FISCO BCOS EN Documentation

Release v2.9.0

fisco-dev

Jan 12, 2024

INTRODUCTION

1 Introduction 3

2 Compatibility 9

3 Installation 13

4 Build the first blockchain application 21

5 More Tutorials 31

6 Build Blockchain Network 33

7 Application Development 109

8 FAQ (Revision in progress) 147

9 Chain building script 151

10 Console 161

11 Blockchain browser 245

12 Enterprise deployment tool 253

13 JSON-RPC API 295

14 《深入浅出FISCO BCOS》 335

15 Community 337

FISCO BCOS is a reliable, secure, efficient and portable blockchain platform with proven success from many partners and successful financial-grade applications.

- Github homepage
- Insightful articles
- Code contribution
- Feedback
- Application cases
- WeChat group
- WeChat official account

Overview

- To fast build a blockchain system based on FISCO BCOS 2.0+, please read Installation
- To deploy multi-group blockchain and the first blockchain application based on FISCO BCOS 2.0+, please read Quick Guide
- To know more about functions of FISCO BCOS 2.0+, please read Config files and items, Node access, Parallel transactions, Distributed storage, OSCCA computing in Operation Tutorial
- Console：Interactive command tool to visit blockchain nodes and check status, deploy or call contract, etc.
- Deployment tool(Generator)：to support operations like building blockchain, expansion, etc., recommended for business level applications. You can learn the operation methods in Quick Guide
- SDK：offer APIs for node status, blockchain system configuration modification and nodes to send transactions.
- The detailed introduction of browser is in Browser
- JSON-RPC interface is introduced in JSON-RPC API
- System design documentation: System design

Key features

- Multi-group: Quick Guide Operation Tutorial Design Documentation
- Parallel computing: Operation Tutorial Design documentation
- Distributed storage: Operation Tutorial Design documentation

---

Important:

- This technical documentation is only adaptable for FISCO BCOS 2.0+. For FISCO BCOS 1.3.x users, please check Ver.1.3 Documentation
- FISCO BCOS 2.0+ and its adaptability are illustrated here

CHAPTER

ONE

INTRODUCTION

FISCO BCOS is the first safe and controllable enterprise-level financial consortium blockchain platform open source by domestic enterprises. It is jointly created by the FISCO open source working group and officially launched in December 2017.

The community links multiple parties with open source. As of May 2020, more than 1000 enterprises and institutions and more than 10,000 community members have joined to build and co-governance, and developed into the largest and most active domestic consortium blockchain platform ecosystem. The underlying platform is highly available and easy to use after extensive application and practice. Hundreds of application projects are developed based on the FISCO BCOS underlying platform, and over 80 have been steadily operating in the production environment, covering cultural copyright, judicial services, government services, Internet of Things, finance, smart communities and other fields.

Note: FISCO BCOS takes the actual needs of the consortium blockchain as a starting point, taking into account performance, security, maintainability, ease of use, and scalability, and supports multiple SDK, and provides visual middleware tools, greatly reducing the time to build chains, develop and deploy applications. In addition, FISCO BCOS passed the two evaluations of the Trusted Blockchain evaluation function and performance of the Information Communication Institute, and the single-chain TPS can reach 20,000.

1.1 Key Features

1.2 Architecture

In 2.0, FISCO BCOS innovatively proposed a “one-body, two-wing, multi-engine” architecture to achieve horizontal expansion of system throughput and greatly improve performance. It has industry in terms of security, operation and maintenance, ease of use, and scalability, and leading edge.

The 'One-body' refers to the group structure, supports the rapid formation of consortium blockchain, and allows companies to build chains as easily as chat groups. According to business scenarios and business relationships, enterprises can choose different groups to form data sharing and consensus of multiple different ledgers, thereby quickly enriching business scenarios, expanding business scale, and greatly simplifying the deployment and operation and maintenance costs of the chain.

The 'two wings' refer to supporting parallel computing models and distributed storage, both of which bring better scalability to the group architecture. The former changes the method of serial execution in the order of transactions in the block, and executes transactions in parallel based on DAG (directed acyclic graph), which greatly improves performance. The latter supports enterprises (nodes) to store data in remote distributed systems, overcoming many limitations of localized data storage.

'Multi-engine' is a summary of a series of functional features. For example, pre-compiled contracts can break through the performance bottleneck of EVM and achieve high-performance contracts; the console allows users to quickly master blockchain usage skills.

The above features all focus on solving the pain points of technology and experience, provide more tool support for development, operation and maintenance, governance and supervision, make the system process faster and

have higher capacity, and make the application operating environment safer and more stable.

1.3 Core module

FISCO BCOS adopts high-throughput scalable multi-group architecture, which can dynamically manage multiple chains and groups to meet the expansion and isolation requirements of multiple business scenarios. Modules include:

- Consensus mechanism: Pluggable consensus mechanism, supporting PBFT, Raft and rPBFT consensus algorithms, low transaction confirmation delay, high throughput, and ultimate consistency. Among them, PBFT and rPBFT can solve Byzantine problems and have higher security.
- Storage: The storage of the world state is changed from the original MPT storage structure to distributed storage, avoids the problem of performance degradation caused by the rapid expansion of the world state. Introduces a pluggable storage engine, supports LevelDB, RocksDB, MySQL and other back-end storage, supports data expansion quickly and easily, and isolates calculation from data, reducing the impact of node failure on node data.
- Network: Support network compression, and implement a good distributed network distribution mechanism based on the idea of load balancing to minimize bandwidth overhead.

1.4 Performance

In order to improve system performance, FISCO BCOS optimizes transaction execution in terms of improving transaction execution efficiency and concurrency, so that transaction processing performance can reach more than 10,000 levels.

- Precompiled contract based on C++: The Precompiled contract written in C++ language is built into the blockchain platform, and the execution efficiency is higher.
- Transaction execution in parallel: Based on the DAG algorithm to build a transaction execution flow within a block based on the mutually exclusive relationship between transactions, maximizing parallel execution of transactions within a block.

1.5 Safety

Considering the high security requirements of the consortium blockchain platform, in addition to the TLS security protocol used for communication between nodes and between nodes and clients, FISCO BCOS also implements a complete set of security solutions:

- Network access mechanism: Restrict nodes from joining and exiting the alliance chain, and delete the malicious nodes of the specified group from the group, ensuring system security.
- Black and white list mechanism: Each group can only receive messages from the corresponding group to ensure the isolation of network communication between the groups; the CA blacklist mechanism can disconnect the network connection from the malicious node in time, ensuring the security of the system.
- Authority management mechanism: Based on distributed storage permission control mechanism, flexible and fine-grained control of permissions for external account deployment contracts and creation, insertion, deletion and update of user tables.
- Support OSCCA-approved algorithm: Support OSCCA-approved encryption, signature algorithm and OSCCA-approved SSL communication protocol.
- Disk encryption algorithm: Support the disk encryption algorithm to ensure the confidentiality of the data on the chain.
- Key management scheme: Based on the disk encryption algorithm, the KeyManager service is used to manage the node key, which is more secure.

- Homomorphic encryption、Group/Ring signature: Homomorphic encryption and group ring signature interfaces are provided on the chain to meet more business needs.

1.6 Operability

In the consortium blockchain platform, the operation and maintenance of the blockchain is crucial. FISCO BCOS provides a complete set of operation and maintenance deployment tools, and introduces contract naming service, data archiving and migration, contract lifecycle management to improve Operation and Management efficiency.

- Operation and Management deployment tool: Convenient tool for deploying, managing and monitoring multi-institution multi-group consortium blockchain, supporting multiple operations such as expanding nodes and expanding new groups.
- Contract naming service: Establish a mapping relationship between the contract address to the contract name and the contract version, so that the caller can easily call the contract on the chain by remembering the simple contract name.
- Data archiving, migration and export functions: Provide data export components, support on-chain data archiving, migration and export, increase the maintainability of on-chain data, and reduce the complexity of operation.
- Contract lifecycle management: Provide contract life cycle management function on the chain, which is convenient for the chain administrator to manage the contract on the chain.

1.7 Ease of use

FISCO BCOS introduces tools such as development and deployment tools, interactive console, blockchain browsers, etc. to improve the ease of use of the system and greatly reduce the time to build chains and deploy applications.

- Development and deployment tools
- Interactive command line tool console based on JavaSDK
- Interactive command line tool console based on Web3SDK
- Blockchain browser

In order to facilitate the rapid development of applications for developers of different languages, FISCO BCOS also supports Java SDK、Node.js SDK、Python SDK and Go SDK

1.8 Community development tools

Relying on the huge open source ecosystem, all partners in the community uphold the co-construction concept of "from developers, for developers", On the bottom platform of FISCO BCOS, independently develop multiple development tools at hand and give back to the community to reduce the difficulty and cost of blockchain application development from different business levels. The following is a partial list, and more institutions or developers are welcome to feedback more useful tools to the community.

- Blockchain middleware platform WeBASE: For a variety of roles, such as developers and operators, and according to different scenarios, including development, debugging, deployment, audit, etc., to create a wealth of functional components and practical tools, providing a friendly and visual operating environment.
- Distributed identity solution WeIdentity: A distributed multi-center technology solution based on blockchain, providing a series of basic layer and application interfaces such as distributed entity identity identification and management, trusted data exchange protocol, etc., which can realize the data of entity objects (people or things) Security authorization and exchange.

- Distributed event-driven architecture WeEvent: Implemented a credible, reliable, and efficient cross-institutional and cross-platform event notification mechanism. Without changing the development language and access protocol of existing commercial systems, realize cross-institution and cross-platform event notification and processing.
- Cross-chain collaboration solution WeCross: Support cross-chain transaction transactions, meet the atomicity of cross-chain transactions, manage cross-chain transactions, support multi-party collaborative management, and avoid single-point risks.
- Scene-style privacy protection solution WeDPR: For hidden payment, anonymous voting, anonymous bidding and selective disclosure and other application solutions, provide an immediately available scenario-based privacy protection and efficient solutions to help various industries to explore data-related businesses legally and compliantly.
- ChainIDE: Provide smart contract cloud development tools to help developers save marginal costs and accelerate the launch of blockchain applications.
- FISCO BCOS Blockchain Toolbox: Work with IDEs such as WeBase/Remix/VSCode/ChainIDE to improve development experience and efficiency.

CHAPTER

TWO

COMPATIBILITY

FISCO BCOS 2.7.0

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.7.0

FISCO BCOS 2.6.0

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.6.0

FISCO BCOS 2.5.0

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.5.0

FISCO BCOS 2.4.0

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.4.0

FISCO BCOS 2.3.0

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.3.0

FISCO BCOS 2.2.0

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.2.0

FISCO BCOS 2.1.0

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.1.0

FISCO BCOS 2.0.0

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.0.0

FISCO BCOS 2.0.0-rc3

New features

- Distributed storage (Operation Manual)
- 'CRUD SDK interface<../sdk/sdk.html#crudservice>'_ (Operation Manual)

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.0.0-rc3

FISCO BCOS 2.0.0-rc2

New features

- Parallel computing model (Operation Manual) (Operation Tutorial)
- Distributed storage (Operation Manual)

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.0.0-rc2

FISCO BCOS 2.0.0-rc1

New features

- Group architecture (Operation Tutorial) (Design Document)
- Console (Installation) (Operation Manual)
- Virtual machine
- Compile contract (Operation Manual)
- CRUD interface contract (Operation Tutorial)
- Key management service (Operation Manual)
- Admission control (Operation Manual)

Change description, compatibility and upgrade instructions

- FISCO BCOS v2.0.0-rc1

FISCO BCOS 1.x Releases

FISCO BCOS 1.3 version:

- FISCO BCOS 1.3.8 Release
- FISCO BCOS 1.3.7 Release
- FISCO BCOS 1.3.6 Release
- FISCO BCOS 1.3.5 Release

- FISCO BCOS 1.3.4 Release
- FISCO BCOS 1.3.3 Release
- FISCO BCOS 1.3.2 Release
- FISCO BCOS 1.3.1 Release
- FISCO BCOS 1.3.0 Release

FISCO BCOS 1.2 version:

- FISCO BCOS 1.2.0 Release

FISCO BCOS 1.1 version:

- FISCO BCOS 1.1.0 Release

FISCO BCOS 1.0 version:

- FISCO BCOS 1.0.0 Release

FISCO BCOS preview version：

- FISCO BCOS 1.5.0 pre-release

View node and data versions

- View node binary version: `./fisco-bcos --version`
- Data format and version of communication protocol: to get it via 'supported_version' configuration item in the configuration file config.ini

CHAPTER

THREE

INSTALLATION

This chapter will introduce the required installations and configurations of FISCO BCOS. For better understanding, we will illustrate an example of deploying a 4-node consortium chain in a local machine using FISCO BCOS. Please use the supported hardware and platform operations according to here.

3.1 To build a single-group consortium chain

This section takes the construction of single group FISCO BCOS chain as an example to operate. We use the `build_chain.sh` script to build a 4-node FISCO BCOS chain locally in `Ubuntu 16.04 64bit` system.

Note:

- To update an existing chain, please refer to compatibility chapter.
- To build OSCCA chain, please refer to ' <../manual/guomi_crypto.html>'_ 。
- It is similar to build a multi-group chain, interested can be referred to here .
- This section uses pre-compiled static fisco-bcos binaries which tested on CentOS 7 and Ubuntu 16.04 64bit.
- build_chain use docker

3.1.1 Prepare environment

- Install dependence

`build_chain.sh` script depends on `openssl, curl` and is installed by using the following instructions. For CentOS system, to replaces `apt` with `yum` in the following command. For macOS system, execute `brew install openssl curl`.

```
# ubuntu
sudo apt install -y openssl curl

# centos
sudo yum install -y openssl openssl-devel
```

- Create operation directory

```
cd ~ && mkdir -p fisco && cd fisco
```

- Download `build_chain.sh` script

```
curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.7.2/build_
↪chain.sh && chmod u+x build_chain.sh
```

Note:

- If the build_chain.sh script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/blob/master-2.0/manual/build_chain.sh && chmod u+x build_chain.sh

3.1.2 Build a single-group 4-node consortium chain

Execute the following command in the fisco directory to generate a single group 4-node FISCO chain. It is necessary to ensure that the `30300~30303, 20200~20203, 8545~8548` ports of the machine are not occupied.

```
bash build_chain.sh -l 127.0.0.1:4 -p 30300,20200,8545
```

Note:

- The -p option specifies the starting port, which are p2p_port, channel_port, and jsonrpc_port.
- For security and ease of use consideration, the latest configuration of v2.3.0 version splits listen_ip into jsonrpc_listen_ip and channel_listen_ip, but still retains the parsing function of listen_ip. For details, please refer to here
- In order to facilitate development and experience, the reference configuration of channel_listen_ip is 0.0.0.0. For security reasons, please modify it to a safe listening address according to the actual business network situation, such as: intranet IP or specific external IP

If the command is executed successfully, `All completed` will be output. If the execution fails, please check the error message in the `nodes/build.log` file.

```
Checking fisco-bcos binary...
Binary check passed.
============================================================
Generating CA key...
============================================================
Generating keys ...
Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1
============================================================
Generating configurations...
Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1
============================================================
[INFO] Execute the download_console.sh script in directory named by IP to get␣
↪FISCO-BCOS console.
e.g.  bash /home/ubuntu/fisco/nodes/127.0.0.1/download_console.sh
============================================================
[INFO] FISCO-BCOS Path   : bin/fisco-bcos
[INFO] Start Port        : 30300 20200 8545
[INFO] Server IP         : 127.0.0.1:4
[INFO] State Type        : storage
[INFO] RPC listen IP     : 127.0.0.1
[INFO] Output Dir        : /home/ubuntu/fisco/nodes
[INFO] CA Key Path       : /home/ubuntu/fisco/nodes/cert/ca.key
============================================================
[INFO] All completed. Files in /home/ubuntu/fisco/nodes
```

3.1.3 Start FISCO BCOS chain

- Execute the following command to start all nodes

```
bash nodes/127.0.0.1/start_all.sh
```

Success will output a response similar to the following, otherwise, please use `netstat -an | grep tcp` to check whether the machine's `30300~30303, 20200~20203, 8545~8548` ports are occupied.

```
try to start node0
try to start node1
try to start node2
try to start node3
 node1 start successfully
 node2 start successfully
 node0 start successfully
 node3 start successfully
```

3.1.4 Check process

- Execute the following command to check whether the process is started

```
ps -ef | grep -v grep | grep fisco-bcos
```

In normal situation, the output will be similar to the following. If the number of processes is not 4, then the reason why the process does not start is that the port is occupied.

```
fisco       5453     1  1 17:11 pts/0    00:00:02 /home/ubuntu/fisco/nodes/127.0.0.
↪1/node0/../fisco-bcos -c config.ini
fisco       5459     1  1 17:11 pts/0    00:00:02 /home/ubuntu/fisco/nodes/127.0.0.
↪1/node1/../fisco-bcos -c config.ini
fisco       5464     1  1 17:11 pts/0    00:00:02 /home/ubuntu/fisco/nodes/127.0.0.
↪1/node2/../fisco-bcos -c config.ini
fisco       5476     1  1 17:11 pts/0    00:00:02 /home/ubuntu/fisco/nodes/127.0.0.
↪1/node3/../fisco-bcos -c config.ini
```

3.1.5 Check log output

- Execute the following command to view the number of nodes that node0 links to

```
tail -f nodes/127.0.0.1/node0/log/log*  | grep connected
```

In normal situation, the connecting messages will be output continuously. From the output messages, we can see that node0 has links with the other three nodes.

```
info|2019-01-21 17:30:58.316769| [P2P][Service] heartBeat,connected count=3
info|2019-01-21 17:31:08.316922| [P2P][Service] heartBeat,connected count=3
info|2019-01-21 17:31:18.317105| [P2P][Service] heartBeat,connected count=3
```

- Execute the following command to check whether it is in consensus

```
tail -f nodes/127.0.0.1/node0/log/log*  | grep +++
```

In normal situation, the message will be output `++++Generating seal` continuously to indicate that the consensus is normal.

```
info|2019-01-21 17:23:32.576197|␣
↪[g:1][p:264][CONSENSUS][SEALER]++++++++++++++++Generating seal on,blkNum=1,tx=0,
↪myIdx=2,hash=13dcd2da...
info|2019-01-21 17:23:36.592280|␣
↪[g:1][p:264][CONSENSUS][SEALER]++++++++++++++++Generating seal on,blkNum=1,tx=0,
↪myIdx=2,hash=31d21ab7...
info|2019-01-21 17:23:40.612241|␣
↪[g:1][p:264][CONSENSUS][SEALER]++++++++++++++++Generating seal on,blkNum=1,tx=0,
↪myIdx=2,hash=49d0e830...
```

3.2 Using console

Important:

- `console 1.x` series is based on Web3SDK implementation, `Console 2.6 after` is based on 'Java SDK <../sdk/java_sdk/index.html >'_ implementation, the latest version of the console is based on the `Java SDK` implementation
- For 2.6 and above version console documentation please refer to here
- For 1.x version console documentation, please refer to here
- You can view the current console version through the command `./start.sh --version`

Console links nodes of FISCO BCOS so as to realize functions like blockchain status query, call and deploy contracts. 2.0 version console command detailed introduction refer here, 1.x version console command detailed introduction refer here.

3.2.1 Prepare environment

- Install Java

In macOS, execute `brew cask install java` to install java

```
# ubuntu
sudo apt install -y default-jdk

# centos
sudo yum install -y java java-devel
```

- Get console

```
cd ~/fisco &&  curl -#LO https://github.com/FISCO-BCOS/console/releases/download/
↪v2.9.2/download_console.sh && bash download_console.sh
```

Note:

- If the download_console.sh script cannot be downloaded for a long time due to network problems, try cd ~/fisco && curl -#LO https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/download_console.sh, and modify encryptType to 1 in applicationContext.xml.

- Copy the console configuration file. If the node does not use the default port, please replace 20200 in the file with another port.

```
# The latest version of the console uses the following command to copy the␣
↪configuration file
cp -n console/conf/config-example.toml console/conf/config.toml
```

- Configure the console certificate

```
cp nodes/127.0.0.1/sdk/* console/conf/
```

3.2.2 Start console

- Start console

```
cd ~/fisco/console && bash start.sh
```

If it outputs following information, then the console has been started successfully, otherwise please check if the node ports in conf/config.toml are configured correctly.

```
=====================================================================================================
Welcome to FISCO BCOS console(2.6.0)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
 ________  ______   ______    ______    ______         _______    ______    ______
↪   ______
|        \|      \ /      \  /      \  /      \       |       \  /      \  /      \
↪\  /      \
| $$$$$$$$ \$$$$$$|  $$$$$$\|  $$$$$$\|  $$$$$$\      | $$$$$$$\|  $$$$$$\|  $$$$$
↪$\|  $$$$$$\
| $$__       | $$  | $$___\$$| $$   \$$| $$  | $$      | $$__/ $$| $$   \$$| $$  | $
↪$| $$___\$$
| $$  \      | $$   \$$    \ | $$      | $$  | $$      | $$    $$| $$      | $$  | $
↪$ \$$    \
| $$$$$      | $$   _\$$$$$$\| $$   __ | $$  | $$      | $$$$$$$\| $$   __ | $$  | $
↪$ _\$$$$$$\
| $$_____   _| $$_ |  \__| $$| $$__/  \| $$__/ $$      | $$__/ $$| $$__/  \| $$__/ $
↪$|  \__| $$
| $$     |   $$ \ \$$    $$ \$$    $$ \$$    $$      | $$    $$ \$$    $$ \$$    $
↪$ \$$    $$
 \$$      \$$$$$$  \$$$$$$   \$$$$$$   \$$$$$$        \$$$$$$$   \$$$$$$   \$$$$$
↪$   \$$$$$$

=====================================================================================================
```

3.2.3 Query blockchain status

```
# acquire client ends version information
[group:1]> getNodeVersion
ClientVersion{
    "Build Time":"20200619 06:32:10",
    "Build Type":"Linux/clang/Release",
    "Chain Id":"1",
    "FISCO-BCOS Version":"2.5.0",
    "Git Branch":"HEAD",
    "Git Commit Hash":"72c6d770e5cf0f4197162d0e26005ec03d30fcfe",
    "Supported Version":"2.5.0"
}
# acquire node connection information
[group:1]> getPeers
[
    {
        "IPAndPort":"127.0.0.1:49948",
        "NodeID":
↪"b5872eff0569903d71330ab7bc85c5a8be03e80b70746ec33cafe27cc4f6f8a71f8c84fd8af9d7912cb5ba068901fe
↪",
        "Topic":[]
    },
    {
        "IPAndPort":"127.0.0.1:49940",
        "NodeID":
↪"912126291183b673c537153cf19bf5512d5355d8edea7864496c257630d01103d89ae26d17740daebdd20cbc645c9a
↪",
        "Topic":[]
    },
    {
```

(continues on next page)

(continued from previous page)

```
        "IPAndPort":"127.0.0.1:49932",
        "NodeID":
↪"db75ab16ed7afa966447c403ca2587853237b0d9f942ba6fa551dc67ed6822d88da01a1e4da9b51aedafb8c64e9d20
↪",
        "Topic":[]
    }
]
```

3.3 To deploy or call HelloWorld contract

3.3.1 HelloWorld contract

HelloWorld contract offers 2 interfaces which are `get()` and `set()` and are used to acquire/set contract variety `name`. The contract content is as below:

```
pragma solidity ^0.4.24;

contract HelloWorld {
    string name;

    function HelloWorld() {
        name = "Hello, World!";
    }

    function get()constant returns(string) {
        return name;
    }

    function set(string n) {
        name = n;
    }
}
```

3.3.2 Deploy HelloWorld contract

For quick experience, the console comes with HelloWorld contract and is placed under console folder `contracts/solidity/HelloWorld.sol`. So, users only have to deploy it using the following command.

```
# input the following instruction in console, if it is deployed successfully, the␣
↪contract address will be returned
[group:1]> deploy HelloWorld
transaction hash:␣
↪0xd0305411e36d2ca9c1a4df93e761c820f0a464367b8feb9e3fa40b0f68eb23fa
contract address:0xb3c223fc0bf6646959f254ac4e4a7e355b50a344
```

3.3.3 Call HelloWorld contract

```
# check the current block number
[group:1]> getBlockNumber
1

# call get interface to acquire name variety, the contract address here is the␣
↪returned address of deploy instruction
[group:1]> call HelloWorld 0xb3c223fc0bf6646959f254ac4e4a7e355b50a344 get
```

(continues on next page)

(continued from previous page)

```
----------------------------------------------------------------------------------------
↪----------
Return code: 0
description: transaction executed successfully
Return message: Success
----------------------------------------------------------------------------------------
↪----------
Return values:
[
    "Hello,World!"
]
----------------------------------------------------------------------------------------
↪----------

# check the current block number, it remains the same, because get interface will␣
↪not change the ledger status
[group:1]> getBlockNumber
1

# call set to set name
[group:1]> call HelloWorld 0xb3c223fc0bf6646959f254ac4e4a7e355b50a344 set "Hello,␣
↪FISCO BCOS"
transaction hash:␣
↪0x7e742c44091e0d6e4e1df666d957d123116622ab90b718699ce50f54ed791f6e
----------------------------------------------------------------------------------------
↪----------
transaction status: 0x0
description: transaction executed successfully
----------------------------------------------------------------------------------------
↪----------
Output
Receipt message: Success
Return message: Success
----------------------------------------------------------------------------------------
↪----------
Event logs
Event: {}

# check the current block number again, if it increased, then it has generated␣
↪block and the ledger status is changed
[group:1]> getBlockNumber
2

# call get interface to acquire name variety, check if the setting is valid
[group:1]> call HelloWorld 0xb3c223fc0bf6646959f254ac4e4a7e355b50a344 get
----------------------------------------------------------------------------------------
↪----------
Return code: 0
description: transaction executed successfully
Return message: Success
----------------------------------------------------------------------------------------
↪----------
Return values:
[
    "Hello,FISCO BCOS"
]
----------------------------------------------------------------------------------------
↪----------

# log out console
[group:1]> quit
```

CHAPTER

FOUR

BUILD THE FIRST BLOCKCHAIN APPLICATION

This chapter will introduce a whole process of business application scenario development based on FISCO BCOS blockchain. The introduce includes business scenario analysis, contract design implementation, contract compilation, and blockchain development. Finally, we introduce an application module implementation which is to implement calling access to the contract on blockchain through the Java SDK we provide.

