString());
 }
 }
```

3.3.6. 业务合约调用

合约调用

- 调用跨子网请求接口

> ⚠ **重要** 业务合约调用前，需子网B给子网A授权区块查询权限。同理，查询其他信息时，也需要提前配置对应权限。

以跨子网查询区块为例，子网A查询子网B的第5个区块，示例代码如下。

- Solidity合约调用

```
private void callContractCrossSubnetRequest() {
        Identity account = generateIdentity(this.accountName);
        Identity contract = generateIdentity(getDemoContractId(contractDemoType));

        byte[] bytes =  hexStringToByteArray("044852B2A670ADE5407E78FB2863C51DE9FCB96542A07186FE3AEDA6BB8A116F");
        EVMParameter parameters = new EVMParameter("crossSubnetRequest(bytes32,string,string,string,string)");
        parameters.addBytes32(bytes);
        parameters.addString("0000000000000001000000000000000000000000");
        parameters.addString("0000000000000002000000000000000000000000");
        parameters.addString("QueryBlock");
        parameters.addString("{\"number\":5}");


        CallContractRequest request = new CallContractRequest(account, contract, parameters, BigInteger.ZERO, VMType
Enum.EVM);
        request.setGroupId(Fixed20ByteArray.valueOf(this.groupId));

        CallContractResponse response = sdk.getContractService().callContract(request);
        if (!response.isSuccess()) {
            exit("callSolidityContract", getErrorMsg((int) response.getTransactionReceipt().getResult()));
        } else {
            System.out.println("callSolidityContract success. :" + response.toString());
        }

        // 交易收据
        TransactionReceipt transactionReceipt = response.getTransactionReceipt();
        if (transactionReceipt.getResult() != 0) {
            System.out.println(" fail.返回信息: "+ transactionReceipt.toString());
        } else {
            System.out.println(" success.返回信息: "+ transactionReceipt.toString());
            JSONObject jsonObject = JSON.parseObject(transactionReceipt.toString());

             requestId = jsonObject.getString("output");
        }
    }
```

- WASM合约调用

```
private void callCppContractCrossSubnetRequest() {
        Identity account = generateIdentity(this.accountName);
        Identity contract = generateIdentity(getDemoContractId(contractDemoType));

        WASMParameter wasmParameter = new
WASMParameter("crossSubnetRequest(string,string,string,string,string,string)");
        //ibc_base_address 跨子网业务合约地址
        wasmParameter.addString("64c34185322ab6c40640d526359c08ea1d19a0dc74906453a40ecfdbd30c72c6");
        //biz_id
        wasmParameter.addString("044852B2A670ADE5407E78FB2863C51DE9FCB96542A07186FE3AEDA6BB8A116F");
        wasmParameter.addString("0000000000000001000000000000000000000000");
        wasmParameter.addString("0000000000000002000000000000000000000000");
        wasmParameter.addString("QueryBlock");
        wasmParameter.addString("{\"number\":5}");
        CallContractRequest request = new CallContractRequest(account, contract, wasmParameter, BigInteger.ZERO, VMT
ypeEnum.WASM);
        request.setGroupId(Fixed20ByteArray.valueOf(this.groupId));

        CallContractResponse response = sdk.getContractService().callContract(request);
        if (!response.isSuccess()) {
            exit("callCppContract", getErrorMsg((int) response.getTransactionReceipt().getResult()));
        } else {
            System.out.println("callCppContract success. :" + response.toString());
        }
        // 交易收据
        TransactionReceipt transactionReceipt = response.getTransactionReceipt();
        if (transactionReceipt.getResult() != 0) {
            System.out.println(" fail.返回信息: "+ transactionReceipt.toString());
        } else {
            System.out.println(" success.返回信息: "+ transactionReceipt.toString());
            JSONObject jsonObject = JSON.parseObject(transactionReceipt.toString());

            requestId = jsonObject.getString("output");
        }
    }
```

结果输出如下：

```
success.返回信息:
{"result":0,"output":"7f8b6b088b6d74c2852fc86c796dca07b44eed6fb3daf5e6b59f7c364db14528","gas_used":757727,"logs":
[{"topics":
["call_contract"],"from":"9e5f9e75192fa45becb945bc5cc52212cc0033e435af7904c186a307e211a347","log_data":"","to":"fc33afc4
6b5daf1633427b3989ecdd0fe24d0a2cd1159c2a1aa2c40e553ad"},{"topics":
["868ebf9c03895e350912f3c8c814898dd7f2e09b8235cbce7917be16fcb7bc0f"],"from":"fc33afc47116b5daf1633427b3989ecdd0fe24d0a2c
59c2a1aa2c40e553ad","log_data":"0000000000000000000000000000000000000000000000000000000000000120044852b2a670ade5407e78fb
3c51de9fcb96542a07186fe3aeda6bb8a116f7f8b6b088b6d74c2852fc86c796dca07b44eed6fb3daf5e6b59f7c364db1452800000000000000000000
0000000000000000000000000000000000000000180000000000000000000000000000000000000000000000000000000000000000000001aa0000000000000
0000000000000000000000000000000000181cc85033e0000000000000000000000000000000000000000000000000000000000000001fc33afc47
b5daf1633427b3989ecdd0fe24d0a2cd1159c2a1aa2c40e553ad0000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000283030303030303030303030303030323030303030303030303030303030
3030303030303000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00a51756572794 26c6f636b0000000000000000000000000000000000000000000000000000000000000c7b226e756d626572223a357d","to":"
4185322ab6c40640d526359c08ea1d19a0dc74906453a40ecfdbd30c72c6"}]}
```

- 调用查询结果接口

  入参**requestId**为上一步返回的**output**：`7f8b6b088b6d74c2852fc86c796dca07b44eed6fb3daf5e6b59f7c364db14528` 。

  示例代码如下：

- Solidity合约调用

```
private void callContractGetRequest(String requestId) {
        System.out.println("request_id:" + requestId);
        Identity account = generateIdentity(this.accountName);
        Identity contract = generateIdentity(getDemoContractId(contractDemoType));

        byte[] bytes =  hexStringToByteArray(requestId);
        EVMParameter parameters = new EVMParameter("getRequest(bytes32)");
        parameters.addBytes32(bytes);


        CallContractRequest request = new CallContractRequest(account, contract, parameters, BigInteger.ZERO, VMType
Enum.EVM);
        request.setGroupId(Fixed20ByteArray.valueOf(this.groupId));

        CallContractResponse response = sdk.getContractService().callContract(request);
        if (!response.isSuccess()) {
            exit("callSolidityContract", getErrorMsg((int) response.getTransactionReceipt().getResult()));
        } else {
            System.out.println("callSolidityContract success. :" + response.toString());
        }


        // 交易收据
        TransactionReceipt transactionReceipt = response.getTransactionReceipt();
        if (transactionReceipt.getResult() != 0) {
            System.out.println(" fail.返回信息: "+ transactionReceipt.toString());
        } else {
            System.out.println(" success.返回信息: "+ transactionReceipt.toString());
            JSONObject jsonObject = JSON.parseObject(transactionReceipt.toString());
        }
    }
```

- WASM合约调用

```
private void callCppContractGetRequest(String requestId) {
        System.out.println("request_id:" + requestId);
        Identity account = generateIdentity(this.accountName);
        Identity contract = generateIdentity(getDemoContractId(contractDemoType));

        WASMParameter wasmParameter = new WASMParameter("getRequest(string)");
        wasmParameter.addString(requestId);
        CallContractRequest request = new CallContractRequest(account, contract, wasmParameter, BigInteger.ZERO, VMT
ypeEnum.WASM);
        request.setGroupId(Fixed20ByteArray.valueOf(this.groupId));

        CallContractResponse response = sdk.getContractService().callContract(request);
        if (!response.isSuccess()) {
            exit("callCppContract", getErrorMsg((int) response.getTransactionReceipt().getResult()));
        } else {
            System.out.println("callCppContract success. :" + response.toString());
        }
        // 交易收据
        TransactionReceipt transactionReceipt = response.getTransactionReceipt();
        if (transactionReceipt.getResult() != 0) {
            System.out.println(" fail.返回信息: "+ transactionReceipt.toString());
        } else {
            System.out.println(" success.返回信息: "+ transactionReceipt.toString());
            JSONObject jsonObject = JSON.parseObject(transactionReceipt.toString());
        }
    }
```

结果输出如下：

```
callSolidityContract success.
:
{"sequence":10,"trace_id":"","group_id":"0000000000000001000000000000000000000000","msg_name":"transaction_CallContract_
p","msg_type":21,"return_code":0,"hash":"e0b076c3490676f6dd363da59115683663f9a1845292cfe0e1d37896a6155a87"}
success.返回信息:
{"result":0,"output":"000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000
00000000000000000000000044b7b22686561646572223a7b2268617368223a22353237323164626430353534396333643433646435393761323235
3561356537636261623863393634653364323939333261383438336134346431313532222c2276657273696f6e223a22373230353735393631383837
732383334222c226e756d626572223a2235222c22706172656e745f68617368223a223338663665313566323032663333366131386363353637323838
61386130373833343062363230653165353831333239353765326263643863363534353934222c227472616e73616374696f6e5f726f6f74223a2230
030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303
222c22726563656970745f726f6f74223a22303030303030303030303030303030303030303030303030303030303030303030303030303030303030
0303030303030303030303030303030303030303030222c2273746174655f726f6f74223a226230326662623534646135633162393236316131313534
34363431343939373435333664373835323031353566663931383531643264656637383537336422c226761735f75736564223a2230222c2274696
374616d70223a2231363536343833353439373532222c226c6f675f626c6f6f6d223a2230303030303030303030303030303030303030303030303030
303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030
030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303
303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030
030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303
303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030
030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303
303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030
030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030227d2c226
6479223a7b227472616e73616374696f6e5f6c697374223a22222c22726563656970745f6c697374223a22222c22636f6e73656e7375735f70726f6f
23a22227d7d0000000000000000000000000000000000000000","gas_used":55238,"logs":[{"topics":
["call_contract"],"from":"9e5f9e75192fa45becb945bc5cc52212cc0033e435af7904c186a307e211a347","log_data":"","to":"fc33afc
6b5daf1633427b3989ecdd0fe24d0a2cd1159c2a1aa2c40e553ad"}]}
```


数据解析

1. 解析上述跨子网查询的数据结果（ `output` 字段），解析结果如下。


```
{
 "header":{
 "hash":"52721dbd05549c3d43dd597a225f95a5e7cbab8c964e3d29932a8483a44d1152",
 "version":"720575961887672834",
 "number":"5",
 "parent_hash":"38f6e15f202f336a18cc5672870a8a078340b620e1e58132957e2bcd8c654594",
 "transaction_root":"0000000000000000000000000000000000000000000000000000000000000000",
 "receipt_root":"0000000000000000000000000000000000000000000000000000000000000000",
 "state_root":"b02fbb54da5c1b9261a1154df464149974536d78520155ff91851d2def78573d",
 "gas_used":"0",
 "timestamp":"1656483549752",

"log_bloom":"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000"
 },
 "body":{
 "transaction_list":"",
 "receipt_list":"",
 "consensus_proof":""
 }
}
```


2. 使用mychain_tool查询对应子网B上的区块号 5 的信息，发现与 `output` 信息一致，跨子网通讯成功。

```
[root@mychain bin]# ./mychain_tool block -m 5 -g 2
get input subnet index 2
Execute BlockNumberCommand on subnet : 0000000000000002000000000000000000000000

Execute query block by number succuss :
        hash:                           52721dbd05549c3d43dd597a225f95a5e7cbab8c964e3d29932a8483a44d1152
        number:                         5
        version:                        720575961887672834
        parent hash:                    38f6e15f202f336a18cc5672870a8a078340b620e1e58132957e2bcd8c654594
        transaction root:               0000000000000000000000000000000000000000000000000000000000000000
        receipt root:                   0000000000000000000000000000000000000000000000000000000000000000
        state root:                     b02fbb54da5c1b9261a1154df464149974536d78520155ff91851d2def78573d
        gas used:                       0
        timestamp:                      1656483549752

        transactions:                   [ ]
```

4.运维工具介绍

4.1. traverse_db

4.1.1. 工具简介

假定当前mychain安装在/data1/mychain_deploy目录下。

mychain平台在运行过程中会在本地的磁盘文件系统持久化相关的数据信息，这些数据包含区块数据、交易数据、账户数据、合约存储数据等。这些数据按照一定的协议格式组织存储在本地的rocksdb文件中。traverse_db是mychain平台提供的数据检索工具，该工具不依赖mychain区块链平台的任何接口，直接访问指定的数据库文件检索需要的数据并进行解析后输出到用户终端上。traverse_db 自动适配不同的密码学算法，不需要用户显式指定。

假设用户已经按照**《蚂蚁链私有化标准版用户指南》**搭建了mychain的环境，则可以通过执行如下命令来获取traverse_db工具的使用说明。

```
cd /data1/mychain_deploy/bin
./traverse_db -h
```

成功执行上述命令会在用户终端输出如下的帮助信息。

```
# ./traverse_db -h
traverse_db command line options:
  -h [ --help ]Print this help message and exit.
  -c [ --subnetId ] arg (=0000000000000000000000000000000000000000)
                                        blockchain subnetId that defines
   dynamic subnet[1-X], e.g.

00000000000000010000000000000000000000
0 or subnet index[1-x], default is
subnet0.
  -d [ --database ] arg (="../data")

blockchain storage directory
  -s [ --storage ] arg (=public)blockchain storage name

traverse_db subcommand:
block block subcommand. use --help to show usage
account account subcommand. use --help to show usage
storage storage subcommand. use --help to show usage
```

其中，**-d**参数用于指定待查询的mychain区块链数据库文件的存储目录，一般是mychain区块链节点安装目录下的data/public目录。traverse_db工具通过不同的子命令来完成对不同类型数据的检索，**block**子命令用于查询mychain数据库中的区块信息；**account**子命令用于查询mychain数据库中的账户信息；**storage**子命令用于查询mychain数据库中的智能合约存储信息。

可以通过执行子命令的help操作来进一步获取子命令的帮助信息。

```
cd /data1/mychain_deploy/bin
./traverse_db block -h
```

成功执行上述命令会在用户终端输出block子命令的参数和帮助信息。

```


block command options:
 -h [ --help ]Print this help message and exit
-b [ --block_number ] arg (=-1) the block number you want to look up
```

上述参数的具体信息在后续的章节进行详细介绍，也可以通过调用account和storage子命令来查看相关的使用帮助信息。

4.1.2. 工具使用

本节主要介绍如何使用traverse_db来完成对mychain区块链数据库文件的检索，读取数据库中的区块、账户、存储等相关信息。

- 查询区块
- 查询账户
- 查询torage

查询区块

traverse_db通过**block**子命令检索区块的相关信息，可以通过指定**-b**参数来查询指定的区块信息，如果不指定**-b**参数则会输出mychain数据库文件中的所有区块信息。

```
cd /data1/mychain_deploy/bin
./traverse_db -d ../data/public block -b 2
```

在上述命令中，-d参数指定了数据库文件的存储目录，-b参数指定了待查询的区块号，系统中的区块号从0开始递增。成功执行上述操作会在用户终端输出类似如下的区块信息。

```
last block number: 1144501

block 2 info :
[{"hash":"4d1ce0a081db8d6601f4c1a3053c8c909d7dae074fe179df9440d0ca71b2d320","version":"844429225099522","number":"2","pare
hash":"a27482afbbaac82eb8220d03f9223b45a3899109bee3b394bfbce7d69db983d5","transaction_root":"0000000000000000000000000000
0000000000000000000000000000000","receipt_root":"0000000000000000000000000000000000000000000000000000000000000000","state
ot":"424d24d907a056cf0b74cdb99e2ade1b79093ca873e5649ea2d2e4d9e3a8779f","gas_used":"0","timestamp":"1576033338807","log_blo
:"0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"}
]
consensus_proof info : [f8eff8c9b8414e4d1249b99fa6b88b45b7a0418d34109d1f0bbefaab30c4b8a1402bfb1]
Consensus Algorithm: PBFT
```

在输出信息中会打印mychain数据库文件中最大区块号，同时输出查询的区块信息。输出信息中相关属性的说明如下表所示。

属性名称	属性描述	属性名称	属性描述
hash	区块对应的哈希值	number	区块高度
version	系统版本	parent hash	父区块哈希值
transaction root	本区块交易的merkle root	receipt root	本区块中所有交易收据树根
state root	本区块的世界状态树根	gas used	本区块执行交易消耗的gas总和
timestamp	本区块时间戳	log_boom	交易回执的log_bloom信息

如果查询过程中不特别指定区块号，traverse_db则会遍历数据库文件中的所有区块信息并在用户终端上输出。

查询账户

traverse_db通过**account**子命令检索数据库中的账户信息，在数据库中保留了账户的历史信息，因此可以通过指定**-b**参数来查询指定区块下的历史状态信息，如果不指定**-b**参数则会检索当前数据库中最新区块下的账户信息；可以通过**-i**参数来指定待查询的账户标识，如果不指定**-i**参数则会输出数据库中的所有账户信息。可以使用如下的命令操作来查询账户信息。

```
cd /data1/mychain_deploy/bin
./traverse_db account -d ../data/public
-b 0 -i e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b
```

> **说明** 账户id可以通过crypto_tool工具的hash功能来计算账户的标识信息，具体的操作方法参考crypto_tool工具的介绍。

成功执行上述指令则会在用户终端输出如下的信息。

```
account info : [
    id: e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b,
    balance: 1000000000000000000,
 storage_root: 76be8b528d0075f7aae98d6fa57a6d3c83ae480a8469e668d7b0af968995ac71,
  code_hash: 0000000000000000000000000000000000000000000000000000000000000000,
    auth_root: fc4000fd7982b90eb8c97c92c25b3b490ca7c14c122ea03bee31b5079ac705da,
 recover_key:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000,
    recover_time: 0,
    status: 0,
    version: 2
]
```

在输出信息中会输出查询的账户信息。输出信息中相关属性的说明如下表所示。

属性名称	属性描述	属性名称	属性描述
id	账户id，是一个可读标识的哈希结果	balance	账户的当前余额
storage_root	合约存储的MPT树根，如果是普通账户则会是一个特定的固定值	code_hash	合约代码的哈希值，针对普通账户则是一个固定指定
auth_root	账户的公钥和权重列表的哈希值	recover_key	用于账户重置公钥列表的恢复公钥
recover_time	账户预重置公钥列表的时间，如果没有进行过预重置或者重置操作完成则该值为0	status	账户状态，0表示正常状态，1表示冻结状态，2表示恢复状态。
version	账户版本号，用于指示账户结构是否发生变化	无	无

查询torage

traverse_db可以查询数据库中指定合约账户下的存储信息，通过**storage**子命令检索数据库中的合约账户存储信息，在数据库中保留了合约账户的历史信息，因此可以通过指定**-b**参数来查询指定区块下的历史状态信息，如果不指定**-b**参数则会检索当前数据库中最新区块下的合约账户存储信息；需要通过**-i**参数来指定待查询的账户标识；可以通过**-k**来指定待查询的key，如果不指定该key值则会输出账户下的所有存储信息。可以使用如下的命令操作来查询合约账户的存储信息。

```
cd /data1/mychain_deploy/bin
./traverse_db storage -d ../data/public
-i 87e89abb4c1c551fe08d355d097f18b8de78edca5f556997085681662fce8eed
```

成功执行上述操作会在用户终端输出所有查询到的key-value信。

```
key info :
[ec4916dd28fc4c10d78e287ca5d9cc51ee1ae73cbfde08c6b37324cbfaac8bc5 =
0000000000000000000000000000000000000000000000000000000000000001]
key info :
[0e01bc9c1bd2710b7e43fa3daac3762439ae4dc3c23a5b2144bd2d358237eb67 =
0000000000000000000000000000000000000000000000000000000000000081]
key info :
[a059e24472da068c4a8ff2faadb73556de30c5c7272e1c95c183e56962c72672 =
0000000000000000000000000000000000000000000000000000000000000201]
key info :
[a5bc7f8efd5a71f7932bca9748017a46bbb4bb775bf85ffa718453095a123de9 =
0ae8c43005c79fde048131077b965b8d3ef473afafb8e9a5bd86df5ec3cfd288]
key info :
[66687aadf862bd776c8fc18b8e9f8e20089714856ee233b3902a591d0d5f2925 =
0000000000000000000000000000000000000000000000000000000000000000]
key info :
[2e2ec931311002ee45a63a045e787f3a83be9c75f5905dd59e6e946952700bb6 =
3132372e302e302e3100000000000000000000000000000000000000000012]
key info :
[63d20ea7419ce0fa56cf7c79ebf139349b55c2b8ff8be361eb81d23823627806 =
09c5ae416dce7aed60387e6cbad1859ee524691df3fc99cc8577da319411ae2b]
key info :
[4181606e1974270df1a4cd2617274a67d9bca783ae92ec1649620e8cd21270ad
= 0000000000000000000000000000000000000000000000000000000000004736]
key info :
[ec4916dd28fc4c10d78e287ca5d9cc51ee1ae73cbfde08c6b37324cbfaac8bc5 =
0000000000000000000000000000000000000000000000000000000000000001]
key info :
[6d3ce372f455ec953c1fcd555595b29fa687b04f121877b0d599e0f0efae0d2e =
02d3a42bd705165b979486d803abe7a8dea9a3861ab0eefedcf82de3f197c0ce]
```

在上述输出信息中，key info中前面32字节的字符串是合约账户中存储数据的key值，等号后面是对应的value值。这些值均是有效值的十六进制的表达方式。这些key的编码信息根据不同的智能合约编码规则相关，如果是solidity智能合约则这些存储key值的编码方式可以参考solidity的官方文档介绍；如果是native智能合约则这些存储key值的编码方式会跟具体的合约编码相关。

4.1.3. 支持subnet

在2.19版本后，mychain支持subnet动态组网，traverse_db也可以支持在各个Subnet中进行数据检索。

原有功能保持不变，默认查询subnet0[h160(0)]，对于其他Subnet的操作有两种方式：

- 需要指定subnet_id或者subnet_index[1-x]，如-g 00000000000000010000000000000000000000000表示指定的Subnet1，同-g 1，会转化成操作DBsubnets/0000000000000001。
- 指定storage，如-s subnets/0000000000000001，指定操作的DB。

```
./traverse_db account -g
00000000000000010000000000000000000000000 -d ../data/public -b 0 -i
e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b

./traverse_db account -g 1 -d ../data/public
-b 0 -i e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b
```

4.2. mychain_tool

4.2.1. 工具简介

mychain_tool是区块链平台工具的统一入口，它支持交互式和非交互式（命令行模式）两种模式。可以用mychain_tool来查询区块、账户和发送交易等数据。mychain_tool工具正常运行依赖于节点配置文件node_config.json。同时，假设用户已经按照**《蚂蚁链私有化标准版用户指南》**章节搭建了mychain的环境。

4.2.2. 环境配置

本节主要介绍node_config.json配置文件中的相关属性说明，用户可以根据需要酌情修改。

如下内容是配置文件的参考样例。