This tutorial requires user to be familiar with the Linux operating environment, has the basic skills of Java development, is able to use the Gradle tool, and is familiar with Solidity syntax.

Through the tutorial, you will learn the following:

1. How to express the logic of a business scenario in the form of a contract
2. How to convert Solidity contract into Java class
3. How to configure Java SDK
4. How to build an application and integrate Java SDK into application engineering
5. How to call the contract interface through Java SDK, and to understand its principle

The full project source code for the sample is provided in the tutorial and users can quickly develop their own applications based on it.

Important: Please refer to Installation documentation to complete the construction of the FISCO BCOS blockchain and the download of the console. The operation in this tutorial is assumed to be carried out in the environment of the documentation building.

4.1 Sample application requirements

Blockchain is naturally tamper-proof and traceable. These characteristics make it more attractive to the financial sector. This article will provide an easy example of asset management development and ultimately achieve the following functions:

- Ability to register assets on blockchain
- Ability to transfer funds from different accounts
- Ability to check the amount of assets in the account

4.2 Contract design and implementation

When developing an application on blockchain, for combining with business requirements, it is first necessary to design the corresponding smart contract to determine the storage data that contract needs, and on this basis, to determine the interface provided by the smart contract. Finally, to specifically implement each interface.

4.2.1 Storage design

FISCO BCOS provides a contract CRUD interface development model, which can create table through contracts, and add, delete, and modify the created table. For this application, we need to design a table `t_asset` for storage asset management. The table's fields are as follows:

- account: primary key, asset account (string type)
- asset_value: asset amount (uint256 type)

account is the primary key, which is the field that needs to be passed when the `t_asset` table is operated. The blockchain queries the matching records in the table according to the primary key field. The example of `t_asset` table is as follow:

4.2.2 Interface design

According to the design goals of the business, it is necessary to implement asset registration, transfer, and query functions. The interfaces of the corresponding functions are as follows:

```
// query the amount of assets
function select(string account) public constant returns(int256, uint256)
// asset registration
function register(string account, uint256 amount) public returns(int256)
// asset transfer
function transfer(string from_asset_account, string to_asset_account, uint256␣
↪amount) public returns(int256)
```

4.2.3 Full source

```
pragma solidity ^0.4.24;

import "./Table.sol";

contract Asset {
    // event
    event RegisterEvent(int256 ret, string account, uint256 asset_value);
    event TransferEvent(int256 ret, string from_account, string to_account,␣
↪uint256 amount);

    constructor() public {
        // create a t_asset table in the constructor
        createTable();
    }

    function createTable() private {
        TableFactory tf = TableFactory(0x1001);
        // asset management table, key : account, field : asset_value
        // |  account(primary key)   |  amount       |
        // |-------------------- |-------------------|
        // |        account      |    asset_value    |
        // |---------------------|-------------------|
        //
        // create table
        tf.createTable("t_asset", "account", "asset_value");
    }

    function openTable() private returns(Table) {
        TableFactory tf = TableFactory(0x1001);
        Table table = tf.openTable("t_asset");
        return table;
```

(continues on next page)

(continued from previous page)

```
    }

    /*
    description: query asset amount according to asset account

    parameter:
            account: asset account

    return value:
            parameter1:  successfully returns 0, the account does not exist and␣
↪returns -1
            parameter2:  valid when the first parameter is 0, the amount of assets
    */
    function select(string account) public constant returns(int256, uint256) {
        // open table
        Table table = openTable();
        // query
        Entries entries = table.select(account, table.newCondition());
        uint256 asset_value = 0;
        if (0 == uint256(entries.size())) {
            return (-1, asset_value);
        } else {
            Entry entry = entries.get(0);
            return (0, uint256(entry.getInt("asset_value")));
        }
    }

    /*
    description : asset registration
    parameter :
            account : asset account
            amount  : asset amount
    return value:
            0  regist successfully
            -1 asset account already exists
            -2 other error
    */
    function register(string account, uint256 asset_value) public returns(int256){
        int256 ret_code = 0;
        int256 ret= 0;
        uint256 temp_asset_value = 0;
        // to query whather the account exists
        (ret, temp_asset_value) = select(account);
        if(ret != 0) {
            Table table = openTable();

            Entry entry = table.newEntry();
            entry.set("account", account);
            entry.set("asset_value", int256(asset_value));
            // insert
            int count = table.insert(account, entry);
            if (count == 1) {
                // true
                ret_code = 0;
            } else {
                // false. no permission or other error
                ret_code = -2;
            }
        } else {
            // account already exists
            ret_code = -1;
```

(continues on next page)

(continued from previous page)

```
        }

        emit RegisterEvent(ret_code, account, asset_value);

        return ret_code;
    }

    /*
    description : asset transfer
    parameter :
            from_account : transferred asset account
            to_account : received asset account
            amount :  transferred amount
    return value:
            0  transfer asset successfully
            -1 transfe asset account does not exist
            -2 receive asset account does not exist
            -3 amount is insufficient
            -4 amount is excessive
            -5 other error
    */
    function transfer(string from_account, string to_account, uint256 amount)␣
↪public returns(int256) {
        // query transferred asset account information
        int ret_code = 0;
        int256 ret = 0;
        uint256 from_asset_value = 0;
        uint256 to_asset_value = 0;

        // whather transferred asset account exists?
        (ret, from_asset_value) = select(from_account);
        if(ret != 0) {
            ret_code = -1;
            // not exist
            emit TransferEvent(ret_code, from_account, to_account, amount);
            return ret_code;

        }

        // whather received asset account exists?
        (ret, to_asset_value) = select(to_account);
        if(ret != 0) {
            ret_code = -2;
            // not exist
            emit TransferEvent(ret_code, from_account, to_account, amount);
            return ret_code;
        }

        if(from_asset_value < amount) {
            ret_code = -3;
            // amount of transferred asset account is insufficient
            emit TransferEvent(ret_code, from_account, to_account, amount);
            return ret_code;
        }

        if (to_asset_value + amount < to_asset_value) {
            ret_code = -4;
            // amount of received asset account is excessive
            emit TransferEvent(ret_code, from_account, to_account, amount);
            return ret_code;
        }
```

(continues on next page)

(continued from previous page)

```
        Table table = openTable();

        Entry entry0 = table.newEntry();
        entry0.set("account", from_account);
        entry0.set("asset_value", int256(from_asset_value - amount));
        // update transferred account
        int count = table.update(from_account, entry0, table.newCondition());
        if(count != 1) {
            ret_code = -5;
            // false? no permission or other error?
            emit TransferEvent(ret_code, from_account, to_account, amount);
            return ret_code;
        }

        Entry entry1 = table.newEntry();
        entry1.set("account", to_account);
        entry1.set("asset_value", int256(to_asset_value + amount));
        // update received account
        table.update(to_account, entry1, table.newCondition());

        emit TransferEvent(ret_code, from_account, to_account, amount);

        return ret_code;
    }
}
```

Note: The implementation of the `Asset.sol` contract requires to introduce a system contract interface file `Table.sol` provided by FISCO BCOS. The system contract file's interface is implemented by the underlying FISCO BCOS. When a business contract needs to operate CRUD interface, it is necessary to introduce the interface contract file. `Table.sol` contract detailed interface reference here.

4.3 Contract compiling

In the previous section, we designed the storage and interface of the contract `Asset.sol` according to business requirements, and implemented them completely. However, Java program cannot directly call Solidity contract. The Solidity contract file needs to be compiled into a Java file first.

The console provides a compilation tool that stores the `Asset.sol` contract file in the `console/contract/solidity` directory. Compile with the `sol2java.sh` script provided in the console directory, as follows:

```
$ mkdir -p ~/fisco
# download console
$ cd ~/fisco && curl -#LO https://github.com/FISCO-BCOS/console/releases/download/
↪v2.9.2/download_console.sh && bash download_console.sh
# switch to the fisco/console/ directory
$ cd ~/fisco/console/
# compile the contract, specify a Java package name parameter later, you can␣
↪specify the package name according to the actual project path.
$ ./sol2java.sh -p org.fisco.bcos.asset.contract
```

After successful operation, the java, abi, and bin directories will be generated in the `console/contracts/sdk directory` as shown below.

```
|-- abi # The generated abi directory, which stores the abi file generated by␣
↪Solidity contract compilation.
|   |-- Asset.abi
|   |-- Table.abi
|-- bin # The generated bin directory, which stores the bin file generated by␣
↪Solidity contract compilation.
```

(continues on next page)

(continued from previous page)

```
|   |-- Asset.bin
|   |-- Table.bin
|-- contracts # The source code file that stores Solidity contract. Copy the␣
↪contract that needs to be compiled to this directory.
|   |-- Asset.sol # A copied Asset.sol contract, depends on Table.sol
|   |-- Table.sol # The contract interface file that implements the CRUD operation
|-- java  # Storing compiled package path and Java contract file
|   |-- org
|        |--fisco
|             |--bcos
|                  |--asset
|                       |--contract
|                            |--Asset.java  # Java file generated by the Asset.
↪sol contract
|                            |--Table.java  # Java file generated by the Table.
↪sol contract
|-- sol2java.sh
```

The `org/fisco/bcos/asset/contract/` package path directory is generated in the java directory. The directory contains two files `Asset.java` and `Table.java`, where `Asset.java` is the file required by the Java application to call the Asset.sol contract.

`Asset.java`'s main interface:

```
package org.fisco.bcos.asset.contract;

public class Asset extends Contract {
    // Asset.sol contract  transfer interface generation
    public TransactionReceipt transfer(String from_account, String to_account,␣
↪BigInteger amount);
    // Asset.sol contract  register interface generation
    public TransactionReceipt register(String account, BigInteger asset_value);
    // Asset.sol contract  select interface generation
     public Tuple2<BigInteger, BigInteger> select(String account) throws␣
↪ContractException;

    // Load the Asset contract address, to generate Asset object
    public static Asset load(String contractAddress, Client client, CryptoKeyPair␣
↪credential);

    // Deploy Assert.sol contract, to generate Asset object
    public static Asset deploy(Client client, CryptoKeyPair credential) throws␣
↪ContractException;
}
```

The load and deploy functions are used to construct the Asset object, and the other interfaces are used to call the interface of the corresponding solidity contract. The detailed use will be introduced below.

4.4 SDK configuration

We provide a Java engineering project for development. First, get the Java engineering project:

```
$ mkdir -p ~/fisco
# get the Java project project archive
$ cd ~/fisco
$ curl -#LO https://github.com/FISCO-BCOS/LargeFiles/raw/master/tools/asset-app.
↪tar.gz
# extract the Java project project asset-app directory
$ tar -zxf asset-app.tar.gz
```

Note:

- If the asset-app.tar.gz cannot be downloaded for a long time due to network problems, try curl -#LO https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/FISCO-BCOS/FISCO-BCOS/tools/asset-app.tar.gz

The directory structure of the asset-app project is as follows:

```
|-- build.gradle // gradle configuration file
|-- gradle
|   |-- wrapper
|       |-- gradle-wrapper.jar //  related code implementation for downloading␣
↪Gradle
|       |-- gradle-wrapper.properties //  Configuration information used by the␣
↪wrapper, such as the version of gradle
|-- gradlew // shell script for executing wrapper commands under Linux or Unix
|-- gradlew.bat // batch script for executing wrapper commands under Windows
|-- src
|   |-- main
|   |   |-- java
|   |         |-- org
|   |             |-- fisco
|   |                   |-- bcos
|   |                         |-- asset
|   |                               |-- client // the client calling class
|   |                                       |-- AssetClient.java
|   |                               |-- contract // the Java contract class
|   |                                       |-- Asset.java
|   |-- test
|       |-- resources // resource files
|           |-- applicationContext.xml // project configuration file
|           |-- contract.properties // file that stores the deployment contract␣
↪address
|           |-- log4j.properties // log configuration file
|           |-- contract // Solidity contract files
|                   |-- Asset.sol
|                   |-- Table.sol
|
|-- tool
    |-- asset_run.sh // project running script
```

4.4.1 Project introduced Java SDK

The project's `build.gradle` file has been introduced to Java SDK and no need to be modified. The introduction method is as follows:

- You need to add maven remote repository to the `build.gradle` file:

```
repositories {
    mavenCentral()
    maven {
        url "http://maven.aliyun.com/nexus/content/groups/public/"
    }
    maven { url "https://oss.sonatype.org/content/repositories/snapshots" }
}
```

- introduce the Java SDK jar package

```
compile ('org.fisco-bcos.java-sdk:fisco-bcos-java-sdk:2.7.2')
```

4.4.2 Certificate and configuration file

- Blockchain node certificate configuration

Copy the SDK certificate corresponding to the blockchain node

```
# go to the ~ directory
# copy the node certificate to the project's resource directory
$ cd ~/fisco
$ cp -r nodes/127.0.0.1/sdk/* asset-app/src/test/resources/conf
# if you want to run this app in IDE, copy the certificate to the main resource␣
↪directory
$ mkdir -p asset-app/src/main/resources/conf
$ cp -r nodes/127.0.0.1/sdk/* asset-app/src/main/resources/conf
```

- applicationContext.xml

Note:

If the channel_listen_ip (If the node version is less than v2.3.0, check listen_ip) set in the chain is 127.0.0.1 or 0.0.0.0 and the channel_listen_port is 20200, the `applicationContext.xml` configuration does not need to be modified. If the configuration of blockchain node is changed, you need to modify `applicationContext.xml`.

4.5 Business development

We've covered how to introduce and configure the Java SDK in your own project. This section describes how to invoke a contract through a Java program, as well as an example asset management note. The asset-app project already contains the full source code of the sample, which users can use directly. Now introduces the design and implementation of the core class `AssetClient`.

`AssetClient.java`: The deployment and invocation of the contract is implemented by calling `Asset.java`, The path `/src/main/java/org/fisco/bcos/asset/client`, the initialization and the calling process are all in this class.

- initialization

The main function of the initialization code is to construct the Web3j and Credentials' objects, which are needed to be used when creating the corresponding contract class object (calling the contract class's deploy or load function).

```
@SuppressWarnings("resource")
ApplicationContext context =
        new ClassPathXmlApplicationContext("classpath:applicationContext.xml");
bcosSDK = context.getBean(BcosSDK.class);
// init the client that can send requests to the group one
client = bcosSDK.getClient(1);
// create the keyPair
cryptoKeyPair = client.getCryptoSuite().createKeyPair();
client.getCryptoSuite().setCryptoKeyPair(cryptoKeyPair);
logger.debug("create client for group1, account address is " + cryptoKeyPair.
↪getAddress());
```

- construct contract class object

Contract objects can be initialized using the deploy or load functions, which are used in different scenarios. The former applies to the initial deployment contract, and the latter is used when the contract has been deployed and the contract address is known.

```
// deploy contract
Asset asset = Asset.deploy(client, cryptoKeyPair);
// load contract address
Asset asset = Asset.load(contractAddress, client, cryptoKeyPair);
```

- interface calling

Use the contract object to call the corresponding interface and handle the returned result.

```
// select interface calling
 Tuple2<BigInteger, BigInteger> result = asset.select(assetAccount);
// register interface calling
TransactionReceipt receipt = asset.register(assetAccount, amount);
// transfer interface
TransactionReceipt receipt = asset.transfer(fromAssetAccount, toAssetAccount,␣
↪amount);
```

4.6 Running

So far we have introduced all the processes of the asset management application using the blockchain and implemented the functions. Then we can run the project and test whether the function is normal.

- compilation

```
# switch to project directory
$ cd ~/asset-app
# compile project
$ ./gradlew build
```

After the compilation is successful, the `dist` directory will be generated under the project root directory. There is an `asset_run.sh` script in the dist directory to simplify project operation. Now let's start by verifying the requirements set out in this article.

- deploy the `Asset.sol` contract

```
# enter dist directory
$ cd dist
$ bash asset_run.sh deploy
Deploy Asset successfully, contract address is␣
↪0xd09ad04220e40bb8666e885730c8c460091a4775
```

- register asset

```
$ bash asset_run.sh register Alice 100000
Register account successfully => account: Alice, value: 100000
$ bash asset_run.sh register Bob 100000
Register account successfully => account: Bob, value: 100000
```

- query asset

```
$ bash asset_run.sh query Alice
account Alice, value 100000
$ bash asset_run.sh query Bob
account Bob, value 100000
```

- transfer asset

```
$ bash asset_run.sh transfer Alice Bob  50000
Transfer successfully => from_account: Alice, to_account: Bob, amount: 50000
$ bash asset_run.sh query Alice
account Alice, value 50000
$ bash asset_run.sh query Bob
account Bob, value 150000
```

Summary: So far, we have built an application based on the FISCO BCOS Alliance blockchain through contract development, contract compilation, SDK configuration and business development.

CHAPTER

FIVE

MORE TUTORIALS

This chapter will introduce the basic process and related core concept for quick development of DApp on FISCO BCOS. We will also provide company users a toolkit tutorial for easier development and deployment.

CHAPTER

SIX

BUILD BLOCKCHAIN NETWORK

Setup and deploy blockchain

- Getting Executables
  - Download binary, use docker images and complie source code.
- Chain building script
  - Options and node directory
- Certificate description
  - Certificate format, role and generating process.
- Configuration files and configuration items
  - All configure files' options
- Deploy Multi-Group Blockchain System
  - The guide of deploying Multi-Group Blockchain
- Distributed storage
  - The guide of using distributed storage feature

Use Blockchain

- Console
  - Configuration and options
- Manage blockchain accounts
  - Account generation and using guide.
- SDK
  - The SDK to call smart from outside
- AMOP
  - Send messages between SDKs

Write smart contracts

- Smart contract development
  - Solidity smart contract and precompiled contract
- Parallel contract
  - The guide of writing parallel contract

Management and Security

- Group members management
  - Add/Remove members(nodes) of group
- Permission control
  - Access control among accounts.
- CA blacklist
  - Deny connection from certain node.
- Storage security
  - Encrypt data during writing into disk
- Privacy protection
  - Integrate homomorphic encryption and group/ring signature algorithms in precompiled contracts

Others

- OSCCA-approved cryptography
  - OSCCA-approved cryptography node and SDK

Important:

- Important features
  - Deploy Multi-Group Blockchain System
  - Parallel contract
  - Distributed storage

6.1 Hardware requirements

Note:

- FISCO BCOS supports x86_ 64 and aarch64 (ARM) architecture CPU
- Since multiple nodes share network bandwidth, CPU, and memory resources, it is not recommended to configure too much nodes on one machine in order to ensure the stability of service.

The following table is a recommended configuration for single-group and single-node. Node consumes resources in a linear relationship with the number of groups. You can configure the number of nodes reasonably according to actual business requirement and machine resource.

6.2 Supported Platforms

- CentOS 7.2+
- Ubuntu 18.04

- macOS 10.14+
- Kylin OS V10
- deepin

6.3 Getting Executables

Users can choose any of the following methods to get FISCO BCOS executable. It is recommended to download the precompiled binaries from GitHub.

- The official statically linked precompiled files can be used on Ubuntu 16.04 and CentOS 7.2 version or later.
- docker image is provided officially, welcome to use. docker-hub address
- You can compile from the source code, visit here source code compilation.

6.3.1 Downloading precompiled fisco-bcos

The statically linked precompiled executable provided has been tested on Ubuntu 16.04 and CentOS 7. Please download the latest released pre-compiled executable from the Release.

6.3.2 docker image

From v2.0.0 version, we provide the docker image for the tag version. Corresponding to the master branch, we provide image of `latest` tag. For more docker tags please refer tohere.

build_chain.sh script adds the `-d` option to provide docker mode building for developers to deploy. For details, please refer to here.

Note: For using build_chain.sh script easily, we start docker by using `--network=host` network mode. Users may need to customize and modify according to their own network scenarios when they actually use.

6.3.3 Source code compilation

Note: The source code compilation is suitable for the experienced developers. You are required to download all library dependencies during compilation. Network connection would required and would take 5-20 minutes in total.

FISCO BCOS is using generic CMake to generate platform-specific build files, which means the steps are similar for most operating systems:

1. Install build tools and dependent package (depends on platform).
2. Clone code from FISCO BCOS.
3. Run `cmake` to generate the build file and compile.

Installation dependencies

- Ubuntu

Ubuntu 16.04 or later is recommended. The versions below 16.04 have not been tested. You will require to have the build tools build tools and `libssl` for compiling the source code.

```
sudo apt install -y g++ libssl-dev openssl cmake git build-essential autoconf␣
↪texinfo flex patch bison libgmp-dev zlib1g-dev
```

- CentOS

CentOS7 version or later is recommended.

```
$ sudo yum install -y epel-release centos-release-scl
$ sudo yum install -y openssl-devel openssl cmake3 gcc-c++ git flex patch bison␣
↪gmp-static devtoolset-7
```

- macOS

xcode10 version and above are recommended. macOS dependent package installation depends on Homebrew.

```
brew install openssl git flex bison gmp
```

Code clone

```
git clone https://github.com/FISCO-BCOS/FISCO-BCOS.git

# If you have network issue for exec the command above, please try:
git clone https://gitee.com/FISCO-BCOS/FISCO-BCOS.git

cd FISCO-BCOS && git checkout master-2.0
```

Compile

After compilation, binary files are located at `FISCO-BCOS/build/bin/fisco-bcos`.

```
$ cd FISCO-BCOS
$ git checkout master-2.0
$ mkdir -p build && cd build
$ source /opt/rh/devtoolset-7/enable  # CentOS Please execute
# please use cmake3 for CentOS
$ cmake ..
#To add -j4 to accelerate compilation by 4 compilation processes
# In macOS, if it raises "ld: warning: direct access" when execute make command,␣
↪please ignore it
$ make
```

Note:

- If dependency libs cannot be downloaded for a long time due to network problems, try https://gitee.com/FISCO-BCOS/LargeFiles/tree/master/libs , and put in FISCO-BCOS/deps/src/

Compile options

- TESTS, off by default, unit test compilation flag. To enable it, use `cmake -DTESTS=on ..`
- DEMO, off by default, test program compilation switch. To open it through `cmake -DDEMO=on ...`
- TOOL, off by default, tools program compilation switch. To open it through`cmake -DTOOL=on ...`
- ARCH_NATIVE, off by default, optimize code according to local CPU architecture if on.
- BUILD_STATIC，off by default, static compilation switch, only supports Ubuntu. To open it through `cmake -DBUILD_STATIC=on ...`

- Generate source documentation.

```
# Install Doxygen
$ sudo apt install -y doxygen graphviz
# Generate source documentation locate at build/doc
$ make doc
```

6.4 Certificate description

FISCO BCOS network adopts a CA-oriented access mechanism to support any multi-level certificate structure for ensuring information confidentiality, authentication, integrity, and non-repudiation.

FISCO BCOS uses the x509 protocol certificate format. According to the existing business scenario, a three level certificate structure is adopted by default, and from top to bottom, the three levels are chain certificate, agency certificate, and node certificate respective.

In multi-group architecture, a chain has a chain certificate and a corresponding chain private key, and the chain private key is jointly managed by alliance chain committee. Alliance chain committee can use the agency's certificate request file `agency.csr` to issue the agency certificate `agency.crt`.

Agency private key held by the agency administrator can issue node certificate to the agency's subordinate nodes.

Node certificate is the credential of node identity and uses this certificate to establish an SSL connection with other nodes for encrypted communication.

sdk certificate is a voucher for sdk communicating with node. Agency generates sdk certificate that allows sdk to do that.

The files' suffixes of FISCO BCOS node running are described as follows:

6.4.1 Role definition

There are four roles in the FISCO BCOS certificate structure, namely the alliance chain committee administrator, agency, node, and SDK.

Alliance chain committee

- The alliance chain committee manages private key of chain, and issues agency certificate according to agency's certificate request document `agency.csr`.

```
ca.crt chain certificate
ca.key chain private key
```

When FISCO BCOS performs SSL encrypted communication, only the node with the same chain certificate `ca.crt` can establish a connection.

Agency

- Agency has an agency private key that can issue node certificate and SDK certificate.

```
ca.crt chain certificate
agency.crt agency certificate
agency.csr agency certificate request file
agency.key agency private key
```

Node/SDK

- FISCO BCOS nodes include node certificates and private keys for establishing SSL encrypted connection among nodes;
- SDK includes SDK certificate and private key for establishing SSL encrypted connection with blockchain nodes.

```
ca.crt #chain certificate
node.crt #node certificate
node.key #node private key
sdk.crt #SDK certificate
sdk.key #SDK private key
```

Node certificate `node.crt` includes the node certificate and the agency certificate information. When the node communicates with other nodes/SDKs, it will sign the message with its own private key `node.key`, and send its own `node.crt` to nodes/SDKs to verify.

6.4.2 Certificate generation process

FISCO BCOS certificate generation process is as follows. Users can also use the Enterprise Deployment Tool to generate corresponding certificate

Chain certificate generation

- Alliance chain committee uses openssl command to request chain private key `ca.key`, and generates chain certificate `ca.crt` according to ca.key.

Agency certificate generation

- Agency uses openssl command to generate agency private key `agency.key`
- Agency uses private key `agency.key` to get agency certificate request file `agency.csr`, and sends `agency.csr` to alliance chain committee.
- Alliance chain committee uses chain private key `ca.key` to generate the agency certificate `agency.crt` according to the agency certificate request file `agency.csr`. And send agency certificate `agency.crt` to corresponding agency.

Node/SDK certificate generation

- The node generates the private key `node.key` and the certificate request file `node.csr`. The agency administrator uses the private key `agency.key` and the certificate request file `node.csr` to issue the certificate to the node/SDK.

6.4.3 TODO

6.5 cfca

TODO

6.6 CA blacklist and whitelist

This documents tells you how to use CA blacklist and whitelist. Read the design of CA blacklist and whitelist for more.

6.6.1 CA blacklist

Use blacklist to reject connection coming from certain NodeIDs.

Configure blacklist

Modify `config.ini`

```
[certificate_blacklist]
    ; crl.0 should be nodeid, nodeid's length is 128
    ;crl.0=
```

Restart the node.

```
$ bash stop.sh && bash start.sh
```

Check connections.

```
$ curl -X POST --data '{"jsonrpc":"2.0","method":"getPeers","params":[1],"id":1}'␣
↪http://127.0.0.1:8545 |jq
```

6.6.2 CA whitelist

Use whitelist to reject all connections which NodeID is not belong to whitelist.

Configure whitelist

Modify`config.ini`, If whitelist is empty, the whitelist is disable and accept all connections.

```
[certificate_whitelist]
    ; cal.0 should be nodeid, nodeid's length is 128
    cal.
↪0=7718df20f0f7e27fdab97b3d69deebb6e289b07eb7799c7ba92fe2f43d2efb4c1250dd1f11fa5b5ce687c8283d650
    cal.
↪1=f306eb1066ceb9d46e3b77d2833a1bde2a9899cfc4d0433d64b01d03e79927aa60a40507c5739591b8122ee609cf5
```

If the node is not started, use `start.sh`. The whitelist configuration is loaded during starting up. If the node has started, use `reload_whitelist.sh` . The whitelist configureation is refresh and reject disconnect the node not belongs to whitelist.

```
# If the node is not started.
$ bash start.sh
# If the node is started.
$ cd scripts
$ bash reload_whitelist.sh
node_127.0.0.1_30300 is not running, use start.sh to start and enable whitelist␣
↪directlly.
```

Check connections.

```
$ curl -X POST --data '{"jsonrpc":"2.0","method":"getPeers","params":[1],"id":1}'␣
↪http://127.0.0.1:8545 |jq
```

6.6.3 Usage: Public CA

If we build chain using the certificate coming from CFCA. This chain's CA is CFCA. This CA is public and can be used to build by other blockchain. Cause different chain can connect each other by using a same CA. In this case, we need to use whitelist to reject the connection coming from other blockchain.

Build chain an enable whitelist

1. Build chain.
2. Get every nodes' NodeID.
3. Add every nodes' NodeID into every Node's whitelist.
4. Start the node or use `reload_whitelist.sh` to reconfigure whitelist.

Add node in blockchain and update whitelist

1. Build a node based on a blockchain.
2. Get NodeID of new node.
3. Append new node's NodeID into every Node's whitelist.
4. Copy old node's whitelist to new node.
5. Reconfigure whitelist using `reload_whitelist.sh`.
6. Start new node.
7. Add new node into group (addSealer or addObserver)

6.6.4 Examples

Build blockchain

Build a blockchain contains 4 nodes.

```
$ bash build_chain.sh -l 127.0.0.1:4
```

Get NodeIDs of these nodes.

```
$ cat node*/conf/node.nodeid
219b319ba7b2b3a1ecfa7130ea314410a52c537e6e7dda9da46dec492102aa5a43bad81679b6af0cd5b9feb7cfdc0b395
7718df20f0f7e27fdab97b3d69deebb6e289b07eb7799c7ba92fe2f43d2efb4c1250dd1f11fa5b5ce687c8283d65030aae
f306eb1066ceb9d46e3b77d2833a1bde2a9899cfc4d0433d64b01d03e79927aa60a40507c5739591b8122ee609cf5636e
38158ef34eb2d58ce1d31c8f3ef9f1fa829d0eb8ed1657f4b2a3ebd3265d44b243c69ffee0519c143dd67e91572ea8cb4
```

The NodeIDs are：

- node0: 219b319b....
- node1: 7718df20....
- node2: f306eb10....
- node3: 38158ef3....

Start all nodes.

```
$ cd node/127.0.0.1/
$ bash start_all.sh
```

Check connections. Use node0 as example. (8545 is the rpc port of node0).

```
$ curl -X POST --data '{"jsonrpc":"2.0","method":"getPeers","params":[1],"id":1}'␣
↪http://127.0.0.1:8545 |jq
```

We can see, node0 has connected with 3 nodes.


```
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": [
    {
      "Agency": "agency",
      "IPAndPort": "127.0.0.1:62774",
      "Node": "node3",
      "NodeID":
↪"38158ef34eb2d58ce1d31c8f3ef9f1fa829d0eb8ed1657f4b2a3ebd3265d44b243c69ffee0519c143dd67e91572ea8
↪",
      "Topic": []
    },
    {
      "Agency": "agency",
      "IPAndPort": "127.0.0.1:62766",
      "Node": "node1",
      "NodeID":
↪"7718df20f0f7e27fdab97b3d69deebb6e289b07eb7799c7ba92fe2f43d2efb4c1250dd1f11fa5b5ce687c8283d6503
↪",
      "Topic": []
    },
    {
      "Agency": "agency",
      "IPAndPort": "127.0.0.1:30302",
      "Node": "node2",
      "NodeID":
↪"f306eb1066ceb9d46e3b77d2833a1bde2a9899cfc4d0433d64b01d03e79927aa60a40507c5739591b8122ee609cf56
↪",
      "Topic": []
    }
  ]
}
```


Use blacklist: Reject connection from node1 to node0

Add node1's NodeID into node0's `config.ini`.

```
vim node0/config.ini
```

Like (Let whitelist empty):

```
[certificate_blacklist]
    ; crl.0 should be nodeid, nodeid's length is 128
    crl.
↪0=7718df20f0f7e27fdab97b3d69deebb6e289b07eb7799c7ba92fe2f43d2efb4c1250dd1f11fa5b5ce687c8283d650

[certificate_whitelist]
    ; cal.0 should be nodeid, nodeid's length is 128
    ; cal.0=
```

Restart to enable.

```
$ cd node0
$ bash stop.sh && bash start.sh
```

Check connections.

```
$ curl -X POST --data '{"jsonrpc":"2.0","method":"getPeers","params":[1],"id":1}'␣
↪http://127.0.0.1:8545 |jq
```

We can see node0 has connected with 2 nodes. Haven't connected with node1.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": [
    {
      "Agency": "agency",
      "IPAndPort": "127.0.0.1:30303",
      "Node": "node3",
      "NodeID":
↪"38158ef34eb2d58ce1d31c8f3ef9f1fa829d0eb8ed1657f4b2a3ebd3265d44b243c69ffee0519c143dd67e91572ea8
↪",
      "Topic": []
    },
    {
      "Agency": "agency",
      "IPAndPort": "127.0.0.1:30302",
      "Node": "node2",
      "NodeID":
↪"f306eb1066ceb9d46e3b77d2833a1bde2a9899cfc4d0433d64b01d03e79927aa60a40507c5739591b8122ee609cf56
↪",
      "Topic": []
    }
  ]
}
```

Use whitelist: Reject all connection except node1 and node 2

Add NodeIDs of node1 and node2 into node0's `config.ini`.

```
$ vim node0/config.ini
```

Let blacklist empty and add NodeIDs of node1 and node2 into whitelist

```
[certificate_blacklist]
    ; crl.0 should be nodeid, nodeid's length is 128
    ;crl.0=

[certificate_whitelist]
    ; cal.0 should be nodeid, nodeid's length is 128
    cal.
↪0=7718df20f0f7e27fdab97b3d69deebb6e289b07eb7799c7ba92fe2f43d2efb4c1250dd1f11fa5b5ce687c8283d650
    cal.
↪1=f306eb1066ceb9d46e3b77d2833a1bde2a9899cfc4d0433d64b01d03e79927aa60a40507c5739591b8122ee609cf5
```

Restart the node.

```
$ bash stop.sh && bash start.sh
```

Check connections.

```
$ curl -X POST --data '{"jsonrpc":"2.0","method":"getPeers","params":[1],"id":1}'␣
↪http://127.0.0.1:8545 |jq
```

We can see, the node0 has connected with 2 nodes, not connected with node1.

```
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": [
    {
      "Agency": "agency",
      "IPAndPort": "127.0.0.1:30302",
      "Node": "node2",
      "NodeID":
↪"f306eb1066ceb9d46e3b77d2833a1bde2a9899cfc4d0433d64b01d03e79927aa60a40507c5739591b8122ee609cf563
↪",
      "Topic": []
    },
    {
      "Agency": "agency",
      "IPAndPort": "127.0.0.1:30301",
      "Node": "node1",
      "NodeID":
↪"7718df20f0f7e27fdab97b3d69deebb6e289b07eb7799c7ba92fe2f43d2efb4c1250dd1f11fa5b5ce687c8283d65030
↪",
      "Topic": []
    }
  ]
}
```


Use blacklist and whitelist: The priority of blacklist is higher than whitelist

Modify `config.ini` of node0.

```
$ vim node0/config.ini
```

Add node1 into blacklist and add node1, node2 into whitelist. Both blacklist and whitelist has configured node1.

```
[certificate_blacklist]
    ; crl.0 should be nodeid, nodeid's length is 128
    crl.
↪0=7718df20f0f7e27fdab97b3d69deebb6e289b07eb7799c7ba92fe2f43d2efb4c1250dd1f11fa5b5ce687c8283d6503

[certificate_whitelist]
    ; cal.0 should be nodeid, nodeid's length is 128
    cal.
↪0=7718df20f0f7e27fdab97b3d69deebb6e289b07eb7799c7ba92fe2f43d2efb4c1250dd1f11fa5b5ce687c8283d6503
    cal.
↪1=f306eb1066ceb9d46e3b77d2833a1bde2a9899cfc4d0433d64b01d03e79927aa60a40507c5739591b8122ee609cf5
```

Restart the node.

```
$ bash stop.sh && bash start.sh
```

Check connections.

```
$ curl -X POST --data '{"jsonrpc":"2.0","method":"getPeers","params":[1],"id":1}'␣
↪http://127.0.0.1:8545 |jq
```

We can see. Although node1 has been added into whitelist, node0 hasn't connected with node1. Because node1 has also added into blacklist. The priority of blacklist is higher than whitelist.


```
{
  "id": 1,
  "jsonrpc": "2.0",
```


(continues on next page)

(continued from previous page)


```
  "result": [
    {
      "Agency": "agency",
      "IPAndPort": "127.0.0.1:30302",
      "Node": "node2",
      "NodeID":
↪"f306eb1066ceb9d46e3b77d2833a1bde2a9899cfc4d0433d64b01d03e79927aa60a40507c5739591b8122ee609cf56
↪",
      "Topic": []
    }
  ]
}
```


6.7 Configuration files and configuration items

FISCO BCOS supports multiple ledger. Each chain includes multiple unique ledgers, whose data among them are isolated from each other. And the transaction processing among groups are also isolated. Each node includes a main configuration `config.ini` and multiple ledger configurations `group.group_id.genesis`, `group.group_id.ini`.

- `config.ini`: The main configuration file, mainly configures with RPC, P2P, SSL certificate, ledger configuration file path, compatibility and other information.
- `group.group_id.genesis`: group configurations file. All nodes in the group are consistent. After node launches, you cannot manually change the configuration including items like group consensus algorithm, storage type, and maximum gas limit, etc.
- `group.group_id.ini`: group variable configuration file, including the transaction pool size, etc.. All configuration changes are effective after node restarts.

6.7.1 Main configuration file config.ini

`config.ini` uses `ini` format. It mainly includes the configuration items like ** rpc, p2p, group, secure and log **.

Important:

- The public IP addresses of the cloud host are virtual IP addresses. If listen_ip/jsonrpc_listen_ip/channel_listen_ip is filled in external network IP address, the binding fails. You must fill in 0.0.0.0.
- RPC/P2P/Channel listening port must be in the range of 1024-65535 and cannot conflict with other application listening ports on the machine.
- In order to facilitate development and experience, the reference configuration of listen_ip/channel_listen_ip is 0.0.0.0. For security reasons, please modify it to a safe listening address according to the actual business network situation, such as the internal IP or a specific external IP

Configure RPC

- `channel_listen_ip`: Channel listening IP, to facilitate node and SDK cross-machine deployment, the default setting is `0.0.0.0`;
- `jsonrpc_listen_ip`: RPC listening IP, security considerations, the default setting is 127.0.0.1, if there is an external network access requirement, please monitor node external network IP or `0.0.0.0`;

- `channel_listen_port`: Channel port, is corresponding to `channel_listen_port` in [Java SDK]](../sdk/sdk.html#id2) configuration;
- `jsonrpc_listen_port`: JSON-RPC port.

Note: For security and ease of use consideration, the latest configuration of v2.3.0 version splits listen_ip into jsonrpc_listen_ip and channel_listen_ip, but still retains the parsing function of listen_ip:

- Include only listen_ip in the configuration: The listening IPs of both RPC and Channel are configured listen_ip
- The configuration also contains listen_ip, channel_listen_ip, or jsonrpc_listen_ip: Priority is given to channel_listen_ip and jsonrpc_listen_ip. Configuration items that are not configured are replaced with the value of listen_ip
- Starting from v2.6.0, RPC module support IPV6.

RPC configuration example is as follows:

```
# ipv4
[rpc]
    channel_listen_ip=0.0.0.0
    jsonrpc_listen_ip=127.0.0.1
    channel_listen_port=30301
    jsonrpc_listen_port=30302

# ipv6
[rpc]
    channel_listen_ip=::1
    jsonrpc_listen_ip=::1
    channel_listen_port=30301
    jsonrpc_listen_port=30302
```

Configure P2P

Note:

- In order to facilitate development and experience, the reference configuration of listen_ip is 0.0.0.0. For security reasons, please modify it to a safe listening address according to the actual business network situation, such as the internal IP or a specific external IP.
- Starting from v2.6.0, P2P module support IPV6.

The current version of FISCO BCOS must be configured with `IP` and `Port` of the connection node in the `config.ini` configuration. The P2P related configurations include:

- `listen_ip`: P2P listens for IP, to set `0.0.0.0` by default.
- `listen_port`: Node P2P listening port.
- `node.*`: All nodes' `IP:Port` or `DomainName:Port` which need to be connected to node. This option supports domain names, but suggests users who need to use it manually compile source code.
- `enable_compress`: Enable network compression configuration option. Configuring to true, indicates that network compression is enabled. Configuring to false, indicates that network compression is disabled. For details on network compression, please refer to [here](../design/features/network_compress .md).

P2P configuration example is as follows:

```
# ipv4
[p2p]
    listen_ip=0.0.0.0
    listen_port=30300
    node.0=127.0.0.1:30300
    node.1=127.0.0.1:30304
    node.2=127.0.0.1:30308
    node.3=127.0.0.1:30312

# ipv6
[p2p]
    listen_ip=::1
    listen_port=30300
    node.0=[::1]:30300
    node.1=[::1]:30304
    node.2=[::1]:30308
    node.3=[::1]:30312
```

Configure ledger file path

`[group]` To configure all group configuration paths which this node belongs:

- `group_data_path`: Group data storage path.
- `group_config_path`: Group configuration file path.

  Node launches group according to all `.genesis` suffix files in the `group_config_path` path.

```
[group]
    ; All group data is placed in the node's data subdirectory
    group_data_path=data/
    ; Program automatically loads all .genesis files in the path
    group_config_path=conf/
```

Configure certificate information

For security reasons, communication among FISCO BCOS nodes uses SSL encrypted communication. `[network_security]` configure to SSL connection certificate information:

- `data_path`: Directory where the certificate and private key file are located.
- `key`: The `data_path` path that node private key relative to.
- `cert`: The `data_path` path that certificate `node.crt` relative to.
- `ca_cert`: ca certificate file path.
- `ca_path`: ca certificate folder, required for multiple ca.
- `check_cert_issuer`: sets whether the SDK can only connect the nodes with same organization, which is turned on by default（check_cert_issuer=true）.

```
[network_security]
    data_path=conf/
    key=node.key
    cert=node.crt
    ca_cert=ca.crt
    ;ca_path=
```

Configure blacklist

For preventing vice, FISCO BCOS allows nodes to configure untrusted node blacklist to reject establishing connections with these blacklist nodes. To configure blacklist through `[crl]`:

> `crl.idx`: Blacklist node's Node ID, can get from `node.nodeid` file; `idx` is index of the blacklist node.

For details of the blacklist, refer to [CA Blacklist].(./certificate_list.md)

Blacklist configuration example is as follows:

```
; certificate blacklist
[crl]
    crl.
↪0=4d9752efbb1de1253d1d463a934d34230398e787b3112805728525ed5b9d2ba29e4ad92c6fcde5156ede8baa5aca3
3787c338a4
```

Configure log information

FISCO BCOS supports boostlog. The log configuration is mainly located in the `[log]` configuration item of `config.ini`.

Log common configuration items

FISCO BCOS general log configuration items are as follows:

- `enable`: Enable/disable log. Set to `true` to enable log; set to `false` to disable log. set to true by default. For performance test, to set this option to `false` to reduce the impact of print log on test results
- `log_path`:log file patch.
- `level`: log level, currently includes 5 levels which are `trace`、`debug`、`info`、`warning`、`error`.After setting a certain log level, the log file will be entered with a log equal to or larger than this level.The log level is sorted from large to small by `error > warning > info > debug > trace`.
- `max_log_file_size`: Maximum size per log file, ** unit of measure is bytes, default is 200MB**
- `flush`: boostlog enables log auto-refresh by default. To improves system performance, it is recommended to set this value to false.

boostlog configuration example is as follows:

```
[log]
    ; whether to enable log, set to true by default
    enable=true
    log_path=./log
    level=info
    ; Maximum size per log file, default is 200MB
    max_log_file_size=200
    flush=true
```

Statistics log configuration items

Considering that the real-time monitoring system resource usage is very important in the actual production system, FISCO BCOS v2.4.0 introduced statistical logs, and the statistical log configuration items are located in `config.ini`.

Statistics log enable/disable configuration item

Considering that not all scenarios require network traffic and Gas statistics functions, FISCO BCOS provides the `enable_statistic` option in`config.ini` to turn on and off the function, which is turned off by default.

- `log.enable_statistic` is set to true to enable network traffic and gas statistics
- `log.enable_statistic` is set to false to disable network traffic and gas statistics

The configuration example is as follows:

```
[log]
    ; enable/disable the statistics function
    enable_statistic=false
```

Network statistics log output interval configuration item

Due to the periodic output of network statistics logs, `log.stat_flush_interval` is introduced to control the statistics interval and log output frequency, the unit is seconds, and the default is 60s. The configuration example is as follows:

```
[log]
    ; network statistics interval, unit is second, default is 60s
    stat_flush_interval=60
```

Configure chain attributes

Users can configure attributes of chain through `[chain]` in `config.ini`. The tool will be automatically generated when changing the configuration item to build chain, so users do not need to change it.

- `id`, the ID of chain, 1 by default;
- `sm_crypto`, in 2.5.0 and follow-up versions of FISCO BCOS, node can be launched in SM-Crypto mode or not through this configuration. `true` means SM-Crypto mode will be used and `false` means opposite, `false` by default;
- `sm_crypto_channel`, in 2.5.0 and follow-up versions of FISCO BCOS, connection between SDK and node can be established via SM-SSL. This configuration is used to indicate wheather to use this feature, `false` by default。

Configure node compatibility

All versions of FISCO BCOS 2.0+ are forward compatible. You can configure the compatibility of node through `[compatibility]` in `config.ini`. The tool will be automatically generated when changing the configuration item to build chain, so users do not need to change it.

- `supported_version`: The version of the current node running

Important:

- view the latest version of FISCO BCOS currently supports through the command ‘./fisco-bcos –version | grep “Version” ‘
- In the blockchain node configuration generated by build_chain.sh, supported_version is configured to the current latest version of FISCO BCOS
- When upgrading an old node to a new node, directly replace the old FISCO BCOS binary with the latest FISCO BCOS binary, don’t modify supported_version

`FISCO BCOS 2.2.0` node's `[compatibility]` configuration is as follows:

```
[compatibility]
    supported_version=2.2.0
```

Optional configuration: Disk encryption

In order to protect node data, FISCO BCOS introduces Disk Encryption to ensure confidentiality. Disk Encryption Operation Manual Reference.

`storage_security` in `config.ini` is used to configure disk encryption. It mainly includes (for the operation of the disk encryption, please refer to Operation Manual):

- `enable`: whether to launch disk encryption, not to launch by default;
- `key_manager_ip`: Key Managerservice's deployment IP;
- `key_manager_port`: Key Managerservice's listening port;
- `cipher_data_key`: ciphertext of node data encryption key. For `cipher_data_key` generation, refer to disk encryption operation manual.

disk encryption configuration example is as follows:

```
[storage_security]
enable=true
key_manager_ip=127.0.0.1
key_manager_port=8150
cipher_data_key=ed157f4588b86d61a2e1745efe71e6ea
```

Optional configuration: flow control

In order to realize the flexible service of the blockchain system and prevent the mutual influence of resources between multiple groups, FISCO BCOS v2.5.0 introduces a flow control function, mainly including the request rate limit from SDK to nodes and the flow limit between nodes. Under `[flow_control]` of `config.ini`, it is disabled by default. For detailed design of flow control, please refer to here

SDK request rate limit configuration

The SDK request rate limit is located in the configuration item `[flow_control].limit_req`, which is used to limit the maximum number of requests from the SDK to the node per second. When the request to the node per second exceeds the value of the configuration item, the request will be rejected. The rate limit is disabled by default. To enable this function, you need to remove the `;` in front of the `limit_req` configuration item. Enable the SDK request rate limit and design a node that can accept 2000 SDK requests per second as follows:

```
[flow_control]
    ; restrict QPS of the node
    limit_req=2000
```

Inter-node traffic limit configuration

In order to prevent block sync and AMOP message transmission from occupying too much network traffic and affecting the transmission of message packets of the consensus module, FISCO BCOS v2.5.0 introduces the function of inter-node traffic restriction. This configuration item is used to configure the average bandwidth of the node, but does not limit the flow of block consensus and transaction sync. When the average bandwidth of the node exceeds the configured value, block sync and AMOP message transmission will be paused.

- `[flow_control].outgoing_bandwidth_limit`: Node output bandwidth limit, the unit is `Mbit/s`, When the node output bandwidth exceeds this value, block sync will be paused, and the[AMOP](./amop_protocol.md) request sent by the client will be rejected, but It will not limit the traffic of block consensus and transaction broadcast. This configuration item is disabled by default. To enable the traffic limit function, please remove the `;` in front of the `outgoing_bandwidth_limit` configuration item.

The configuration example of enable the outgoing bandwidth traffic limit of the node and setting it to `5MBit/s` is as follows:

```
[flow_control]
    ; Mb, can be a decimal
    ; when the outgoing bandwidth exceeds the limit, the block synchronization␣
↪operation will not proceed
    outgoing_bandwidth_limit=5
```

6.7.2 Group system configuration instruction

Each group has unique separate configuration file, which can be divided into group system configuration and group variable configuration according to whether it can be changed after launch. group system configuration is generally located in the `.genesis` suffix configuration file in node's `conf` directory.

For example:`group1` system configuration generally names as `group.1.genesis`. Group system configuration mainly includes the related configuration of group ID、consensus, storage and gas.

Important: When configuring the system configuration, you need to pay attention to:

- configuration group must be consistent: group system configuration is used to generate the genesis block (block 0), so the configurations of all nodes in the group must be consistent.
- node cannot be modified after launching： system configuration has been written to the system table as genesis block, so it cannot be modified after chain initializes.
- After chain is initialized, even if genesis configuration is modified, new configuration will not take effect, and system still uses the genesis configuration when initializing the chain.
- Since genesis configuration requires all nodes in the group to be consistent, it is recommended to use build_chain to generate the configuration.

Group configuration

`[group]`configures group ID. Node initializes the group according to the group ID.

group2's configuration example is as follows:

```
[group]
id=2
```

Consensus configuration

`[consensus]` involves consensus-related configuration, including:

- `consensus_type`: consensus algorithm type, currently supports PBFT, Raft and rPBFT. To use PBFT by default;
- `max_trans_num`: a maximum number of transactions that can be packed in a block. The default is 1000. After the chain is initialized, the parameter can be dynamically adjusted through Console;
- `consensus_timeout`: In the PBFT consensus process, the timeout period of each block execution, the default is 3s, the unit is seconds, the parameter can be dynamically adjusted through Console;

- `node.idx`: consensus node list, has configured with the [Node ID] of the participating consensus nodes. The Node ID can be obtained by the `${data_path}/node.nodeid` file (where `${data_path}` can be obtained by the configuration item `[secure].data_path` of the main configuration `config.ini`)

FISCO BCOS v2.3.0 introduced the rPBFT consensus algorithm, The rPBFT related configuration is as follows:

- `epoch_sealer_num`: The number of nodes participating in the consensus is selected in a consensus period. The default is the total number of all consensus nodes. After the chain is initialized, this parameter can be dynamically adjusted through [Console] (../console/console.html#setsystemconfigbykey)
- `epoch_block_num`: The number of blocks generated in a consensus period, the default is 1000, which can be dynamically adjusted through [Console] (../console/console.html#setsystemconfigbykey)

Note: rPBFT configuration does not take effect on other consensus algorithms

```
; Consensus protocol configuration
[consensus]
    ;consensus algorithm, currently supports PBFT(consensus_type=pbft) and␣
↪Raft(consensus_type=raft)
    consensus_type=pbft
    ; maximum number of transactions in a block
    max_trans_num=1000
    epoch_sealer_num=4
    epoch_block_num=1000
    ; leader node's ID lists
    node.
↪0=123d24a998b54b31f7602972b83d899b5176add03369395e53a5f60c303acb719ec0718ef1ed51feb7e9cf4836f26
e01789233a
    node.
↪1=70ee8e4bf85eccda9529a8daf5689410ff771ec72fc4322c431d67689efbd6fbd474cb7dc7435f63fa592b98f22b1
3494db8776
    node.
↪2=7a056eb611a43bae685efd86d4841bc65aefafbf20d8c8f6028031d67af27c36c5767c9c79cff201769ed80ff220b
922aa0ef50
    node.
↪3=fd6e0bfe509078e273c0b3e23639374f0552b512c2bea1b2d3743012b7fed8a9dec7b47c57090fa6dcc5341922c32
4a5aed2b4a
```

State mode configuration

state is used to store blockchain status information. It locates in the genesis file `[state]`:

- `type`: state type, currently supports storage state and MPT state, defaults to Storage state. storage state storing the transaction execution result in the system table, which is more efficient. MPT state storing the transaction execution result in the MPT tree, which is inefficient but contains complete historical information.

Important: The storage state is recommended.