```
{
    "node_config" : {
        "identity" : "Tester001",
        "key_path" : "../../user",
        "user_keys" : [
            {
                "key_name" : "Tester001.key",
                "key_passwd" : "123abc"
            }
        ],
        "node" : {
            "endpoints" : [
                "127.0.0.1:10016"
            ],
            "ca_path" : "../certs/ca.crt",
            "cert_path" : "../certs/client.crt",
            "key_path" : "../certs/client.key",
            "passwd" : "123abc"
        }
    }
}
```

上述配置文件中的相关参数的介绍如下表所示。

参数名称	参数说明	配置要求	示例
identity	待发送交易的账户名	需要与创建账户时使用的账户名保持一致	`"identity" : "Tester001"`
key_path	发送交易账户的私钥文件存储目录	无	`"key_path" : "../../user"`
user_keys	发送交易账户的密钥文件列表	因为考虑到存在多重签名的账户，因此该处可以配置多个密钥文件，多重签名密钥文件的生成请参考 <<账户操作>> 文档中的介绍	多重签名需要配置多个账户的key，示例如下： `"user_keys" : [ { "key_name" : "Tester001.key", "key_passwd" : "123abc" }, { "key_name" : "Tester002.key", "key_passwd" : "123abc" } ]`
key_name	发送交易账户的密钥文件路径	密钥文件的产生可以参考<<账户操作>>文档中的说明	无
key_passwd	发送交易账户的密钥文件加密密码	创建账户密钥文件时指定的密码	无
node.endpoints	接收交易的节点接入点信息	需要指定当前机器能够访问的节点接入IP地址，可以指定多个	指定多个节点，需要配置多个IP地址，示例如下： `"endpoints" : [ "127.0.0.1:10016", "127.0.0.1:10017" ]`

node.ca_path	与节点通讯的SSL链路CA证书路径	合约平台的认证ca，客户端使用用来验证区块链平台服务	`"ca_path" : "../certs/ca.crt"`
node.cert_path	与节点通讯的SSL链路的客户端证书路径	客户端的证书文件	`"cert_path" : "../certs/client.crt"`
node.key_path	与节点通讯的SSL链路的客户端私钥路径	客户端的私钥文件	`"key_path" : "../certs/client.key"`
node.passwd	与节点通讯的SSL链路的客户端私钥文件的密码	客户端的私钥文件密码	`"passwd" : "123abc"`

4.2.3. 工具使用

4.2.3.1. 使用说明

获取工具使用说明

您可以通过以下两种方式获取工具使用说明：

- 使用帮助命令获取帮助，其命令为：

```
cd /data1/mychain_deploy/bin
./mychain_tool
> help
```

输入mychain_tool命令之后，进入到一个交互式命令行，输入help命令可以看到帮助信息如下所示。

```
Available Commands:
    account                     Account operation command
    account_queue               Account queue information
    black_list                  update black list
    block                       Query block according to block number
    block_header                Query block header according to block number or hash
    block_queue                 Block queue in cache: verified、unverified
    chain_version               chain version operations
    consensus                   Query consensus plugin information
    contract                    Contract operation command
    dev_contract                develop contract operations
    dump                        dump database
    event                       event contract、block subscribe information
    group                       group related operations
    log                         log level query、set
    metrics                     query metrics info
    node                        Get node information: p2p、contract node、host status
    sender                      transaction sender operations
    sync                        Get synchronize status
    system_contract             System contract information
    transaction                 Transaction operation command
    transaction_queue           Transaction queue in cache: verified、unverified
    update_crl                  update crl
```

其中，主要包含account、account_queue、black_list、block、block_header、block_queue、chain_version、consensus、contract、dev_contract、dump、 event、group、log、metrics、node、sender、sync、system_contract、transaction、transaction_queue、update_crl这些子模块。

子命令的主要作用如表1所示。

子命令	作用
account	用于查询账户信息
account_queue	用于查询当前账号队列的信息，暂未启用

black_list	用于配置账户黑名单
block	用于查询区块信息
block_header	用于查询区块头信息
block_queue	用于查询区块队列的信息
chain_version	用于Mychain的渐进式升级
consensus	用于查询共识模块相关的信息
contract	用于查询合约账户信息
dev_contract	用于合约开发相关的操作，包括如下操作：初始化创建、编译和部署
dump	用于dump平台的数据库
event	用于订阅区块、合约、账户和Topic的事件
group	用于增加group分组
log	用于查询和设置log
metrics	用于查询统计交易等信息
node	用于查询当前系统节点的信息，增加、删除节点
sender	用于交易发送
sync	用于查询同步模块的信息
system_contract	用于系统合约的查询
transaction	用于查询交易的信息
transaction_queue	用于查询交易队列的信息
update_crl	用于更新客户端吊销证书列表

对于**account**、**block**、**block_header**、**transction**等子命令，进一步帮助信息可以使用如下命令。

```
./mychain_tool
>[subcommand] -h
```

其中[subcommand]是指具体某种子命令。

- 使用如下的非交互模式获取mychain_tool的使用帮助。

```
./mychain_tool -h
```

通过如下命令，获取子命令的使用帮助。

```
./mychain_tool [subcommand] -h
```

命令帮助显示内容同上表所示。

进入交互命令行

首先用户需要使用如下命令进入交互命令行。

```
./mychain_tool
```

执行上述命令输出如下结果。

```
********************************************************************
Instrunction:
    help                        list sub command
    quit                        normal end
********************************************************************
>
```

4.2.3.2. account子命令

mychain_tool的**account**子命令主要用来查询存在于区块链平台中的账户信息，其支持两种不同模式的查询：一是指定账户实体可读标识，通过**-a**参数标识，二是指定账户在区块链中存储的唯一标识，通过**-i**参数标识，如下所示。

```
> account -h
usage: account {-i, -a} {hash, name}
options:
   -h [ --help ]                    produce help message
   -i [ --identity ]                account hash identity
   -a [ --account ]                 account prior to identity
```

其执行命令如下。

```
> account -a Administrator
> account -i e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b
```

其中 `e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b` 由Administrator进行Sha256运算得出，具体可参见crypto_tool工具使用介绍。

执行上述命令输出如下结果。

```
Execute query accout succuss :
    id:                             e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b
    status:                         0
    balance:                        1000000000000000000
    recover key:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000
    recover time:                   0
    encryption key:
    version:                        2
    auth_map:                       {

ca51a93af8d63f3e06ea6753aedad678d3edcd9109ad9bde69eb01fbd2056341885ef45171258fd1a904140f1e2fb04e47529f9c5ac26db7f719315e0
62f : 40

d55794383aefca32470a072e06e5a120e794bee6ca207cf9927d24d42dee145da146c0f5b88b4dba38e6d437b33402408829f24ecb8c792ee8dee709f
970 : 40
                         }
```

可以看出，Administrator账户是一个多签账户，需满足3/4权限控制。结果描述信息如下所示。

属性名称	属性描述
id	账号/合约全局唯一标识
status	正常、冻结、恢复中的三个状态
balance	账号的余额
recover key	恢复公钥，用于重置AuthMap
recover time	最后一次重置的时间
encryption key	加密交易金额的公钥
version	账号数据结构的版本
auth_map	账号的公钥集合

4.2.3.3. black_list子命令

mychain_tool的**black_list**子命令主要用来配置平台账号黑名单，平台将拒绝处理涉及黑名单列表中账号的所有交易，其执行命令如下。

```
>black_list -p ./path/black_list.json

黑名单文件示例：./path/black_list.json
{"black_list" : ["e8d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6a"]}
```

执行上述命令输出如下查询结果。

```
Sending update blackList to mychain completes!
```

4.2.3.4. block子命令

mychain_tool的**block**子命令主要用来查询存在区块链平台中的区块信息，其支持两种不同模式的查询：一是通过区块号来查询，通过**-m**参数来

标识；二是通过区块哈希值来查询，通过-i参数来标识。三是查询最新区块头，如下所示。

```
> block -h

usage: block {-i, -m} {hash, number}

options:
   -h [ --help ]                    produce help message
   -m [ --number ]                  query block by number
   -i [ --identity ]                quer block by hash, if hash = "last", query last block
```

其具体执行命令为：

```
> block -i last
> block -m 0
> block -i d3af3c6ca22aacdaf60d5cdbbe8cae10dca4e62995c4551b1d06b299615ef0c6
```

执行上述命令输出如下查询结果。

```
Execute query  block succuss:

    hash:                           d3af3c6ca22aacdaf60d5cdbbe8cae10dca4e62995c4551b1d06b299615ef0c6
    number:                         0
    version:                        1
    parent hash:                    0000000000000000000000000000000000000000000000000000000000000000
    transaction root:               0000000000000000000000000000000000000000000000000000000000000000
    receipt root:                   0000000000000000000000000000000000000000000000000000000000000000
    state root:                     9470abb4bd3be8e1666e931a58b88ef06f53f74ff8f74807996aaf0b81c8cb57
    gas used:                       0
    timestamp:                      0
    transactions:                    [ ]
```

结果描述信息如下所示。

属性名称	属性描述
hash	区块头的哈希
number	区块号
version	区块数据结构版本
parent hash	上一区块哈希
transaction root	区块体中的交易构成的默克尔哈希根
receipt root	区块体中的收据构成的默克尔哈希根
state root	世界状态的默克尔哈希根
gas used	交易执行的总消耗量
timestamp	时间戳
transactions	区块中的交易集合

4.2.3.5. block_header子命令

mychain_tool的**block_header**子命令主要用来查询存储在区块链平台中的区块头信息，当不关注区块存储的交易以及交易收据具体信息时，可以使用**block_header**子命令，其支持两种不同模式的查询：一是通过区块号来查询，通过**-m**参数来标识；二是通过区块哈希值来查询，通过**-i**参数来标识。三是查询最新区块头，如下所示。

```
> block_header -h

usage: block_header {-i, -t, -m} {identity, timestamp, number}

options:
   -h [ --help ]                    produce help message
   -m [ --number ]                  query block header by number
   -t [ --timestamp ]               query block header by timestamp
   -i [ --identity ]                query block header by hash identity, if hash = "last", query last block header
```

具体执行命令为：

```
> block_header -i last
> block_header -m 48
> block_header -i eb24bd8caacbb6258726c848ab4fc8b5b454692d30d0fbeeb73611391f19e7be
```

执行上述命令输出如下结果。

```
Execute query block header by number succuss :
    hash:                           eb24bd8caacbb6258726c848ab4fc8b5b454692d30d0fbeeb73611391f19e7be
    number:                         48
    version:                        844429225099522
    parent hash:                    94071010c16cad493ae0bfafad5c126a3fe231dc3a58d1f89f4e6d2218df70f9
    transaction root:               0000000000000000000000000000000000000000000000000000000000000000
    receipt root:                   0000000000000000000000000000000000000000000000000000000000000000
    state root:                     c88fe97db9707c00008bb15939981f1d0276b4e2b4197844f4e3ecf7f20285ea
    gas used:                       0
    timestamp:                      1565854463740
```

4.2.3.6. block_queue子命令

mychain_tool的**block_queue**子命令主要用于查询mychain节点区块缓存队列的信息，执行命令如下。

```
> block_queue
```

执行上述命令后，输出结果为：

```
Execute block cache status query succuss :
    unverified queue limit:         128
    unverified queue usage:         0
    unverified max block NO.:       0
    verified queue limit:           256
    verified queue usage:           0
    verified max block NO.:         0
    verified p-n-f limit:           256
    verified p-n-f usage:           0
    verified p-n-f max block NO.:   0
```

其结果对应描述信息为：

属性名称	属性描述
unverified queue limit	未验证队列最大容量（以区块为单位）
unverified queue usage	未验证队列已使用大小
unverified max block NO.	未验证队列中保存的最大区块号
verified queue limit	验证队列最大容量（以区块为单位）
verified queue usage	验证队列已使用大小
verified max block NO.	验证队列中保存的最大区块号
verified parent not found(p-n-f) limit.	验证但父区块不存在的队列缓存区块最大容量（以区块为单位）
verified parent not found(p-n-f) usage	验证但父区块不存在的队列已使用大小
verified parent not found(p-n-f) max block NO	验证但父区块不存在的队列中保存的最大区块号

4.2.3.7. chain_version子命令

为方便进行渐进式升级，mychain_tool新增chain_version子命令。

```
> chain_version -h
usage: update_chain_version [-v, -c]

options:
   -h [ --help ]                  print help message
   -v [ --version ]               the version you want to update
   -i [ --admin_id ]              the admin identity, default=sha256(Administrator)
   --update                       do update chain version operation
   --parse                        do parse version operation
```

- **执行解析版本命令**

```
>chain_version --parse -v 0.10.2.12
```

执行上述命令后，输出结果为：

```
the block version is 1970329131942146
the vm version is 7
the tee version is 1
the consensus version is 0
the storage version is 0
the pexe version is 1
the pbds(block/tx/account) version is 2
```

- **执行升级版本命令**

```
>chain_version --update -v 0.10.2.12
```

执行上述命令后，输出结果为：

```
call contract transaction:
[{"hash":"c856c50b33047959afceef03cf36aecb2379b08330c478298c64827adea9945c","type":"11","timestamp":"1576673988691","no
:"15012543175202714052","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"87
bb4c1c551fe08d355d097f18b8de78edca5f556997085681662fce8eed","value":"0","gas":"10000000","group_id":"000000000000000000
00000000000000000","version":"2","data":"55bc5c7c000000000000000000000000000000000000000000000000000000000000000600000000
0000000000000000000000000000000000000000000000000a000000000000000000000000000000000000000000000000000000000000000100000000
0000000000000000000000000000000000000000000000000013636861696e2e626c6f636b5f76657273696f6e0000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000010313430373337393137383532303833340000000000000000000000000000
000000","extensions":"","signature":
["987391675f7fee14d39f8fb68384d75d3763c7df90a39f6d92cc1ceb29e8972d2c7aed97d6a97541cf8ff6d715049ed05edc8a940ed7152b98"]}

]
Call contract transaction receipt: [{"result":"0","gas_used":"72910","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"87e89abb4c1c551fe08d355d097f18b8de78ec
f556997085681662fce8eed","topics":["call_contract"],"log_data":""},
{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"87e89abb4c1c551fe08d355d097f18b8de78edc
556997085681662fce8eed","topics":
["4897a472ac8897deca130ff7c0018fee47e2df535ef8905308f45a16b5d6df07"],"log_data":"000000000000000000000000000000000000000000
0000000000000000004000000000000000000000000000000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000000
0000000000000000000000013636861696e2e626c6f636b5f76657273696f6e0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000010313430373337393137383532303833340000000000000000000000000000000000"}],"output":"0000000000
0000000000000000000000000000000000000000000000000000000000"}
]
```

> **说明** mychain_tool的**chain_version**子命令根据传过来的参数以及node_config.json，构造一笔同步调用系统合约**Config**方法**SetConfig**的交易，返回交易**Receipt**给调用方，如果有问题，将错误抛出给调用方。并且node_config.json中需要配置Administrator的密钥，通常是3把，请确保Administrator的key文件存在。

```
{
    "node_config" : {
        "identity" : "Administrator",
        "key_path" : "../../user",
        "user_keys" : [
            {
                "key_name" : "Administrator1.key",
                "key_passwd" : "123abc"
            },
            {
                "key_name" : "Administrator2.key",
                "key_passwd" : "123abc"
            },
            {
                "key_name" : "Administrator3.key",
                "key_passwd" : "123abc"
            }
        ],
        "node" : {
            "endpoints" : [
                "127.0.0.1:10016"
            ],
            "ca_path" : "../certs/ca.crt",
            "cert_path" : "../certs/client.crt",
            "key_path" : "../certs/client.key",
            "passwd" : "123abc"
        }
    }
}
```

4.2.3.8. consensus子命令

mychain_tool的**consensus**子命令主要用于查询mychain节点共识相关信息，执行命令如下。

```
> consensus
```

执行上述命令后，输出结果为：

```
Execute query consensus succuss:
    type:                           PBFT
    phase:                          NORMAL
    N:                              4
    f:                              1
    Node Id:                        d912c8b9ace64ccc0afe37bce418361299d334cb223c8860a74969d23a6694b7
    Leader:                         898c7f81ae52fee6c029cb203a6f13dd411299c012f5881e445039df3481e6d3
    View:                           3889
    Last exec seq:                  83848
    Propose size:                   2000
    Propose time interval:          7000
    Normal time interval:           5000
    Viewchange time interval:       2000
```

结果描述信息如下所示：

属性名称	属性描述
type	共识算法名称（pbft，honeybadger）
phase	共识阶段（starting，normal，viewchange）
N	参与共识的节点数
f	可容错的节点数
Node Id	本节点ID
Leader	当前共识中主节点ID
View	当前共识的视图编号
Last exec seq	最后执行的序列号（等同于区块号）
Propose size	提议中最大交易数量

Propose time interval	上一轮共识结束到提议开始时间间隔
Normal time interval	主节点发起提议到共识结束时间间隔
Viewchange time interval	当前视图切换超时时间间隔，如果是 honeybadger算法，无此属性。

4.2.3.9. contract子命令

contract子命令主要用来查询存在于区块链平台中的合约账户信息，使用方式与 account子命令相同。

4.2.3.10. dump子命令

dump子命令用于dump平台的db数据库，以便快速排查问题。请**dump**命令中**type**的选项，当选择为**all**时执行时间较长，并且需要注意磁盘空间是否足够，执行以下命令dump 数据库。

```
> dump -h
usage: dump -t type -p path
options:
       -h [ --help ]                    produce help message
       -t [ --type ]                    select [all or spv] dump type: all, spv
       -p [ --path ]                    set dump path
```

参数**-t**用于指定dump的类型，**all**代表对平台中的交易做 `snapshot` ，**spv**是指对spv功能所做的 `snapshot` ， **0**表示dump所有。 参数**-p**用于指定dump下来的文件要保存的服务端路径。

执行命令示例如下：

```
>dump -t spv -p ./
```

执行上述命令输出如下结果：

```
Sending dump db to mychain completes! Please check dump path.
```

4.2.3.11. event子命令

event子命令主要用于事件订阅的相关操作。主要包括：区块生成事件、合约事件、账户事件和Topic订阅，如下所示。

```
> event -h
usage: event -s {contract, block, topic, account, vm_type} [-i, -a, -t, -v] label

options:
   -h [ --help ]                    produce help message
   -s [ --subscribe ]               subscribe contract, block, topic, account event
   -i [ --identity ]                contract, account identity
   -a [ --account ]                 contract, account name
   -t [ --topic ]                    topic
   -v [ --vm_type ]                  vm_type, wasm/evm
```

其中**-s**是指事件订阅；**-i**是指根据账户id订阅账户事件，还需要输入账户id，则系统中发生所有与该账户相关的事件都会返回给用户；**-a**是指根据账户名称订阅账户事件，还需要输入账户名称，功能与**-i**命令基本一致，**-t**是指订阅的事件主题，**-v**是指虚拟机类型，具体使用方式如下所示。

例如订阅区块，可以使用如下命令：

```
>event -s block
```

执行上述命令输出如下查询结果：

```
Subscribe-block
The : 83 th subscribe-block, subscribe-block size : 1
   ------------------------------------------------------
   hash:                            41b58cf29237ff6e9f579000ebed125b7c8c19e65f111c968a8e3428ea8c2769
   number:                          1171029
   version:                         1970329131942146
   parent hash:                     97d5b4a42acf4baec138c543b05b90b0ce59875aabed3ad4fde7680c21d8bd55
   transaction root:                b7fdd863b6261fd6ef9adf6369f76ef7cc11c2ba6c296debff74968532c64fc0
   receipt root:                    05326b11cd33f75eef0cd01f743ee1f8fb5babae118b5915a97e5625b2c83f1f
   state root:                      2e7eb75e7b88d2fb8613e76b858850c8eed4a05299695dc985bc2c4c4cbde010
   gas used:                        2786723944
   timestamp:                       1576674818528
```

执行以下命令来订阅合约事件：

```
>event -s contract -i
a7937b64b8caa58f03721bb6bacf5c78cb235febe0e70b1b84cd99541461a08e
```

可以依据参数名称、合约id进行合约事件的订阅，通过-a指定合约名称订阅合约事件，通过-i指定合约的标识订阅合约事件。

执行上述命令输出如下查询结果：

```
Subscribe-contract: a7937b64b8caa58f03721bb6bacf5c78cb235febe0e70b1b84cd99541461a08e
The : 1 th subscribe-contract, subscribe-contract size : 1
   ----------------------------------------------------
   id:                                a7937b64b8caa58f03721bb6bacf5c78cb235febe0e70b1b84cd99541461a08e
   status:                            0
   balance:                           0
   recover key:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000
   recover time:                      0
   encryption key:
   version:                           2
   storage root:                      e82fb0983fb22f7f4eff53a170d247549165640d0672288efa1f3bf469ba685c
   code hash:                         d60e894a07dbdf8cc9f8b2ecb8c64525c0845ce80a8ff0d77c1dd75558448b62
   code:
01608060405260043610610078576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff1680630479717
1007d578063427830200160002060008282540192505081905550610 3ef565b600160008082825401925050819055506001806000338152602001908l
200160002081905550 5b5600a165627a7a72305820662234 5dde63ac1d16aef65918d5e1d062108bf90f116fae1ecdad1bf88ca1bc0029
   auth_map:                          {

0df77ed252f69c40dd30de80d303c2456f49b454178f42c6672992deca62887a91226f3e094c46f4621ce33cd9a1f7db59ada9b411c3876011ffee539
e81 : 40

889ce6ebd670786644eb14e1834baa517b6d3f72d60d5d8809b004b93df17dec0f57fdfa3bb01c0f540376b92c666202b63e2576a133e5073e238d897
264 : 40

a7336828867cab5bd9ab14c63cc671a9179a8863c8cdebb14bcad87b1026907c83dc5790b40784fadca5401b5e39ae2519571c524d4b3c7561f7f87e5
0fb : 40

b4ac832f4ae5b53f0c29cf90b6e8a83a9b0c252f98f14e376484e7d774e987a540a09ac22d3bd936a5c138b6878573c247e3c9172190a75379231f63e
1c9 : 40
                                      }
```

执行以下命令来订阅账户变动事件：

```
>event -s account [-a contract_name] [-i contract_id]
```

执行上述命令输出如下查询结果：

```
Subscribe-account: c60a9d48105950a0cca07a4c6320b98c303ad42d694a634529e8e1a0a16fcdb5
The : 1 th subscribe-account, subscribe-account size : 1
   ----------------------------------------------------
   id:                              c60a9d48105950a0cca07a4c6320b98c303ad42d694a634529e8e1a0a16fcdb5
   status:                          0
   balance:                         999999999997000000
   recover key:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000
   id:                                  recover time:                     0
   encryption key:
c60a9d48105950a0cca07a4c6320b98c303ad42d694a634529e8e1a0a16fcdb5
   status:                          0
   balance:                         999999999997000000
   recover key:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000
   recover time:                    0
   version:                             encryption key:
   version:                         2
2
   auth_map:                        {

74c9553fbea2752c35fca5b18da5eb9cf649a2e876c5e793815bcfb86bec585e00a66ef5f0ba79a4ec47dfcd49e986f78c2e449ee64f2b30ea34a3647
195 :  auth_map:                        100{
                                    }