```
[state]
    type=storage
```

Gas configuration

FISCO BCOS is compatible with Ethereum virtual machine (EVM). In order to prevent DOS from attacking EVM, EVM introduces the concept of gas when executing transactions, which is used to measure the computing and storage resources consumed during the execution of smart contracts. The measure includes the maximum gas

limit of transaction and block. If the gas consumed by the transaction or block execution exceeds the gas limit, the transaction or block is discarded.

FISCO BCOS is alliance chain that simplifies gas design. It retains only maximum gas limit of transaction, and maximum gas of block is constrained together by consensus configuration max_trans_num and transaction maximum gas limit.

FISCO BCOS configures maximum gas limit of the transaction through genesis `[tx].gas_limit`. The default value is 300000000. After chain is initialized, the gas limit can be dynamically adjusted through the console command.

```
[tx]
    gas_limit=300000000
```

EVM configuration

FISCO BCOS v2.4.0 introduces the `Free Storage` Gas measurement mode to increase the proportion of CPU and memory in Gas consumption. For details, please refer to [here] (../design/virtual_machine/gas.html#evm-gas) The opening and closing of `Free Storage` Gas mode is controlled by the `evm.enable_free_storage` configuration item in the `genesis` file.

Note:

- `evm.enable_free_storage` is supported in v2.4.0. This feature is not supported when `supported_version` is less than v2.4.0, or the old chain directly replaces binary upgrade
- When the chain is initialized, `evm.enable_free_storage` is written to the genesis block; after the chain is initialized, the node reads the `evm.enable_free_storage` configuration item from the genesis block, manually modifying the `genesis` configuration item will not take effect
- `evm.enable_free_storage` is set to false by default

- `evm.enable_free_storage` is set to true: enable `Free Storage` Gas mode
- `evm.enable_free_storage` is set to false: turn off `Free Storage` Gas mode

The configuration example is as follows:

```
[evm]
    enable_free_storage=false
```

6.7.3 Ledger variable configuration instruction

Variable configuration of the ledger is located in the file of the `.ini` suffix in the node `conf` directory.

For example: `group1` variable configuration is generally named `group.1.ini`. Variable configuration mainly includes transaction pool size, PBFT consensus message forwarding TTL, PBFT consensus packing time setting, PBFT transaction packaging dynamic adjustment setting, parallel transaction settings, etc..

Configure storage

Storage currently supports three modes: RocksDB, MySQL, and Scalable. Users can choose the DB to use according to their needs. RocksDB has the highest performance. MySQL supports users to use MySQL database for viewing data. Since the RC3 version, we have used RocksDB instead of LevelDB for better performance, but still supports LevelDB.

Note:

- Starting from v2.3.0, in order to facilitate chain maintenance, it is recommended to use MySQL storage mode instead of External storage mode

---

Public configuration item

Important: If you want to use MySQL, please set type to MySQL.

- `type`: The stored DB type, which supports `RocksDB`, `MySQL` and `External`. When the DB type is RocksDB, all the data of blockchain system is stored in the RocksDB local database; when the type is `MySQL`, the node accesses MySQL database according to the configuration. All data of blockchain system is stored in MySQL database. For accessing MySQL database, to configure the amdb-proxy. Please refer to here for the amdb-proxy configuration.
- `max_capacity`: configures the space size of the node that is allowed to use for memory caching.
- `max_forward_block`: configures the space size of the node that allowed to use for memory block. When the blocks exceeds this value, the node stops the consensus and waits for the blocks to be written to database.
- `binary_log`: default is false. when set to `true`, enable binary log, and then disable the wal of RocksDB.
- `cached_storage`: controls whether to use the cache. The default is `true`.

Database related configuration item

- `topic`: When the type is `External`, you need to configure this field to indicate the amdb-proxy topic that blockchain system is interested in. For details, please refer to here.
- `max_retry`: When the type is `External`, you need to configure this field to indicate the number of retries when writing fails. For details, please refer to here.
- `scroll_threshold_multiple`: when the type is `Scalable`, this configuration item is used to configure the handover threshold of the block database. The default value is 2, so Block data is stored in different RocksDB instances every 2000 blocks.
- `db_ip`: When the type is `MySQL`, you need to configure this field to indicate the IP address of MySQL.
- `db_port`: When the type is `MySQL`, you need to configure this field to indicate the port number of MySQL.
- `db_username`: When the type is `MySQL`, you need to configure this field to indicate the MySQL username.
- `db_passwd`: When the type is `MySQL`, you need to configure this field to indicate the password corresponding to the MySQL user.
- `db_name`: When the type is `MySQL`, you need to configure this field to indicate the database name used in MySQL.
- `init_connections`: When the type is `MySQL`, this field can be optionally configured to indicate the initial number of connections established with MySQL. The default value is 15, and it is fine to use it.
- `max_connections`: When the type is `MySQL`, this field can be optionally configured to indicate the maximum number of connections established with MySQL. The default value is 20, and it is fine to use it.

The following is an example of the configuration of [storage]:

```
[storage]
    ; storage db type, RocksDB / MySQL / external, RocksDB is recommended
    type=RocksDB
    max_capacity=256
    max_forward_block=10
    ; only for external
    max_retry=100
    topic=DB
    ; only for MySQL
    db_ip=127.0.0.1
    db_port=3306
    db_username=
    db_passwd=
    db_name=
```

Transaction pool configuration

FISCO BCOS opens the transaction pool capacity configuration to users. Users can dynamically adjust the transaction pool according to their business size requirements, stability requirements, and node hardware configuration.

Transaction pool capacity limit

In order to prevent excessive accumulating transactions occupy too much memory, FISCO BCOS provides two configuration items `[tx_pool].limit` and `[tx_pool].memory_limit` to limit the transaction pool capacity:

- `[tx_pool].limit`: limit the maximum number of transactions that can be accommodated in the transaction pool. The default is `150000`, after the limit is exceeded, transactions sent by the client to the node will be rejected
- `[tx_pool].memory_limit`: The memory size limit of transactions in the transaction pool, the default is `512MB`, after this limit is exceeded, the transaction sent by the client to the node will be rejected

The transaction pool capacity is configured as follows:

```
[tx_pool]
    limit=150000
    ; transaction pool memory size limit, MB
    memory_limit=512
```

Transaction pool push thread number configuration

In order to improve the performance of the blockchain system, FISCO BCOS uses the asynchronous push logic of transaction receipts. When the transaction is chained, the push thread in the transaction pool will asynchronously push the receipt of the transaction on the chain to the client. More system resources, and in order to prevent too few push threads from affecting the timeliness of transaction push, FISCO BCOS provides `[tx_pool].notify_worker_num` configuration item to configure the number of asynchronous push threads:

- `[tx_pool].notify_worker_num`: Number of asynchronous push threads, the default is 2, it is recommended that the value does not exceed 8

The number of push threads in the transaction pool is configured as follows:

```
[tx_pool]
    ; number of threads responsible for transaction notification,
    ; default is 2, not recommended for more than 8
    notify_worker_num=2
```

Transaction Expiration Configuration

In order to prevent the txs from pending in the transaction pool for too long when the system is abnormal, FISCO BCOS 2.9.0 introduces the transaction expiration time configuration `txs_expiration_time`. When the txs pending in the transaction pool exceeds `txs_expiration_time`, the transaction pool will actively clear the transaction.

- `[tx_pool].txs_expiration_time`: Transaction expiration time, the default is 10 minutes, the value is required to be not less than the consensus_timeout.

An example of txs_expiration_time configuration is as follows:

```
; transaction expiration time, in seconds, default is 10 minute
txs_expiration_time=600
```

PBFT consensus configurations

In order to improve the performance, availability, and network efficiency of the PBFT algorithm, FISCO BCOS has made a series of optimizations for block packaging algorithms and networks, including PBFT block packaging dynamic adjustment strategies, PBFT message forwarding optimization, and PBFT Prepare packet structure optimization.

Note: Due to protocol and algorithm consistency requirements, it is recommended to ensure that the PBFT consensus configuration of all nodes is consistent.

PBFT consensus message broadcast configuration

In order to ensure the maximum network fault tolerance of the consensus process, each consensus node broadcasts the message to other nodes after receiving a valid consensus message. In smooth network environment, the consensus message forwarding mechanism will waste additional network bandwidth, so the `ttl` is introduced in the group variable configuration item to control the maximum number of message forwarding. The maximum number of message forwarding is `ttl-1`, and the configuration item is valid only for PBFT.

Setting consensus message to be forwarded at most once configuration example is as follows:

```
; the ttl for broadcasting pbft message
[consensus]
ttl=2
```

PBFT consensus packing time configuration

The PBFT module packing too fast causes only 1 to 2 transactions to be pack in some blocks. For avoiding wasting storage space, FISCO BCOS v2.0.0-rc2 introduces `min_block_generation_time` configuration item in the group variable configuration `group.group_id.ini`'s `[consensus]` to manager the minimum time for PBFT consensus packing. That is, when the consensus node packing time exceeds `min_block_generation_time` and the number of packaged transactions is greater than 0, the consensus process will start and handle the new block generated by the package.

Important:

- `min_block_generation_time` is 500ms by default
- The longest packing time of consensus node is 1000ms. If the time is exceeded 1000ms and the number of transactions packed in the new block is still 0, the consensus module will enter the logic of empty block generation, and the empty block will not be written to disk;

- `min_block_generation_time` cannot exceed the time of empty block generation which is 1000ms. If the set value exceeds 1000ms, the system defaults min_block_generation_time to be 500ms.

```
[consensus]
;min block generation time(ms), the max block generation time is 1000 ms
min_block_generation_time=500
```

PBFT transaction package dynamic adjustment

For the impact causing by CPU loading and network latency on system processing power, PBFT provides an algorithm that dynamically adjusts the maximum number of transactions that can be packed in a block. The algorithm dynamically can adjust the maximum number of transactions according to the state of historical transaction processing. The algorithm is turned on by default, and it can be turned off by changing the `[consensus].enable_dynamic_block_size` configuration item of the variable configuration `group.group_id.ini` to `false`. At this time, the maximum number of transactions in the block is the `[consensus].max_trans_num` of `group.group_id.genesis`.

The configuration of closing the dynamic adjustment algorithm for the block package transaction number is as follows:

```
[consensus]
    enable_dynamic_block_size=false
```

PBFT message forwarding configuration

FISCO BCOS v2.2.0 optimizes the PBFT message forwarding mechanism to ensure that PBFT message packets can reach each consensus node as much as possible in the network disconnection scenario, while reducing redundant PBFT message packets in the network. For PBFT message forwarding optimization strategies. You can use the `[consensus].enable_ttl_optimization` configuration item of `group.group_id.ini` to enable or disable the PBFT message forwarding optimization strategy.

- `[consensus].enable_ttl_optimization` is configured as `true`: Enable PBFT message forwarding optimization strategy
- `[consensus].enable_ttl_optimization` is configured as `false`: Disable PBFT message forwarding optimization strategy
- When `supported_version` is not less than v2.2.0, the PBFT message forwarding strategy is enabled by default; when`supported_version` is less than v2.2.0, the PBFT message forwarding optimization strategy is disabled by default

Disable PBFT message forwarding optimization strategy configuration as follows:

```
[consensus]
    enable_ttl_optimization=false
```

PBFT Prepare package structure optimization

Considering that in the PBFT algorithm, transactions in blocks in the Prepare packet broadcast by the Leader have a high probability of hitting in the transaction pools of other consensus nodes. In order to save network bandwidth, FISCO BCOS v2.2.0 has optimized the Prepare packet structure: The block only contains a list of transaction hashes. After other consensus nodes receive the Prepare packet, they will first obtain the hit transaction from the local transaction pool and request the missing transaction from Leader. This policy can be enabled or disabled through the `[consensus].enable_prepare_with_txsHash` configuration item of `group.group_id.ini`.

- `[consensus].enable_prepare_with_txsHash` is configured as `true`: Enable the structure optimization of the Prepare package. The blocks in the Prepare message package only contain the transaction hash list.
- `[consensus].enable_prepare_with_txsHash` is configured as `false`: Disable the structure optimization of the Prepare packet, the block in the Prepare message packet contains the full amount of transactions
- When `supported_version` is not less than v2.2.0, `[consensus].enable_prepare_with_txsHash` defaults to `true`; when `supported_version` is less than v2.2.0, `[consensus].enable_prepare_with_txsHash` defaults to `false`

Note: Due to protocol consistency requirements, all nodes must ensure enable_prepare_with_txsHash configuration is consistent

Disable the PBFT Prepare package structure optimization configuration as follows:

```
[consensus]
    enable_prepare_with_txsHash=false
```

rPBFT consensus configurations

FISCO BCOS v2.3.0 introduces the rPBFT consensus algorithm. In order to ensure the load balance of the network traffic of the rPBFT algorithm, the tree broadcast policy of the Prepare packet is introduced, Corresponding fault tolerance scheme.

- `[consensus].broadcast_prepare_by_tree`: Enable/disable switch for Prepare tree broadcast policy. Set to `true` to enable the tree broadcast policy for Prepare packets. Set to`false` to disable the tree broadcast policy for Prepare packets. Default is `true`.

The following is the fault-tolerant configuration after the Prepare packet tree broadcast policy is enabled:

- `[consensus].prepare_status_broadcast_percent`: The percentage of the randomly selected nodes that receive the prepare status, The value ranges from 25 to 100, and the default is 33.
- `[consensus].max_request_prepare_waitTime`: When the node's Prepare cache is missing, the longest delay for waiting for the parent node to send a Prepare packet is 100ms by default. After this delay, the node will request from other nodes that own the Prepare packet.

The following is the configuration of load balancing after enabling Prepare package structure optimization in rPBFT mode:

- `[consensus].max_request_missedTxs_waitTime`: After the transaction in the node's Prepare packet is missing, the longest delay for waiting for the parent node or other non-leader node to synchronize the Prepare packet status is 100ms by default, if the packet status is synchronized to the parent node or non-leader node within the waiting delay window, a random node will be selected to request the missing transaction, otherwise, directly request the missing transaction from the leader.

rPBFT default configuration is as follows:

```
; Tree broadcast policy for Prepare packets is enabled by default
broadcast_prepare_by_tree=true
; Only effective when the prepare package tree broadcast is enabled
; Each node randomly selects 33% consensus nodes to synchronize the prepare packet␣
↪status
prepare_status_broadcast_percent=33
; Under the prepare package tree broadcast strategy,
; the node missing the prepare package takes more than 100ms and
; does not wait for the prepare package forwarded by the parent node
; to request the missing prepare package from other nodes.
max_request_prepare_waitTime=100
```

(continues on next page)

(continued from previous page)

```
; The maximum delay for a node to wait for a parent node
;or other non-leader node to synchronize a prepare packet is 100ms
max_request_missedTxs_waitTime=100
```

Sync configurations

The synchronization module is a "big network consumer", including block synchronization and transaction synchronization. FISCO BCOS optimizes the efficiency of the consensus module network using the principle of load balancing.

Note: Due to protocol consistency requirements, it is recommended to ensure that the PBFT consensus configuration of all nodes is consistent.

Block synchronization optimization configuration

In order to enhance the scalability of the blockchain system under the condition of limited network bandwidth, FISCO BCOS v2.2.0 has optimized block synchronization. For detailed optimization strategies. You can use the `[sync].sync_block_by_tree` of `group.group_id.ini` to enable or disable the block synchronization optimization strategy.

- `[sync].sync_block_by_tree` is configured as `true`:Enable block synchronization optimization strategy
- `[sync].sync_block_by_tree` is configured as `false`: Turn off block synchronization optimization strategy
- When `supported_version` is not less than v2.2.0, `[sync].sync_block_by_tree` defaults to `true`; when `supported_version` is less than v2.2.0,`[sync].sync_block_by_tree` defaults to `false`

In addition, in order to ensure the robustness of tree topology block synchronization, FISCO BCOS v2.2.0 also introduced the gossip protocol to periodically synchronize the block status. The related configuration items of the gossip protocol are located in `[sync]` of `group.group_id.ini` The details are as follows:

- `gossip_interval_ms`: gossip protocol synchronization block status period, default is 1000ms
- `gossip_peers_number`: Each time a node synchronizes the block status, the number of randomly selected neighbor nodes, the default is 3

Note:

1. gossip protocol configuration item, only effective when block tree broadcast optimization is enabled
2. Must ensure that all nodes sync_block_by_tree configuration is consistent

---

The optimized configuration of enabling block tree broadcasting is as follows:

```
[sync]
    ; Block tree synchronization strategy is enabled by default
    sync_block_by_tree=true
    ; Every node synchronizes the latest block status every 1000ms
    gossip_interval_ms=1000
    ; Each node randomly selects 3 neighbor nodes at a time to synchronize the␣
↪latest block status
    gossip_peers_number=3
```

Optimal configuration of transaction tree broadcast

In order to reduce the peak outbound bandwidth of SDK directly connected nodes and improve the scalability of the blockchain system, FISCO BCOS v2.2.0 introduced a transaction tree broadcast optimization strategy. You can use the `[sync].send_txs_by_tree` of `group.group_id.ini` to enable or disable the transaction tree broadcast strategy. The detailed configuration is as follows:

- `[sync].sync_block_by_tree`: Set to `true` to enable transaction tree broadcast strategy; set to`false` to disable transaction tree broadcast strategy

The configuration of the disabled transaction tree broadcast policy is as follows:

```
[sync]
    ; Transaction tree broadcast strategy is enabled by default
    send_txs_by_tree=false
```

Note:

- Due to protocol consistency requirements, all nodes must ensure that the tree broadcast switch send_txs_by_tree is configured consistently
- When supported_version is not less than v2.2.0, the transaction tree broadcast optimization strategy is turned on by default; when supported_version is less than v2.2.0, the transaction tree broadcast strategy is turned off by default

---

Optimized transaction forwarding configuration

In order to reduce the traffic overhead caused by transaction forwarding, FISCO BCOS v2.2.0 introduced a state packet-based transaction forwarding strategy. You can configure the maximum number of forwarding nodes for the transaction status through `[sync].txs_max_gossip_peers_num` of `group.group_id.ini`.

Note: To ensure that transactions reach each node and minimize the traffic overhead introduced by transaction status forwarding, it is not recommended to set txs_max_gossip_peers_num too small or too large, just use the default configuration

---

The maximum number of nodes for transaction status forwarding is configured as follows:

```
[sync]
    ; Each node randomly selects up to 5 neighbor nodes to synchronize the latest␣
↪transaction status.
    txs_max_gossip_peers_num=5
```

Parallel transaction configuration

FISCO BCOS supports execution of transactions in parallel. Turning on the transaction parallel execution switch to enable for improving throughput. Execution of the transaction in parallel is only effective in the storage state mode.

Note:

In order to simplify system configuration, v2.3.0 removes the enable_parallel configuration item, which only takes effect when supp

- storageState mode: enable parallel transaction
- mptState mode: disable parallel transactions

```
[tx_execute]
    enable_parallel=true
```

Optional configuration: group flow control

In order to prevent the mutual influence of resources between multiple groups, FISCO BCOS v2.5.0 introduces a flow control function, which supports group-level SDK request rate limit and flow limit, Configure `[flow_control]` located in `group. {Group_id}.ini`, disabled by default， For detailed design of flow control, please refer to here.

SDK to group request rate limit configuration

The SDK request rate limit within the group is located in the configuration item `[flow_control] . limit_req`, Used to limit the maximum number of SDK requests to the group per second, when the request to the node per second exceeds the value of the configuration item, the request will be rejected, SDK to group request rate limit is disabled by default, to enable this function, you need to remove the `;` in front of the `limit_req` configuration item, An example of enable the SDK request rate limit and configuring the group to accept 1000 SDK requests per second is as follows:

```
[flow_control]
    ; restrict QPS of the group
    limit_req=1000
```

Traffic limit configuration between groups

In order to prevent block sync from occupying too much network traffic and affecting the message packet transmission of the consensus module, FISCO BCOS v2.5.0 introduces group-level traffic limit, which configures the upper limit of the average bandwidth of the group, but does not limit the block consensus and transaction sync, when the average bandwidth of the group exceeds the configured value, the block transmission will be suspended.

- `[flow_control].outgoing_bandwidth_limit`: Group output bandwidth limit, the unit is `Mbit/s`, when the group output bandwidth exceeds this value, it will suspend sending blocks, but will not limit the block consensus and transaction broadcast traffic, this configuration item is disabled by default, to enable the traffic limit function, remove the `;` before the `outgoing_bandwidth_limit` configuration item.

The configuration example of enable the group outbound traffic limit and setting it to `2MBit/s` is as follows:

```
[flow_control]
    ; Mb, can be a decimal
    ; when the outgoing bandwidth exceeds the limit, the block synchronization␣
↪operation will not proceed
    outgoing_bandwidth_limit=2
```

Optional configuration: SDK allowlist configuration

In order to achieve access control from SDK to group, FISCO BCOS v2.6.0 introduced a group-level SDK allowlist access control mechanism. The configuration of `[sdk_allowlist]` located in `group. {group_id}.ini` is disabled by default. Please refer to here for the group-level SDK allowlist mechanism.

Important: FISCO BCOS v2.6.0 disables the allowlist access control from SDK to group by default, that is, the SDK can communicate with all groups by default. To enable the allowlist-based access control function between SDK and group, you need to change the ;public_key.0 and other configuration items before the semicolon removed

---

- `public_key.0`、`public_key.1`、...、`public_key.i`: Configure the SDK public key public key list allowed to communicate with the group.

SDK allowlist configuration example is as follows:

```
[sdk_allowlist]
; When sdk_allowlist is empty, all SDKs can connect to this node
; when sdk_allowlist is not empty, only the SDK in the allowlist can connect to␣
↪this node
; public_key.0 should be nodeid, nodeid's length is 128
public_key.
↪0=b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a995
```

6.7.4 Dynamically configure system parameters

FISCO BCOS system currently includes the following system parameters (other system parameters will be extended in the future):

Console provides setSystemConfigByKey command to modify these system parameters. getSystemConfigByKey command can view the current value of the system parameter:

Important: It is not recommended to modify tx_count_limit and tx_gas_limit arbitrarily. These parameters can be modified as follows:

- Hardware performance such as machine network or CPU is limited: to reduce tx_count_limit for reducing business pressure;
- gas is insufficient when executing transactions for comlicated business logic: increase tx_gas_limit.

rpbft_epoch_sealer_num and rpbft_epoch_block_num are only valid for the rPBFT consensus algorithm. In order to ensure the performance of the consensus, it is not recommended to frequently switch the consensus list dynamically, that is, it is not recommended that the rpbft_epoch_block_num configuration value is too small

---

```
# To set the maximum number of transactions of a packaged block to 500

> setSystemConfigByKey tx_count_limit 500
# inquiry tx_count_limit
> getSystemConfigByKey tx_count_limit
[500]

# To set transaction gas limit as  400000000
> setSystemConfigByKey tx_gas_limit 400000000
> getSystemConfigByKey tx_gas_limit
[400000000]

# Under the rPBFT consensus algorithm, set a consensus period to select the number␣
↪of nodes participating in the consensus to 4
[group:1]> setSystemConfigByKey rpbft_epoch_sealer_num 4
Note: rpbft_epoch_sealer_num only takes effect when rPBFT is used
{
    "code":0,
    "msg":"success"
}
# query rpbft_epoch_sealer_num
[group:1]> getSystemConfigByKey rpbft_epoch_sealer_num
Note: rpbft_epoch_sealer_num only takes effect when rPBFT is used
4

# Under the rPBFT consensus algorithm, set a consensus period to produce 10,000␣
↪blocks
```

(continues on next page)

(continued from previous page)

```
[group:1]> setSystemConfigByKey rpbft_epoch_block_num 10000
Note: rpbft_epoch_block_num only takes effect when rPBFT is used
{
    "code":0,
    "msg":"success"
}
# query rpbft_epoch_block_num
[group:1]> getSystemConfigByKey rpbft_epoch_block_num
Note: rpbft_epoch_block_num only takes effect when rPBFT is used
10000

# get consensus_timeout
[group:1]> getSystemConfigByKey consensus_timeout
3

# set consensus_timeout to 5s
[group:1]> setSystemConfigByKey consensus_timeout 5
{
    "code":0,
    "msg":"success"
}
```

6.8 OSCCA-approved cryptography

For fully supporting the OSCCA-approved cryptography algorithm, FISCO integrates the national encryption, decryption, signature, verification, hash algorithm and SSL communication protocol in the FISCO BCOS platform based on the OSCCA-approved cryptography standard. The design documents can be found in the FISCO BCOS Design Manual. OSCCA-approved cryptography Version.

6.8.1 Initial deployment of FISCO BCOS OSCCA-approved cryptography version

This section uses the `build_chain` script to build a 4-nodes FISCO BCOS chain locally, and uses `Ubuntu 16.04` system as an example to operate. This section uses pre-compiled static `fisco-bcos` binaries for testing on CentOS 7 and Ubuntu 16.04.