74c9553fbea2752c35fca5b18da5eb9cf649a2e876c5e793815bcfb86bec585e00a66ef5f0ba79a4ec47dfcd49e986f78c2e449ee64f2b30ea34a3647
195 : 100
                                    }
```

执行以下命令来订阅指定的Topic：

```
>event -s topic -t deploy_contract -v evm
```

-t参数指定了要订阅的topic列表。

执行上述命令输出如下查询结果：

```
event -s topic -t deploy_contract -v evm
Subscribe-topics: deploy_contract
The : 1 th subscribe-topics, subscribe-topics size : 1
   ----------------------------------------------------
   topic:                           deploy_contract
   data:
```

4.2.3.12. log子命令

log子命令主要用于日志的查询和修改运行节点的日志级别。比如在排查线上问题时，修改日志级别，如下所示。

```
> log -h
usage: 1-query: log     2-set: log -l <0-5>
options:
   -h [ --help ]                     produce help message
   -l [ --level ]                    select [0-5] log level: trace-0, debug-1, info-2, warning-3, error-4, fatal-5
```

执行以下命令进行日志的查询：

```
log
```

执行命令之后，输出的内容如下：

```
Execute query log level succuss:
    log level:                       info
```

执行以下命令进行日志的设置：

```
log -l <{0|1|2|3|4|5}>
```

设置成功之后会显示如下信息：

```
Execute set log level succuss!
```

4.2.3.13. metrics子命令

metrics子命令主要用于在某一个时间段内交易和查询的统计，用户可以使用如下命令来查询其使用方式。

```
> metrics -h

usage: metrics -t type -i node_identity [-b begin_timestamp -e end_timestamp]

options:
   -h [ --help ]                  produce help message
   -t [ --type ]                  transaction or query or tps_latency.
   -i [ --id ]                    node identity.
   -b [ --begin ]                 (optional) metrics begin timestamp.
   -e [ --end ]                   (optional) metrics end timestamp.
```

- 参数**-t**用来指定统计查询的类型，包括: transaction、query、tps_latency
- 参数**-i**用来指定统计查询的节点的identity
- 参数**-b**、参数**-e**两者成对出现。**-b**指定开始时间，**-e**用来指定结束时间。时间格式为 `%Y-%m-%d-%H:%M:%S` 。

比如查询tps和延迟，则执行命令示例如下：

```
>metrics -t tps_latency -i ed59c7792c33f3ddbf7aa53243e4b5f5d168644c6d5f7253fb4c9c27a81286da
```

执行上述命令输出如下结果：

```
Execute query tps and latency metrics succuss :
    granularity:                   1800000
    save duration:                 43200000
    start time:                    1576672200000


    type:                          METRIC_TPS
    counter:                       347

    type:                          METRIC_TX_LATENCY
    counter:                       500
```

4.2.3.14. node子命令

node子命令用于查询系统合约中节点信息和动态增删及更新节点的命令，其中动态增删及更新节点操作需要在node_config.json文件中配置Administrator用户。帮助信息如下：

```
> node -h

usage: node [-t type [-i node_id]]
   node [-u <add|update> -p <publickey> -e <endpoints> -nt <nodetype> -s <nodestate> [-o <domain name>]]
   node [-i <nodeid>]

options:
   -h [ --help ]                  produce help message
   -t [ --type ]                  node info type: contract, p2p, host, node_timestamp
   -i [ --id ]                    node id
   -u [ --change_node ]           change node operations: add, update, please use an admin user
   -p [ --publickey ]             node public key
   -nt [ --node_type ]             node type: consensus, nonconsensus
   -s [ --node_state ]            node state: unactived, normal
   -e [ --endpoints ]             endpoints: ip:port|ip:port|..., use | split
   -o [ --domain_name ]           domain name: NULL or real domain name
   -d [ --delete_node ]           delete node by node id, please use an admin user
```

- 功能一：节点信息查询
  mychain_tool的node子命令支持两种类型查询：一是查询mychain节点系统合约中的信息，通过-t contract 来标识；二是查询节点p2p运行状态，通过-t p2p 来标识。执行命令如下：

  ```
  > node -t contract
  ```

  执行上述命令输出如下结果：

```
Execute query contract node succuss:
    node id:                        fdadce54c4c0780525e1834c264a9495ebcc62e3b860cd51843447b18edc09a7
  endPoints:                      [ 10.1.6.146 : 18230 ]
    public key:
09927d4e7c3eff07dc0705d8dd8c5e05db6d8007c32bbf02a6e24bdcf1df19a79d900d2d7f48d2491991d0d23b1f85dbcd4e97851214dfa23ac4f58
6ff6a
    node role:                      node role consensus
    node state:                     node state normal

    node id:                        d912c8b9ace64ccc0afe37bce418361299d334cb223c8860a74969d23a6694b7
    endPoints:                      [ 10.1.6.147 : 18230 ]
    public key:
d58794dd2ff2c060392de0c13f17107a1da71ae87bc63745d979c541b47c07911c27e9230af1d69783aac64891a668dbeec2dc5d50697fbb850efd3
4e0e6
    node role:                      node role consensus
    node state:                     node state normal

    node id:                        898c7f81ae52fee6c029cb203a6f13dd411299c012f5881e445039df3481e6d3
    endPoints:                      [ 10.1.6.148 : 18230 ]
    public key:
3e34bab059e54c22d1fd708398d3469fd757315f43ad4e00c12634725dad0225c68a9f3502553a46ebb75cca67bb906f47da68a8060b412b779f201
c4d68
    node role:                      node role consensus
    node state:                     node state normal

    node id:                        79dc0026a226a2bb675278d852a43f043c4e5681fef803be652053032c76fe58
    endPoints:                      [ 10.1.6.149 : 18230 ]
    public key:
73942c170b709a5afc432261daa87f078c664b813b3238a0c09d94a47a7a815306396f18b6e8808a4724a9da2ed9963b8ba6a46e846ec7c029fc993
48954
    node role:                      node role consensus
    node state:                     node state normal
```

结果描述信息如下所示：

属性名称	属性描述
node id	节点系统合约配置中节点标识ID
endPoints	节点监听p2p的IP地址和端口或域名信息
public key	节点的公钥信息
node role	节点类型（consensus代表共识节点，non consensus代表非共识节点，unknown代表未知）
node status	节点状态（normal代表正常，unactived代表未激活，deleted代表已删除，unknown代表未知）

查询节点的P2P运行状态，执行如下命令：

```
> node -t p2p
```

执行上述命令输出如下结果：

```
Execute node query succuss:
    node count:                     3
    consensus node:                 3
    non consensus node:             0
    connected consensus node:       3
    connected non consensus node:   0

    node status:            [
                              connected: true; node id:
898c7f81ae52fee6c029cb203a6f13dd411299c012f5881e445039df3481e6d3; (10.1.6.148, 18230, 2) ,
                              connected: true; node id:
d912c8b9ace64ccc0afe37bce418361299d334cb223c8860a74969d23a6694b7; (10.1.6.147, 18230, 2) ,
                              connected: true; node id:
fdadce54c4c0780525e1834c264a9495ebcc62e3b860cd51843447b18edc09a7; (10.1.6.146, 18230, 2)
                            ]
```

结果信息描述如下。

属性名称	属性描述
node count	当前节点总数（不包括本身）

consensus node	系统中存在的共识节点的数量（不包括本身）
non consensus node	系统中存在的非共识节点的数量（不包括本身）
connected consensus node	当前节点连接的共识节点数量（不包括本身）
connected non consensus node	当前节点连接的非共识节点数量（不包括本身）
node status	当前节点连接的节点状态信息

- **功能二：动态增加或更新节点**

  执行命令如下：

```
node [-u <add|update> -p <publickey> -e <endpoints> -n <nodetype> -s <nodestate> [-o <domain name>]]
```

  参数描述如下表所示。

参数	描述
-u	指定操作类型，可选项为add和update
-p	新增或更新的节点的公钥
-e	节点使用的ip和port集合，参考格式：ip:port|ip:port|...，多个ip需要使用|隔开
-n	节点类型，可选项为consensus和nonconsensus
-s	节点状态，可选项为unactived和normal
-o	节点域名，选项为可选项，不指定该选项则兼容以前的版本

- **功能三：删除节点**
  执行命令如下：

```
>node -d 427584248eebf37ce46c42bd344645c7478650759f3e0a890fb31c2dd869e915
```

  参数描述：

  -d：指定要删除的节点id。

  命令输出内容如下：

```
delete node succuss
tx hash: f090623c318aef1d48d444cb34534fae0a418e5672407d61860c2dfdf387c37a
recepit result: {"result":"0","gas_used":"87692","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"e93372533f323b2f12783aa3a586135cf4214
9c2cdcde47411b78f9839ec","topics":["call_contract"],"log_data":""},
{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"e93372533f323b2f12783aa3a586135cf42148
c2cdcde47411b78f9839ec","topics":
["3f9cd4381b3cc4b69a3c0503f011996a6e19b7974dc79c26a9b1df8e649f70fd","427584248eebf37ce46c42bd344645c7478650759f3e0a890f
2dd869e915"],"log_data":"0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000
0000000000000000000000000000003000000000000000000000000000000000000000000000000000000000000040c3649d5ec39a52027d410b7c2
aafe669a31f9e4d25f80907beb27ec37b60c216e594928eaa2aee9078de23688fbb17c7a8dce3f9336e5f0ab764e2041812"}],"output":"000000
0000000000000000000000000000000000000000000000000000"}
```

4.2.3.15. sender子命令

sender是mychain平台提供的交易发送工具，目前最新版本的sender集成在mychain_tool当中，用来模拟客户端向mychain区块链网络的某一节点发送交易请求，并读取解析mychain区块链网络节点返回的交易执行结果，显式打印在用户的执行终端上。sender通过读取配置文件来获取访问的区块链网络环境信息和本地的密钥配置信息，同时通过读取配置文件来设置sender的交易发送类型和相关交易参数，通过这些参数构造指定类型的交易并进行编码后通过网络发送给区块链节点。

sender工具的使用相对简单，主要包含如下三个步骤。

1. 修改sender依赖的配置文件node_config.json，确保交易发送的环境信息和账户密钥信息。
2. 修改sender依赖的配置文件transaction.json，指定需要发送交易的类型和相关的交易属性。
3. 执行sender命令发送交易并检查交易的执行结果。

假设用户已经按照《蚂蚁链私有化标准版用户指南》搭建了mychain的环境，则可以通过执行的如下命令来获取sender工具的使用说明。

```
cd /data1/mychain_deploy/bin
./mychain_tool sender -h
```

成功执行上述命令会在用户终端输出如下的帮助信息。

```
usage: sender [-c config] [-t transaction]

options:
       -h [ --help ]                    produce help message
       -t [ --transaction ]            give transaction config file path.
       -c [ --config ]                 give node config file path.
       --rawtx                         send raw transaction
```

其中，-t参数用于指定待发送给区块链网络节点的交易属性配置文件，-c参数用于指定待访问节点的环境信息以及交易中用户的公私钥配置信息。如果不指定上述两个参数的值，则sender会使用默认的配置文件路径为与bin目录同级的conf目录下的transaction.json文件和node_config.json文件，如下内容是配置文件transaction.json的参考样例。

```
{
    "transactions" : {
        "CreateAccount" : {
            "enable" : true,
            "local" : false,
            "from" : "Tester001",
            "to" : "Tester002",
            "gas" : 10000000,
            "value" : 0,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "public_keys" : [
                {
                        "public_key" :
"0x52ea38936829a7d1717ba7d8202df03a95abc76f40ad0672e643dcfd54ea14bae05fca0377ba2054fec8535fd1bdcc6d5516e011ff4d40076c8d280
7f56b5",
                        "weight" : 100
                }
             ],
                "recover_key" :
"0x52ea38936829a7d1717ba7d8202df03a95abc76f40ad0672e643dcfd54ea14bae05fca0377ba2054fec8535fd1bdcc6d5516e011ff4d40076c8d280
7f56b5",
            "block_num" : 0
        },
        "TransferBalance" : {
            "enable" : true,
            "local" : false,
            "from" : "Tester001",
            "to" : "Tester002",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "value" : 1000000,
            "block_num" : 0
        },
        "SetRecoverkey" : {
            "enable" : false,
            "local" : false,
            "from" : "Tester002",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "recover_key" :
"0x825dbfd4d3690c7e691d16e46e4344e67d7efe16e9151d37349a46cea6fb24e9e66b43d359c3ce2c9d0c952017d3ee71c0553a4dc2268920bd24e5c
29fe24",
            "block_num" : 0
        },
        "PresetPubkey" : {
            "enable" : false,
            "local" : false,
            "from" : "Tester002",
            "to" : "Tester002",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "block_num" : 0
        },
```

```
        "ResetPubkey" : {
            "enable" : false,
            "local" : false,
            "from" : "Tester002",
            "to" : "Tester002",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "public_keys" : [
                {
                    "public_key" :
"0x825dbfd4d3690c7e691d16e46e4344e67d7efe16e9151d37349a46cea6fb24e9e66b43d359c3ce2c9d0c952017d3ee71c0553a4dc2268920bd24e5c
29fe24",
                    "weight" : 100
                }
            ],
            "block_num" : 0
        },
        "UpdateAuthmap" : {
            "enable" : false,
            "local" : false,
            "from" : "Tester002",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "public_keys" : [
                {
                    "public_key" :
"0x825dbfd4d3690c7e691d16e46e4344e67d7efe16e9151d37349a46cea6fb24e9e66b43d359c3ce2c9d0c952017d3ee71c0553a4dc2268920bd24e5c
29fe24",
                    "weight" : 100
                }
            ],
            "block_num" : 0
        },
        "DeployContract" : {
            "enable" : false,
            "local" : false,
            "from" : "Tester001",
            "to" : "first",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "value" : 0,
            "code" : "../conf/compiled_test_contracts/first.evm",
            "contract_type" : "evm",
            "method" : "",
            "parameters" : [
            ],
            "block_num" : 0
        },
        "CallContract" : {
            "enable" : false,
            "local" : false,
            "from" : "Tester001",
            "to" : "first",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "value" : 0,
            "contract_type" : "evm",
            "method" : "set(uint256)",
            "parameters" : [
                100
            ],
            "block_num" : 0
        },
        "UpdateContract" : {
            "enable" : false,
```

```
            "local" : false,
            "from" : "first",
            "gas" : 100000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "code" : "../conf/compiled_test_contracts/first_new.evm",
            "contract_type" : "evm",
            "block_num" : 0
        },
        "freezeAccount" : {
            "enable" : false,
            "local" : false,
            "from" : "Tester001",
            "to" : "Tester002",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "block_num" : 0
        },
        "UnfreezeAccount" : {
            "enable" : false,
            "local" : false,
            "from" : "Tester001",
            "to" : "Tester002",
            "gas" : 1000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "block_num" : 0
        },
        "EncryptionEnvelope" : {
            "enable" : false,
            "local" : false,
            "from" : "0",
            "to" : "0",
            "group_id" : "32f0874e9dbddf4a8b89232890912f23630617f6",
            "gas" : 100000000,
            "nonce" : 0,
            "timestamp" : 0,
            "period" :0,
            "data" : [
                {
                    "type" : "DeployContract",
                    "from" : "Tester001",
                    "to" : "private_contract",
                    "group_id" : "32f0874e9dbddf4a8b89232890912f23630617f6",
                    "gas" : 100000,
                    "contract_type" : "evm",
                    "code_path" : "../hello.bin",
                    "method" : "(uint256)",
                    "parameters" : [
                        "100"
                    ],
                    "key_files" : [
                        {
                            "path" : "../Tester001.key",
                            "user_passwd" : "123456"
                        }
                    ]
                },
                {
                    "type" : "CallContract",
                    "from" : "Tester001",
                    "to" : "private_contract",
                    "group_id" : "32f0874e9dbddf4a8b89232890912f23630617f6",
                    "gas" : 10000000,
                    "contract_type" : "evm",
                    "method" : "set(uint256)",
                    "parameters" : [
                        "1000"
                    ],
```

```
                "key_files" : [
                    {
                        "path" : "../Tester001.key",
                        "user_passwd" : "123456"
                    }
                ]
            },
            {
                "type" : "UpdateContract",
                "from" : "private_contract",
                "to" : "private_contract",
                "group_id" : "32f0874e9dbddf4a8b89232890912f23630617f6",
                "gas" : 100000000,
                "contract_type" : "evm",
                "code_path" : "../hello.bin",
                "key_files" : [
                    {
                        "path" : "../Tester001.key",
                        "user_passwd" : "123456"
                    }
                ]
            }
        ],
        "block_num" : 0
    },
    "DepositData" : {
        "enable" : false,
        "local" : false,
        "from" : "Tester001",
        "to" : "Tester002",
        "gas" : 1000000,
        "nonce" : 0,
        "timestamp" : 0,
        "period" :0,
        "value" : 0,
        "data" : "user-defined-deposit-data",
        "block_num" : 0
    },
    "RelatedDepositData" : {
        "enable" : false,
        "local" : false,
        "from" : "Tester001",
        "to" : "Tester001",
        "gas" : 1000000,
        "nonce" : 0,
        "timestamp" : 0,
        "period" :0,
        "value" : 0,
        "deposit_flag" : 0,
        "data" : "user-defined-deposit-data",
        "block_num" : 0
    },
    "IssueAssets" : {
        "enable" : false,
        "local" : false,
        "from" : "Administrator",
        "to" : "Tester001",
        "gas" : 1000000,
        "nonce" : 0,
        "timestamp" : 0,
        "period" : 0,
        "value" : 1000000,
        "block_num" : 0
    },
    "RedeemAssets" : {
        "enable" : false,
        "local" : false,
        "from" : "Administrator",
        "to" : "Tester001",
        "gas" : 1000000,
        "nonce" : 0,
        "timestamp" : 0,
        "period" : 0,
        "value" : 1000000,
```

```
            "block_num" : 0
        }
    }
}
```

上述配置文件时mychain平台提供的参考配置文件，在该配置文件中配置了sender需要读取和发送的交易属性内容。该文件的组织形式按照平台支持的不同交易类型分别予以配置，sender读取该文件中的所有交易配置，根据配置要求组装交易发往区块链的节点。如下表展示了该配置文件中的交易属性说明。

参数名称	参数介绍	配置说明
enable	是否发送该交易	配置成true则sender会发送该交易，配置为false则sender不发送该交易
local	是否作为本地交易发送	配置成true则发送本地交易，配置成false则发送非本地交易，本地交易只会在接收交易的节点被执行并返回结果，不经过区块链网络的共识，且不会写入区块链，交易执行所产生的数据变化不会在区块链数据上生效，本地交易往往用于交易的测试和数据查询操作
from	交易的发送方账户标识	交易的发送方账户标识是对区块链上账户的可读标识，需要保证区块链网络上的唯一性，系统内部会对该标识进行哈希后来标识账户。并且此参数值与node_config.user_keys的密钥是一一对应关系。
to	交易的接收方账户标识	交易的接收方账户标识含义与from类似，不过该to账户标识可以是合约账户也可以是普通账户
gas	交易所允许使用的gas费用	可以指定交易执行允许消耗的gas费用，用来限制交易执行的复杂度，如果指定为0或者指定的值超过系统允许的最大gas限制，则会被系统强制设置成系统的最大gas值。系统的最大gas值在系统合约中配置，配置见genesis.json文件中的 chain.gaslimit
nonce	交易的随机值	默认为0时，由sender帮忙生成随机值，不为0时，使用该值作为交易的nonce值
timestamp	交易的时间戳，	默认为0时，由sender帮忙生成当前时间戳，不为0时，使用该值作为交易的时间戳
period	保留字段，以便将来扩展时使用	默认为0时，由sender帮忙生成，不为0时，使用该值作为交易的period
value	交易中需要从发送账户转账给接收方账户的账户金额	在交易执行过程中，如果value值大于零，则会从发送账户扣除value大小的金额，并将金额累加到接收方账户上，value值不能超过发送方账户的当前余额，接收方账户在接收value值后账户余额不能超过系统允许的最大值 。系统的最大value值在系统合约中配置，配置见genesis.json文件中accounts的Administrator.balance
public_keys	待指定的账户公钥列表	考虑到系统支持多重签名账户，因此该处可以指定账户的签名公钥列表
public_key	账户公钥	该处是账户公钥的十六进制字符串表达形式，账户公钥是在账户创建时生成的私钥文件推导出的对应内容，可以通过crypto_tool工具加载私钥文件进行查看获取对应的值
weight	账户公钥在账户签名中所对应的权重	配置public_key所对应的权重信息，该权重是一个不大于100的整形数值，需要确保账户中所有公钥列表的权重之和不小于100
recover_key	重置账户公钥列表所对应的恢复公钥	可以通过该公钥签名交易完成对签名公钥列表的重置
block_number	交易执行的区块号	该属性主要应用于本地交易的发送过程中，用来指定本地交易执行时所依赖的区块号，等同于指定了需要在特定的区块链历史状态下指定该交易
code	合约代码	主要应用于合约的创建和合约的升级，用来指定区块链网络上的智能合约账户代码，该代码的生成可以通过编写智能合约后结合mychain平台提供的智能合约工具进行编译获得。此参数在部署合约和升级合约中使用，合约最大限制大小在系统合约中配置，配置见genesis.json文件中的tx.payload.limit
parameters	调用合约的参数列表	指定调用合约的参数列表，在发布合约的交易中特指调用智能合约构造函数的参数列表，使用逗号分隔的字符串列表，如果参数是一个数组类型，则在字符串参数中使用逗号分隔，类似如下的格式： "100","abc","true","1,2,3","true,false,true","0x1234567890"。支持的参数类型可以参考<<SDK用户手册>>文档中的说明
method	指定需要调用合约的方法	该处是指智能合约中的函数签名信息，即类似的函数原型，包含调用的函数名称和函数参数类型列表，类似如下的形式： "set(uint256,string,bool,bytes)"，注意不允许空格出现
data	用户自定义的data字段	不同的交易类型有不同的data字段格式的要求，如数据隔离的私有交易需要data内需要定义易中合约交易，关联存证交易则是由用户去自行定义data

在上表中罗列了mychain平台交易的所有属性，在系统中支持不同类型的交易，每种交易所依赖和配置的交易属性并不完全一致，所以在不同类型的交易中，可以根据参考配置文件中所指定的交易属性来配置。各种交易类型所代表的含义如下所示。

交易名称	交易含义	交易名称	交易含义
交易名称	交易含义	交易名称	交易含义
CreateAccount	创建普通账户	UpdateAuthmap	更新账户权限公钥列表
TransferBalance	普通账户转账	DeployContract	部署智能合约

SetRecoverkey	设置恢复公钥	CallContract	调用智能合约
PresetPubkey	预重置账户权限公钥列表	UpdateContract	更新智能合约
ResetPubkey	重置账户权限公钥列表	DepositData	存证交易
RelatedDepositData	关联存证交易	DepositEnvelope	数据隔离私有交易
IssueAssets	资产发行交易	RedeemAssets	资产回收交易

在完成sender所依赖的配置文件的设置后，即可执行如下的指令发送交易给mychain区块链网络的节点，执行指令的方式如下。