```
# rely on the installation of Ubuntu16
$ sudo apt install -y openssl curl
# prepare environment
$ cd ~ && mkdir -p fisco && cd fisco
# download build_chain.sh script
$ curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.7.2/
↪build_chain.sh

# If you have network issue for exec command above, please try:
$ curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/build_
↪chain.sh

$ chmod u+x build_chain.sh
```

- build a 4-nodes FISCO BCOS chain

```
# Generate a 4-nodes FISCO chain. All nodes belong to group1. The following␣
↪instructions are executed in the fisco directory.
# -p specifies the starting ports which are p2p_port, channel_port, jsonrpc_port
# According to the following instructions, it needs to ensure that the 30300~30303,
↪ 20200~20203, 8545~8548 ports of the machine are not occupied.
```

(continues on next page)

(continued from previous page)

```
# -g It will generate a chain of OSCCA-approved cryptography.
$ ./build_chain.sh -l 127.0.0.1:4 -p 30300,20200,8545 -g
```

For the `build_chain.sh` script option, please [refer to here] (../manual/build_chain.md). The command that execute normally will output `All completed`. (If there is no output, refer to `nodes/build.log` for checking).

```
[INFO] Downloading tassl binary ...
Generating CA key...
Generating Guomi CA key...
============================================================
Generating keys ...
Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1
============================================================
Generating configurations...
Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1
============================================================
[INFO] FISCO-BCOS Path   : bin/fisco-bcos
[INFO] Start Port        : 30300 20200 8545
[INFO] Server IP         : 127.0.0.1:4
[INFO] State Type        : storage
[INFO] RPC listen IP     : 127.0.0.1
[INFO] Output Dir        : /mnt/c/Users/asherli/Desktop/key-manager/build/nodes
[INFO] CA Key Path       : /mnt/c/Users/asherli/Desktop/key-manager/build/nodes/
↪gmcert/ca.key
[INFO] Guomi mode        : yes
============================================================
[INFO] All completed. Files in /mnt/c/Users/asherli/Desktop/key-manager/build/nodes
```

After the deployment of the OSCCA-approved cryptography alliance chain is completed, the rest of operations are same as [installation] (../installation.md).

6.8.2 OSCCA-approved cryptography configuration information

The nodes of FISCO BCOS OSCCA-approved cryptography version message through using SSL secure channel. The main configuration items of the SSL certificate are concentrated in the following configuration items:

```
[network_security]

data_path: path where the certificate file is located
key: the path of the node private key relative to the data_path
cert: the path of the certificate gmnode.crt relative to data_path
ca_cert: the path of certificate gmca

;certificate configuration
[network_security]
    ;directory the certificates located in
    data_path=conf/
    ;the node private key file
    key=gmnode.key
    ;the node certificate file
    cert=gmnode.crt
    ;the ca certificate file
    ca_cert=gmca.crt
```

After Fisco-BCOS version 2.5.0, nodes and SDKS can communicate using either SSL or national secret SSL connections. The configuration is as follows:

```
[chain]
    ; use SM crypto or not, should nerver be changed
    sm_crypto=true
    ; use SM SSL connection with SDK
    sm_crypto_channel=true
```

6.8.3 using OSCCA-approved cryptography SDK

For details, refer to [SDK Documentation] (../sdk/sdk.html#id8).

6.8.4 OSCCA-approved cryptography console configuration

6.8.5 using OSCCA-approved cryptography console

The function of OSCCA-approved cryptography console is used in the same way as the standard console. Console version 2.6 and above does not require additional configuration. For the configuration method of console version 1.x, please refer to Console Operation Manual.

6.8.6 OSCCA-approved cryptography configuration

OSCCA-approved cryptography Key Manager

Other steps are same as the standard Key Manager. Please refer to key-manager repository.

OSCCA-approved cryptography node configuration

FISCO BCOS OSCCA-approved cryptography version adopts dual certificate mode, so two sets of certificates are needed for disk encryption. They are the conf/gmnode.key and conf/origin_cert/node.key. Other operations of disk encryption are the same as [Standard Edition Loading Encryption Operation] (./storage_security.md).

```
cd key-manager/scripts
#encrypt conf/gmnode.key parameter: ip port  Node private key file cipherDataKey
bash encrypt_node_key.sh 127.0.0.1 8150 nodes/127.0.0.1/node0/conf/gmnode.key␣
↪ed157f4588b86d61a2e1745efe71e6ea
#encrypt conf/origin_cert/node.key parameter: ip port  Node private key file␣
↪cipherDataKey
bash encrypt_node_key.sh 127.0.0.1 8150 nodes/127.0.0.1/node0/conf/origin_cert/
↪node.key ed157f4588b86d61a2e1745efe71e6ea
```

6.9 Deploy Multi-Group Blockchain System

This chapter takes the star networking and parallel multi-group networking as an example to guide you to the following.

- Learn to deploy multi-group blockchain with `build_chain.sh` shell script;
- Understand the organization of the multi-group blockchain created by `build_chain.sh`
- Learn how to start the blockchain node and get the consensus status of each group through the log;
- Learn how to send transactions to the given group and get the block generation status through the log;
- Understand node management of the blockchain system, including how to add/remove the given consensus node;
- Learn how to create a new group.

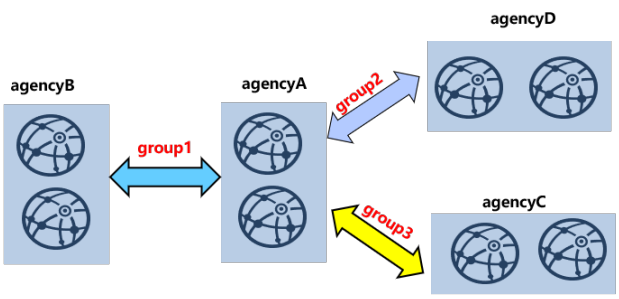
Important:

- `build_chain.sh` is suitable for developers and users to use quickly
- Build an enterprise-level business chain, recommend to use generator

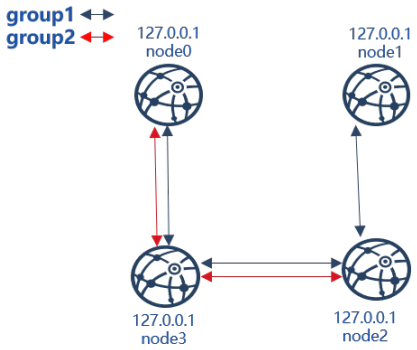
6.9.1 Introduction of star networking topology and parallel multi-group networking topology

As shown in the following figure, the star networking topology and the parallel multi-group networking topology are two widely used networking methods in blockchain applications.

- Star networking topology: The nodes of the central agency belong to multiple groups, and runs multiple institutional applications. Nodes of the other agencies belongs to different groups and runs their respective applications;
- Parallel multi-group networking topology: Each node in the blockchain belongs to multiple groups and can be used for multi-service expansion or multi-node expansion of the same service.



Star networking topology



Parallel multi-group networking topology

The following is a detailed description of how to deploy a eight-node star networking blockchain system and a four-node parallel multi-group networking blockchain.

6.9.2 Installation dependency

Before deploying the FISCO BCOS blockchain, you need to install software such as `openssl`, `curl`, etc. The specific commands are as follows:

```
# CentOS
$ sudo yum install -y openssl curl

# Ubuntu
$ sudo apt install -y openssl curl

# Mac OS
$ brew install openssl curl
```

6.9.3 Star networking topology

In this chapter, we deploy a star networking blockchain system with four agencies, three groups and eight nodes in the local machine.

Here is the detailed configuration of star networking blockchain:

- `agencyA`: belongs to `group1`、`group2`、`group3`, including 2 nodes with the same IP address `127.0.0.1`;
- `agencyB`: belongs to `group1`, including 2 nodes with the same IP address `127.0.0.1`;
- `agencyC`: belongs to `group2`, including 2 nodes with the same IP address `127.0.0.1`;
- `agencyD`: belongs to `group3`, including 2 nodes with the same IP address `127.0.0.1`.

In a star network topology, the core node (in this case, the agencyA node) belongs to all groups and has a high load. It is recommended to deploy it separately on a machine with better performance.

Important:

- In the actual application scenario, it is not recommended to deploy multiple nodes on the same machine. It is recommended to select the number of deployed nodes in one machine according to the machine loading. Please refer to the hardware configuration
- In a star network topology, the core node (in this case, the agencyA node) belongs to all groups and has a high load. It is recommended to deploy it separately on a machine with better performance.
- When operating in different machines, please copy the generated IP folder to the corresponding machine to start. The chain building operation only needs to be performed once!

Generate configuration for star networking blockchain

`build_chain.sh` supports to generate configurations for blockchain with any topology, you can use this script to build configuration for the star networking blockchain.

Prepare for dependency

- Create an operation directory

```
mkdir -p ~/fisco && cd ~/fisco
```

- Download the build_chain.sh script

```
curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.7.2/build_
↪chain.sh && chmod u+x build_chain.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/gen_node_cert.sh

---

Generate configuration for star networking blockchain system

```
# Generate ip_list(configuration)
$ cat > ipconf << EOF
127.0.0.1:2 agencyA 1,2,3
127.0.0.1:2 agencyB 1
127.0.0.1:2 agencyC 2
127.0.0.1:2 agencyD 3
EOF

# Check the content of ip_list
$ cat ipconf
# Meaning of the space-separated parameters:
# ip:num: IP of the physical machine and the number of nodes
# agency_name: agency name
# group_list: the list of groups the nodes belong to, groups are separated by comma
```

(continues on next page)

(continued from previous page)

```
127.0.0.1:2 agencyA 1,2,3
127.0.0.1:2 agencyB 1
127.0.0.1:2 agencyC 2
127.0.0.1:2 agencyD 3
```

Create node configuration folder for star networking blockchain using build_chain script

Please refer to the build_chain for more parameter descriptions of `build_chain.sh`.

```
# Generate a blockchain of star networking and make sure ports 30300~30301, 20200~
↪20201, 8545~8546 of the local machine are not occupied
$ bash build_chain.sh -f ipconf -p 30300,20200,8545
Generating CA key...
================================================================
Generating keys ...
Processing IP:127.0.0.1 Total:2 Agency:agencyA Groups:1,2,3
Processing IP:127.0.0.1 Total:2 Agency:agencyB Groups:1
Processing IP:127.0.0.1 Total:2 Agency:agencyC Groups:2
Processing IP:127.0.0.1 Total:2 Agency:agencyD Groups:3
================================================================
......Here to omit other outputs......
================================================================
[INFO] FISCO-BCOS Path   : ./bin/fisco-bcos
[INFO] IP List File      : ipconf
[INFO] Start Port        : 30300 20200 8545
[INFO] Server IP         : 127.0.0.1:2 127.0.0.1:2 127.0.0.1:2 127.0.0.1:2
[INFO] State Type        : storage
[INFO] RPC listen IP     : 127.0.0.1
[INFO] Output Dir        : /home/ubuntu16/fisco/nodes
[INFO] CA Key Path       : /home/ubuntu16/fisco/nodes/cert/ca.key
================================================================
[INFO] All completed. Files in /home/ubuntu16/fisco/nodes

# The generated node file is as follows:
nodes
|-- 127.0.0.1
|   |-- fisco-bcos
|   |-- node0
|   |   |-- conf  # node configuration folder
|   |   |   |-- ca.crt
|   |   |   |-- group.1.genesis
|   |   |   |-- group.1.ini
|   |   |   |-- group.2.genesis
|   |   |   |-- group.2.ini
|   |   |   |-- group.3.genesis
|   |   |   |-- group.3.ini
|   |   |   |-- node.crt
|   |   |   |-- node.key
|   |   |   `-- node.nodeid # stores the information of Node ID
|   |   |-- config.ini # node configuration file
|   |   |-- start.sh  # shell script to start the node
|   |   `-- stop.sh   # shell script to stop the node
|   |-- node1
|   |   |-- conf
......omit other outputs here......
```

Note: If the generated nodes belong to different physical machines, the blockchain nodes need to be copied to the corresponding physical machine.

Start node

FISCO-BCOS provides the `start_all.sh` and `stop_all.sh` scripts to start and stop the node.

```
# Switch to the node directory
$ cd ~/fisco/nodes/127.0.0.1

# Start the node
$ bash start_all.sh

# Check node process
$ ps aux | grep fisco-bcos
ubuntu16         301  0.8  0.0 986644  7452 pts/0    Sl   15:21   0:00 /home/
↪ubuntu16/fisco/nodes/127.0.0.1/node5/../fisco-bcos -c config.ini
ubuntu16         306  0.9  0.0 986644  6928 pts/0    Sl   15:21   0:00 /home/
↪ubuntu16/fisco/nodes/127.0.0.1/node6/../fisco-bcos -c config.ini
ubuntu16         311  0.9  0.0 986644  7184 pts/0    Sl   15:21   0:00 /home/
↪ubuntu16/fisco/nodes/127.0.0.1/node7/../fisco-bcos -c config.ini
ubuntu16      131048  2.1  0.0 1429036 7452 pts/0    Sl   15:21   0:00 /home/
↪ubuntu16/fisco/nodes/127.0.0.1/node0/../fisco-bcos -c config.ini
ubuntu16      131053  2.1  0.0 1429032 7180 pts/0    Sl   15:21   0:00 /home/
↪ubuntu16/fisco/nodes/127.0.0.1/node1/../fisco-bcos -c config.ini
ubuntu16      131058  0.8  0.0 986644  7928 pts/0    Sl   15:21   0:00 /home/
↪ubuntu16/fisco/nodes/127.0.0.1/node2/../fisco-bcos -c config.ini
ubuntu16      131063  0.8  0.0 986644  7452 pts/0    Sl   15:21   0:00 /home/
↪ubuntu16/fisco/nodes/127.0.0.1/node3/../fisco-bcos -c config.ini
ubuntu16      131068  0.8  0.0 986644  7672 pts/0    Sl   15:21   0:00 /home/
↪ubuntu16/fisco/nodes/127.0.0.1/node4/../fisco-bcos -c config.ini
```

Check consensus status of groups

When no transaction is sent, the node with normal consensus status will output +++ log. In this example, `node0` and `node1` belong to `group1`, `group2` and `group3`; `node2` and `node3` belong to `group1`; `node4` and `node5` belong to `group2`; `node6` and `node7` belong to `group3`. Check the status of node by `tail -f node*/log/* | grep "++"`.

Important:

Node with normal consensus status prints +++ log, fields of +++ log are defined as:

- `g:`: group ID;
- `blkNum`: the newest block number generated by the Leader node;
- `tx`: the number of transactions included in the new block;
- `nodeIdx`: the index of this node;
- `hash`: hash of the newest block generated by consensus nodes.

```
# Check if node0 group1 is normal(Ctrl+c returns to the command line)
$ tail -f node0/log/* | grep "g:1.*++"
info|2019-02-11 15:33:09.914042| [g:1][p:264][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=2,hash=72254a42....

# Check if node0 group2 is normal
$ tail -f node0/log/* | grep "g:2.*++"
info|2019-02-11 15:33:31.021697| [g:2][p:520][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=3,hash=ef59cf17...

# ... To check node1, node2 node for each group is normal or not, refer to the␣
↪above operation method...

# Check if node3 group1 is normal
$ tail -f node3/log/*| grep "g:1.*++"
```

(continues on next page)

(continued from previous page)

```
info|2019-02-11 15:39:43.927167| [g:1][p:264][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=3,hash=5e94bf63...

# Check if node5 group2 is normal
$ tail -f node5/log/* | grep "g:2.*++"
info|2019-02-11 15:39:42.922510| [g:2][p:520][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=2,hash=b80a724d...
```

Configuration console

The console connects to the FISCO BCOS node, it is used to query the blockchain status and deploy the calling contract, etc. Please refer to here for the console manual for version 2.6 and above, and here for the console manual for version 1.x.

Important: The console relies on Java 8 and above, and Ubuntu 16.04 system needs be installed with openjdk 8. Please install Oracle Java 8 or above for CentOS.

```
# Switch back to ~/fisco folder
$ cd ~/fisco

# Download console
$ curl -#LO https://github.com/FISCO-BCOS/console/releases/download/v2.9.2/
↪download_console.sh

# If you have network issue for exec command above, please try:
$ curl -#LO https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/download_
↪console.sh

$ bash download_console.sh

# Switch to console directory
$ cd console

# Copy node certificate of group 2 to the configuration directory of console
$ cp ~/fisco/nodes/127.0.0.1/sdk/* conf/

# Obtain channel_listen_port of node0
$ grep "channel_listen_port" ~/fisco/nodes/127.0.0.1/node*/config.ini
/home/ubuntu16/fisco/nodes/127.0.0.1/node0/config.ini:    channel_listen_port=20200
/home/ubuntu16/fisco/nodes/127.0.0.1/node1/config.ini:    channel_listen_port=20201
/home/ubuntu16/fisco/nodes/127.0.0.1/node2/config.ini:    channel_listen_port=20202
/home/ubuntu16/fisco/nodes/127.0.0.1/node3/config.ini:    channel_listen_port=20203
/home/ubuntu16/fisco/nodes/127.0.0.1/node4/config.ini:    channel_listen_port=20204
/home/ubuntu16/fisco/nodes/127.0.0.1/node5/config.ini:    channel_listen_port=20205
/home/ubuntu16/fisco/nodes/127.0.0.1/node6/config.ini:    channel_listen_port=20206
/home/ubuntu16/fisco/nodes/127.0.0.1/node7/config.ini:    channel_listen_port=20207

# configure the console
$ cp ~/fisco/console/conf/config-example.toml ~/fisco/console/conf/config.toml
```

Important: When connecting node with the console, we should make sure that the connected nodes are in the group configured by the console

---

Start the console

```
$ bash start.sh
# The following information output indicates that the console is successfully␣
↪started. If the startup fails, check whether the certificate configuration and␣
↪the channel listen port configuration are correct.
=====================================================================================
Welcome to FISCO BCOS console(2.6.1)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
 ________ _______  _______   _______  _______       ________  _______  _______  _______
|        |       \/       \ /       \ /       \     |        \ /       \ /       \ /       ␣
↪\
| $$$$$$$$\$$$$$|  $$$$$$|  $$$$$$|  $$$$$$\    | $$$$$$$|  $$$$$$|  $$$$$$|  $$$$$
↪$\
| $$__      | $$ | $$___\$| $$   \$| $$  | $$    | $$__/ $| $$   \$| $$  | $| $$___\
↪$$
| $$  \     | $$  \$$    \| $$      | $$  | $$    | $$    $| $$      | $$  | $$\$$    ␣
↪\
| $$$$$     | $$  _\$$$$$$| $$   __| $$  | $$    | $$$$$$$| $$   __| $$  | $$_\$$$$$
↪$\
| $$       _| $$_|  \__| $| $$__/  | $$__/ $$    | $$__/ $| $$__/  | $$__/ $|  \__|
↪$$
| $$      |   $$ \\$$    $$\$$    $$\$$    $$    | $$    $$\$$    $$\$$    $$\$$
↪$$
 \$$       \$$$$$$ \$$$$$$  \$$$$$$  \$$$$$$      \$$$$$$$  \$$$$$$  \$$$$$$  \$$$$$
↪$

=====================================================================================
[group:1]>
```

Send transactions to groups

In the above section, we learned how to configure the console, this section will introduce how to send transactions to groups through the console.

Important: In the group architecture, the ledgers are independent in each group. And sending transactions to one group only increases the block number of this group but not others

---

Send transactions through console

```
# ... Send HelloWorld transaction to group1...
$ [group:1]> deploy HelloWorld
transaction hash:␣
↪0xd0305411e36d2ca9c1a4df93e761c820f0a464367b8feb9e3fa40b0f68eb23fa
contract address:0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744
# Check the current block number of group1, if the block number is increased to 1,␣
↪ it indicates that the blockchain is normal. Otherwise, please check if group1␣
↪is abnormal.
$ [group:1]> getBlockNumber
1

# ... Send HelloWorld transaction to group2...
# Switch to group2
$ [group:1]> switch 2
Switched to group 2.
# Send transaction to group2, return a transaction hash indicates that the␣
↪transaction is deployed successfully, otherwise, please check if the group2␣
↪works normally.
$ [group:2]> deploy HelloWorld
transaction hash:␣
↪0xd0305411e36d2ca9c1a4df93e761c820f0a464367b8feb9e3fa40b0f68eb23fa
```

(continues on next page)

(continued from previous page)

```
contract address:0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744
# Check the current block number of group2, if the block number is increased to 1,␣
↪ it indicates that the blockchain is normal. Otherwise, please check if group1␣
↪is abnormal
$ [group:2]> getBlockNumber
1

# ... Send transaction to group3...
# Switch to group3
$ [group:2]> switch 3
Switched to group 3.
# Send transaction to group3, return a transaction hash indicates that the␣
↪transaction is deployed successfully, otherwise, please check if the group2␣
↪works normally.
$ [group:3]> deploy HelloWorld
contract address:0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744
# Check the current block number of group3, if the block number is increased to 1,␣
↪ it indicates that the blockchain is normal. Otherwise, please check if group1␣
↪is abnormal
$ [group:3]> getBlockNumber
1

# ... Switch to group 4 that does not exist: the console prompts that group4 does␣
↪not exist, and outputs the current group list ...
$ [group:3]> switch 4
Group 4 does not exist. The group list is [1, 2, 3].

# Exit the console
$ [group:3]> exit
```

Check the log

After the block is generated, the node will output `Report` log, which contains fields with following definitions:

```
# Switch the node directory
$ cd ~/fisco/nodes/127.0.0.1

# Check the block generation status of group1: new block generated
$ cat node0/log/* |grep "g:1.*Report"
info|2019-02-11 16:08:45.077484| [g:1][p:264][CONSENSUS][PBFT]^^^^^^^^Report,num=1,
↪sealerIdx=1,hash=9b5487a6...,next=2,tx=1,nodeIdx=2

# Check the block generation status of group2: new block generated
$ cat node0/log/* |grep "g:2.*Report"
info|2019-02-11 16:11:55.354881| [g:2][p:520][CONSENSUS][PBFT]^^^^^^^^Report,num=1,
↪sealerIdx=0,hash=434b6e07...,next=2,tx=1,nodeIdx=0

# Check the block generation status of group3: new block generated
$ cat node0/log/* |grep "g:3.*Report"
info|2019-02-11 16:14:33.930978| [g:3][p:776][CONSENSUS][PBFT]^^^^^^^^Report,num=1,
↪sealerIdx=1,hash=3a42fcd1...,next=2,tx=1,nodeIdx=2
```

Node joins the group

Through the console, FISCO BCOS can add the specified node to the specified group, or delete the node from the specified group. For details, please refer to the node admission management manual, for console configuration, please reference console operation manual.