```
cd /data1/mychain_deploy/bin
./mychain_tool sender -t ../conf/transaction.json -c ../conf/node_config.json
```

执行上述指令后，sender程序会读取node_config.json文件中的配置信息，建立与mychain区块链网络指定节点的链路，并从配置信息中加载出用户的私钥，读取transaction.json文件中enable属性为true的交易配置，构建一个对应的交易数据结构使用私钥进行签名后发往指定节点。同时采用同步的方式接受区块链网络节点上该交易的执行结果，解析执行结果并在用户的终端输出。sender指令会在成功收到mychain区块链网络节点的交易执行结果后，解析执行结果在用户的终端输出，输出样例如下。

```
Transfer balance transaction receipt:
[
    {
        "result":"0",
        "gas_used":"20000",
        "logs":[
            {
                "from":"c60a9d48105950a0cca07a4c6320b98c303ad42d694a634529e8e1a0a16fcdb5",
                "to":"cb84ac09120827b41e01de5494cd25bb06fd7b709879a34f72b8e44b0e6b276f",
                "topics":[
                    "transfer_balance"
                    ],
                "log_data":""
            }
        ],
        "output":""
    }
]
```

> **重要** 因为发送交易类型的差异可能与上述输出存在些许偏差。如果指定发送了多个交易则在用户终端会输出多个交易执行结果。

针对上述输出内容，主要包含交易的执行结果和返回内容，同时包含交易在执行过程所产生的平台或者合约的日志事件信息，针对每个属性的含义如下表所示。

属性名称	属性描述
result	交易结果，该结果是在mychain平台角度来评估的结果，并非包含交易在合约层面的执行结果。返回为0代表平台执行交易c成功，非0表示平台执行交易失败，具体的错误码含义参考 <<错误码参考手册>> 章节说明
gas_used	执行交易所消耗的gas值，该gas值从一定程度上代表了交易执行的复杂度
logs	平台执行交易过程中所产生的日志和事件信息，一般每个交易平台会包含一个日志事件，如果是执行的合约调用则可能还会包含合约中产生的日志和事件信息
logs.from	该日志事件所涉及的发起方，一般为交易的发送方账户标识，该标识是账户可读标识的哈希摘要值
logs.to	该日志事件所涉及的接收方，一般为交易的接收方账户标识，该标识是账户可读标识的哈希摘要值
logs.log_data	该日志事件的的内容，针对平台所产生的日志，一般为空，如果是智能合约所产生的日志事件，则是由智能合约指定
logs.topics	该日志事件所涉及的主题，一般使用主题来作为事件的可读标识和分类，可以基于该主题来快速的进行事件的监听和过滤。一个日志事件可以最多包含4个主题

4.2.3.16. sync子命令

mychain_tool的sync子命令主要用于查看当前mychain节点的同步状态信息，执行命令如下。

```
> sync
```

执行上述命令输出如下结果。

```
Execute synchronize status query succuss:
    Syncing:                    Unsynchronized
    My best:                    84238
    Target:                     84238
```

结果描述如下所示。

属性名称	属性描述
Syncing	同步状态(Unsynchronized，Synchronizing)
My best	当前本地最佳区块号
Target	表示当前需要同步的目标区块号

4.2.3.17. system_contract子命令

system_contract子命令主要用于查询系统合约的配置和修改系统配置，其中修改系统配置操作需要在node_config.json文件中配置Administrator用户。帮助信息如下：

```
> system_contract -h

usage: system_contract --query
         system_contract [-k <key> -v <value>]

options:
   -h [ --help ]                   produce help message
   -q [ --query ]                  query all configurations
   -k [ --key ]                    modify the configuration item key, please use an admin user
   -v [ --value ]                  modify the configuration item value
```

- 功能一：查询系统配置信息
  执行命令如下：

  ```
  system_contract -q
  ```

  命令输出内容如下：

```
Execute query sysytem contract config succuss :
    account.recover_duration                        100
    admin.account                                   0xe7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b
    chain.allow_self_create                         true
    chain.block_version                             844429225099522
    chain.check_create_account_admin                false
    chain.concurrent_execution                      false
    chain.consensus_period                          500
    chain.gaslimit                                  10000000000
    chain.gasprice                                  0
    chain.max_propose_bytes                         4000000
    chain.min_query_block                           0
    chain.null_tx_block_period                      3000
    chain.recycleid                                 0x4a501e12f78568e19b208e4f4c798d346d5b4f30aa4bc0080b4d074aab63e19d
    consensus.algorithm                             pbft
    consensus.batch_size                            256
    consensus.honeybadger.threads                   4
    consensus.is_batch_transaction_hashes           false
    consensus.normal_timeout_interval               3000
    consensus.pbft.be_primary_in_turn               false
    consensus.pbft.viewchange_timeout_interval      3000
    consensus.pbft.viewchange_timeout_max_interval  30000
    consensus.proposal_time_limit                   300000
    consensus.propose_timeout_interval              5000
    consensus.water_level                           5
    contract.native.enable                          true
    nonce.after                                     1800000
    nonce.before                                    3600000
    p2p.codec                                       rlp
    p2p.protocol                                    ssl
    storage.type                                    rocksdb
    tee.rsa_key_version                             0
    tee.sym_key_version                             1
    tx.payload.limit                                1000000
    vm.cache_limit                                  20
    vm.call_depth_limit                             1024
    vm.memory_limit                                 16777216
    vm.stack_limit                                  65536
    vm.value_stack_limits                           32768
```

- **功能二：修改系统配置信息**
  执行命令如下：

```
system_contract -k consensus.batch_size -v 512
```

参数描述：

- -k：指定要修改的配置项名称，参考查询命令结果中的配置项
- -v：指定-k对应的配置项的新值

命令输出内容如下：

```
Execute query system contract config succuss :
    account.recover_duration                        100
    admin.account                                   0xe7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b
    chain.allow_self_create                         true
    chain.block_version                             844429225099522
    chain.check_create_account_admin                false
    chain.concurrent_execution                      false
    chain.consensus_period                          500
    chain.gaslimit                                  10000000000
    chain.gasprice                                  0
    chain.max_propose_bytes                         4000000
    chain.min_query_block                           0
    chain.null_tx_block_period                      3000
    chain.recycleid                                 0x4a501e12f78568e19b208e4f4c798d346d5b4f30aa4bc0080b4d074aab63e19d
    consensus.algorithm                             pbft
    consensus.batch_size                            256
    consensus.honeybadger.threads                   4
    consensus.is_batch_transaction_hashes           false
    consensus.normal_timeout_interval               3000
    consensus.pbft.be_primary_in_turn               false
    consensus.pbft.viewchange_timeout_interval      3000
    consensus.pbft.viewchange_timeout_max_interval  30000
    consensus.proposal_time_limit                   300000
    consensus.propose_timeout_interval              5000
    consensus.water_level                           5
    contract.native.enable                          true
    nonce.after                                     1800000
    nonce.before                                    3600000
    p2p.codec                                       rlp
    p2p.protocol                                    ssl
    storage.type                                    rocksdb
    tee.rsa_key_version                             0
    tee.sym_key_version                             1
    tx.payload.limit                                1000000
    vm.cache_limit                                  20
    vm.call_depth_limit                             1024
    vm.memory_limit                                 16777216
    vm.stack_limit                                  65536
    vm.value_stack_limits                           32768

 modify system configuration succuss
 tx hash: d24dd3de09d56c3b93fd6eb7514f28299181ebc7c5f500cf98be3c5db20f3339
 recepit result: {"result":"0","gas_used":"75566","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"87e89abb4c1c551fe08d355d097f18b8de78ec
f555997085681662fce8eed","topics":["call_contract"],"log_data":""},
{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"87e89abb4c1c551fe08d355d097f18b8de78ed
555997085681662fce8eed","topics":
["4897a472ac8897deca130ff7c0018fee47e2df535ef8905308f45a16b5d6df07"],"log_data":"000000000000000000000000000000000000000
000000000000000000040000000000000000000000000000000000000000000000000000000000000000000000000000000800000000000000000000000000000000
000000000000000000000000014636f6e73656e7375732e62617463685f73697a650000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000003353132000000000000000000000000000000000000000000000000000000000000000000000"}],"output":"000000000
0000000000000000000000000000000000000000000000000000000000000"}
```

4.2.3.18. transaction子命令

mychain_tool的**transaction**子命令主要用于查询保存在区块链中的交易以及交易收据信息，其支持两种模式的查询：一是通过hash值查询交易信息，通过**-t**来标识；二是通过hash值查询交易收据信息，通过**-r**来标识；三是通过交易哈希查询交易的内容和收据信息，如下所示。

```
> transaction -h

usage: transaction {-t, -l, -r} hash

options:
   -h [ --help ]                  produce help message
   -t [ --txHash ]                query transaction by hash
   -l [ --txHashList ]            query transaction by hash list Limit 5，Hash separated by comma
   -r [ --txReceiptHash ]         query transaction receipt by hash
   -a [ --all ]                   query transaction and receipt by hash
```

下面分别介绍，查询交易信息执行如下命令。

```
transaction -t 7f3e2dd7355533183de4ad7cb1b838fb0611f6692a86cba80aced08709929d07
```

执行上述命令输出如下结果：

```
Execute query transaction by hash succuss:
    block number:                   63005
    tx index:                       0
    hash:                           7f3e2dd7355533183de4ad7cb1b838fb0611f6692a86cba80aced08709929d07
    type:                           TRANSFER_BALANCE
    timestamp:                      1539915250168
    nonce:                          493864955596744559
    period:                         1539915250168
    from:                           c60a9d48105950a0cca07a4c6320b98c303ad42d694a634529e8e1a0a16fcdb5
    to:                             cb84ac09120827b41e01de5494cd25bb06fd7b709879a34f72b8e44b0e6b276f
    value:                          100
    gas:                            1000000
    data:
    signatures:                     [
1be750c310be88ab958de56646eb939cb626dae6c9c8c5cb346b3dfe4513c4ba7418ba5c452fee8c32c03ddbf2d75145cf740f98ead6362b4caaedc8d
90d00 ]
```

其结果信息描述如下所示。

属性名称	属性描述
属性名称	属性描述
block_number	交易所在区块高度
tx index	交易在区块中的索引
hash	交易的哈希
type	交易类型
timestamp	交易时间戳
nonce	交易的nonce值
period	交易预留时间
from	交易发送方
to	交易接收方
value	交易的value值
gas	交易发送的gas
data	交易输入数据
signature	交易签名

查询交易收据信息，执行如下命令。

```
transaction -r 7f3e2dd7355533183de4ad7cb1b838fb0611f6692a86cba80aced08709929d07
```

执行上述命令输出如下结果。

```
Execute query transaction receipt by hash succuss:
    block number:                   63005
    index:                          0
    result:                         SUCCESS
    gas used:                       20000
    output:
    log entries:
                                1.  from:       c60a9d48105950a0cca07a4c6320b98c303ad42d694a634529e8e1a0a16fcdb5
                                    to:         cb84ac09120827b41e01de5494cd25bb06fd7b709879a34f72b8e44b0e6b276f
                                    topics:     [ 7472616e736665725f62616c616e6365 ]
                                    log data:
```

结果信息描述如下所示。

属性名称	属性描述
属性名称	属性描述
block number	交易所在区块号

index	交易在区块中的索引
result	交易结果
gas_used	交易执行消耗的gas
output	交易输出
log_entry.from	交易日志中记录交易发送方
log_entry.to	交易日志中记录交易接收方
log_entry.log_data	交易日志中记录交易数据
log_entry.topic	交易日志中记录主题

查询交易和收据信息。

```
transaction -a 7f3e2dd7355533183de4ad7cb1b838fb0611f6692a86cba80aced08709929d07
```

执行上述命令后输出结果如下所示。

```
Execute query transaction and receipt by hash succuss :
    Transaction index :             0
    ----------transaction info----------:
    block number:                   63005
    tx index:                       0
    hash:                           7f3e2dd7355533183de4ad7cb1b838fb0611f6692a86cba80aced08709929d07
    type:                           TRANSFER_BALANCE
    timestamp:                      1539915250168
    nonce:                          493864955596744559
    period:                         1539915250168
    from:                           c60a9d48105950a0cca07a4c6320b98c303ad42d694a634529e8e1a0a16fcdb5
    to:                             cb84ac09120827b41e01de5494cd25bb06fd7b709879a34f72b8e44b0e6b276f
    value:                          100
    gas:                            1000000
    data:
    signatures:                     [
1be750c310be88ab958de56646eb939cb626dae6c9c8c5cb346b3dfe4513c4ba7418ba5c452fee8c32c03ddbf2d75145cf740f98ead6362b4caaedc8d
90d00

    ----------receipt info----------:
    result:                         SUCCESS
    gas used:                       20000
    output:
    log entries:
                                1.  from:           c60a9d48105950a0cca07a4c6320b98c303ad42d694a634529e8e1a0a16fcdb5
                                    to:             cb84ac09120827b41e01de5494cd25bb06fd7b709879a34f72b8e44b0e6b276f
                                    topics:      [ 7472616e736665725f62616c616e6365 ]
                                    log data:
```

4.2.3.19. transaction_queue子命令

mychain_tool的transaction_queue子命令主要用于查看mychain节点的交易缓存队列信息，执行命令如下。

```
> transaction_queue
```

用户可以直接执行该命令，其结果如下所示。

```
Execute transaction cache query succuss:
    unverified queue limit:         20480
    unverified queue usage:         0
    verified queue limit:           40960
    verified queue usage:           0
```

其结果描述信息如下。

属性名称	属性描述
unverified queue limit	未验证队列最大容量（以交易为单位）
unverified queue usage	未验证队列中已使用大小
verified queue limit	验证队列最大容量（以交易为单位）

verified queue usage	验证队列中已使用大小

4.2.3.20. update_crl子命令

mychain_tool的**update_crl**子命令主要用于更新客户端吊销证书列表。执行命令如下。

```
update_crl -p ../certs/client.crl
```

其中-p是指调吊销证书列表的路径，用户可以直接执行该命令，其结果如下所示。

```
Sending update crl to mychain completes!
```

4.2.3.21. system_manage子命令

mychain_tool的system_manage子命令用来升级或者查询系统合约。

可用选项如下表所示。

可用选项	描述
-i	用来发交易的账户，默认是Administrator。发送交易更新系统合约或调用系统合约的Version()接口
--name	指定要操作的合约名称，可选值有 `Node|Config|Group|Govern|Subnet|Authority`
-u	如果要升级系统合约，使用**-u**选项
-f	如果要升级系统合约，**-f**用来指定新合约的字节码文件路径
-q	如果要查询系统合约，使用-q选项。如果对2.19以前的系统合约进行查询操作，会得到错误码 `10201` ，这是因为2.19以前的系统合约并没有Version()这个函数接口

示例1：更新Node合约

```
./mychain_tool system_manage --name Node -f ../conf/compiled_system_contracts/contract_node.bin-runtime -u

call contract transaction:
[{"hash":"c23b901552b7f7910116650b7f19e1edae31dd89dbe0a6022405965b4f7550f2","type":"12","timestamp":"1602308131037","nonce
3188504985787704603","period":"0","from":"e93372533f323b2f12783aa3a586135cf421486439c2cdcde47411b78f9839ec","to":"e9337253
23b2f12783aa3a586135cf421486439c2cdcde47411b78f9839ec","value":"0","gas":"100000000","group_id":"0000000000000000000000000
000000000000","version":"2","data":"0160806040526004361061006100f1576000357c0100000000000000000000000000000000000000000000000000000
00000900463ffffffff1680630c9f3eaa146100f65780630d94bfc014610181578063261b40a4146101d257"]}]

Update contract transaction receipt: [{"result":"0","gas_used":"329440","logs":
[{"from":"e93372533f323b2f12783aa3a586135cf421486439c2cdcde47411b78f9839ec","to":"e93372533f323b2f12783aa3a586135cf421486
2cdcde47411b78f9839ec","topics":["update_contract"],"log_data":""}],"output":""}]
```

示例2：查询Node合约

```
> system_manage -q --name Node

call contract transaction:
[{"hash":"7c3af16b0d58aad344f9335c1e1d6a96e36300b36b47a9b00cd771cfce5b585f","type":"11","timestamp":"1602308020567","nonce
4556387513396268744","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"e9337253
23b2f12783aa3a586135cf421486439c2cdcde47411b78f9839ec","value":"0","gas":"10000000","group_id":"0000000000000000000000000
00000000000","version":"2","data":"bb62860d","extensions":"","signature":
["55e6fa7caedf7f6703130068db87d1e5a1da38372380a8779ab3e8b7aeca547b38962d62c165e"]}]

Call contract transaction receipt: [{"result":"0","gas_used":"20847","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"e93372533f323b2f12783aa3a586135cf421486
2cdcde47411b78f9839ec","topics":
["call_contract"],"log_data":""}],"output":"0000000000000000000000000000000000000000000000000000000000000020000000000000000
0000000000000000000000000000000000000000000008322e31392e302e3000000000000000000000000000000000000000000000000000000000000000"}]

current version is: 2.19.0.0
```

4.2.3.22. subnet子命令

mychain_tool的**subnet**子命令用来灵活使用subnet的功能特性。

可用选项：

-m: 指定要操作的系统合约的功能，可选值有 add | quit | join | stop | start | list | query等方式

其他参数为各个功能操作的参数，如：add(-c) | quit(-i -p) | join(-i -p) | stop(-i) | start(-i) | list | query(-i)

示例1

```
> ./mychain_tool subnet -h
usage: subnet --method [add | quit | join | stop | start | list | query] [help info]
options:
   -h [ --help ]                  print help message
   -m [ --method ]                the method you want to call, like add(-c) | quit(-i -p) | join(-i -p) | stop(-i) | st
art(-i) | list | query(-i)
   -c [ --genesis_file ]          the genesis of a new subnet
   -i [ --subnet_index ]          the subnet index of all subnets
   -p [ --publick_key ]           the public key of node
```

示例2 [AddSubnet]

```
> ./mychain_tool subnet -m add -c ../conf/genesis_subnet.json
```

> **重要** 4节点测试时，若新建Subnet，可以直接将../conf/genesis.json改成../conf/genesis_subnet.json，并删除contracts中的subnet合约信息。

示例3 [JoinSubnet | QuitSubnet]

```
> ./mychain_tool subnet -m join -i 1 -p
0xFA5D1F80155CF8F0670F00EDD129CA5162E2DFC092965B971B6208180C73059CF90337BBC6543078BF9BD3A51A1BDFB948C0CBA45E91202C7CAD7D2
E8B81
```

示例4 [StartSubnet | StopSubnet]

```
> ./mychain_tool subnet -m start -i 1
call contract transaction:
[{"hash":"536cd21fb18ce5abb0fe56c2ce31f17b03caf91cbad55e51b2d0f8a20a28e9c4","type":"11","timestamp":"1602610009881","nonce
10474366896927529382","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"bfea90e
1816a3935322c35705a585a2bc319ff57377428455a234d6371472","value":"0","gas":"10000000","group_id":"0000000000000000000000000
000000000000","version":"2","data":"99927d1b0000000000000000000000000000000000000000000000000000000000000000000000000001","extensions"
,"signature":["be3d98625fc8cb7c55f8341418522f0bddf263a31d80b444240c33318123ae357c88c4cd73661a96"]}]

Call contract transaction receipt: [{"result":"0","gas_used":"34577","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"bfea90e5ae1816a3935322c35705a585a2bc319
377428455a234d6371472","topics":["call_contract"],"log_data":""},
{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"bfea90e5ae1816a3935322c35705a585a2bc319f
77428455a234d6371472","topics":
["a8f7aad3170c13d4df4bee045423729a46e2642677fd6c50ff0906535fd01767","0000000000000000000000000000000000000000000000000000000000000
00000001","0000000000000000000000000000000000000000000000000000000000000001"],"log_data":""}],"output":""}]

Call action successfully
```

示例5 [ListSubnet]

```
> ./mychain_tool subnet -m list
start to execute the action: list

call contract transaction:
[{"hash":"84eded8f9340c48a175cf47863b9be9c23d3891232dfafc93532847e01bf6c10","type":"11","timestamp":"1602610026606","nonce
2495547067553278 24","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"bfea90e5a
16a3935322c35705a585a2bc319ff57377428455a234d6371472","value":"0","gas":"10000000","group_id":"00000000000000000000000000
0000000000","version":"2","data":"73758557","extensions":"","signature":
["f5fefd0c58f1fb3f84b5eeaf8678ca7d647d66acba1e456eeba7e14f5890bcc45f4b636a23a6dc9d4cf0faf0425bc837cf828a7973358daa53b1757e
5203f01","c6adca0e6974456218844b55a74f0f31054718be1c83ea52c836b0010247a4b9677c973a444611aedb056413271d06c1a0e27056d550757a
3e7ef0f08073301","a21a0b2749a5feb4dc157d3b8d8cc59ce9e1e5235921754da13ff116ea122cd8043407a2bc4406c54ed9516339b19f290336256c
11d1d23d924e3b630985001"]}]

Call contract transaction receipt: [{"result":"0","gas_used":"275819","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"bfea90e5ae1816a3935322c35705a585a2bc319
377428455a234d6371472","topics":
["call_contract"],"log_data":""}],"output":"00000000000000000000000000000000000000000000000000000000000000200000000000000
0000000000000000000000000000000000000000002fe7b20227375626e65747322203a205b207b20227375626e6574496422203a20312c20226e6f6
5f636f756e7422203a20312c20226e6f64655f7075626b65797322203a205b7b20227075626b657922203a2022307838374645363846384145444137
44230303031344646334645331453841434242363939363938303033135353646373939 4644"}]

Call action successfully
Receipt output is: { "subnets" : [ { "subnetId" : 1, "node_count" : 1, "node_pubkeys" : [{ "pubkey" :
"0x87FE68F8AEDA7CADB00014FF3FE31E8ACBB6996980031556F799FD4D49DA075CCEE9B0692D813E0A5703CF031DAB8F54E7B0A585334D273E709544
CCF691" }], "subnetState" : 1 }, { "subnetId" : 2, "node_count" : 3, "node_pubkeys" : [{ "pubkey" :
"0xFA5D1F80155CF8F0670F00EDD129CA5162E2DFC092965B971B6208180C73059CF90337BBC6543078BF9BD3A51A1BDFB948C0CBA45E91202C7CAD7D2
8E8B81" }, { "pubkey" :
"0x2B1EEDD1C894EDEB696A0398A84EF37199C76EFBC3D7734380AC538298DDFEC5D9184DD5EE709D91D9B8C8085726009D2E5E51C4D1632303CFDCFB9
F87A1A" }, { "pubkey" :
"0x87FE68F8AEDA7CADB00014FF3FE31E8ACBB6996980031556F799FD4D49DA075CCEE9B0692D813E0A5703CF031DAB8F54E7B0A585334D273E709544
CCF691" }], "subnetState" : 1 } ] }
```

4.2.3.23. subnet_access_control子命令使用

`subnet_access_control` 子命令主要用于跨子网通讯请求中保证安全性的功能（即子网访问权限）。它按照用户指定的子网 S_1 上允许其他子网 S_i 访问，且支持不同功能，如查询区块、区块头、交易、LocalCall。

用户可以使用如下命令来查询其使用方式：

```
>./mychain_tool subnet_access_control -h

 usage: access --type [addq | deleteq | findq | addc | deletec | findc | list] --mychainId[SubnetX] [help info]

 options:
    -h [ --help ]                   print help message
    -t [ --type ]                   the method type you want to call, like addq(-i -m) | | deleteq(-i -m) | findq(-i -m)
| addc(-i -m -c -f) | deletec(-i -m -c -f) | findc(-i -m -c -f) | list
    -i [ --permitted_mychainId ]    the permitted mychainId that can access, e.g.
0000000000000001000000000000000000000000 or subnet index[0-x].
    -m [ --method ]                 the method you want to add access control, like QueryBlock, QueryTransaction
    -c [ --contract_id ]            the id of defined contract
    -f [ --contract_function ]      the function of defined contract
    -g [ --mychainId ]               the contract in which dynamic subnet[1-X], e.g.
0000000000000001000000000000000000000000 or subnet index[1-x], default is subnet0.
```

- 参数 `-t` 用来指定子网访问权限的方法类型。
- 参数 `-i` 用来指定允许的子网标识，可以是 h160 或者子网的序号。
- 参数 `-m` 用来指定允许子网的访问方法。
- 参数 `-c` 用来指定子网访问权限为“LocalCall”的合约 id。
- 参数 `-f` 用来指定子网访问权限为“LocalCall”的合约 id 中的方法。
- 参数 `-g` 用来指定操作该子命令执行时所在的子网，默认为主网。

在指定 `-t` 后可以根据上述中的信息，指定各个方法需要的参数。

功能一：新增子网查询权限

- 目标子网账本信息查询：QueryBlock、QueryBlockHeader、QueryTransaction
  执行命令示例如下：

  ```
  >./mychain_tool subnet_access_control -t addq -m QueryBlock -i 1 -g 2
  ```

  执行上述命令输出如下结果：

```
get input subnet index 2
Execute SysContractCommand on subnet : 0000000000000002000000000000000000000000

start to execute the action:

parameter(permitted_mychainId): get input permitted_subnet index 1

call contract transaction:
[{"hash":"c515ca69d83271af73a95d5c512f7bdd2f0462d0420c04d5f0a564ff49f6571f","type":"11","timestamp":"1619593174830","no
:"6491182380235466824","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21f
8b64bde0491f9347cbc73854576bba59bcf9dd8cf03e39edb44ace130","value":"0","gas":"10000000","group_id":"000000000000000200
0000000000000000","version":"2","data":"911f16c900000000000000000000000000000000000000000000000000000000000000400000000
0000000000000000000000000000000000000000000000000a000000000000000000000000000000000000000000000000000000000000000283
0303030303030303030303031303030303030303030303030303030303030303030303030000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000a5175657279426c6f636b000000000000000000000000000000000
00000","extensions":"","signature":
["76d094276d2c00dec183fab1e725f18fc38ff01d5f3fee2d3d87e90f0dd087216e785f29fbe6992af3c22e4e040feb5780dcfeae5718ed9eb3b5f
066fe8c01","2a23cb8fd7f31978d95e41cd6707ef5c95de422381b5c0e3f8f68ab8cc51c0553f1fc6f5d3a085a1614938f949fdb8d5aa44dff4697
86cbd4c95c531f73801","1ed72950be725efe4fad5ab9298f3116da547c17b80a9e2c29e002eeefc58e9d18a54f52cb976d0e6c34617739195b79b
e4a4b77f9f1503788860aabd72900"]}]

Call contract transaction receipt: [{"result":"0","gas_used":"154340","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288b64bde0491f9347cbc73854576bba5
f9dd8cf03e39edb44ace130","topics":["call_contract"],"log_data":""},
{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288b64bde0491f9347cbc73854576bba5
9dd8cf03e39edb44ace130","topics":
["f2dd28bb9b919a0b704e9e35b487de296d534026d7d3927d3b9d87c1d2f9dfdc"],"log_data":"000000000000000000000000000000000000000
0000000000000000400000000000000000000000000000000000000000000000000000000000000a000000000000000000000000000000000000
0000000000000000000000283030303030303030303030303031303030303030303030303030303030303030303030303000000000000000000
0000000000000000000000000000000000000000000000000000000000000a5175657279426c6f636b000000000000000000000000000000000
0000000000000000000000000000000000"}],"output":"0000000000000000000000000000000000000000000000000000000000000000"}]


Call action successfully
AddAcQuery done: [0000000000000001000000000000000000000000]->[0000000000000002000000000000000000000000]:QueryBlock
```

- 目标子网中合约调用：LocalCall
  执行命令示例如下：

```
>./mychain_tool subnet_access_control -t addc -m LocalCall -c simple -f "set(uint256)" -i 2 -g 1
```

> **说明** 以上是子网1 给子网2 授权合约 `simple` 的 `set(uint256)` 方法。

功能二：删除子网查询权限

执行命令示例如下：

```
>./mychain_tool subnet_access_control -t deleteq -m QueryBlock -i 1 -g 2
```

执行上述命令输出如下结果：

```
get input subnet index 2
Execute SysContractCommand on subnet : 0000000000000002000000000000000000000000

start to execute the action:

parameter(permitted_mychainId): get input permitted_subnet index 1

call contract transaction:
[{"hash":"fe0b3ea0a88985199f2150d07d67e99d035b671a7bab6d76aa21fceab5f7a9ac","type":"11","timestamp":"1619593398169","nonce
17428319179306030645","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb228
4bde0491f9347cbc73854576bba59bcf9dd8cf03e39edb44ace130","value":"0","gas":"10000000","group_id":"000000000000000200000000
000000000000","version":"2","data":"b71e0c4c00000000000000000000000000000000000000000000000000000000400000000000000000000
0000000000000000000000000000000000000000000a0000000000000000000000000000000000000000000000000283030303030303030303030303030
303030303030303031303030303030303030303030303030303030303030303030300000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000a5175657279426c6f636b000000000000000000000000000000000000000000","ex
sions":"","signature":
["3d99c73bd07185c64f1e7dcbe3cf3a051fd2468e992d5d5cd7932fb09e03038e67604995abbbbe498c8939739440ae60ce7a271f10f4010cbb29506
484b000","ea14a36c417daa992a82b078f829c1932d71df08d1112c440dffbb842ceabec410158a0982092e922f59b9de32f4cb06cb1dc458c87bea8
e27467b99820900","f6907847bf95c92599f178e59292f266446b53867b8d8f3939017ea2907f24372827827f9eb2d6ec615814ac2b6bb557632118a
cab314408e7c089626b5d00"]}]

Call contract transaction receipt: [{"result":"0","gas_used":"50652","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288b64bde0491f9347cbc73854576bba59b
dd8cf03e39edb44ace130","topics":["call_contract"],"log_data":""},
{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288b64bde0491f9347cbc73854576bba59b
d8cf03e39edb44ace130","topics":
["1871acce2498e0e69006f4d3839bf42f62780c30163150998f7529bf27ec8914"],"log_data":"0000000000000000000000000000000000000000
0000000000000000400000000000000000000000000000000000000000000000000000000000000a000000000000000000000000000000000000000000
00000000000000000000283030303030303030303030303131303030303030303030303030303030303030303030303030303000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000a5175657279426c6f636b000000000000000
00000000000000000000000000000"}],"output":"0000000000000000000000000000000000000000000000000000000000000000"}]

Call action successfully
```

功能三：查找子网查询权限

执行命令示例如下：

```
>./mychain_tool subnet_access_control -t findq -m QueryBlock -i 1 -g 2
```

执行上述命令输出如下结果：

```
get input subnet index 2
Execute SysContractCommand on subnet : 0000000000000002000000000000000000000000

start to execute the action:

parameter(permitted_mychainId): get input permitted_subnet index 1

call contract transaction:
[{"hash":"0fd29147146428a51231a13c956d859cd38dd9b9c881b09054e591bc20148031","type":"11","timestamp":"1619593389647","nonce
3678009694658756563","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288
bde0491f9347cbc73854576bba59bcf9dd8cf03e39edb44ace130","value":"0","gas":"10000000","group_id":"0000000000000002000000000
00000000000","version":"2","data":"e78f80e0000000000000000000000000000000000000000000000000000000004000000000000000000000
000000000000000000000000000000000000000000a0000000000000000000000000000000000000000000000000283030303030303030303030303030
0303030303030303130303030303030303030303030303030303030303030303030300000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000a5175657279426c6f636b000000000000000000000000000000000000000000","ext
ions":"","signature":
["7127e5cb989f5b2a8558594cb269905d2437e896eac2b4c15ee742e61d7d69c629e0f185b1cd447ed9c50c588a89a4607b1593e91964918714218f2
7adc600","06b6f224e971e034e1f24f1105259e7b0e1bab757f32622305c0a94402b4c5a23236b52592ed96a508e2dbe78390c6817e127af429dc24d
9a6cb8abea60e01","cd14abd6a8131a708e8522d1fe446f35d88757f5274114f63cb3ce5396a8a7c91b9c7616ebdb82d0ff89452570dae67330a6670
bad79e682cc15f768055800"]}]

Call contract transaction receipt: [{"result":"0","gas_used":"32589","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288b64bde0491f9347cbc73854576bba59b
dd8cf03e39edb44ace130","topics":
["call_contract"],"log_data":""}],"output":"0000000000000000000000000000000000000000000000000000100000000000000000000000
000000000000000000000000000000000000000000000"}]

Call action successfully
Receipt output is: exist:1, index:0
```

功能四：展示子网查询权限列表

执行命令示例如下：

```
>./mychain_tool subnet_access_control -t list -g 2
```

执行上述命令输出如下结果：

```
get input subnet index 2
Execute SysContractCommand on subnet : 00000000000000020000000000000000000000000

start to execute the action:


call contract transaction:
[{"hash":"a5cfb66fe98110c5d053149524d74445107f41b3a204bed2a90e9de70b64d9a2","type":"11","timestamp":"1619595072354","nonce
7949266017897904980","period":"0","from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288
bde0491f9347cbc73854576bba59bcf9dd8cf03e39edb44ace130","value":"0","gas":"10000000","group_id":"000000000000000020000000000
00000000000","version":"2","data":"dd9feb5b","extensions":"","signature":
["6b7986780e0e54e5f1244c0ca08a67572286a95e7ca239d86bbbf04d78e3399a373f8183742dcb1c3f305ae8aac3bdac2fc4524874e5e96ebbd8b841
55f9201","de8ecb28cc981ffd81e4ba3133ae57195723c15c453cf474d5cb293a4ddf06d54812e7a12fb49079d5a9db3d419493790bf67c12b1d96c41
1c3c77e2d87c300","607f691e6338f4d92df6d7366f0085dc24be40e6c8b64b9205482cd0e0f1b4e60868486623dbecfd1f6d47cc3e01d3d088273793
9f730fb03f22d3b26ceaf00"]}]

Call contract transaction receipt: [{"result":"0","gas_used":"41508","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288b64bde0491f9347cbc73854576bba59b
dd8cf03e39edb44ace130","topics":
["call_contract"],"log_data":""}],"output":"0000000000000000000000000000000000000000000000000000000000000020000000000000000
0000000000000000000000000000000000000000000000867b202261635f717565727922203a205b207b20227065726d69747465645f6d79636861696e5
6422203a2022303030303030303030303030303030303130303030303030303030303030303030303030303030303030222c20226d6574686f6422203a2022
5657279426c6f636b22207d5d2c202261635f63616c6c22203a205b20205d207d0000000000000000000000000000000000000000000000000000000000000000"}]


Call action successfully
Receipt output is: { "ac_query" : [ { "permitted_mychain_id" : "0000000000000001000000000000000000000000", "method" : "
QueryBlock" }], "ac_call" : [  ] }
```

更多信息展示：

```
Call contract transaction receipt: [{"result":"0","gas_used":"166880","logs":
[{"from":"e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b","to":"21fb2288b64bde0491f9347cbc73854576bba59
dd8cf03e39edb44ace130","topics":
["call_contract"],"log_data":""}],"output":"0000000000000000000000000000000000000000000000000000000000000020000000000000
0000000000000000000000000000000000000000030a7b202261635f717565727922203a205b207b20227065726d69747465645f6d79636861696e5
6422203a2022303030303030303030303030303030323030303030303030303030303030303030303030303030303030222c20226d6574686f6422203a2022
56572795472616e73616374696f6e22207d2c207b20227065726d69747465645f6d79636861696e5f696422203a2022303030303030303030303030303030
32303030303030303030303030303030303030303030303030303030222c20226d6574686f6422203a2022517565727926c6f636b22207d2c207b2022706572
9747465645f6d79636861696e5f696422203a2022303030303030303030303030303030303230303030303030303030303030303030303030303030303030
20226d6574686f6422203a2022517565727926c6f636b486561646572222207d5d2c202261635f63616c6c22203a205b207b20227065726d6974746564
d79636861696e5f696422203a2022303030303030303030303030303030323030303030303030303030303030303030303030303030222c20226d657
6f6422203a20224c6f63616c43616c6c222c2022636f6e74726163745f696422203a2022386234343339316362323863613432626466466626636366234
5393763623063623964373636356165643330336664343638663865626337616230666643466222c2022636f6e74726163745f66756e6374696f6e2220
227365742875696e743235362922207d2c207b20227065726d69747465645f6d79636861696e5f696422203a2022303030303030303030303030303030
030303030303030303030303030303030303030303030303030222c20226d6574686f6422203a20224c6f63616c43616c6c222c2022636f6e74726163745f6
22203a2022323733323165303731393765346439306431393665386466456339333733343465366337383033663164323033643265353634656233313
5386231636531222c2022636f6e74726163745f66756e6374696f6e22203a20227365742875696e743235362922207d205d207d000000000000000000
0000000000000000000000"}]

{
    "ac_query":[
        {
            "permitted_mychain_id":"0000000000000002000000000000000000000000",
            "method":"QueryTransaction"
        },
        {
            "permitted_mychain_id":"0000000000000002000000000000000000000000",
            "method":"QueryBlock"
        },
        {
            "permitted_mychain_id":"0000000000000002000000000000000000000000",
            "method":"QueryBlockHeader"
        }
    ],
    "ac_call":[
        {
            "permitted_mychain_id":"0000000000000002000000000000000000000000",
            "method":"LocalCall",
            "contract_id":"8b44391cb28ca42bddfbf66b4be97cb0cb9d7665aed303fd468f8ebc7ab0fd4f",
            "contract_function":"set(uint256)"
        },
        {
            "permitted_mychain_id":"0000000000000002000000000000000000000000",
            "method":"LocalCall",
            "contract_id":"27321e07197e4d90d196e8ddec937344e6c7803f1d203d2e564eb3185e8b1ce1",
            "contract_function":"set(uint256)"
        }
    ]
}
```

4.2.4. 支持subnet

对于工具使用中的所有子命令都支持subnet操作。

在2.19版本后，mychain支持subnet动态组网。

目前交易已经支持，**sender**子命令暂时不添加该功能，**sender**作为交易发送工具，需要在合约交易（json文件）中添加 `group_id` 字段，即可将交易发送到对应的Subnet中（当前节点需要在该Subnet中），交易如下。

```
"CallContract" : {
    "enable" : true,
    "local" : false,
    "from" : "Administrator",
    "to" : "Node",
    "gas" : 10000000,
    "nonce" : 0,
    "timestamp" : 0,
    "period" : 0,
    "value" : 0,
    "contract_type" : "evm",
      "group_id": "0000000000000001000000000000000000000000",
    "method" : "DeleteNode(bytes32)",
    "parameters" : [
        "0x6007ef66a285f600fe810a944c3beceabb924fb82533aeccc388cab8a561bad6"
    ],
    "block_num" : 0
}
```

2.20迭代中计划：mychain_tool的各个子命令也可以支持在各个Subnet中进行数据检索。

原有功能保持不变，默认查询 `subnet0[h160(0)]` ，对于其他Subnet则需要指定 `subnet_id` 或者 `subnet_index[1-x]` 。

如 `-g 0000000000000001000000000000000000000000` 表示指定的Subnet1，例如：

```
> account -i e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b -g
0000000000000001000000000000000000000000

> account -i e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b -g 1
```

4.3. archive_tool

4.3.1. 工具简介

archive_tool 是一个独立提供的命令行实用工具，用于对 mychain-0.10 区块链系统产生的老旧数据在指定的范围内进行归档和数据清理操作，其主要目的是为了解决块链数据不断增长和单机存储空间有限之间的矛盾。

数据归档的过程可以大致上描述如下。

1. 节点运行停止，世界状态不再更新。
2. 初始化，打开对应数据库的读写句柄，在mychain-0.10中，主要对应着 block / state / extra 数据库。
3. 确定需要归档的区块、需要保留的区块范围。
4. 对区块无关的数据，直接进行kv拷贝到新建的临时数据库，对需要保留的块，遍历其状态树、存储树，拷贝对应的kv到临时数据库。
5. 将整个老数据库拷贝到指定的归档位置，随后删除老数据库，启用新建的临时数据库。
6. 归档完成，重启节点。

4.3.2. 工具获取

archive_tool 的获取途径有两种：

- 第一种是预先编译的二进制可执行文件形式，此种情况下 archive_tool 会随 mychain-0.10 的RHEL二进制发布包一同提供（从 v0.10.2.1 开始），解压 tar 包文件后路径是 bin/archive_tool。
- 第二种是源码编译形式，archive_tool 随平台代码一同编译，参考mychain-0.10 的源码编译文档进行编译，即可得到 archive_tool 工具。

4.3.3. 工具使用

基本使用语法

archive_tool 的基本使用语法是：

```
archive_tool [--help] --data-dir <path> --archive-dir <path> [--time <date_str>] [--retain-num <number>]
```

各个选项的意义如下表所示。

选项	是否可选	是否带参数	描述
--help	可选	不带参数	当给出该选项时，则忽略其他所有命令行选型，打印 archive_tool 自带的帮助信息，并以返回值0成功退出。
--data-dir	必选	带一个参数：一个目录路径	指定需要被归档的 mychain-0.10 数据库路径，也就是当前数据库路径。例如 --data-dir /path/to/data。

--archive-dir	必选	带一个参数：一个目录路径	指定归档数据的目的路径，也就是老旧数据存放的路径。通常，可以是一个挂载了大容量磁盘的路径，例如 --archive-data /path/to/archive-01。
--time	可选	带一个参数：一个ISO-8601 标准格式的时刻信息 (CCYYMMDD) 或者 ISO-8601 扩展格式 (CCYY-MM-DD) 的时刻信息	指定一个时刻，其意义是，对于当前系统所产生的所有区块，时间戳在该时刻前的区块，均需要被归档到前面指定的路径。例如 --time 20181201。
--retain-num	可选	带一个参数：一个正整数	指定多少个区块需要在归档后被保留在当前的存储系统中。例如 --retain-num 10。
--version	可选	不带参数	打印 archive_tool 的版本信息，并以返回值0成功退出。

重要 **--time**和**--retain-num**选项均可以指定数据归档的范围。当两个选项同时出现，且参数均合法时，**--time**选项具有更高的优先级，也就是这种情况下归档范围以**--time**的参数为准。当两个选项均不给出，系统会以**--retain-num**的默认值（10）进行归档操作。

基本使用流程

在一个联盟区块链系统中，根据所选择的共识协议的容错模型，可能会有2f+1和3f+1的容错类型区别。在做数据归档时，应当由各个节点的管理员进行沟通，依次轮流进行节点的停止、数据归档、重启，保证联盟区块链系统作为一个整体对外能够继续提供服务且保持一致性。过程中还需要考虑归档节点完成数据归档后重启的数据同步过程。

4.4. crypto_tool

4.4.1. 工具简介

mychain平台在使用过程中会需要相关的工具来执行数据操作，例如哈希、加解密、签名验签、证书生成、签名证书、密钥处理、交易编解码等。mychain平台通过提供加解密工具crypto_tool来提供上述功能。

假设用户已经按照**《蚂蚁链私有化标准版用户指南》**搭建了mychain的环境，则可以通过执行如下命令来获取crypto_tool工具的使用说明。

```
cd /data1/mychain_deploy/bin
./crypto_tool -h
```

成功执行上述命令会在用户终端输出子命令的参数和帮助信息。

```
-h [ --help ] Print this help message and exit
-t [ --type ] arg (=sha256) digest algorithm type:sha256(default)/sm3/keccak256
-d [ --data ] arg  the data to hash. data with prefix "0x" will be treated as hex data, and string otherwise.
```

说明 上述参数的具体信息在后续的章节进行详细介绍，也可以通过调用其他子命令来查看相关的使用帮助信息。

4.4.2. 工具使用

4.4.2.1. hash子命令

crypto_tool的hash子命令主要用来完成对指定数据的哈希计算，这个操作可以应用主对账户可读标识的哈希计算获取账户标识。

执行如下的命令操作来完成指定数据的哈希计算，其中**-t**参数用于指定hash计算采用的算法（可选算法：sha256(default)/sm3/keccak256），省略该参数时默认采用sha256算法进行哈希计算。

```
cd /home/mychain/mychain-0.10.2.12/node/bin
./crypto_tool hash -d Administrator -t sha256
```

说明 如果需要对不可见的bytes数据进行哈希计算，则可以使用0x开头的十六进制字符串表示。

成功执行上述命令会在用户终端输出如下哈希计算结果。

```
hash result: e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b
```

4.4.2.2. key子命令

crypto_tool的key子命令主要用来加载和读取指定私钥文件的内容，并在用户终端输出公私钥信息。

key子命令执行方式如下。

```
cd /home/mychain/mychain-0.10.2.12/node/bin
./crypto_tool key -k ../../user/Tester001.key
```

执行上述命令会要求用户在终端输入 `key.key` 的解密密码，输入过程会屏蔽密码明文，在上述命令操作中，**-k**参数用于指定需要读取的私钥文件路径，当然也可以使用**-p**参数用于指定私钥文件的密码，为了安全，建议不指定**-p**选项，成功执行上述命令会在用户终端输出如下的信息。

```
private key: 000107deee0bdda4465d708fcd1a11d4c1eddc62561e3a897dc6e4c9e98471bde943
public key:
c798bdae2187ed4529fa6381c0c832e7a683425517c219c84910dcca0cba801575f8ab544198229e4a75a24141767d650e42a62dffdcc828fba9b25069
81d
```

> **说明** 在上述输出中，`private key` 标识私钥文件的私钥信息；`publickey` 标识私钥文件对应的公钥信息。输出信息为十六进制的字符串表达方式。

4.4.2.3. signature子命令

crypto_tool的**signature**子命令主要完成对指定数据的签名操作，生成对应的签名结果并输出。

signature子命令的操作可以应用到对交易哈希的签名，通过指定签名私钥文件和私钥密码来完成对私钥文件加载和私钥的解析，然后调用ECDSA签名算法完成对数据的签名。签名所使用的的操作算法根据所指定的密钥文件确定，当前支持的密钥类型有：Secp256k1, Secp256r1(Prime256v1), SM2, RSA2048。其命令执行方式如下所示。

```
cd /home/mychain/mychain-0.10.2.12/node/bin
./crypto_tool signature -d 0xb62f1aa433957586c65e6af6648f68d2d1abfacd53fba506f8614aab31a8831b -k
../../user/Tester001.key
```

在上述命令行中，各参数选项说明如下。

- **-d**参数用于指定需要签名的哈希数据，使用十六进制的格式输出，可以通过crypto_tool的**hash**子命令来计算获取。
- **-k**用于指定签名的私钥文件路径，可以通过crypto_tool的**keygen**子命令来生成。
- **-p**用于指定私钥文件的加密密码，是在生成私钥文件时所输入的密码。如果不指定**-p**则需要用户手动在命令行输入密码，这样更为安全。

成功执行上述命令会在用户终端输出如下的签名结果。