Taking how to join node2 to group2 as an example, this chapter introduces how to add a new node to an existing group.

Copy group2 configuration to node2

```
# Switch to node directory
$ cd ~/fisco/nodes/127.0.0.1

# ... Copy group2 configuration of node0 to node2 ...
$ cp node0/conf/group.2.* node2/conf

# ...Restart node2(make sure the node is in normal consensus after restart)...
$ cd node2 && bash stop.sh && bash start.sh
```

Get the ID of node2

```
# Please remember the node ID of node2. Add node2 to group2 needs the node ID.
$ cat conf/node.nodeid
6dc585319e4cf7d73ede73819c6966ea4bed74aadbbcba1bbb777132f63d355965c3502bed7a04425d99cdcfb7694a1c1
```

Send commands to group2 through the console to add node2 into group2

```
# ...Go back to the console directory and launch the console (direct boot to␣
↪group2)...
$ cd ~/fisco/console && bash start.sh 2

# ...Join node2 as a consensus node through the console...
# 1. Check current consensus node list
$ [group:2]> getSealerList
[
    ␣
↪9217e87c6b76184cf70a5a77930ad5886ea68aefbcce1909bdb799e45b520baa53d5bb9a5edddeab94751df179d54d41
↪
    ␣
↪227c600c2e52d8ec37aa9f8de8db016ddc1c8a30bb77ec7608b99ee2233480d4c06337d2461e24c26617b6fd53acfa61
↪
    ␣
↪7a50b646fcd9ac7dd0b87299f79ccaa2a4b3af875bd0947221ba6dec1c1ba4add7f7f690c95cf3e796296cf4adc989f4
↪
    ␣
↪8b2c4204982d2a2937261e648c20fe80d256dfb47bda27b420e76697897b0b0ebb42c140b4e8bf0f27dfee64c946039
]
# 2. Add node2 to the consensus node
# The parameter after addSealer is the node ID obtained in the previous step
$ [group:2]> addSealer␣
↪6dc585319e4cf7d73ede73819c6966ea4bed74aadbbcba1bbb777132f63d355965c3502bed7a04425d99cdcfb7694a1c
{
    "code":0,
    "msg":"success"
}
# 3. Check current consensus node list
$ [group:2]> getSealerList
[
    ␣
↪9217e87c6b76184cf70a5a77930ad5886ea68aefbcce1909bdb799e45b520baa53d5bb9a5edddeab94751df179d54d41
↪
    ␣
↪227c600c2e52d8ec37aa9f8de8db016ddc1c8a30bb77ec7608b99ee2233480d4c06337d2461e24c26617b6fd53acfa61
↪
    ␣
↪7a50b646fcd9ac7dd0b87299f79ccaa2a4b3af875bd0947221ba6dec1c1ba4add7f7f690c95cf3e796296cf4adc989f4
↪
    ␣
↪8b2c4204982d2a2937261e648c20fe80d256dfb47bda27b420e76697897b0b0ebb42c140b4e8bf0f27dfee64c946039
↪
    ␣
↪6dc585319e4cf7d73ede73819c6966ea4bed74aadbbcba1bbb777132f63d355965c3502bed7a04425d99cdcfb7694a1c
↪# new node
```

(continues on next page)

(continued from previous page)

```
]
# Get the current block number of group2
$ [group:2]> getBlockNumber
2

#... Send transaction to group2
# Deploy the HelloWorld contract and output contract address. If the contract␣
↪fails to deploy, please check the consensus status of group2
$ [group:2] deploy HelloWorld
contract address:0xdfdd3ada340d7346c40254600ae4bb7a6cd8e660

# Get the current block number of group2, it increases to 3. If not, please check␣
↪the consensus status of group2
$ [group:2]> getBlockNumber
3

# Exit the console
$ [group:2]> exit
```

Check the block generation status of the new node through log

```
# Switch to the node directory
cd ~/fisco/nodes/127.0.0.1
# Check the consensus status of the node(Ctrl+c returns the command line)
$ tail -f node2/log/* | grep "g:2.*++"
info|2019-02-11 18:41:31.625599| [g:2][p:520][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=4,tx=0,nodeIdx=1,hash=c8a1ed9c...
......Other outputs are omitted here......

# Check the block generation status of node2 and group2: new block generated
$ cat node2/log/* | grep "g:2.*Report"
info|2019-02-11 18:53:20.708366| [g:2][p:520][CONSENSUS][PBFT]^^^^^Report:,num=3,
↪idx=3,hash=80c98d31...,next=10,tx=1,nodeIdx=1
# node2 also reports a block with block number 3, indicating that node2 has joined␣
↪group2.
```

Stop the node

```
# Back to the node directory && stop the node
$ cd ~/fisco/nodes/127.0.0.1 && bash stop_all.sh
```

6.9.4 Parallel multi-group networking topology

Deploying parallel multi-group networking blockchain is similar with deploying star networking blockchain. Taking a four-node two-group parallel multi-group blockchain as an example:

- group 1: includs 4 nodes with the same IP `127.0.0.1`;
- group 2: includs 4 nodes with the same IP `127.0.0.1`.

In a real application scenario, it is not recommended to deploy multiple nodes on the same machine. It is recommended to select the number of deployed nodes according to the machine load. To demonstrate the parallel multi-group expansion process, only group1 is created here first. In a parallel multi-group scenario, node join and exit group operations are similar to star networking topology.

Important:

- In a real application scenario, it is not recommended to deploy multiple nodes the same machine, It is recommended to determine the number of deployed nodes according to the machine load
- To demonstrate the parallel multi-group expansion process, only group1 is created here first
- In a parallel multi-group scenario, the operations of node joining into a group or leaving from a group are similar to star networking blockchain.

Build blockchain with a single group and 4 nodes

Generate a single-group four-node blockchain node configuration folder with the build_chain.sh script

```
$ mkdir -p ~/fisco && cd ~/fisco
# Download build_chain.sh script
$ curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.7.2/
↪build_chain.sh

# If you have network issue for exec command above, please try:
$ curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/build_
↪chain.sh

$ chmod u+x build_chain.sh
#Build a local single-group four-node blockchain (in a production environment, it␣
↪is recommended that each node be deployed on a different physical machine)
$ bash build_chain.sh -l 127.0.0.1:4 -o multi_nodes -p 20000,20100,7545
Generating CA key...
================================================================
Generating keys ...
Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1
================================================================
Generating configurations...
Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1
================================================================
[INFO] FISCO-BCOS Path   : bin/fisco-bcos
[INFO] Start Port        : 20000 20100 7545
[INFO] Server IP         : 127.0.0.1:4
[INFO] State Type        : storage
[INFO] RPC listen IP     : 127.0.0.1
[INFO] Output Dir        : /home/ubuntu16/fisco/multi_nodes
[INFO] CA Key Path       : /home/ubuntu16/fisco/multi_nodes/cert/ca.key
================================================================
[INFO] All completed. Files in /home/ubuntu16/fisco/multi_nodes
```

Start all nodes

```
# Switch to the node directory
$ cd ~/fisco/multi_nodes/127.0.0.1
$ bash start_all.sh

# Check process
$ ps aux | grep fisco-bcos
ubuntu16       55028  0.9  0.0 986384  6624 pts/2    Sl   20:59   0:00 /home/
↪ubuntu16/fisco/multi_nodes/127.0.0.1/node0/../fisco-bcos -c config.ini
ubuntu16       55034  0.8  0.0 986104  6872 pts/2    Sl   20:59   0:00 /home/
↪ubuntu16/fisco/multi_nodes/127.0.0.1/node1/../fisco-bcos -c config.ini
ubuntu16       55041  0.8  0.0 986384  6584 pts/2    Sl   20:59   0:00 /home/
↪ubuntu16/fisco/multi_nodes/127.0.0.1/node2/../fisco-bcos -c config.ini
ubuntu16       55047  0.8  0.0 986396  6656 pts/2    Sl   20:59   0:00 /home/
↪ubuntu16/fisco/multi_nodes/127.0.0.1/node3/../fisco-bcos -c config.ini
```

Check consensus status of nodes

```
# Check consensus status of node0(Ctrl+c returns to the command line)
$ tail -f node0/log/* | grep "g:1.*++"
info|2019-02-11 20:59:52.065958| [g:1][p:264][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=2,hash=da72649e...

# Check consensus status of node1
$ tail -f node1/log/* | grep "g:1.*++"
info|2019-02-11 20:59:54.070297| [g:1][p:264][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=0,hash=11c9354d...

# Check consensus status of node2
$ tail -f node2/log/* | grep "g:1.*++"
info|2019-02-11 20:59:55.073124| [g:1][p:264][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=1,hash=b65cbac8...

# Check consensus status of node3
$ tail -f node3/log/* | grep "g:1.*++"
info|2019-02-11 20:59:53.067702| [g:1][p:264][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=3,hash=0467e5c4...
```

Add group2 to the blockchain

The `genesis` configuration files in each group of parallel multi-group networking blockchain are almost the same, except group ID [group].id, which is the group number.

```
# Switch to the node directory
$ cd ~/fisco/multi_nodes/127.0.0.1

# Copy the configuration of group 1
$ cp node0/conf/group.1.genesis node0/conf/group.2.genesis
$ cp node0/conf/group.1.ini node0/conf/group.2.ini

# Modify group ID
$ sed -i "s/id=1/id=2/g" node0/conf/group.2.genesis
$ cat node0/conf/group.2.genesis | grep "id"
# Have modified to    id=2

# Update the list of consensus nodes in the "group.2.genesis" to remove obsolete␣
↪consensus nodes

# Copy the configuration to each node
$ cp node0/conf/group.2.genesis node1/conf/group.2.genesis
$ cp node0/conf/group.2.genesis node2/conf/group.2.genesis
$ cp node0/conf/group.2.genesis node3/conf/group.2.genesis
$ cp node0/conf/group.2.ini node1/conf/group.2.ini
$ cp node0/conf/group.2.ini node2/conf/group.2.ini
$ cp node0/conf/group.2.ini node3/conf/group.2.ini

# Restart node
$ bash stop_all.sh
$ bash start_all.sh
```

Check consensus status of the group

```
# Check the consensus status of node0 group2
$ tail -f node0/log/* | grep "g:2.*++"
info|2019-02-11 21:13:28.541596| [g:2][p:520][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=2,hash=f3562664...
```

(continues on next page)

(continued from previous page)

```
# Check the consensus status of node1 group2
$ tail -f node1/log/* | grep "g:2.*++"
info|2019-02-11 21:13:30.546011| [g:2][p:520][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=0,hash=4b17e74f...

# Check the consensus status of node2 group2
$ tail -f node2/log/* | grep "g:2.*++"
info|2019-02-11 21:13:59.653615| [g:2][p:520][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=1,hash=90cbd225...

# Check the consensus status of node3 group2
$ tail -f node3/log/* | grep "g:2.*++"
info|2019-02-11 21:14:01.657428| [g:2][p:520][CONSENSUS][SEALER]++++++++Generating␣
↪seal on,blkNum=1,tx=0,nodeIdx=3,hash=d7dcb462...
```

Send transactions to groups

Download console

```
# If you have never downloaded the console, please do the following to download␣
↪the console, otherwise copy the console to the ~/fisco directory:
$ cd ~/fisco
# Download console
$ curl -#LO https://github.com/FISCO-BCOS/console/releases/download/v2.9.2/
↪download_console.sh

# If you have network issue for exec command above, please try:
$ curl -#LO https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/download_
↪console.sh

bash download_console.sh
```

Configuration console

```
# Get channel_port
$ grep "channel_listen_port" multi_nodes/127.0.0.1/node0/config.ini
multi_nodes/127.0.0.1/node0/config.ini:    channel_listen_port=20100

# Switch to console subdirectory
$ cd console
# Copy node certificate
$ cp ~/fisco/multi_nodes/127.0.0.1/sdk/* conf

# Copy the console configuration
$ cp ~/fisco/console/conf/config-example.toml ~/fisco/console/conf/config.toml

# Modify the node port of the console connection to 20100 and 20101
# The linux system uses the following commands:
$ sed -i 's/127.0.0.1:20200/127.0.0.1:21000/g' ~/fisco/console/conf/config.toml
$ sed -i 's/127.0.0.1:20201/127.0.0.1:21001/g' ~/fisco/console/conf/config.toml

# The mac system uses the following commands:
$ sed -i .bkp 's/127.0.0.1:20200/127.0.0.1:21000/g' ~/fisco/console/conf/config.
↪toml
$ sed -i .bkp 's/127.0.0.1:20201/127.0.0.1:21001/g' ~/fisco/console/conf/config.
↪toml
```

Send transactions to groups via console

```
# ... Start console ...
$ bash start.sh
# The following information output indicates that the console is successfully␣
↪started. If the startup fails, check whether the certificate configuration and␣
↪the channel listen port configuration are correct.
================================================================================
Welcome to FISCO BCOS console(2.6.1)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
 ________ _______  _______   _______  _______       ________   _______   _______   _______
|        |       \/       \ /       \ /       \     |        \ /       \ /       \ /       ␣
↪\
| $$$$$$$$\$$$$$|  $$$$$$$| $$$$$$$| $$$$$$$\    | $$$$$$$$| $$$$$$$| $$$$$$$| $$$$$
↪$$\
| $$__      | $$ | $$__   \$| $$__   \$| $$  | $$    | $$__/ $$| $$__   \$| $$  | $$| $$___\
↪$$
| $$  \     | $$ | $$  \     | $$  \     | $$  | $$    | $$    $$| $$  \     | $$  | $$\$$   ␣
↪\
| $$$$$     | $$ | $$$$$  __  | $$$$$  __  | $$  | $$    | $$$$$$$$| $$$$$  __  | $$  | $$_\$$$$$
↪$$\
| $$       _| $$_| $$_____|  \| $$_____|  \| $$__/ $$    | $$__/ $$| $$_____|  \| $$__/ $$|  \__|
↪$$
| $$      |   $$ \\$$     $$\$$     $$\$$     $$    | $$    $$\$$     $$\$$     $$\$$
↪$$
 \$$       \$$$$$$ \$$$$$$   \$$$$$$   \$$$$$$       \$$$$$$$  \$$$$$$   \$$$$$$   \$$$$$
↪$

================================================================================
# ... Send transaction to group 1...
# Get the current block number
$ [group:1]> getBlockNumber
0
# Deploy the HelloWorld contract to group1. If the deployment fails, check whether␣
↪the consensus status of group1 is normal
$ [group:1]> deploy HelloWorld
transaction hash:␣
↪0xd0305411e36d2ca9c1a4df93e761c820f0a464367b8feb9e3fa40b0f68eb23fa
contract address:0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744
# Get the current block number. If the block number is not increased, please check␣
↪if the group1 is normal
$ [group:1]> getBlockNumber
1

# ... send transaction to group 2...
# Switch to group2
$ [group:1]> switch 2
Switched to group 2.
# Get the current block number
$ [group:2]> getBlockNumber
0
# Deploy HelloWorld contract to group2
$ [group:2]> deploy HelloWorld
transaction hash:␣
↪0xd0305411e36d2ca9c1a4df93e761c820f0a464367b8feb9e3fa40b0f68eb23fa
contract address:0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744
# Get the current block number. If the block number is not increased, please check␣
↪if the group1 is normal
$ [group:2]> getBlockNumber
1
# Exit console
$[group:2]> exit
```

Check block generation status of nodes through log

```
# Switch to the node directory
$ cd ~/fisco/multi_nodes/127.0.0.1/

# Check block generation status of group1, and to see that the block with block␣
↪number of 1 belonging to group1 is reported.
$ cat node0/log/* | grep "g:1.*Report"
info|2019-02-11 21:14:57.216548| [g:1][p:264][CONSENSUS][PBFT]^^^^^Report:,num=1,
↪sealerIdx=3,hash=be961c98...,next=2,tx=1,nodeIdx=2

# Check block generation status of group2, and to see that the block with block␣
↪number of 1 belonging to group2 is reported.
$ cat node0/log/* | grep "g:2.*Report"
info|2019-02-11 21:15:25.310565| [g:2][p:520][CONSENSUS][PBFT]^^^^^Report:,num=1,
↪sealerIdx=3,hash=5d006230...,next=2,tx=1,nodeIdx=2
```

Stop nodes

```
# Back to nodes folder && stop the node
$ cd ~/fisco/multi_nodes/127.0.0.1 && bash stop_all.sh
```

6.10 Distributed storage

6.10.1 Install MySQL

The currently supported distributed database is MySQL. Before using distributed storage, you need to set up the MySQL service. The configuration on Ubuntu and CentOS servers is as follows:

Ubuntu: Execute the following three commands to configure the root account password during the installation process.

```
sudo apt install -y mysql-server mysql-client libmysqlclient-dev
```

Start the MySQL service and log in: root account password.

```
sudo service mysql start
mysql -uroot -p
```

CentOS: Perform the following two commands to install.

```
yum install mysql*
# some versions of linux need to install mariadb which is a branch of mysql
yum install mariadb*
```

Start the MySQL service. Log in and set a password for the root user.

```
service mysqld start
#If mariadb is installed, to use the following command to start
service mariadb start
mysql -uroot
mysql> set password for root@localhost = password('123456');
```

6.10.2 Configure MySQL parameters

check the configuration file my.cnf

```
mysql --help | grep 'Default options' -A 1
```

After execution, you can see the following data:

```
Default options are read from the following files in the given order:
/etc/mysql/my.cnf /etc/my.cnf ~/.my.cnf
```

Configure my.cnf

mysql loads the configuration from /etc/mysql/my.cnf, /etc/my.cnf, ~/.my.cnf in turn. Search these files in turn, find the first file that exists, and add the following content in the [mysqld] section (modify the value if it exists).

```
max_allowed_packet = 1024M
sql_mode =STRICT_TRANS_TABLES
ssl=0
default_authentication_plugin = mysql_native_password
```

Restart mysql-server and verify parameters

Ubuntu：Execute the following command to restart

```
service mysql restart
```

CentOS：Execute the following command to restart

```
service mysqld start
# If mariadb is installed, use the following command to start:
service mariadb start
```

Validation parameter:

```
mysql -uroot -p
# Execute the following command to view the value of max_allowed_packet
MariaDB [(none)]>  show variables like 'max_allowed_packet%';
+--------------------+------------+
| Variable_name      | Value      |
+--------------------+------------+
| max_allowed_packet | 1073741824 |
+--------------------+------------+
1 row in set (0.00 sec)

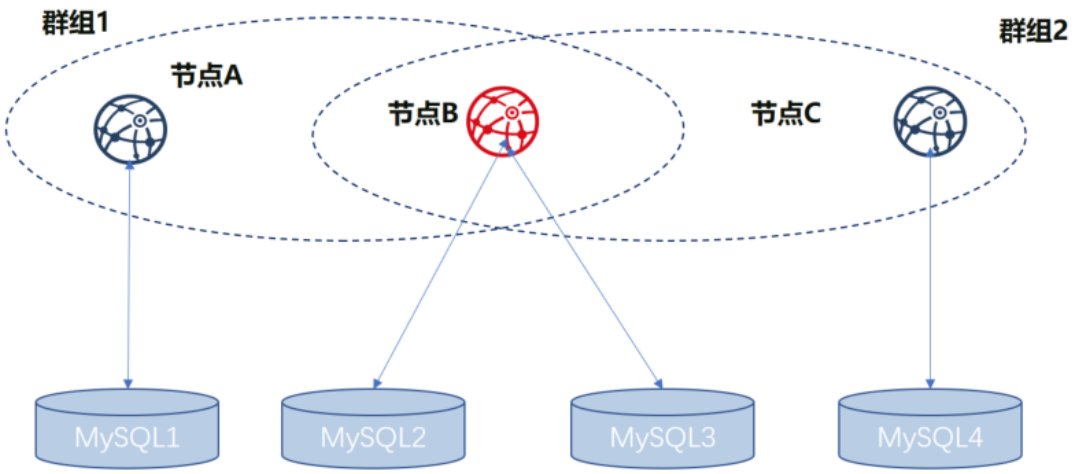
# Execute the following command to view the value of sql_mode
MariaDB [(none)]>  show variables like 'sql_mode%';
+---------------+---------------------+
| Variable_name | Value               |
+---------------+---------------------+
| sql_mode      | STRICT_TRANS_TABLES |
+---------------+---------------------+
1 row in set (0.00 sec)
```

6.10.3 Node directly connected to MySQL

FISCO BCOS in version 2.0.0-rc3 supports nodes directly connected to MySQL through connection pool. Compared to the proxy access MySQL mode, this configuration is simple. No need to manually create a database. Please refer to the configuration method:

Logical architecture diagram

The multi-group architecture means that blockchain node supports launching multiple groups. The transaction processing, data storage, and block consensus among the groups are isolated from each other. Therefore, each node in the group corresponds to a database instance. For example, in blockchain network, there are three nodes A, B, and C, where A and B belong to Group1, and B and C belong to Group2. Nodes A and C correspond to one database instance respectively, and Node B corresponds to two database instances. The logical architecture diagram is as follows.



As shown in the above figure, NodeB belongs to multiple groups. The database instances which are corresponded by the same node in different groups are separate. For distinguishing the same node in different groups, the nodes of A, B, and C are respectively represented with Group1_A (NodeA in Group1, same as below), Group1_B, Group2_B, and Group2_C.

We use the above figure as an example to describe the setup configuration process in following.

Build node

Before using distributed storage, you need to complete the establishment of the alliance chain and the configuration of multiple groups. For details, refer to the following steps.

Prepare dependence

```
mkdir -p ~/fisco && cd ~/fisco
# Download build_chain.sh script
curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.7.2/build_
↪chain.sh && chmod u+x build_chain.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/gen_node_cert.sh

Generate configuration file

```
# generate blockchain configuration file ipconf
cat > ipconf << EOF
127.0.0.1:1 agencyA 1
127.0.0.1:1 agencyB 1,2
127.0.0.1:1 agencyC 2
EOF

# view configuration file
cat ipconf
127.0.0.1:1 agencyA 1
127.0.0.1:1 agencyB 1,2
127.0.0.1:1 agencyC 2
```

Build blockchain with build_chain

```
### build blockchain (please confirm the ports of 30300~30302, 20200~20202, 8545~
↪8547 are not occupied)
### The difference right here is that the parameter "-s MySQL" is appended to the␣
↪command and the port is changed.
bash build_chain.sh -f ipconf -p 30300,20200,8545 -s MySQL
=============================================================
Generating CA key...
=============================================================
Generating keys ...
Processing IP:127.0.0.1 Total:1 Agency:agencyA Groups:1
Processing IP:127.0.0.1 Total:1 Agency:agencyB Groups:1,2
Processing IP:127.0.0.1 Total:1 Agency:agencyC Groups:2
=============================================================
Generating configurations...
Processing IP:127.0.0.1 Total:1 Agency:agencyA Groups:1
Processing IP:127.0.0.1 Total:1 Agency:agencyB Groups:1,2
Processing IP:127.0.0.1 Total:1 Agency:agencyC Groups:2
=============================================================
Group:1 has 2 nodes
Group:2 has 2 nodes
```

Modify node ini file

In group.[group].ini configuration file, the configuration information of MySQL is related to this feature. Suppose that the MySQL configuration information is as follows:

```
|node|db_ip|db_port|db_username|db_passwd|db_name|
|Group1_A|127.0.0.1|3306|root|123456|db_Group1_A|
|Group1_B|127.0.0.1|3306|root|123456|db_Group1_B|
|Group2_B|127.0.0.1|3306|root|123456|db_Group2_B|
|Group2_C|127.0.0.1|3306|root|123456|db_Group2_C|
```

Modify the group.1.ini configuration in node0

Modify the content in the section ~/fisco/nodes/127.0.0.1/node0/conf/group.1.ini[storage] and add the following content. Db_passwd is the corresponding password.

```
            db_ip=127.0.0.1
            db_port=3306
```

(continues on next page)

(continued from previous page)

```
db_username=root
db_name=db_Group1_A
db_passwd=
```

Modify the group.1.ini configuration in node1

Modify the content in the section ~/fisco/nodes/127.0.0.1/node1/conf/group.1.ini[storage] and add the following content. Db_passwd is the corresponding password.