```
signature result:
bc62083ef3fc5489ffd4a800b0312373b0070b1beaba6a9465a00753e98fece066208810d5f3041f63751e5741c5debd17d7f2f9380da2e7f355dccb14
09a01
```

4.4.2.4. recover子命令

crypto_tool的**recover**子命令主要用来验证签名是否是一个合法的签名数据，如果是合法的签名数据则会恢复出签名私钥所对应的公钥信息。

> **重要** recover子命令仅用于secp256k1类型的密钥做出的签名。

recover子命令的执行方式如下所示。

```
cd /home/mychain/mychain-0.10.2.12/node/bin
./crypto_tool recover -d 0xb62f1aa433957586c65e6af6648f68d2d1abfacd53fba506f8614aab31a8831b -s
0x308ff758524e3636c6c039f089c139db167716d809a6072781739e79442eebf1353e0f3551c40cec1cbeb1bda2cca1e5f37044699f3b23345cebc7e7
90b0600
```

在上述命令中，**-d**参数用于指定签名的哈希数据，同signature子命令中-d参数；**-s**参数用于指定签名结果数据，同signature子命令的输出结果。成功执行上述命令会在用户终端输出如下信息。

```
Public key recovery succeeded, public key:
66215a05074f23763f60a162ccac59020a28d94c956aadb28a2dbe39565b56841891720e854b68ac68fe3143c3ac484f4c705e3886ce0651edb2fa3b7a
eaf
```

> **说明** 上述输出的公钥信息可以通过crypto_tool的**key**命令来解析私钥文件输出公钥进行对比。

4.4.2.5. cert子命令

crypto_tool的**cert**子命令主要用来完成对指定证书请求的签发，通过使用ca的私钥完成对请求信息的加载签名生成对应的证书文件，证书文件满足openssl x509的pem格式要求。

cert子命令的指令操作方式如下所示。

```
cd /home/mychain/mychain-0.10.2.12/node/bin
./crypto_tool cert -c ../../ca/ -i ca -o ../../ca/ -r ../certs/user.req -d 3650
```

在上述命令中，各参数选项说明如下：

- **-c**参数用于指定ca文件的存储目录。
- **-i**参数用于指定ca的标识，此标识需与生成ca时指定标识保持一致。
- **-o**参数用于指定输出的用户证书文件存储目录。

- **-r**用于指定证书请求文件的路径，证书请求文件的生成可以参考**req**子命令的操作说明。
- **-d**用于指定签发证书的有效期。
- **-p**用于指定ca私钥文件的加密密码，如果不指定则会要求在用户终端输入密码。

成功执行指令会在在用户的终端输出如下信息。

```
Success: create cert file ../../ca/user.crt
```

在成功完成上述命令操作后，会在指定的输出目录下生成账户证书文件user.crt。

4.4.2.6. req子命令

crypto_tool的req子命令主要用来生成账户的私钥文件和证书请求文件，req子命令会为用户随机生成一个私钥文件，通过指定用户信息和私钥文件对应的公钥信息来生成证书申请文件，可以应用于后续的证书签发过程。

req子命令的具体指操作方式如下所示。

```
cd /home/mychain/mychain-0.10.2.12/node/bin
./crypto_tool req -i user -s '/C=CN/ST=HZ/L=xihu/O=ali/OU=ant/CN=www.alipay.com/emailAddress=user@example.com' -o
../certs/ -p 123abc
```

> **重要** 上述参数中的**-s**参数指定的身份信息不能与创建ca时输入的身份信息相同，可以酌情修改邮件地址等信息。

在上述命令中，各参数选项的说明如下：

- **-i**参数用于指定账户的标识。
- **-s**参数用于指定证书用户的身份信息，具体格式参考openssl x509证书说明。
- **-o**参数用于指定生成的私钥文件和证书请求文件的存储目录。
- **-p**用于指定生成私钥文件的加密密码，如果不通过命令行参数指定，则会要求通过用户终端输入，请两次输入相同的密码并确认。

成功执行上述命令会在用户终端输出如下信息。

```
Success: create cert key : ../certs/user.key, and cert req file : ../certs/user.req
```

在完成上述命令的操作后，会在指定的输出目录生成两个文件，分别是代表用户私钥的文件user.key和代表用户证书申请请求的文件user.req。其中user.key被使用输入的密码加密，后续应用于交易的签名过程，请妥善保管。

4.4.2.7. keygen子命令

crypto_tool程序的**keygen**子命令主要用来生成一个账户私钥文件，该私钥文件可以应用于交易签名过程。

keygen子命令的执行方式如下所示。

```
cd /home/mychain/mychain-0.10.2.12/node/bin
./crypto_tool keygen -o ../certs/ -i user02 -t sm2
```

在上述命令中，各参数选项说明如下：

- -o参数用于指定生成的私钥文件的存储目录。
- -i参数用于指定生成的私钥标识，如果不指定则默认为 `key` 。
- -p用于指定生成私钥文件的加密密码，如果不通过命令行参数指定，则会要求通过用户终端输入，请两次输入相同的密码并确认。

  > **说明** 建议不使用-p参数，直接通过终端输入密码。

- -t用于指定生成的密钥类型，可选择的项有： `secp256k1(ecck1)/eccr1(secp256r1,prime256v1)/sm2/rsa2048` 。 不使用**-t**参数时，默认生成 `secp256k1` 类型的密钥。

成功执行上述命令会在用户终端输出如下信息。

```
Success: generate user key file : ../certs/user02.key
```

在完成上述命令的操作后，会在指定的输出目录生成用户私钥的文件user02.key，生成的user02.key文件被使用输入的密码加密，后续应用于交易的签名过程，请妥善保管。

> **重要** 密钥类型如下：
> - ecck1 = secp256k1
> - eccr1 = secp256r1 = prime256v1
> - sm2 = sm2
> - rsa2048 = rsa2048

4.4.2.8. decode子命令

crypto_tool程序的**decode**子命令主要用来解码虚拟机当中的输出结果。不同的虚拟机当中采用的编解码方式不同，decode子命令工具提供给用户解码虚拟机输出的工具，目前仅支持evm输出的相关解码。

decode子命令的执行方式如下所示。

```
cd /home/mychain/mychain-0.10.2.12/node/bin
./crypto_tool decode -h
decode command options:
  -h [ --help ]            Print this help message and exit
  -d [ --data ] arg        the contract data to decode
  -t [ --type ] arg        the type of contract code to decode
  -r [ --return_type ] arg the return data type for decode contract code
```

其中，**-d**选项是用来指定解码的数据；**-t**是用来指定解码的数据类型，当前只支持evm；**-r**是指解码的数据的返回值类型，其执行方式如下。

```
./crypto_tool decode -d
0000000000000000000000000000000000000000000000000000000000000000e0000000000000000000000000000000000000000000000000000000000000000
7e3000000000000000000000000000000000000000000000000000000000000001200000000000000000000000000000000000000000000000000000000000000
0d0598000000000000000000000000000000000000000000000000000000000001800000000000000000000000000000000000000000000000000000000000000
0000001c0000000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000
0000000000076d79636861696e0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000200000000000000000000000000000000000000000000000000000000000000007e2000000000000000000000000000000000000000000000000
0000000000000007e3000000000000000000000000000000000000000000000000000000000000000a626c6f636b636861696e0000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000002616e7400000000000000000000000000000000000000000000000
0000000000000000000000626c6f636b636861696e0000000000000000000000000000000000000000 -t evm -r
string,uint,uint[],uint256,bytes,bytes[],bool[],identity
The decode data is { "string" : "mychain" }, { "uint" : 2019 }, { "uint[]" : 2018, 2019,  }, { "uint256" : 362019620248
}, { "bytes" : "626c6f636b636861696e" }, { "bool[]" : 1, 1,  }, { "identity" :
"0000000000000000000000000000000000000000000000000000000000000001" },
```

5.MyChain节点运维

5.1. 节点起停

节点启停指的是根据需要来完成区块链节点程序的启动和停止操作。这种操作主要发生在节点异常退出、系统升级以及数据备份等特定场景下。

为了后续文档的描述方案，特做如下的系统假设。

MyChain节点程序被安装在节点Linux操作系统的MyChain用户根目录下，假设用户已按照标准版管控软件搭建了MyChain的环境，安装目录为：/data1/mychain_depoy/。

> **说明** 该文中所有的Linux指令操作均为使用MyChain用户登录节点Linux操作系统完成。

节点启动

执行如下的指令来完成节点mychain程序的启动。

```
cd /data1/mychain_depoy/bin
./mychain -d
```

执行mychain程序的启动后，mychain会自动切换至后台运行，在切换至后台运行之前，会通过标准输入来要求输入相关的密码，包括：P2P网络节点SSL私钥密码、客户端SSL接入私钥密码，需要依次输入这些密码，如果密码正确则mychain会切换至后台运行；如果密码错误则mychain程序会退出。

如果不想 mychain 切换至后台运行，则启动时不用加**-d**选项。

> **重要** 待输入的SSL私钥密码是在节点部署过程中设置的对应密码，在输入过程中因为取消了屏显，所以没有输入提示，请直接按照密码内容完成输入，输入完成后以回车键结束。

状态检查

状态检查是指通过相关操作来检查mychain节点程序的运行状态，用于快速的验证mychain节点程序是否正确启动并保持运行，是作为后台运行的mychain程序的有效check手段，可以通过如下的指令来完成。

```
cd /data1/mychain_depoy/bin./mychain --status
```

执行上述指令会输出mychain的运行状态信息，如果mychain节点程序正处于运行状态，则会在用户操作终端输出如下信息。

```
mychain is running, pid is 7186.
```

> **重要** 上述输出中的pid是mychain运行程序的进程标识，是一个每次启动会发生变化的数字。

如果mychain节点程序未能正确启动或者运行，执行状态检查操作会在用户操作终端输出如下信息。

```
mychain is not running.
```

节点停止

可以根据需要在适时情况下停止节点mychain程序，通过执行如下的指令来完成节点mychain的停止运行。

```
cd /data1/mychain_deploy/bin./mychain --stop
```

执行如上指令则mychain会完成当前的操作并安全退出，确保系统相关资源的有效释放。停止完成后可以通过状态检查操作来检查节点mychain程序的运行状态。

5.2. 系统监控

系统监控是指通过mychain提供的monitor系统来完成节点的运行时状态查看，这些运行时状态包括但不限于：网络拓扑结构、节点的相关运行状态、系统中的交易和网络延时、系统的吞吐量和处理性能、系统中的相关日志事件和告警等。

系统监控主要有如下三方面的作用：

- 通过适时的查看monitor监控系统来监视节点的正常运行状态
- 通过延时和吞吐量来来指导系统的扩容
- 通过日志和告警信息来完成异常的处理

下面例举了解常见的手动操作监控查询命令，长期监控数据获取与接口：

网络拓扑

当前系统合约网络成员列表
执行命令如下：

```
./mychain_tool node -t contract
```

示例输出如下：

```
Execute query contract node succuss :
    node id:                        88bbee402d654e23d584f39019c07259aae4a34a748772f6448d4cfbfc8e320f
    endPoints:                      [ 192.168.1.10 : 18230 ]
    domain name:
    public key:
874355b12cb52423b8e558678d38e1a222693cc79b401878d05cf07228c50ed8f76361a29eda2eddbd15225d466326cf2f61785814d02bf2600e07b1fe
e13
    node role:                      node role consensus
    node state:                     node state normal

    node id:                        58a9415ea57bf763aeeaa0a0d148229a86869b8e5eefa8d017977868bc1ce160
    endPoints:                      [ 192.168.1.11 : 18231 ]
    domain name:
    public key:
0971a963b1556b268319d0be5358499574717ce95fc3d5ce901bb1953abcfd609c0d6b8bc2adade6a85db037659b30eb44f50a0d41999a49776cee2ec0
a5c
    node role:                      node role consensus
    node state:                     node state normal

    node id:                        6cb019530eef6d9a0e2c8518da8ab8fccae31552e6b469f824ecc7d1aff65388
    endPoints:                      [ 192.168.1.12 : 18232 ]
    domain name:
    public key:
e9dbb87528dd81fd680809d920a2b4edc6849e17f91044decbf3f31971e8ad34b64f2409eb199fcb92f1b9973cc05a248c887279f9892649361959b514
2bc
    node role:                      node role consensus
    node state:                     node state normal

    node id:                        2952108cef43f8f31398e5ccc556fb3990ef27f2950524ca57c9592fab3e9a19
    endPoints:                      [ 192.168.1.13 : 18233 ]
    domain name:
    public key:
66e818a628d7ad9f0e24f755133cd49fa2cd69c653203cc809dd4ec4a6bdf61df6677f7df478bdd42abd013b665d72688e9915283c96243df76fa48671
745
    node role:                      node role consensus
    node state:                     node state normal
```

当前实际网络成员列表

- **查询节点本身**
  执行命令如下：

```
./mychain_tool node -t host
```

示例输出结果如下：

```
Execute host info query success :
    node id:                        88bbee402d654e23d584f39019c07259aae4a34a748772f6448d4cfbfc8e320f
    endPoints:                      [ 192.168.1.10 : 18230 ]
    domain name:
    public key:
874355b12cb52423b8e558678d38e1a222693cc79b401878d05cf07228c50ed8f76361a29eda2eddbd15225d466326cf2f61785814d02bf2600e07b1
78e13
    node role:                      node role consensus
    node state:                     node state normal
```

- **查询节点的对端节点**
  执行命令如下：

```
./mychain_tool node -t p2p
```

示例输出如下：

```
Execute node query succuss :
    node count:                     3
    consensus node:                 3
    non consensus node:             0
    connected consensus node:       3
    connected non consensus node:   0

    node status:                    [
                                            connected: true, cct: p2p; node id:
2952108cef43f8f31398e5ccc556fb3990ef27f2950524ca57c9592fab3e9a19; ([ 192.168.1.13 : 18233 ], 2) ,
                                            connected: true, cct: btn; node id:
58a9415ea57bf763aeeaa0a0d148229a86869b8e5eefa8d017977868bc1ce160; ([ 192.168.1.11 : 18231 ], 2) ,
                                            connected: true, cct: btn; node id:
6cb019530eef6d9a0e2c8518da8ab8fccae31552e6b469f824ecc7d1aff65388; ([ 192.168.1.12 : 18232 ], 2)
                                    ]
```

- **共识信息与参数**
  执行命令如下：

```
./mychain_tool consensus
```

示例输出如下：

```
Execute query consensus succuss :
    type:                           0
    phase:                          NORMAL
    N:                              4
    f:                              1
    Node Id:                        88bbee402d654e23d584f39019c07259aae4a34a748772f6448d4cfbfc8e320f
    Leader:                         58a9415ea57bf763aeeaa0a0d148229a86869b8e5eefa8d017977868bc1ce160
    View:                           1
    Last exec seq:                  19234
    Propose size:                   256
    Propose time interval:          5000
    Normal time interval:           3000
    Viewchange time interval:       3000
```

网络延时

通过网络拓扑获得节点IP信息，然后通过网络工具获得网络时延。

系统吞吐

命令行：

```
./mychain_tool metrics -t tps_latency -i 58a9415ea57bf763aeeaa0a0d148229a86869b8e5eefa8d017977868bc1ce160
```

示例结果：

```
Execute query tps and latency metrics succuss :
    granularity:                    1800000
    save duration:                  43200000
    start time:                     1574319600000
    type:                           METRIC_TPS
    counter:                        0
    type:                           METRIC_TX_LATENCY
    counter:                        0
```

5.3. 例行检查

5.3.1. 例行检查说明

例行检查指的是日常针对节点的巡查行为，用来快速的验证节点的正常运行，整个区块链网络未出现相关的异常和不可预期的错误。

例行的巡查可以根据系统的重要程度和IT管理策略酌情调整频率。

例行检查主要分两个方向进行检查。

- 后台环境检查
- 管控平台检查

5.3.2. 后台环境检查

检查mychain节点程序的运行状态是否正常

```
./mychain --status
```

使用启动mychain的用户执行此命令，看到pid的值说明mychain实例已启动。

```
ps -ef|grep 2839
```

使用启动mychain的用户执行此命令，看到pid对应的进程存在说明mychain正在运行。

```
./mychain_tool node -t host
```

使用启动mychain的用户执行此命令，看到node state是normal说明mychain运行状态正常。

检查mychain节点的网络连接状态是否正常

```
./mychain_tool node -t p2p
```

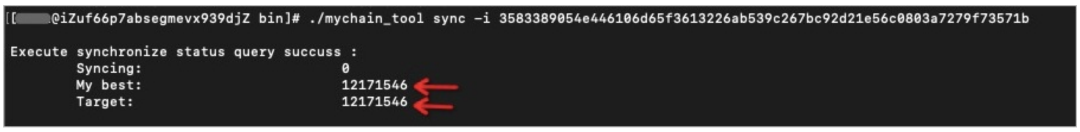
使用启动mychain的用户执行此命令，看到当前节点与其他节点连接状态均为true表示当前节点与其他节点的P2P连接正常。

```
./mychain_tool node -t p2p -i b7910fe525e80067f932a71c0d1c9080e7a2987a6abb83759be0c299eeff74ad
```

使用启动mychain的用户执行此命令，看到指定节点与其他节点连接状态均为true表示指定节点与其他节点的P2P连接正常，请依次检查所有节点与其他节点的P2P连接状态。

检查mychain节点的区块延迟状态是否正常

```
./mychain_tool sync -i 3583389054e446106d65f3613226ab539c267bc92d21e56c0803a7279f73571b
```



使用启动mychain的用户执行此命令，看到My best和Target差距在100以内，说明区块同步正常，请依次检查所有节点的区块同步状态。

检查mychain节点的共识状态是否正常

```
./mychain_tool consensus -i b7910fe525e80067f932a71c0d1c9080e7a2987a6abb83759be0c299eeff74ad
```

```
[    @iZuf66p7absegmevx939djZ bin]# ./mychain_tool consensus -i b7910fe525e80067f932a71c0d1c9080e7a2987a6abb83759be0c299eeff74ad

Execute query consensus succuss :
        type:                           0
        phase:                          NORMAL
        N:                              9
        f:                              2
        Node Id:                        b7910fe525e80067f932a71c0d1c9080e7a2987a6abb83759be0c299eeff74ad
        Leader:                         b7910fe525e80067f932a71c0d1c9080e7a2987a6abb83759be0c299eeff74ad
        View:                           7934
        Last exec seq:                  12171862
        Propose size:                   256
        Propose time interval:          5000
        Normal time interval:           3000
        Viewchange time interval:       3000
        Consensus proposal status:      0
```

使用启动mychain的用户执行此命令，看到phase状态为NORMAL表示当前节点共识状态正常，请依次检查所有节点的区块同步状态。

检查mychain节点的交易处理状态是否正常

```
./mychain_tool transaction_queue -i b7910fe525e80067f932a71c0d1c9080e7a2987a6abb83759be0c299eeff74ad
```

```
[    @iZuf66p7absegmevx939djZ bin]# ./mychain_tool transaction_queue -i b7910fe525e80067f932a71c0d1c9080e7a2987a6abb83759be0c299eeff74ad

Execute transaction cache query succuss :
        unverified queue limit:         20480
        unverified queue usage:         0
        verified queue limit:           40960
        verified queue usage:           0
```

使用启动mychain的用户执行此命令，看到未验证队列使用情况和已验证队列使用情况小于最大深度(limit对应的数值队列最大深度)，说明交易没有堆积，请依次检查所有节点的交易堆积情况。

```
./mychain_tool block -i last
```

```
[    @iZuf66p7absegmevx939djZ bin]# ./mychain_tool block -i last

Execute query last block succuss :
        hash:                           1af650d99fbb915d2cf23fc782d94c4758bf74dca7f2061e2bfe144ec35f0f66
        number:                         12172048
        version:                        1688854155231490
        parent hash:                    d1327a610f5e1f2a52450bfab528badfb7df1ea57387c9645938e5a6b53efb79
        transaction root:               0000000000000000000000000000000000000000000000000000000000000000
        receipt root:                   0000000000000000000000000000000000000000000000000000000000000000
        state root:                     eb739a54524d89576ae7c5d78facad665ae70a9b68f15563d6cda0f3a397e227
        gas used:                       0
        timestamp:                      1605496829976

        trasactions:                    [ ]
```

使用启动mychain的用户执行此命令，请每隔1s执行一次，多次执行，看到number所示的区块高度在逐步增长，说明出块正常，交易可以正常被打包进入区块。

检查mychain节点日志是否正常

```
tail -f mychain.log|grep GenerateNewBlock
```

使用启动mychain的用户执行此命令，看到区块不停产生表示正常，否则表示区块链已经停止出块，需要及时上报并联系产品提供方定位问题。

```
tail -f mychain.log | grep CheckSyncStatus
```

```
[root@iZuf66p7absegmevx939djZ log]# tail -f mychain.log | grep CheckSyncStatus
[2020-11-16 16:45:35:411] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178548, synchronize target block number: 12178548, nothing to sync
[2020-11-16 16:45:35:911] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178548, synchronize target block number: 12178548, nothing to sync
[2020-11-16 16:45:36:411] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178548, synchronize target block number: 12178548, nothing to sync
[2020-11-16 16:45:36:912] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178549, synchronize target block number: 12178549, nothing to sync
[2020-11-16 16:45:37:412] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178549, synchronize target block number: 12178549, nothing to sync
[2020-11-16 16:45:37:912] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178549, synchronize target block number: 12178549, nothing to sync
[2020-11-16 16:45:38:412] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178549, synchronize target block number: 12178549, nothing to sync
[2020-11-16 16:45:38:913] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178549, synchronize target block number: 12178549, nothing to sync
[2020-11-16 16:45:39:413] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178549, synchronize target block number: 12178549, nothing to sync
[2020-11-16 16:45:39:913] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178550, synchronize target block number: 12178550, nothing to sync
[2020-11-16 16:45:40:414] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178550, synchronize target block number: 12178550, nothing to sync
[2020-11-16 16:45:40:914] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178550, synchronize target block number: 12178550, nothing to sync
[2020-11-16 16:45:41:414] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178550, synchronize target block number: 12178550, nothing to sync
[2020-11-16 16:45:41:915] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178550, synchronize target block number: 12178550, nothing to sync
[2020-11-16 16:45:42:415] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178550, synchronize target block number: 12178550, nothing to sync
[2020-11-16 16:45:42:915] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178551, synchronize target block number: 12178551, nothing to sync
[2020-11-16 16:45:43:416] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178551, synchronize target block number: 12178551, nothing to sync
[2020-11-16 16:45:43:916] [p:16097/t:16302] [mychain:info] [synchronizer.cpp:463:CheckSyncStatus] My local best block number: 12178551, synchronize target block number: 12178551, nothing to sync
```

使用启动mychain的用户执行此命令，看到区块在同步表示正常，否则表示当前节点已经停止同步区块，需要及时上报并联系产品提供方定位问题。

检查mychain节点的物理资源是否正常