```
db_ip=127.0.0.1
db_port=3306
db_username=root
db_name=db_Group1_B
db_passwd=
```

Modify the group.2.ini configuration in node1

Modify the content in the section ~/fisco/nodes/127.0.0.1/node1/conf/group.2.ini[storage] and add the following content. Db_passwd is the corresponding password.

```
db_ip=127.0.0.1
db_port=3306
db_username=root
db_name=db_Group2_B
db_passwd=
```

Modify the group.2.ini configuration in node2

Modify the content in the section ~/fisco/nodes/127.0.0.1/node2/conf/group.2.ini[storage] and add the following content. Db_passwd is the corresponding password.

```
db_ip=127.0.0.1
db_port=3306
db_username=root
db_name=db_Group2_C
db_passwd=
```

Start node

```
cd ~/fisco/nodes/127.0.0.1;sh start_all.sh
```

Check process

```
ps -ef|grep fisco-bcos|grep -v grep
fisco   111061      1  0 16:22 pts/0    00:00:04 /data/home/fisco/nodes/127.0.0.1/
↪node2/../fisco-bcos -c config.ini
fisco   111065      1  0 16:22 pts/0    00:00:04 /data/home/fisco/nodes/127.0.0.1/
↪node0/../fisco-bcos -c config.ini
fisco   122910      1  1 16:22 pts/0    00:00:02 /data/home/fisco/nodes/127.0.0.1/
↪node1/../fisco-bcos -c config.ini
```

If it starts successfully, you can see there are 3 fisco-bcos processes. If it fails, please refer to the log to confirm whether the configuration is correct.

Check output of log

Execute the following command to view the number of nodes connected to node0 (similar to other nodes)

```
tail -f nodes/127.0.0.1/node0/log/log*  | grep connected
```

Normally, you will see an output similar to the following, and you can see that node0 is connecting to the other two nodes from it.

```
info|2019-05-28 16:28:57.267770|[P2P][Service] heartBeat,connected count=2
info|2019-05-28 16:29:07.267935|[P2P][Service] heartBeat,connected count=2
info|2019-05-28 16:29:17.268163|[P2P][Service] heartBeat,connected count=2
info|2019-05-28 16:29:27.268284|[P2P][Service] heartBeat,connected count=2
info|2019-05-28 16:29:37.268467|[P2P][Service] heartBeat,connected count=2
```

Execute the following command to check if it is in consensus

```
tail -f nodes/127.0.0.1/node0/log/log*  | grep +++
```

Normally, the output will continue to output ++++Generating seal to indicate that the consensus is normal.

```
info|2019-05-28 16:26:32.454059|[g:1][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=28,tx=0,nodeIdx=3,hash=c9c859d5...
info|2019-05-28 16:26:36.473543|[g:1][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=28,tx=0,nodeIdx=3,hash=6b319fa7...
info|2019-05-28 16:26:40.498838|[g:1][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=28,tx=0,nodeIdx=3,hash=2164360f...
```

Send transaction by console

Prepare dependence

```
cd ~/fisco;
curl -#LO https://github.com/FISCO-BCOS/console/releases/download/v2.9.2/download_
↪console.sh && bash download_console.sh
cp -n console/conf/config-example.toml console/conf/config.toml
cp nodes/127.0.0.1/sdk/* console/conf/
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/download_console.sh

---

Modify configuration file

Modify ~/fisco/console/conf/applicationContext.xml to the following configuration (partial information)

```
peers=["127.0.0.1:20300", "127.0.0.1:20301"]    # The peer list to connect
```

Start console

```
cd ~/fisco/console
sh start.sh 1
#deploy TableTest contract
```

(continues on next page)

(continued from previous page)

```
[group:1]> deploy TableTest
contract address:0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744
```

view the table in the database

```
MySQL -uroot -p123456 -A db_Group1_A
use db_Group1_A;
show tables;
+-------------------------------------------------------+
| Tables_in_db_Group1_A                                 |
+-------------------------------------------------------+
| c_8c17cf316c1063ab6c89df875e96c9f0f5b2f744 |
| c_f69a2fa2eca49820218062164837c6eecc909abd |
| _sys_block_2_nonces_                                  |
| _sys_cns_                                             |
| _sys_config_                                          |
| _sys_consensus_                                       |
| _sys_current_state_                                   |
| _sys_hash_2_block_                                    |
| _sys_number_2_hash_                                   |
| _sys_table_access_                                    |
| _sys_tables_                                          |
| _sys_tx_hash_2_block_                                 |
+-------------------------------------------------------+
12 rows in set (0.02 sec)
```

view the table in the database

```
show tables;
+-------------------------------------------------------+
| Tables_in_db_Group1_A                                 |
+-------------------------------------------------------+
| c_8c17cf316c1063ab6c89df875e96c9f0f5b2f744 |
| c_f69a2fa2eca49820218062164837c6eecc909abd |
| _sys_block_2_nonces_                                  |
| _sys_cns_                                             |
| _sys_config_                                          |
| _sys_consensus_                                       |
| _sys_current_state_                                   |
| _sys_hash_2_block_                                    |
| _sys_number_2_hash_                                   |
| _sys_table_access_                                    |
| _sys_tables_                                          |
| _sys_tx_hash_2_block_                                 |
| u_t_test                                              |
+-------------------------------------------------------+
```

Inserting a record to the database

```
#insert data into the table
call TableTest 0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744 insert "fruit" 100 "apple
↪"
0x082ca6a5a292f1f7b20abeb3fb03f45e0c6f48b5a79cc65d1246bfe57be358d1
```

open the MySQL client and query the u_t_test table data

```
#view data in the user table
select * from u_t_test\G;
*************************** 1. row ***************************
     _id_: 31
   _hash_: 0a0ed3b2b0a227a6276114863ef3e8aa34f44e31567a5909d1da0aece31e575e
```

(continues on next page)

(continued from previous page)

```
    _num_: 3
 _status_: 0
     name: fruit
  item_id: 100
item_name: apple
1 row in set (0.00 sec)
```

6.11 SDK allowlist mechanism

FISCO BCOS 2.0 started to support multiple groups, but did not control the SDK's access rights to multiple groups. As long as the SDK can establish a connection with the node, it can access all the groups of the node, which will bring security risks.

FISCO BCOS v2.6.0 introduces a group-level SDK allowlist mechanism, controls the SDK's access rights to groups, and provides scripts to support the dynamic loading of the SDK allowlist list, further improving the security of the blockchain system.

Note:

- When the number of SDK allowlist lists in the configuration item is 0, the node does not enable the SDK allowlist control function, and any SDK can access the group;
- The SDK allowlist is a node-level access control mechanism. The node that enables this function controls the SDK's access rights to the node group based on the locally configured allowlist;
- The SDK allowlist mechanism controls SDK access to all group-level interfaces of the node

---

6.11.1 Configuration

Note: The SDK allowlist configuration of each group is located in the [sdk_allowlist] configuration item of the group.{group_id}.ini configuration file. For details, please refer to here

---

6.11.2 Get SDK public key

Before adding the SDK to the allowlist, you first need to obtain the SDK public key, which is used to set the `public_key` of `group.*.ini`. The methods for obtaining the SDK public key for each version are as follows:

Newly built chain

The SDK certificate generated by the newly built chain comes with SDK private key information. The standard version is `sdk.publickey`, and the OSCCA-approved version is `gmsdk.publickey`:

```
# Assuming that the certificate has been copied to the SDK, enter the SDK␣
↪directory and execute the following command (sdk is located in the ~/fisco␣
↪directory)
$ cd ~/fisco/java-sdk

# Obtain the public key of the OSCCA-approved SDK
$ cat dist/conf/sdk.publickey

# Obtain the public key of the standard SDK
$ cat dist/conf/gmsdk.publickey
```

Old chain

The old chain needs to use openssl or tassl commands to generate `sdk.publickey` (the OSCCA-approved version is `gmsdk.publickey`), and load the public key from the generated file, as follows:

Standard Version

```
# Enter the SDK directory and execute the following command:
$ openssl ec -in dist/conf/sdk.key -text -noout 2> /dev/null | sed -n '7,11p' | tr␣
↪-d ": \n" | awk '{print substr($0,3);}' | cat > dist/conf/sdk.publickey
# Obtain the public key
$ cat dist/conf/sdk.publickey
```

OSCCA-approved Version

```
# Note: It must be ensured that ~/.fisco/tassl exists
$ ~/.fisco/tassl ec -in dist/conf/gmsdk.key -text -noout 2> /dev/null | sed -n '7,
↪11p' | sed 's/://g' | tr "\n" " " | sed 's/ //g' | awk '{print substr($0,3);}' ␣
↪| cat > dist/conf/gmsdk.publickey
# Get SDK public key
$ cat dist/conf/gmsdk.publickey
```

6.11.3 Dynamically modify the SDK allowlist

In order to facilitate users to modify the SDK allowlist list, the `reload_sdk_allowlist.sh` script is provided in the `scripts` directory of each node to reload the SDK allowlist.

Note: There is no reload_sdk_allowlist.sh script on the old chain nodes, and the script can be downloaded by the command curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/FISCO-BCOS/master-2.0/tools/reload_sdk_allowlist.sh.

6.11.4 Example

This section takes adding and modifying the console to the whitelist as an example to show in detail how to use the SDK allowlist mechanism.

Build a blockchain and copy the certificate to the console

Please refer to Installation.

Get console public key information

```
# Enter the console directory
$ cd ~/fisco/console/

# Obtain console public key information through sdk.publickey
$ cat conf/sdk.publickey
ebf5535c92f7116310ed9e0f9fc9bfc66a607415d4fa444d91f528485eff61b15e40a70bc5d73f0441d3959efbc7718c2
```

Enable the SDK allowlist mechanism

Add the public key of a console to the allowlist of the group.[group_id].ini configuration file of `node0`:

```
[sdk_allowlist]
public_key.
↪0=b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a995
```

Reload the SDK allowlist

```
$ bash node0/scripts/reload_sdk_allowlist.sh
 [INFO] node0 is trying to reload sdk allowlist. Check log for more information.

# After the SDK allowlist is successfully hot-loaded, the node outputs the␣
↪following log:
parseSDKAllowList,
↪sdkAllowList=[b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf98
↪enableSDKAllowListControl=true
```

Console access node

Note: Since the SDK allowlist is a node-level access control mechanism, in order to demonstrate the access control function of node0 to the SDK, the console only connects to node0.

Since `node0` does not configure the console's access rights to the group, the result of the deployment contract is as follows:

```
# Deploy HelloWorld contract in the console
[group:1]> deploy HelloWorld
sendRawTransaction The SDK is not allowed to access this group
```

Add the console to the SDK allowlist

Configure the console to the allowlist of `node0`:

```
[sdk_allowlist]
public_key.
↪0=b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a995
public_key.
↪1=ebf5535c92f7116310ed9e0f9fc9bfc66a607415d4fa444d91f528485eff61b15e40a70bc5d73f0441d3959efbc77
```

Reload the SDK whitelist list:

```
$ bash node0/scripts/reload_sdk_allowlist.sh
 [INFO] node0 is trying to reload sdk allowlist. Check log for more information.

# After the SDK allowlist is successfully hot-loaded, the node outputs the␣
↪following log:
parseSDKAllowList,
↪sdkAllowList=[b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf98
↪ebf5535c92f7116310ed9e0f9fc9bfc66a607415d4fa444d91f528485eff61b15e40a70bc5d73f0441d3959efbc7718
↪enableSDKAllowListControl=true
```

Console access node

After `node0` is added to the console to the allowlist, the console can deploy the contract normally, as follows:

```
[group:1]> deploy HelloWorld
contract address: 0xcd4ccd96c86fe8e4f27b056c0fdb7eb4ca201f0f
```

6.12 Storage security

The data of the alliance chain is only visible to members of the alliance. Disk encryption ensures the security of the data running on the alliance chain on the hard disk. Once the hard drive is taken out from the intranet environment of alliance chain, the data will not be decrypted.

Disk encryption encrypts the content stored on the hard disk by the node. The encrypted content includes: the data of the contract and the private key of the node.

For specific disk encryption introduction, please refer to: Introduction of Disk Encryption

6.12.1 Key Manager deployment

Each agency has a Key Manager. For specific deployment steps, please refer to Key Manager README or Key Manager Gitee README

6.12.2 Node building

Use the script [build_chain.sh] (../installation.md) to build a node with normal operations.

```
curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.9.1/build_
↪chain.sh && chmod u+x build_chain.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/build_chain.sh

```
bash build_chain.sh -l 127.0.0.1:4 -p 30300,20200,8545
```

Important: The node cannot be launched until the dataKey is configured. Before the node runs for the first time, it must be configured to use the disk encryption or not. Once the node starts running, it cannot be switched its state.

6.12.3 Key Manager launch

To launch key-manager directly. If key-manager is not deployed, refer to Key Manager README Introduction.

```
# parameter: port, superkey
./key-manager 8150 123xyz
```

launch successfully and print the log.

```
[1546501342949][TRACE][Load]key-manager started,port=8150
```

6.12.4 DataKey configuration

Important: The node configured by the dataKey must be newly generated node and has not been launched.

To execute the script, define dataKey, and get cipherDataKey

```
cd key-manager/scripts
bash gen_data_secure_key.sh 127.0.0.1 8150 123456

CiherDataKey generated: ed157f4588b86d61a2e1745efe71e6ea
Append these into config.ini to enable disk encryption:
[storage_security]
enable=true
key_manager_ip=127.0.0.1
key_manager_port=8150
cipher_data_key=ed157f4588b86d61a2e1745efe71e6ea
```

The script for getting `cipherDataKey` automatically prints out the ini configuration required for the disk encryption (see below). Now, the cipherDataKey is:`cipher_data_key=ed157f4588b86d61a2e1745efe71e6ea`

To write the ini configuration that has been disk encryption to the node configuration file (config.ini).

```
vim nodes/127.0.0.1/node0/config.ini
```

To put it at last like this.

```
[storage_security]
enable=true
key_manager_ip=127.0.0.1
key_manager_port=8150
cipher_data_key=ed157f4588b86d61a2e1745efe71e6ea
```

6.12.5 Encrypted node private key

To execute script and encrypt node private key

```
cd key-manager/scripts
# parameter:ip port node private key file cipherDataKey
bash encrypt_node_key.sh 127.0.0.1 8150 ../../nodes/127.0.0.1/node0/conf/node.key␣
↪ed157f4588b86d61a2e1745efe71e6ea
```

The node private key is automatically encrypted after execution, and the files before encryption is backed up to the file `node.key.bak.xxxxxx`. Please take care of the backup private key and delete the backup private key generated on the node

```
[INFO] File backup to "nodes/127.0.0.1/node0/conf/node.key.bak.1546502474"
[INFO] "nodes/127.0.0.1/node0/conf/node.key" encrypted!
```

If you check the `node.key`, you can see that it has been encrypted as ciphertext.

```
8b2eba71821a5eb15b0cbe710e96f23191419784f644389c58e823477cf33bd73a51b6f14af368d4d3ed647d9de681893
```

Important: All files that need to be encrypted are listed below. If they are not encrypted, the node cannot be launched.

- standard version: conf/node.key
- national cryptography version: conf/gmnode.key和conf/origin_cert/node.key

---

6.12.6 Node running

to launch node directly

```
cd nodes/127.0.0.1/node0/
./start.sh
```

6.12.7 Correct judgment

(1) The node runs and generates block normally, and the block information is continuously output.

```
tail -f nodes/127.0.0.1/node0/log/* | grep +++
```

(2) `key-manager` will print a log each time the node launches. For example, when a node launches, the log directly output by Key Manager is as follows.

```
[1546504272699][TRACE][Dec]Respond
{
   "dataKey" : "313233343536",
   "error" : 0,
   "info" : "success"
}
```

6.13 Group members management

FISCO BCOS introduces free nodes, observer nodes and consensus nodes which can be converted to each other by the console.

- Group member
  - Consensus nodes (Sealer)
  - Observer nodes (Observer)
- Non-Group member
  - Free nodes (The node waiting for joining the group)

6.13.1 Operation command

The console provides three commands of AddSealer, AddObserver, and RemoveNode to convert the specified node to Sealer, Oberserver, and RemoveNode, and can use getSealerList, getObserverList, and getNodeIDList to view the current list of Sealer, Observer, and other nodes.

- addSealer: to set the corresponding node as the Sealer according to the NodeID;
- addObserver: to set the corresponding node as the Observer according to the NodeID;
- removeNode: to set the corresponding node as the RemoveNode according to the NodeID;
- getSealerList: to view the Sealer in the group;
- getObserverList: to view the Observer in the group;
- getNodeIDList: to view the NodeID in the group;

For example, to convert the specified nodes to Sealer, Observer, and RemoveNode, the main operation commands are as follows:

Important: Before accessing the node, please ensure that:

- Node ID exists and can execute cat that is getting from conf/node.nodeid in the node directory
- All Sealers are normal, and they will output +++ logs.

```
# to get Node ID (to set the directory as ~/nodes/192.168.0.1/node0/)
$ cat ~/fisco/nodes/192.168.0.1/node0/conf/node.nodeid
7a056eb611a43bae685efd86d4841bc65aefafbf20d8c8f6028031d67af27c36c5767c9c79cff201769ed80ff220b9695

# to connect console (to set the console in the ~/fisco/console directory)
$ cd ~/fisco/console

$ bash start.sh
```

(continues on next page)

(continued from previous page)

```
# to convert the specified node to Sealer
[group:1]> addSealer␣
↪7a056eb611a43bae685efd86d4841bc65aefafbf20d8c8f6028031d67af27c36c5767c9c79cff201769ed80ff220b96
# to view the list of Sealer
[group:1]> getSealerList
[
        7a056eb611a43bae685efd86d4841bc65aefafbf20d8c8f6028031d67af27c36c5767c9c79cff201769ed80ff
]

# to convert the specified node to Observer
[group:1]> addObserver␣
↪7a056eb611a43bae685efd86d4841bc65aefafbf20d8c8f6028031d67af27c36c5767c9c79cff201769ed80ff220b96

# to view the list of Observer
[group:1]> getObserverList
[
        7a056eb611a43bae685efd86d4841bc65aefafbf20d8c8f6028031d67af27c36c5767c9c79cff201769ed80ff
]

# to convert the specified node to removeNode
[group:1]> removeNode␣
↪7a056eb611a43bae685efd86d4841bc65aefafbf20d8c8f6028031d67af27c36c5767c9c79cff201769ed80ff220b96

# to view the list of NodeID
[group:1]> getNodeIDList
[
        7a056eb611a43bae685efd86d4841bc65aefafbf20d8c8f6028031d67af27c36c5767c9c79cff201769ed80ff
]
[group:1]> getSealerList
[]
[group:1]> getObserverList
[]
```

6.13.2 Operation cases

The following describes the operations of group expansion and node exit in detail with specific operation cases. Group expansion is divided into two phases, namely adding nodes to the network and adding nodes to the group. Also, node exit is divided into two phases, namely exiting node from the group and exiting node from the network.

Operation methods

- Node configuration modification: After the node modifies its own configuration, it need to be restarted to takes effect. The involved operations include network adding/exit and CA blacklist inclusion/removal.
- Transaction uploading to the chain: To modify the transaction, the configuration of group consensus is needed, and the operation involves the modification of node type. The current sending transaction path is the pre-compiled service interface provided by the console and SDK.
- RPC inquiry： To use the command curl to inquire the information on the chain, and the operation involves the query of the group node.

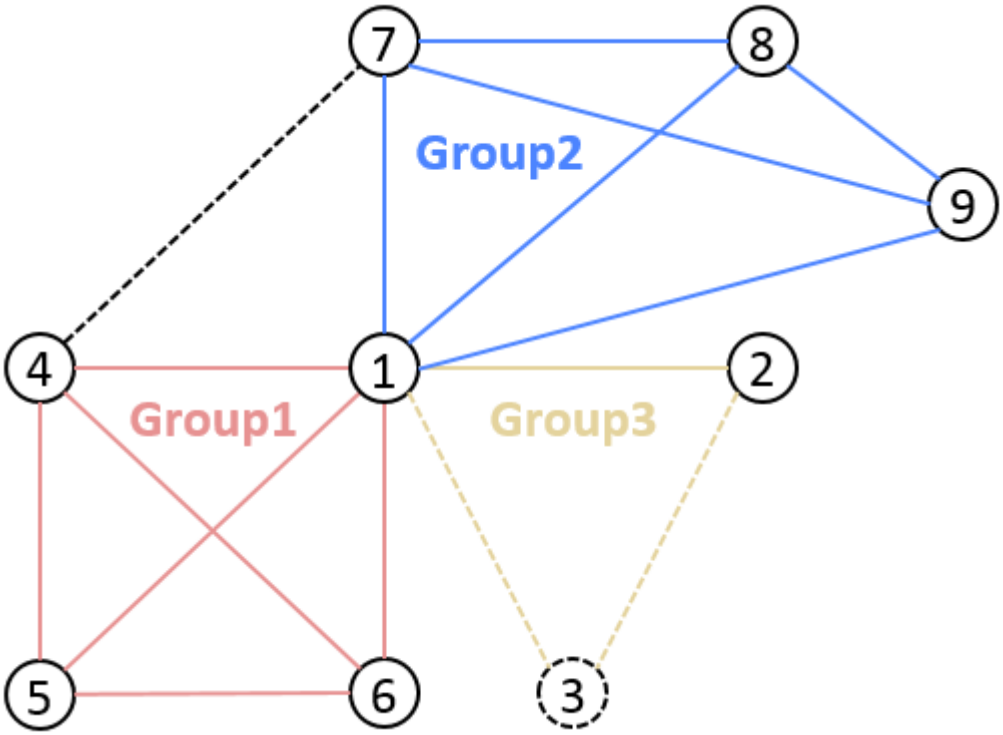
Operation steps

In this section, the following figure is taken as an example to describe the operations of expansion and network exit show above.

The dotted line indicates that network communication can be performed among nodes, and the solid line indicates that nodes have group relationships base on the communication among nodes, and colors' difference distinguish

different group relationships.

The below figure below shows a network with three groups of which Group3 has three nodes. Whether Group3 has intersection nodes with other groups does not affect the versatility of the following operation process.



The folder name of node 1 is `node0`, IP port 127.0.0.1:30400, the former 4 bytes of nodeID b231b309...

The folder name of node 2 is `node1`, IP port 127.0.0.1:30401, the former 4 bytes of nodeID aab37e73...

The folder name of node 3 is `node2`, IP port 127.0.0.1:30402, the former 4 bytes of nodeID d6b01a96...

Node A to join the network

Background:

Node 3 is outside the network and wants to join it now.

Operation steps:

1 . enter nodes folder and execute `gen_node_cert.sh` to generate node folder. Here we name the folder as node2, which contains `conf/` folder;

```
# acquire script
$ curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/FISCO-BCOS/master-2.0/
↪tools/gen_node_cert.sh && chmod u+x gen_node_cert.sh
# execute, -c is the ca route given when the node was generated, agency is the␣
↪agency name, -o is the name of the node folder to be generated ( use -g when the␣
↪node type is GM )
$ ./gen_node_cert.sh -c nodes/cert/agency -o node2
```

Note:

- If download failed, please try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/gen_node_cert.sh

2 . copy node 2 under `nodes/127.0.0.1/`, parallel with other node folder (`node0`, `node1`);

```
$ cp -r ./node2/ nodes/127.0.0.1/
```

3 . enter `nodes/127.0.0.1/`, copy `node0/config.ini`, `node0/start.sh` and `node0/stop.sh` to node2 directory;

```
$ cd nodes/127.0.0.1/
$ cp node0/config.ini node0/start.sh node0/stop.sh node2/
```

4 . modify `node2/config.ini`. For `[rpc]` model, modify `channel_listen_port` and `jsonrpc_listen_port`; for `[p2p]` model, modify `listen_port` and add its node information in `node`..

Note: For the convenience of development and experience, the recommended configuration of channel_listen_ip/listen_ip is 0.0.0.0. For security considerations, please modify it to a safe listening address according to the actual business network situation, such as the internal IP or a specific external IP

```
$ vim node2/config.ini
[rpc]
    ;rpc listen ip
    channel_listen_ip=0.0.0.0
    jsonrpc_listen_ip=127.0.0.1
    ;channelserver listen port
    channel_listen_port=20302
    ;jsonrpc listen port
    jsonrpc_listen_port=8647
[p2p]
    ;p2p listen ip
    listen_ip=0.0.0.0
    ;p2p listen port
    listen_port=30402
    ;nodes to connect
    node.0=127.0.0.1:30400
    node.1=127.0.0.1:30401
    node.2=127.0.0.1:30402
```

5 . node 3 copies `node1/conf/group.3.genesis`(which contains initial list of group nodes) and `node1/conf/group.3.ini` to `node2/conf` folder, without modification;

```
$ cp node1/conf/group.3.genesis node2/conf/
$ cp node1/conf/group.3.ini node2/conf/
```

6 . execute `node2/start.sh` and start node 3;

```
$ ./node2/start.sh
```

7 . confirm that node 3 is connected with node 1, 2, then it has joined the network now.