```
free
```

```
[root@iZuf66p7absegmevx939djZ ~]# free
              total        used        free      shared  buff/cache   available
Mem:       32780404     1714112      274224         496    30792068    30618576
Swap:             0           0           0
```

空闲内存低于4G时，需要及时上报并联系产品提供方调整应用部署方式或者更换更高配置服务器，或者直接在云服务器扩容。

```
df
```

```
[root@host136 /home/admin]
$df
Filesystem      1K-blocks      Used  Available Use% Mounted on
devtmpfs        131865224         0  131865224   0% /dev
tmpfs           131913552         0  131913552   0% /dev/shm
tmpfs           131913552      7156  131906396   1% /run
tmpfs           131913552         0  131913552   0% /sys/fs/cgroup
/dev/nvme0n1p1 1845816148  84377940 1667653172   5% /
/dev/sda1          999320    112004     818504  13% /boot
/dev/nvme1n1   1844891500 356833496 1394272900  21% /data1
```

磁盘使用率高于80%时，需要及时上报并发起对数据的归档处理或者升级扩容硬盘空间。

```
top
```

```
top - 16:06:00 up 440 days, 22:15,  1 user,  load average: 0.07, 0.10, 0.13
Tasks: 185 total,   1 running, 184 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.1 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 32780404 total,   263448 free,  1716272 used, 30800684 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 30616460 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1661 root      20   0  146160   2168   1444 R   0.3  0.0   0:00.03 top
    1 root      20   0  189596   4336   2080 S   0.0  0.0 107:35.96 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:26.51 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:17.78 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      rt   0       0      0      0 S   0.0  0.0   0:28.17 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S   0.0  0.0  86:15.50 rcu_sched
   10 root      rt   0       0      0      0 S   0.0  0.0   1:21.79 watchdog/0
   11 root      rt   0       0      0      0 S   0.0  0.0   1:11.96 watchdog/1
   12 root      rt   0       0      0      0 S   0.0  0.0   3:17.53 migration/1
   13 root      20   0       0      0      0 S   0.0  0.0   0:12.02 ksoftirqd/1
   15 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
   16 root      rt   0       0      0      0 S   0.0  0.0   1:01.16 watchdog/2
   17 root      rt   0       0      0      0 S   0.0  0.0   0:22.76 migration/2
```

CPU使用率高于80%时，需要及时上报并联系产品提供方调整应用部署方式或者更换更高配置服务器，或者直接在云服务器扩容。

```
date +%s && ./mychain_tool node -t node_timestamp
```

```
[root@iZuf66p7absegmevx939djZ bin]# date +%s && ./mychain_tool node -t node_timestamp
1605515160

Execute node query succuss:
        node timestamp:                 1605515160697
```

查看mychain节点的时间戳与系统时间戳是否一致，如出现偏差，应及时重启mychain节点。

检查mychain节点系统时间是否同步

```
date +%s && ./mychain_tool node -t node_timestamp
```

```
[root@iZuf66p7absegmevx939djZ bin]# date +%s && ./mychain_tool node -t node_timestamp
1605515160

Execute node query succuss:
        node timestamp:                 1605515160697
```

查看mychain节点的时间戳与系统时间戳是否一致，如出现偏差，应及时重启mychain节点。

5.3.3. 管控平台检查

查看节点列表

1. 登录管控平台之后按照如下两步检查区块链状态。

2. 单击**节点管理**，进入节点管理页面。

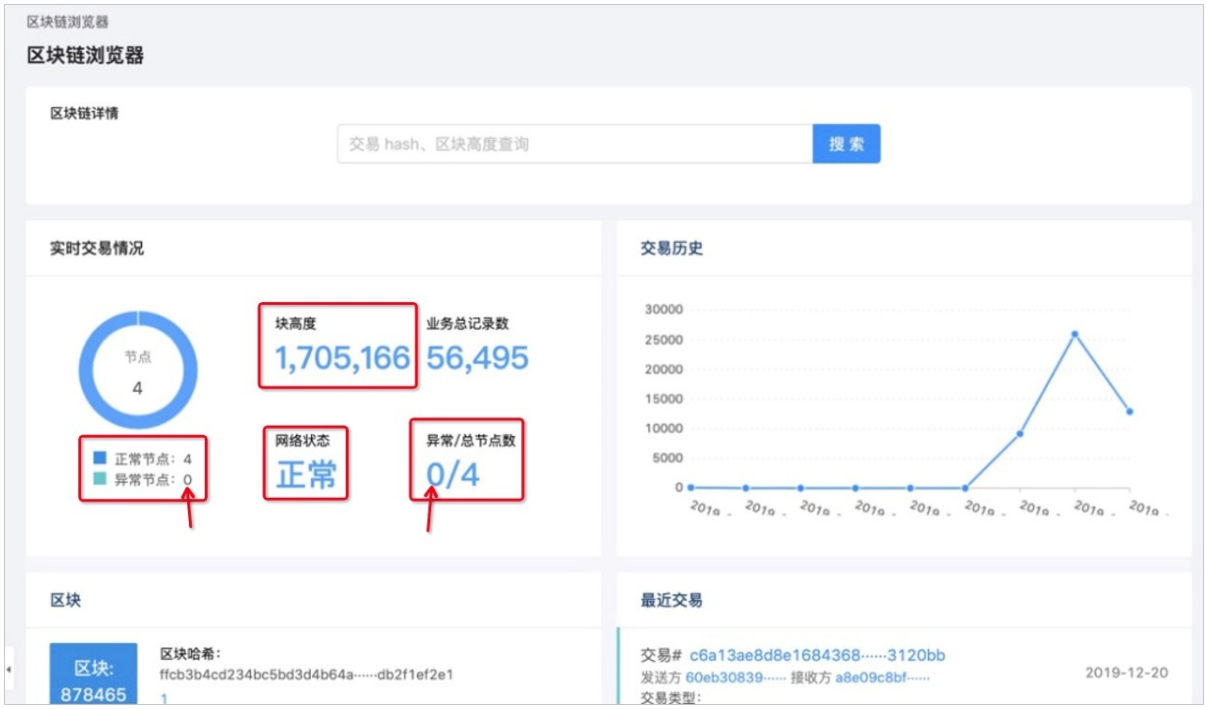
3. 在节点管理中，确认以下三处是否正常。
   - 所有节点运行状态均为运行中
   - 区块高度在不断增高
   - 告警信息处无最近的告警

   如上述三处存在问题，需要后台登录服务器，按照5.1章节给出的检查步骤检查该服务器是否正常，如有问题请联系产品提供方协助解决。

查看区块链浏览器

1. 登录管控平台之后按照如下两步检查区块链状态。

2. 单击**区块链浏览器**，进入**区块链浏览器**详情页面。



3. 在**区块链浏览器**中，确认以下三处是否正常。
   - 异常节点数量为0
   - 网络状态为正常
   - 区块高度在不断增高

   如上述三处存在问题，需要后台登录服务器，按照5.1章节给出的检查步骤检查该服务器是否正常，如有问题请联系产品提供方协助解决。

5.4. 备份与恢复

本节

数据备份是对系统数据的可靠性保证，通过数据备份可以防止节点程序的异常或者磁盘损坏导致的数据丢失，并在出现节点异常时可以通过数据恢复来确保数据的完整性和有效性。

节点上的区块链数据是可以通过节点同步机制来完成数据的恢复，但是节点相关密钥的私钥信息是没法恢复的，只能通过备份恢复机制来确保数据的可靠性。

数据备份

节点数据备份的主要内容包括：节点密钥信息（包括P2P连接私钥和证书、客户端接入私钥和证书、CA公钥证书等）、区块链数据库文件、节点可执行程序、系统配置文件等。

> **说明** 建议执行数据备份的时间间隔为1周，备份数据保留1个月，备份数据文件多目的地保留3份。

- 手动备份
  执行数据备份操作的相关指令如下。

```
cd /data1/mychain_deploy/
mkdir -p /data1/mychain_deploy/bakup >/dev/null 2>/dev/null
tar -zcvf /data1/mychain_deploy/bakup/mychain_data_20180904.tar.gz bin data certs conf
```

> **重要** 上述执行备份输出的压缩文件名中日期可根据实际情况调整为运行时的日期。

执行上述指令操作后，将打包压缩的文件mychain_data_20180904.tar.gz下载并备份至有效的目的地。

- crontab自动备份
  备份操作也可以通过配置Linux的**crontab**来自动完成数据的备份操作，通过将数据备份和下载备份文件等操作写入特定的SHELL脚本文件，然后将脚本文件配置到**crontab**中自动运行，参考的脚本文件样例如下。

```
#!/bin/bash

DATA=`date +%Y%m%d`

cd /data1/mychain_deploy
mkdir -p /data1/mychain_deploy/bakup >/dev/null 2>>/dev/null
tar -zcvf /data1/mychain_deploy/bakup/mychain_data_${DATA}.tar.gz bin data certs conf
```

> **说明** 上述脚本仅仅完成了备份数据的压缩打包过程，并未完成备份数据的目的地拷贝，需要自行完成。或者在脚本中通过ftp或者scp指令来完成。

例如按每周日00:00执行一次，配置的crontab如下。

```
0 0 * * 0 /home/mychain/data_bak.sh
```

> **重要** 可根据系统需要自行调整运行间隔，具体的脚本路径根据实际情况调整。

数据恢复

数据恢复是指在发生数据异常的情况下，通过备份数据来有效恢复历史数据，确保节点的正常持续运行。通过将最近的有效数据备份文件上传到节点的/home/mychain/mychain/bakup目录下，执行如下的指令来完成数据恢复过程。

```
cd /data1/mychain_deploy
rm -rf bin data certs conf
tar -zxvf /data1/mychain_deploy/bakup/mychain_data_20180904.tar.gz ./
```

> **重要** 上述备份文件中的日期根据实际情况调整。

执行完数据的恢复后，即可通过节点的启动操作来完成节点的运行，并通过状态检查或者monitor监控系统来检查mychain节点程序的运行状态。

5.5. 数据归档

本节给出数据归档的具体操作步骤。

操作步骤

1. 查询当前链上的系统合约值，确认当前的chain.min_query_block值。
   例如以下的例子中的**chain.min_query_block**的值为**0**。

   ```
   $./mychain_tool system_contract -q | grep chain.min_query_block
   chain.min_query_block   0
   ```

2. 查询当前最高区块高度。
   例如以下示例中，最高块为：31152。

   ```
   tail -f ../log/mychain.log | grep GenerateNewBlock

   [2020-10-23 16:12:29:974] [p:23066/t:23184] [mychain:info] [block_chain.cpp:695:GenerateNewBlock] [subnet:00000000] Ta
   rget Subnet:[00000000000000000000000000000000000000000], [opt]GenerateNewBlock success:
   [{"hash":"3ac98548513147f75425124c5c4b8d17d7683017404452f19f486c01c23f8cf4","version":"288230397660104450","number":"31
   ,"parent_hash":"d2bb6addab8d7f54b6e91b1824c6e24585f8729b757a2090fbd7bce7f587fea9","transaction_root":"0000000000000000000
   0000000000000000000000000000000000000000000000000","receipt_root":"0000000000000000000000000000000000000000000000000000000000000000000
   00","state_root":"578fa6e5574f1dd5ab92b3f021c71b39ab9bb32eb660fdb1b897858700b59b03","gas_used":"0","timestamp":"1603440
   23","log_bloom":"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"}]  total tx [0]
   cache miss [0] cache total [0] cache cnt[0] cache size [0]
   ```

3. 修改链上的chain.min_query_block值为适当的值。
   示例中定为31145。

   ```
   ./mychain_tool system_contract -k chain.min_query_block -v 31145
   ```

4. 利用数据库快照工具创建一个数据库快照。

   ```
   ./mychain_tool dump -t all -p test_dump/
   Sending dump db to mychain completes! Please check dump path.
   ```

   > **重要** 快照的位置为当前节点存储路径的相对路径。上例子中该快照的路径为: ../data/test_dump/dump。

5. 归档。

   ```
   ./archive_tool -d ../data/test_dump/ --storage-name dump --online-storage-name public
   INFO: I do not see any MIN_BLOCK_NUMBER in DB, seem to be the first time archive attempt for this storage; take 0 as
   my assumption.
   Blocks [0, 31142) will be archived.
   Estimated time: 3 seconds.
   Confirm to execute the archive? [yes/no]: yes
   Archive started.
   Migrate all headers.
   Archive finished.
   ```

6. 通过瞬时重启命令完成节点重启，同时切换到刚刚创建的数据快照。

```
./mychain --restart -d
```

7. 转存快照之前的数据,此时之前的数据目录成为静态文件目录,可以直接拷贝备份。

5.6. 数据清理

mychain节点在运行过程中，会产生一些冗余的数据文件，包含：历史日志内容、数据库文件、备份文件。随着运行时间的持续，会在节点的磁盘空间产生一些历史数据，占用一定的磁盘空间，可以根据需要适时的进行数据清理，防止磁盘空间被消耗殆尽。

日志文件清理

日志文件的清理策略是：每天清理一次，删除超过7天的日志文件。

执行的操作指令如下：

```
find /data1/mychain_deploy/log/ -type f -mtime +7 -exec rm -f {} \;
```

> **说明** 上述指令中的路径需要根据实际情况调整。指令中+7表示删除7天之前的文件，可根据需要酌情修改。

可以通过配置Linux的crontab来自动执行删除任务，具体的配置样例如下。

```
0 0 * * * "find /data1/mychain_deploy/log/ -type f -mtime +7 -exec rm -f {} \;"
```

备份文件清理

出于数据可靠性和安全考虑，系统需要定期的备份相关文件。这些备份文件也需要定期的清理以释放磁盘空间。备份文件的清理策略是：每天清理一次，删除超过1个月的备份文件。执行的操作指令如下。

```
find /data1/mychain_deploy/bakup/ -type f -mtime +30 -exec rm -f {} \;
```

> **重要** 上述指令中的路径需要根据实际情况调整。指令中+30表示删除30天之前的文件，可根据需要酌情修改。

可以通过配置Linux的crontab来自动执行删除任务，具体的配置样例如下。

```
0 0 * * * "find /data1/mychain_deploy/bakup/ -type f -mtime +30 -exec rm -f {} \;"
```

数据库清理

mychain节点程序在运行过程中会不断的产生和记录区块数据，为了保证系统的活性和部分时间上的精度要求，系统会定期的产生一些空的区块数据，这些空的区块数据不包含任何实际有效的交易信息，可以按照一定的要求进行数据清理，以压缩数据库文件的大小，降低磁盘空间占用，提升信息检索效率。

数据库清理操作并不是一个强制策略，可以根据需要和磁盘空间依赖来进行策略调整，如果磁盘空间相对较大，或者业务需求上的依赖可以选择不清理。数据库清理的策略是：每周清理一次，删除超过一定时间的空区块信息。具体的操作指令如下。

```
/data1/mychain_deploy/bin/mychain_cleaner -t 30 -d /data1/mychain_deploy/data/
```

> **重要** 上述指令中的30代表清理30天以前的空区块信息，后续的目录代表数据库文件的存储目录，视实际情况调整。

可以通过配置Linux的crontab来自动执行删除任务，具体的配置样例如下。

```
0 0 * * 0 "/data1/mychain_deploy/bin/mychain_cleaner -t 30 -d /data1/mychain_deploy/data/"
```

5.7. 日志说明

5.7.1. 日志配置

日志输出是mychain系统中定位问题和查看系统运行的有效手段，mychain节点程序通过使用相关的日志组件来记录系统的运行日志，系统通过配置文件来配置日志的输出格式和日志输出路径，通过日志级别来控制输出日志的详细程度。

在mychain的节点安装部署目录下，conf/log.conf配置文件用来控制日志内容的相关输出信息。如下是一个有效日志配置文件的参考样例。

```
{
    "mychain" : {
        "filename" : "../log/mychain.log",
        "max_file_size" : 104857600,
        "max_files" : 100,
        "level" : "debug"
    },
    "vm" : {
        "filename" : "../log/vm.log",
        "max_file_size" : 104857600,
        "max_files" : 100,
        "level" : "info"
    },
    "fdmt" : {
        "filename" : "../log/fdmt.log",
        "max_file_size" : 104857600,
        "max_files" : 100,
        "level" : "info"
    },
    "mywasm" : {
        "filename" : "../log/mywasm.log",
        "max_file_size" : 104857600,
        "max_files" : 100,
        "level" : "info"
    }
}
```

上述日志配置文件中主要配置了两个日志输出选项，也是mychain节点程序运行所必须依赖的两个日志对象：mychain、vm，分别用来输出mychain节点运行日志和智能合约运行日志。在上述日志配置文件中，核心的配置主要包含如下几项：上述日志配置文件中主要配置了两个日志输出选项，也是mychain节点程序运行所必须依赖的两个日志对象：mychain、vm、mywasm，分别用来输出mychain节点运行日志和智能合约运行日志。

在上述日志配置文件中，核心的配置主要包含如下几项：

- filename：日志文件输出的路径，可以是绝对路径，也可以是相对路径（相对于mychain安装目录下的bin目录）。
- max_file_size：用于控制每个日志输出文件的最大文件大小，单位为字节。
- max_files：用于备份的日志文件数量，每当记录的日志文件大小达到MaxFileSize时则会备份当前文件，重新记录新的文件，该选项配置最多备份的文件数量。
- level：用于设置日志输出的级别，日志级别候选为trace、debug、info、warn、error、critical。

5.7.2. 日志格式

本节介绍如何通过配置文件来配置日志的输出格式。

基于日志配置，mychain节点程序会在日志文件中按照如下的格式输出日志内容。

```
[YYYY-MM-DD HH-MI-SS] [p:pid/t:tid] [mychain:Level] [File:Line:Function] Log Data
```

上述日志格式的每个部分内容含义如下：

- YYYY-MM-DD：记录日志的日期，YYYY代表年份，MM代表月份，DD代表日期
- HH-MI-SS：记录日志的时间，HH代表24小时制小时，MI代表分钟，SS代表秒
- pid：mychain节点程序的进程编号
- tid：运行于mychain中的记录日志的线程编号
- Level：记录日志的级别，该级别包含6种可能的取值：trace、debug、info、warning、error、critical
- File：代表记录日志的源文件名
- Line：代表记录日志在源文件中的代码行号
- Function：代表记录日志的源代码函数名称
- Log Data：具体的日志内容

按照日志配置，如下是一个输出的日志内容样例。

```
[2018-08-30 17:21:27:632] [p:563/t:563] [mychain:info ] [event_plugin.cpp:21:Initialize] EventPlugin initialize successf
ully...
[2018-08-30 17:21:27:683] [p:563/t:563] [mychain:info ] [vm_plugin.cpp:23:Initialize] VMPlugin initialize successfully..
.
[2018-08-30 17:21:28:486] [p:563/t:563] [mychain:debug] [block_chain.cpp:889:ConfigureDatabase] ConfigureDatabase succes
s!
```

5.7.3. 级别调整

mychain节点程序默认的日志级别为Info级别，意味着输出日志的最低级别为Info级别，低于Info级别的Trace和Debug级别的日志不会被输出到日志文件中。可以通过mychain节点程序的工具来查看和调整日志级别。

日志级别查询

可以通过如下操作来查询mychain节点程序的日志级别。

```
cd /data1/mychain_deploy/bin/
./mychain_tool
> log
```

> **重要** 执行上述指令，要求mychain节点程序必须处于运行状态，并且相关配置文件必须配置正确。

执行上述指令，会在用户的终端输出mychain节点程序的当前日志级别，输出样例如下。

```
Execute query log level succuss:
    log level:                    debug
```

日志级别调整

可以通过如下操作来调整mychain节点程序的运行日志级别。

```
cd /data1/mychain_deploy/bin/
./mychain_tool
> log -l [ --level ] select [0-5] log level: trace-0, debug-1, info-2, warning-3, error-4, fatal-5
```

> **重要** 执行上述指令，要求mychain节点程序必须处于运行状态，并且相关配置文件必须配置正确。

执行上述指令，会在用户的终端输出日志级别调整的结果，输出样例如下。

```
Execute set log level succuss!
```

成功调整完日志级别后，mychain节点程序则会按照新的日志级别来输出相关日志。

> **说明** 关于调整日志级别的操作还可以参考mychian_tool中log子命令的内容。

5.8. 常见问题处理

5.8.1. 维修处理方法和流程

本节主要给出在遇到一体机硬件故障时，如何执行维修处理流程。

蚂蚁链一体机是蚂蚁自主打造，针对区块链技术特点，定制融合软硬件技术，集区块链平台、中间件和场景服务于一体的区块链基础平台产品，为用户提供高性能、强隐私、高安全的区块链节点基础设施。用户开箱即用，降低启动成本，加速场景落地。

一体机器是在成熟的服务器的基础上整合自定义硬件/软件而来,所以其维修涉及到两种故障：

- 基础故障, 如电源/内存/主板/机箱等基础部件的故障。
- 与一体机定制硬件相关的故障。

一体机的外观图如下所示。

发生故障后处理流程由蚂蚁整体对外，硬件供应商在后端支撑蚂蚁团队，遇到硬件固件升级、漏洞升级，厂商可主动识别，报备蚂蚁确认升级窗口。

5.8.2. 如何处理数据盘容量不足

由于区块链本身技术特点，节点的账本容量在链的生存周期中会一直增长. 当数据盘剩余容量不足时，需要及时运维，进行归档或者扩容。

背景信息

扩容和归档是应对剩余容量不足的两种方法，决定扩容还是归档主要依据是：

- 当前业务中，历史区块会不会经常被访问
- 当前部署中，单节点是否可以容允相当长时间的停服运维

如果上述答案是否，应该选择归档。

两种方法对比如下表所示。

内容	扩容	归档
区分冷热数据	否，所有账本数据在同一个数据盘	是，不常访问的账本数据半离线存储
运维操作复杂度	简单	中等
停服成本	单个节点停止一段时间，链和业务不需要停	单个节点短暂停切换到新的存储点，链和业务不需要停
新增硬件成本	需购买新数据盘替换旧数据盘	将冷数据备份到半离线存储设备上，涉及成本较低
对运维后的合约约束	无影响	归档之后需要对合约能查询的区块范围进行约束

扩容操作方法

对比两种方式后，如果选用扩容方式，步骤如下。

1. 确定扩容容量，完成新数据盘采购/格式化. 新的数据盘的文件系统应该和被更换的数据盘保持一致。
2. 通知相关人员，做好应急措施。
3. 停止节点运行，包括蚂蚁链和相关服务(如节点监控服务)，断开节点与链的网络连接。
4. 将旧的数据盘内容拷贝到新的数据盘上。
5. 关机，硬件运维人员完成数据盘更换后开机。
6. 不联网的情况下，启动蚂蚁链和相关服务，检查单节点运行情况。
7. 开启节点与链的网络连接，全链状态检查，开始同步区块。
8. 通知相关人员，扩容完成。
9. 更换下来的旧的数据盘，需要妥善保存一段时间以防硬件变更风险

> **说明** 此后如果应用于其他用途，务必对原有数据进行清除以防数据泄露。

5.8.3. 区块链密码卡硬件故障处理

如果密码卡发生故障，可联系整机售后支持部门进行处理。支持部门会针对密码卡发生故障种类的不同，给出不同的处理方案。

更换区块链密码卡

根据故障种类不同，支持部门可能对损坏的卡进行维修或者更换。如果维修成功，则只需在硬件上重新安装即可，在遵循共识容错规则的情况下对整条链和业务无影响。如果需要更换新的密码卡，则除了硬件运维之外还需要执行以下操作。

1. 检查共识密钥是否被之前的密码卡托管，如是则需要更新节点，具体操作参见更新节点。
2. 检查TLS密钥是否被之前的密码卡托管，如是则需要重新节点TLS密钥和重新签发节点证书，具体操作参见更新节点TLS密钥和重新签发证书。

5.8.4. 更新节点

当区块链密码卡需要更换时，如果共识密钥被之前的密码卡托管，则需要更新节点。本节给出更新节点的操作步骤。

1. 生成新的节点共识密钥。

   ```
   $ /data1/application_package/deps/antcard/scripts/generate_key.sh ${passwd} SM2 hw > server_l7.key
   ```

   运维使用交付软件包中提供的脚本工具获取该密钥的公钥用于节点更新交易。

   ```
   ./crypto_tool key -k server_l7.key -p ${passwd}
   ```

2. 发送节点更新交易。
   节点更新交易配置如下。

```
"CallContract" : {
    "enable" : true,
    "local" : false,
    "from" : "Administrator",
    "to" : "Node",
    "gas" : 1000000,
    "nonce" : 0,
    "timestamp" : 0,
    "period" :0,
    "value" : 0,
    "contract_type" : "evm",
    "method" : "UpdateNode(bytes32,uint32[],uint32[],bytes,uint8,uint8)",
    "parameters" : [
        "0x5a78c275127a52eebab3d8aceb0078136b1b9e437478e107b4693f3a3afb7726",
        "2130706433",
        "18330",
"0x20ebdddba332117ae24d572808cae432acc15cf93d6e9647e5ac977027db0d6490bab024784d461003f7064612c4ec0477e0e4d43db252020bcf2
4fe100c2",
        "1",
        "1"
    ],
    "block_num" : 0
}
```

以上交易中，`UpdateNode(bytes32,uint32[],uint32[],bytes,uint8,uint8)` 中定义的6个参数说明如下：

- 第一个参数是bytes32类型，指的是节点的ID。
- 第二个参数是uint32[]类型，指的是节点IP的点分十进制数组。
- 第三个参数是uint32[]类型，指的是节点端口号的数组。

  > **说明** 这里的端口号指的是p2p端口号，需要与被更新节点的mychain.conf修改的端口号保持一致。

- 第四个参数是bytes类型，指的是节点的公钥。
- 第五个参数是uint8类型，指的是节点的类型，节点类型主要包括 `{Unknow，Consensus， Unconsensus}` ，其值的范围为[0,2]。
- 第六个参数是uint8类型，指的是节点的状态，节点状态包括 `{Unknown, UnActived, Normal, Deleted}` ，值的范围为[0,3]

该交易需要使用Administrator来发送交易，所以需要修改node_config.json文件，修改结果如下所示。

```
{
    "node_config" : {
        "identity" : "Administrator",
        "key_path" : "../../user",
        "user_keys" : [
            {
                "key_name" : "Administrator1.key",
                "key_passwd" : "123abc"
            },
            {
                "key_name" : "Administrator2.key",
                "key_passwd" : "123abc"
            },
            {
                "key_name" : "Administrator3.key",
                "key_passwd" : "123abc"
            }
        ],
        "node" : {
            "endpoints" : [
                "127.0.0.1:10016"
            ],
            "ca_path" : "../certs/ca.crt",
            "cert_path" : "../certs/client.crt",
            "key_path" : "../certs/client.key",
            "passwd" : "123abc"
        },
        "crypto_suite" : "classic"
    }
}
```

3. 发送该交易。
   发送成功后，可以收到类似如下所示的结果。

```
Call contract transaction receipt: [result: 0, gas_used: 22620, output:
0000000000000000000000000000000000000000000000000000000000000000, log_entry: [{from:
e7d3e769f3f593dadcb8634cc5b09fc90dd3a61c4a06a79cb0923662fe6fae6b, to:
ca598a384aec7bc667e208b55d15059f2e1444b645d3d162a19391eae28aed3f, log_data: , topics: [call_contract]}]]
```

4. 查看**result**和**output**的值，如果**result**与**output**值均为**0**，则代表交易成功，否则交易失败。

> **说明** 确定节点更新交易正确执行后，则可以启动该节点。从链的角度来看，以上操作实质上是删除了之前的节点，新增加了一个节点，该变化对上层业务无任何影响。

5.8.5. 更新节点TLS密钥和重新签发证书

当区块链密码卡需要更换时，如果TLS密钥是否被之前的密码卡托管，则需要更新节点TLS密钥和重新签发证书。本节给出了更新节点TLS密钥和重新签发证书方法。

操作步骤

1. 生成节点TLS密钥。

```
$ /data1/application_package/deps/antcard/scripts/generate_key.sh ${passwd} SM2 hw > server.key
```

2. 生成签名请求。

```
$ /data1/application_package/deps/antcard/scripts/generate_csr.sh server.key ${passwd} nodeId@example.com SM2 HW >
server.csr
```

3. 重新签发节点证书。

```
$ /data1/Babassl-install/bin/openssl x509 -req -in ./server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -CAserial
serial.txt -passin pass:123abc -days 3650 -out server.crt
```

> **说明** ca.key要使用和其他节点一致的密钥，否则该节点无法与链建立连接，至此可以启动该节点。重新生成TLS密钥和重新签发证书仅影响网络层连接，一旦建立连接，该节点在链上的角色未发生变化，对上层业务无任何影响

5.9. 监控能力输出

本节描述蚂蚁链私有化标准版提供的基础监控能力，分为标准版管控和蚂蚁链智能合约平台两部分。对于用户具备集成基础子系统能力的条件下可选配进行进程，形成完整监控链路。

管控浏览器开放接口

查询区块链大盘数据 – queryDashboardInfo

请求参数

参数名	类型	必需	说明
bizId	String	是	链编号

响应参数

参数名	类型	必需	说明
code	String	是	响应状态码 • 成功：AE0000000000 • 失败：非AE0000000000
message	String	否	失败时的响应描述
data	DashboradInfoResponse	否	区块链大盘数据实体，见下表

DashboradInfoResponse

参数名	类型	必需	说明
networdStatus	boolean	是	区块链网络状态
blockCount	long	是	区块总数
createTime	Date	是	建链时间
nodeNum	int	是	节点总数
normalNodeNum	int	是	正常节点数量

abnormalNodeNum	int	是	异常节点数量
transactionCount	long	是	当前上链总数据了（业务总记录数）
historyInfos	List<HistoryInfo>	是	链上近七日数据统计列表，见下表

HistoryInfo

参数名	类型	必需	说明
bizId	String	是	链编号
item	String	是	统计类型，固定值 - transactions
value	Long	是	当日交易数
period	Date	是	统计日期

示例

```
curl http://localhost:8089/api/handy/queryDashboardInfo?bizId=4c05c445
{"code":"AE0000000000","message":null,"id":null,"data":
{"networkStatus":true,"blockCount":15319,"createTime":1596512838000,"nodeNum":4,"normalNodeNum":4,"abnormalNodeNum":0,"tra
ctionCount":1,"historyInfos":
[{"id":630647,"bizId":"M200804114786","item":"transactions","value":1,"period":1596412800000,"gmtCreate":1596514062000,"gm
dified":1596514062000}]}}
```

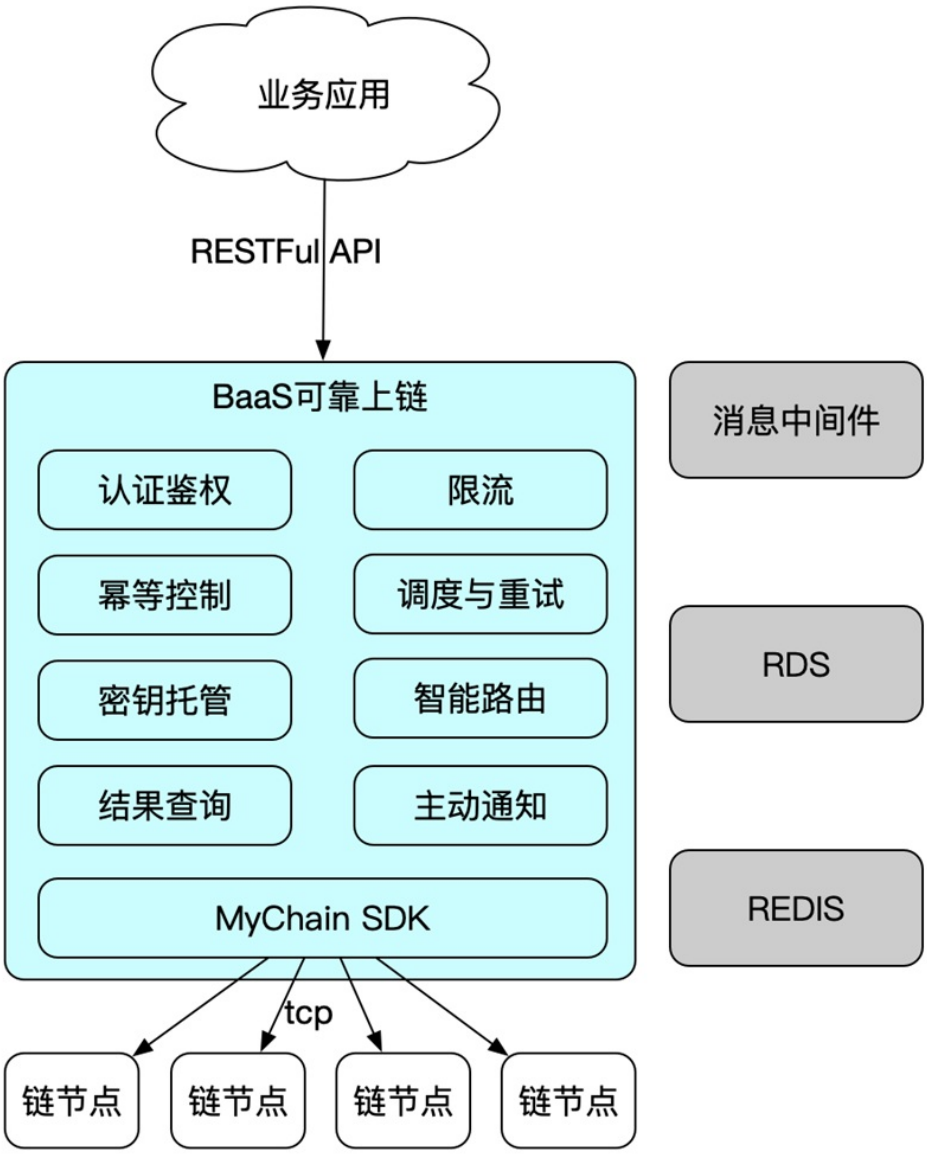
6.REST服务运维

6.1. REST服务概述

REST是蚂蚁链BaaS平台官方提供的接入服务，目标是为上层区块链应用提供易用、高效、可靠的统一区块链接入服务。

REST提供合约管理、账户管理、合约交易、存证交易、区块查询等蚂蚁链常用交互功能，以及MYTF链下安全计算应用管理和应用执行操作功能，同时集成和利用蚂蚁生态中的区块链基础服务设施和核心技术能力，包括KMS和TEE，提供多种灵活的区块链访问方式，同步/异步调用，外部/内部签名，普通/私密交易。

以下是BaaS REST 可靠上链的完整的功能示意图。



REST 接入服务的具体优势如下：

- 标准化接口：跨编程语言标准的RESTful API接口。
- 灵活性编程：支持同步调用、异步调用、主动轮询、异步消息、托管代签等模式。
- 可靠性服务：幂等控制、可靠上链、可靠推送、异常重试、交易数据持久化，智能路由能够选择最优节点来发起上链交易。
- 稳定性保障：限流降级、细粒度监控告警。
- 安全性护航：AK/SK及访问令牌、TEE链加密交易。
- 高性能支撑：结合数据分片、水平扩容、交易缓冲等技术，具备日均10亿。

6.2. 实例调整与性能优化

随着业务发展，业务量增加，可以新增服务器，进行BaaS REST的部署，并挂载到负载均衡中，从而具备对外提供服务的能力。

扩容时，可以从既有服务器中下载相关安装文件和配置文件，进行部署启动。

6.3. 日志查看与问题排查

1. `<baasrest安装目录>/logs/restgateway/rest-default.log`

   rest运行核心日志，排查问题重要的日志文件。

2. `<baasrest安装目录>/logs/restgateway/rest-error.log`

   rest运行过程中的一些error的日志，排查问题快速的日志文件（如果没有信息，可以看上面 `rest-default.log` 的内容）。

   如：业务处理，提供了<交易hash>，可以进入 `<baasrest安装目录>/logs/restgateway/` 目录，根据交易hash进行定位排查，执行命令如下：

   ```
   grep -C 10 <交易hash> *.log
   ```

3. 监控配置。

   i. 调用情况配置。

   调用情况可以通过采集目录： `<baasrest安装目录>/logs/goc/goc.log` 。

   日志格式：

   ```
   时间戳,traceId,spanId,方法,类型(web),链ID,业务Method,状态,执行线程,总用时,访问的AK,请求URL,消息
   ```

   其中traceId、spanId、类型(web)可以忽略，重要的是：链ID、Method、状态、总用时、访问的AK。

   状态针对5XX的为异常的，2XX的一般属于正常（客户问题导致token验证失败为202，系统运行稳定后，请从客户端检查原因，可以看看机器时间戳是否相差过大），示例如下。

   ```
   2022-12-06 17:19:03.344,ac1100011670318343340239320107,0,chainCallForBiz,web,M220907154730,ADDACCESSKEY,202,[http-nio-8088-exec-2],4,baas_admin,/chainCallForBiz/0f9115a7-09b7-4ec1-8bf9-b7c6ad31f850,chain cost:3
   2022-12-06 17:19:04.027,ac1100011670318343855239520107,0,chainCallForBiz,web,M220907154730,ADDACCESSKEY,200,[http-nio-8088-exec-8],172,baas_admin,/chainCallForBiz/0f9115a7-09b7-4ec1-8bf9-b7c6ad31f850,chain cost:171
   ```

   ii. 异常配置

   可以监控日志文件 `<baasrest安装目录>/logs/restgateway/rest-error.log` 中的异常数量。

6.4. 服务起停

- 应用服务停服

  进入<baasrest安装目录>/bin目录，执行 `sh stop.sh` 。

- 应用服务启动

  进入<baasrest安装目录>/bin目录，执行 `sh start.sh` 。

6.5. 备份与恢复

BaaS REST从2.2.0版本之后的数据都建议存在数据库中，相关备份与恢复按照MySQL的备份与恢复方式即可。

如果链访问数据使用的是本地磁盘保存方式，而非数据库存储方式，可以对相关目录下的文件进行备份。

目录位置，请查看 `<baasrest安装目录>/conf/baasrest.config` 中 `chain.rootpath` 的配置。

6.6. 数据清理

日志存放路径： `<baasrest安装目录>/logs` 。

该部分日志一般情况下无需清理，已经按照日期进行拆分，且有历史版本数量。磁盘紧张时，可以进入进行历史版本的清除。

6.7. 常见问题处理

客户端访问rest握手失败，无法获取token，返回码：204

1. 查看客户端与服务端的主机时间戳是否一致（相差不能太大），重要是检查服务器是否进行了时间同步。如果有问题，进行调整。

   baasrest日志中有关键字：overtime signature timestamp。

2. 客户端配置错误： 客户端签名的ak、私钥是否正确，如果不确定私钥是否正确，可以尝试重置后，更新客户端的私钥信息进行重试。

   baasrest日志中有关键字：fail to check signature for ak。

3. 客户端签名处理错误：使用Java Java SDK的一般不会出现这个问题，但是非java语言的，很有可能是这个原因导致握手失败。

   您也可以执行以下命令确认一下ak是否存在，对应的状态是否正常，存在结果，且status为normal则表示ak存在且状态正常。

   ```
   Select * from user_private_key where access_id = <客户端使用的ak>
   ```

客户端访问业务出现202导致无法完成业务，返回码：202

1. 确认请求是否存在access token。
2. 确认token是否存在，可以看rest日志，看token相关的处理。

客户端无权限访问，报错 209、210

检查客户端ak与链ID的关联关系是否存在，且状态正常。

```
select * from accessid_bizid_map where access_id = '<客户端AK>' and biz_id = '<链ID>';
```

- 如果有数据，且状态为normal，则表示正常。
- 如果有数据，状态不为normal，业务确认无误，可以调整状态为normal。
- 如果没有数据，说明该AK不能访问该链。

能连接REST，并获取TOKEN，但是上链失败（存证、创建账户、部署、调用合约等）

- 如果是非托管账户，详细查看交易回执信息进行排查。
- 如果是托管账户，检查托管账户是否存在，托管账户以及对应的租户、keyId是否匹配，不匹配需要客户端进行调整。

```
-- 检查账户是否存在
select * from tenantid_account_map WHERE biz_id = '<链Id>' and account_name = '<账户名称>' and tenantid = '<租户ID>';

-- 检查ak关系是否匹配
select * from kms_entity WHERE baccount = '<账户名称>' and tenantid = '<租户ID>' and keyid = '<keyId>';
```

能连接REST，并获取TOKEN，调用出现415错误

1. 检查链节点是否正常，是否正常出块。如果不正常，解决链节点的问题，过一阵子后重试。
2. 如果链节点正常，进入BaaS REST的服务器，检查REST和当前链是否建立连接：`netstat -an|grep 链节点IP`，如：`netstat -an|grep 10.20.13.12`。

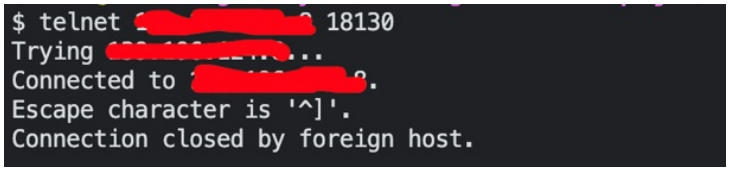
   如果有结果信息，则表示建立了连接，没有则说明和链没有建立连接，需要排查链节点建立连接失败的问题。

BaaS REST和链连接建立失败

rest-error.log中有以下日志，则表示建连失败，或者通过`netstat -an|grep 172.28.41.20`没有结果信息，也表示建连失败。

```
2023-06-12 15:20:50.740 [main] WARN  c.a.m.rest.core.mychain.BlockChainClientFactory - [connectfailedcloseclient]
connect failed and start to close mychain client:172.28.41.20 18130
2023-06-12 15:20:50.740 [main] ERROR c.a.m.rest.core.mychain.BlockChainClientFactory - [blockchainclientfactory]
blockchain M230316162857 ip 172.28.41.20 port 18130 initialize failed,try next time
```

1. 看客户端连接链节点的连通性： telnet 链节点IP 18130 看是否OK（下面则表示网络连通性没有问题）



2. 检查链配置
   i. blockchain_config表中的数据
   ii. keys_and_other_data 中该链的相关数据

   如果**blockchain_config表中的数据**不正常，则进行相关处理。如果**keys_and_other_data 中该链的相关数据** 缺失，则需要先将的**blockchain_config表中的数据**进行删除，通过中途岛重新发起rest开通，看看是什么问题导致。

   特别是对于国密链，需要确认**blockchain_config**表中的**prop_value的数据**，是否包含：**"smTLSSupport":true，如果没有，需要加上该配置。**
3. 清除REST日志，重启REST，如果依然有问题，查看相关的日志

   restgateway/rest-default.log;

   restcore/rest-error.log;

   restcore/mychain-sdk.log

7.数据服务运维

查看 baas-data-gateway 日志

1. 查看数据服务启动脚本 `start.sh` 中配置的根日志路径 `LOG_PATH` ；一般建议 `/root/datagw/logs` 。
2. 进入 baas-data-gateway 的日志路径，如进入 `/root/datagw/logs/baas-data-gateway` 目录。
3. common-default.log 为数据服务后端日志，common-error.log 为错误日志。
4. 重新进入配置的日志路径，查看启动脚本的 std 日志。如 `/root/datagw/logs/gateway-std.log` 。

查看 baas-csi 日志

1. 查看数据服务启动脚本 `start.sh` 中配置的根日志路径 `LOG_PATH` ；一般建议 `/root/datagw/logs` 。
2. 进入根日志路径，如进入 `/root/datagw/logs` 目录。
3. `csi-std.log` 为 csi 启动脚本的 std 日志， `csi.log` 为 csi 后端日志， `csi-trace.log` 为 csi 外部请求日志， `csi-trigger.log` 为 csi 事件触发日志。

重启数据服务

1. 确认数据服务启动脚本 start.sh 所在路径，如 `/root/datagw/target` 。
2. 执行 `bash stop.sh` 。
3. 等待 5 秒左右后执行如下命令：
   i. 执行 `ps -ef | grep gateway | grep -v grep` 检查 baas-data-gateway 和 baas-csi 是否还在运行。
   ii. 执行 `ps -ef | grep elasticsearch | grep -v grep` 检查 ES 是否还在运行。
   iii. 若有还在运行的应用，通过 PID （第一列的数字） 杀死进程。杀死方法 `kill -9 ${PID}` 。
4. 执行 `bash start.sh` 重启数据服务。

登录数据服务数据库

1. 使用 start.sh 配置的数据库用户名（ `DB_USER` ）及密码（ `DB_PWD` ）登陆数据库，如 mysql -u root -p
2. 执行以下命令查看数据库列表。

```
show databases;

示例输出：
+--------------------+
| Database           |
+--------------------+
| information_schema |
| baas-csi-db        |
| data-statistic-db  |
| datagw-db          |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
7 rows in set (0.03 sec)
```

3. datagw-db 为 baas-data-gateway 的数据库，是数据服务的后端数据库，baas-csi-db 为 baas-csi 的数据库，用于存储统计任务等相关信息，data-statistic-db 为统计信息数据库，用于存储统计结果信息。
4. 执行以下命令切换到目标数据库，并列出数据表，如切换到 datagw-db。

```
use `datagw-db`;
show tables;
```

5. 查询表的含义及各个字段的含义 (按照实际库名修改 datagw-db)。

```
SELECT
  a.table_name 表名,
  a.table_comment 表说明,
  b.COLUMN_NAME 字段名,
  b.column_comment 字段说明,
  b.column_type 字段类型,
  b.column_key 约束a
FROM
 information_schema.TABLES a
LEFT JOIN information_schema.COLUMNS b ON a.TABLE_NAME = b.TABLE_NAME
WHERE
 a.table_schema = 'datagw-db'
ORDER BY
 a.table_name;
```

Java 虚拟机排查工具用法

- 查看运行的 Java 进程
  - jps -l
- 显示Java堆详细信息 (pid 可以通过查看运行的 Java 进程得到)
  - jmap -heap <pid>
- 查看 Java 线程输出 (pid 可以通过查看运行的 Java 进程得到)
  - jstack -l <pid>
- 排查CPU高的问题
  - top -H -p <pid> #找到load占比大的pid
  - printf "%x\n" <pid> #十六进制转换
  - jstack pid|grep <45d8 左边是16进制pid> -A 30 #定位线程情况