```
# Open the DEBUG log to check the number and ID of nodes connected with node 2
# The following log information indicates that node 2 is connected with 2␣
↪nodes(with the former 4 bytes being b231b309 and aab37e73)
$ tail -f node2/log/log*  | grep P2P
debug|2019-02-21 10:30:18.694258| [P2P][Service] heartBeat ignore connected,
↪endpoint=127.0.0.1:30400,nodeID=b231b309...
debug|2019-02-21 10:30:18.694277| [P2P][Service] heartBeat ignore connected,
↪endpoint=127.0.0.1:30401,nodeID=aab37e73...
info|2019-02-21 10:30:18.694294| [P2P][Service] heartBeat connected count,size=2
```

- If CA whitelist is enabled, you should configure every nodeID into every nodes' whitelist configuration and reload the configurati

- The other configurations of config.ini copied from node 1 remain the same;
- Theoretically, node 1, 2 can accomplish the extension of node 3 without changing their p2p connecting nodes list;
- The group to be chosen in step 5 are recommended to be the group to be joined by node 3;
- To keep in full connection status, we recommend users to add the information of node 3 to the p2p connecting nodes list in config.ini of node 1, 2, and restart node 1, 2.

Node A to quit the network

Background:

Node 3 is in the network and connected with node 1, 2. Now node 3 needs to quit the network.

Operation steps:

1 . For node 3, clear up the P2P connecting nodes list, and restart node 3;

```
# execute under node 2
$ ./stop.sh
$ ./start.sh
nohup: appending output to 'nohup.out'
```

2 . For node 1, 2, remove node 3 form their P2P connecting nodes list(if has), and restart node 1, 2;

3 . Confirm that node 3 has been disconnected with node 1, 2, and it has quitted the network now.

Note:

- node 3 has to quit the group before quitting the network, which is guaranteed by users and will not be verified by the system;
- the networking process is started by nodes. If missing step 2, node 3 can still get the p2p connecting request from node 1, 2 and start connection. It can be stopped by using CA blacklist.
- If CA whitelist is enabled, you should delete the node from every nodes' whitelist configuration and reload the configuration. Read "CA blacklist and whitelist" for more.

Node A to join a group

Background:

Group 3 contains node 1, 2, either generates block in turn. Now node 3 wants to join the group.

Operation steps:

1. Node 3 joins the network;
2. Set node 3 as the consensus node using console addSealer according to the node ID;
3. check if the node ID of node 3 is included in the consensus nodes of group 3 through console getSealerList. If is, then it has joined the group successfully.

Note:

- node ID of node 3 can be acquired through cat nodes/127.0.0.1/node2/conf/node.nodeid;
- the first start of node 3 will write the configured group node initial list to the node system list, when the blocks stop synchronizing, the node system lists of each node are the same;
- node 3 needs to have access to the network before joining the group, which will be verified by the system;

- the group fixed configuration file of node 3 should be the same with node 1, 2.

Node A to quit the group

Background:

Group 3 contains node 1, 2, 3, either of which generates block in turn. Now node 3 wants to quit the group.

Operation steps:

1. set node 3 as free node according to its ID using console removeNode;
2. check if the node ID of node 3 is included in the consensus nodes of group 3 through console getSealerList. If not, then node 3 has quited the group.

Additional:

Note:

- node 3 can quit the group as a consensus node or observer node.

6.14 Permission control

6.14.1 TODO: add Role Based Access control

Roles and Permissions

6.14.2 Permission conrtol based on Table permission

This section will introduce the operations concerning permission control, for details please check Design of Permission Control.

For the system is defaulted with no permission setting record, any account can perform permission setting. For example, account 1 gives permission of contract deployment to itself, but account 2 also sets itself with permission of contract deployment. So the setting of account 1 becomes meaningless for every other node can add permissions freely. Therefore, before building consortium chain, confirming permission setting rules is needed. We can use grantPermissionManager instruction to set manager account, that is to give some account access to permission setting, which other accounts don't have.

Operations

The operations concerning permission control of following functions are introduced in this section:

- Permission of chain manager
- Permission of system manager
- Permission of contract deployment and user table creation
- Permission of Contract deployment using CNS
- Permission of node management
- Permission to modify system parameter
- Permission to write user table

Environment configuration

Configure and start the nodes and console of FISCO BCOS 2.0+. For reference please check Installation.

Tools for permission control

FISCO BCOS offers permission control of console commands (developers can call PermissionService API of SDK API for permission control). The involved permission control commands are:

Permission control example

Console provides script `get_account.sh` to generate accounts. The account files will be stored in `accounts` folder. Console can set active accounts. The operation method is introduced in Console tutorial. Therefore, through console we can set account to experience permission control. For account safety, we will generate 3 PKCS12 account files under the root folder of console by `get_account.sh` script. Please remember the password during generation. The 3 PKCS12 account files are:

```
# account 1
0x2c7f31d22974d5b1b2d6d5c359e81e91ee656252.p12
# account 2
0x7fc8335fec9da5f84e60236029bb4a64a469a021.p12
# account 3
0xd86572ad4c92d4598852e2f34720a865dd4fc3dd.p12
```

Now we can open 3 Linux terminal and log in console with the 3 accounts separately.

Log in with account 1:

```
$ ./start.sh 1 -p12 accounts/0x2c7f31d22974d5b1b2d6d5c359e81e91ee656252.p12
```

Log in with account 2:

```
$ ./start.sh 1 -p12 accounts/0x7fc8335fec9da5f84e60236029bb4a64a469a021.p12
```

Log in with account 3:

```
$ ./start.sh 1 -p12 accounts/0xd86572ad4c92d4598852e2f34720a865dd4fc3dd.p12
```

Grant permission of chain manager

The 3 accounts play 3 kinds of roles. Account 1 performs chain manager, account 2 performs system manager and account 3 the regular account. Chain manager has permission for access control, namely granting permissions. System manager can manager permissions related to system functions, each of which should be granted independently, including deploying contract, creating user table, managing nodes and deploying contract with CNS and modifying system parameter. Chain manager can grant other accounts to be chain manager or system manager, or grant regular accounts to write table list.

Initial status of chain contains no permission records. Now, we can enter the console of account 1 and set itself as the chain manager, so other accounts are regular accounts.

```
[group:1]> grantPermissionManager 0x2c7f31d22974d5b1b2d6d5c359e81e91ee656252
{
    "code":0,
    "msg":"success"
}

[group:1]> listPermissionManager
-------------------------------------------------------------------------------
↪----------
```

(continues on next page)

(continued from previous page)

```
|                    address                    |                  enable_num
↪          |
| 0x2c7f31d22974d5b1b2d6d5c359e81e91ee656252  |                       1
↪          |
-------------------------------------------------------------------------------------
↪----------
```

Account 1 is set as the chain manager.

Grant permission of system manager

Grant permission to deploy contract and create user table

Account 1 grants permission of system manager to account 2. At first, grant account 2 with permission to deploy contract and create user table.

```
[group:1]> grantDeployAndCreateManager 0x7fc8335fec9da5f84e60236029bb4a64a469a021
{
    "code":0,
    "msg":"success"
}

[group:1]> listDeployAndCreateManager
-------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0x7fc8335fec9da5f84e60236029bb4a64a469a021  |                       2
↪          |
-------------------------------------------------------------------------------------
↪----------
```

Log in console with account 2 and deploy TableTest contract offered by console. Code of TableTest.sol contract is here. The CRUD operations of user table t_test are also provided.

```
[group:1]> deploy TableTest.sol
contract address:0xfe649f510e0ca41f716e7935caee74db993e9de8
```

Call create API of TableTest to create user table t_test.

```
[group:1]> call TableTest.sol 0xfe649f510e0ca41f716e7935caee74db993e9de8 create
transaction hash:0x67ef80cf04d24c488d5f25cc3dc7681035defc82d07ad983fbac820d7db31b5b
-------------------------------------------------------------------------------------
↪----------
Event logs
-------------------------------------------------------------------------------------
↪----------
createResult index: 0
count = 0
-------------------------------------------------------------------------------------
↪----------
```

User table t_test is created successfully.

Log in console with account 3 and deploy TableTest contract.

```
[group:1]> deploy TableTest.sol
{
    "code":-50000,
```

(continues on next page)

(continued from previous page)

```
    "msg":"permission denied"
}
```

Account 3 fails to deploy contract as it has no permission.

- Note: deploying contract and creating user table are "2-in-1" control items. When using CRUD interface contracts, we suggest to create the needed tables (creating tables in building function of contract) when deploying contract, otherwise "table-missing" error may occur when reading or writing table. If it is needed to dynamically create table, the permission should be granted to minority accounts, otherwise there will be many invalid tables on blockchain.

Grant permission to deploy contract using CNS

Console provides 3 commands involving CNS:

Note: permission of deployByCNS command is controllable and needs permission to deploy contract and use CNS at the same time, permissions of callByCNS and queryCNS commands are not controllable.

Log in console with account 1, grant account 2 with permission to deploy contract using CNS.

```
[group:1]> grantCNSManager 0x7fc8335fec9da5f84e60236029bb4a64a469a021
{
    "code":0,
    "msg":"success"
}

[group:1]> listCNSManager
-------------------------------------------------------------------------------------------
↪----------
|                   address                    |                 enable_num
↪          |
| 0x7fc8335fec9da5f84e60236029bb4a64a469a021   |                    13
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

Log in console with account 2, deploy contract using CNS

```
[group:1]> deployByCNS TableTest.sol 1.0
contract address:0x24f902ff362a01335db94b693edc769ba6226ff7

[group:1]> queryCNS TableTest.sol
-------------------------------------------------------------------------------------------
↪----------
|                   version                    |                   address
↪          |
|                     1.0                      |
↪0x24f902ff362a01335db94b693edc769ba6226ff7   |
-------------------------------------------------------------------------------------------
↪----------
```

Log in console with account 3, deploy contract using CNS

```
[group:1]> deployByCNS TableTest.sol 2.0
{
    "code":-50000,
    "msg":"permission denied"
}

[group:1]> queryCNS TableTest.sol
```

(continues on next page)

(continued from previous page)

```
-------------------------------------------------------------------------------------------
↪----------
|                    version                    |                    address
↪          |
|                      1.0                      |
↪0x24f902ff362a01335db94b693edc769ba6226ff7  |
-------------------------------------------------------------------------------------------
↪----------
```

Account 3 fails to deploy contract by CNS due to lack of permission

Grant permission to manage nodes

Console provides 5 commands related to node type management:

- Note: permissions of addSealer, addObserver and removeNode commands are controllable, while permissions of getSealerList and getObserverList commands are not.

Log in console with account 1, grant account 2 with permission to manage nodes.

```
[group:1]> grantNodeManager 0x7fc8335fec9da5f84e60236029bb4a64a469a021
{
    "code":0,
    "msg":"success"
}

[group:1]> listNodeManager
-------------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0x7fc8335fec9da5f84e60236029bb4a64a469a021  |                      20
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

Log in console with account 2, view consensus node list.

```
[group:1]> getSealerList
[
    "01cd46feef2bb385bf03d1743c1d1a52753129cf092392acb9e941d1a4e0f499fdf6559dfcd4dbf2b3ca418caa09d95
↪
    "279c4adfd1e51e15e7fbd3fca37407db84bd60a6dd36813708479f31646b7480d776b84df5fea2f3157da6df9cad078
↪
    "320b8f3c485c42d2bfd88bb6bb62504a9433c13d377d69e9901242f76abe2eae3c1ca053d35026160d86db1a563ab2a
↪
    "c26dc878c4ff109f81915accaa056ba206893145a7125d17dc534c0ec41c6a10f33790ff38855df008aeca3a27ae7d9
]
```

View observer node list:

```
[group:1]> getObserverList
[]
```

Set the first node ID as the observer node:

```
[group:1]> addObserver␣
↪01cd46feef2bb385bf03d1743c1d1a52753129cf092392acb9e941d1a4e0f499fdf6559dfcd4dbf2b3ca418caa09d953
{
    "code":0,
    "msg":"success"
}

[group:1]> getObserverList
[
    ␣
↪01cd46feef2bb385bf03d1743c1d1a52753129cf092392acb9e941d1a4e0f499fdf6559dfcd4dbf2b3ca418caa09d953
]

[group:1]> getSealerList
[
    ␣
↪279c4adfd1e51e15e7fbd3fca37407db84bd60a6dd36813708479f31646b7480d776b84df5fea2f3157da6df9cad078c
↪
    ␣
↪320b8f3c485c42d2bfd88bb6bb62504a9433c13d377d69e9901242f76abe2eae3c1ca053d35026160d86db1a563ab2ac
↪
    ␣
↪c26dc878c4ff109f81915accaa056ba206893145a7125d17dc534c0ec41c6a10f33790ff38855df008aeca3a27ae7d9(
]
```

Log in console with account 3, add observer node to consensus node list.

```
[group:1]> addSealer␣
↪01cd46feef2bb385bf03d1743c1d1a52753129cf092392acb9e941d1a4e0f499fdf6559dfcd4dbf2b3ca418caa09d953
{
    "code":-50000,
    "msg":"permission denied"
}

[group:1]> getSealerList
[
    ␣
↪279c4adfd1e51e15e7fbd3fca37407db84bd60a6dd36813708479f31646b7480d776b84df5fea2f3157da6df9cad078c
↪
    ␣
↪320b8f3c485c42d2bfd88bb6bb62504a9433c13d377d69e9901242f76abe2eae3c1ca053d35026160d86db1a563ab2ac
↪
    ␣
↪c26dc878c4ff109f81915accaa056ba206893145a7125d17dc534c0ec41c6a10f33790ff38855df008aeca3a27ae7d9(
]

[group:1]> getObserverList
[
    ␣
↪01cd46feef2bb385bf03d1743c1d1a52753129cf092392acb9e941d1a4e0f499fdf6559dfcd4dbf2b3ca418caa09d953
]
```

Account 3 fails to add consensus node for lack of permission to manage nodes. Now only account 2 has permission to add observer node to consensus node list.

Grant permission to modify system parameter

Console provides 2 commands about system parameter modification:

- Note: currently we support system parameter setting with key tx_count_limit or tx_gas_limit. Permission of setSystemConfigByKey command is controllable, while permission of getSystemConfigByKey command

is not.

Log in console with account 1, grant account 2 with permission to modify system parameter.

```
[group:1]> grantSysConfigManager 0x7fc8335fec9da5f84e60236029bb4a64a469a021
{
    "code":0,
    "msg":"success"
}

[group:1]> listSysConfigManager
-------------------------------------------------------------------------------------------
↪----------
|                   address                    |                 enable_num
↪          |
| 0x7fc8335fec9da5f84e60236029bb4a64a469a021  |                    23
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

Log in console with account 2, modify the value of system parameter tx_count_limit to 2000.

```
[group:1]> getSystemConfigByKey tx_count_limit
1000

[group:1]> setSystemConfigByKey tx_count_limit 2000
{
    "code":0,
    "msg":"success"
}

[group:1]> getSystemConfigByKey tx_count_limit
2000
```

Log in console with account 3, modify value of parameter tx_count_limit to 3000.

```
[group:1]> setSystemConfigByKey tx_count_limit 3000
{
    "code":-50000,
    "msg":"permission denied"
}

[group:1]> getSystemConfigByKey tx_count_limit
2000
```

Account 3 fails to set parameter due to no permission.

Grant permission to write user table

Account 1 can grant account 3 with permission to write user table t_test.

```
[group:1]> grantUserTableManager t_test 0xd86572ad4c92d4598852e2f34720a865dd4fc3dd
{
    "code":0,
    "msg":"success"
}
[group:1]> listUserTableManager t_test
-------------------------------------------------------------------------------------------
↪----------
|                   address                    |                 enable_num
↪          |
```

(continues on next page)

(continued from previous page)

```
| 0xd86572ad4c92d4598852e2f34720a865dd4fc3dd  |                          6
↪          |
------------------------------------------------------------------------------------------
↪----------
```

Log in console with account 3, insert a record in user table t_test and inquire the records.

```
[group:1]> call TableTest.sol 0xfe649f510e0ca41f716e7935caee74db993e9de8 insert
↪"fruit" 1 "apple"

transaction hash:0xc4d261026851c3338f1a64ecd4712e5fc2a028c108363181725f07448b986f7e
------------------------------------------------------------------------------------------
↪----------
Event logs
------------------------------------------------------------------------------------------
↪----------
InsertResult index: 0
count = 1
------------------------------------------------------------------------------------------
↪----------

[group:1]> call TableTest.sol 0xfe649f510e0ca41f716e7935caee74db993e9de8 select
↪"fruit"
[[fruit], [1], [apple]]
```

Log in console with account 2, update the record inserted by account 3 and inquire the records.

```
[group:1]> call TableTest.sol 0xfe649f510e0ca41f716e7935caee74db993e9de8 update
↪"fruit" 1 "orange"
{
    "code":-50000,
    "msg":"permission denied"
}
[group:1]> call TableTest.sol 0xfe649f510e0ca41f716e7935caee74db993e9de8 select
↪"fruit"
[[fruit], [1], [apple]]
```

Account 2 fails to update information for it has no permission to write user table t_test.

- Account 1 revoke permission of account 3 to write user table t_test.

```
[group:1]> revokeUserTableManager t_test 0xd86572ad4c92d4598852e2f34720a865dd4fc3dd
{
    "code":0,
    "msg":"success"
}

[group:1]> listUserTableManager t_test
Empty set.
```

Revoked successfully.

- Note: now there is no account with permission to write user table t_test, so it is back to initial status, that is, all accounts have permission to write table. Therefore, account 1 can grant another account, like account 2, with permission to write this table.

Under construction

6.15 国内镜像和CDN加速攻略

TODO: translate this into english 本节为访问GitHub较慢的用户提供国内镜像下载地址，以及CDN加速访问介绍。

6.15.1 FISCO BCOS源码与二进制程序

源码同步

FISCO BCOS当前所有仓库源码位于https://github.com/FISCO-BCOS/FISCO-BCOS/tree/master-2.0，每个新的版本发布会将代码和入master分支。

为了方便国内用户，我们同样在gitee上提供了镜像仓库https://gitee.com/FISCO-BCOS/FISCO-BCOS/tree/master-2.0，每次新版本发布后，镜像仓库会同步GitHub上官方仓库的更新，如果从GitHub下载失败，请尝试使用gitee镜像仓库。

二进制程序

FISCO BCOS每个新版本发布会在GitHub的tag中提供对应的二进制程序和部署工具，当前所提供的二进制程序包括：

1. fisco-bcos.tar.gz： 静态二进制程序，支持CentOS 7 和Ubuntu 16.04以上版本
2. fisco-bcos-macOS.tar.gz： 对应macOS系统的二进制程序
3. build_chain.sh ： 对应版本的开发部署工具，依赖openssl和curl，支持CentOS 7/Ubuntu 16.04以上/macOS 10.15以上版本

用户使用开发部署工具(build_chain)，工具先尝试从GitHub下载所需要的二进制程序，如果下载失败则尝试从官网下载。

用户运维部署工具(generator)的时候，工具默认从GitHub下载所需要的二进制程序，可以通过–cdn参数指定从官网下载。例如`./generator --download_fisco ./meta --cdn`

6.15.2 FISCO BCOS文档

FISCO BCOS文档使用readthedocs管理，全部开源于https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/，同样提供国内镜像http://docs.fisco-bcos.org。

每个版本发布会为上个版本的文档打Tag，新版本的文档会和入主干分支，文档由于会持续改进，所以是下个版本发布才打上个版本的tag。readthedocs文档支持下载PDF格式，方便用户使用。

6.15.3 FISCO BCOS配套工具

控制台

FISCO BCOS控制台是一个交互式命令行工具，使用Java开发，代码位于https://github.com/FISCO-BCOS/console/tree/master-2.0，国内镜像https://gitee.com/FISCO-BCOS/console/tree/master-2.0/。

控制台每个版本发布会提供编译好的包，用户下载后配置后即可使用，为了下载控制台用户需要获取download_console.sh脚本。此脚本会从GitHub下载最新版本console.tar.gz，如果下载失败则尝试从官网CDN下载。下面的指令从国内镜像获取download_console.sh脚本并执行。

```
curl -#LO https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/download_
↪console.sh && bash download_console.sh
```

TASSL

FISCO BCOS国密版本需要使用TASSL生成国密版本的证书，部署工具会自动从GitHub下载，解压后放置于~/.fisco/tassl，如果碰到下载失败，请尝试从https://gitee.com/FISCO-BCOS/LargeFiles/blob/master/tools/tassl.tar.gz下载并解压后，放置于~/.fisco/tassl

账户生成脚本

FISCO BCOS在国密模式下使用sm2曲线和对应签名算法，在非国密场景使用secp256k1曲线和ecdsa签名算法。为方便用户提供了生成脚本，脚本生成私钥并以账户地址命名，支持PEM和PKCS12两种格式。详情请参考这里https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/account.html

get_account.sh脚本依赖于openssl指令，用于生成secp256k1私钥，如果从GitHub下载失败，可以尝试镜像地址 https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/get_account.sh

get_gm_account.sh脚本用于生成sm2私钥，依赖于TASSL。如果从GitHub下载失败，可以尝试镜像地址 https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/get_gm_account.sh

6.15.4 举例：使用国内镜像建链

本节以搭建2.4.0国密版本为例，使用国内镜像建链，非国密版本的操作类似，参考https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/installation.html

下载开发部署工具

```
curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.4.0/build_
↪chain.sh
```

如果下载失败请尝试curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/manual/build_chain.sh

下载二进制程序

开发部署工具（build_chain）会自动下载二进制程序，下载失败自动切换官网CDN，不需要用户关注。用户也可以手动下载二进制程序或编译源码，通过开发部署工具的-e选项指定，此时工具不会再去下载。-e选项参考https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/build_chain.html#e-optional

搭建2.4.0国密FISCO BCOS链

搭建国密版本时，开发部署工具还依赖tassl，工具会自动下载，如果失败请用户参考TASSL手动下载方法，下载解压后放置于~/.fisco/tassl。执行下面的指令，输出All completed即表示执行成功。

```
bash build_chain.sh -l 127.0.0.1:4 -p 30300,20200,8545 -g -v 2.4.0
```

6.15.5 举例：使用国内源码镜像编译

本节以CentOS 7 为例，从gitee镜像下载源码并编译，其他操作系统编译流程类似，请参考https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/get_executable.html#id2

安装依赖

```
sudo yum install -y epel-release
sudo yum install -y openssl-devel openssl cmake3 gcc-c++ git
```

下载源码

```
git clone https://gitee.com/FISCO-BCOS/FISCO-BCOS.git -b master-2.0
```

下载依赖包

FISCO BCOS在编译时会自动下载依赖包，每个依赖包有多个源。如果在编译阶段下载依赖包失败，请根据提示从下面的国内镜像手动下载，放置于FISCO-BCOS/deps/src目录下，再次make

https://gitee.com/FISCO-BCOS/LargeFiles/tree/master/libs

编译源码

```
cd FISCO-BCOS && mkdir build && cd build
cmake3 ..
make -j2
```

6.16 Log description

All group logs of FISCO BCOS are outputted in the log directory to the file `log_% YYYY% mm% dd% HH.% MM`, and the log format is customized to facilitate users to view the status of each group through the log. Please refer to the log configuration instructions Log configuration instructions

6.16.1 Log format

Each log record format is as follows:

```
# Log format:
log_level|time|[g:group_id][module_name] content

# Example log:
info|2019-06-26 16:37:08.253147|[g:3][CONSENSUS][PBFT]^^^^^^^^Report,num=0,
↪sealerIdx=0,hash=a4e10062...,next=1,tx=0,nodeIdx=2
```

The meaning of each field is as follows:

- `log_level`: Log level, currently mainly including`trace`, `debug`, `info`, `warning`, `error` and `fatal`, Where `fatal` is output when an extremely serious error occurs
- `time`: Log output time, accurate to nanoseconds
- `group_id`: Output log group ID
- `module_name`: Module keyword, for example, SYNC module keyword is SYNC and consensus module keyword is CONSENSUS
- `content`: Logging content

6.16.2 Common log description

Consensus packaging block log

Note:

- Only the consensus node will periodically output the consensus packaging log (you can use the command `tail -f log/* | grep "${group_id}.*++"` under the node directory to view the specified group consensus packaging log)
- The packaging log can check whether the consensus node of the specified group is abnormal, abnormal consensus nodes will not output packed logs

---

The following is an example of a consensus packaging log:

```
info|2019-06-26 18:00:02.551399|[g:2][CONSENSUS][SEALER]++++++++++++++++
↪Generating seal on,blkNum=1,tx=0,nodeIdx=3,hash=1f9c2b14...
```

The meaning of each field in the log is as follows:

- `blkNum`: Height of packed blocks
- `tx`: The number of transactions contained in the packed block
- `nodeIdx`: Index of the current consensus node
- `hash`: Hash of packed block

Consensus exception log

Network jitter, network disconnection, or configuration errors (like the inconsistency of a group's genesis block files) may all cause abnormal node consensus, The PBFT consensus node will output the `ViewChangeWarning` log, the example is as follows:

```
warning|2019-06-26 18:00:06.154102|[g:1][CONSENSUS][PBFT]ViewChangeWarning: not
↪caused by omit empty block ,v=5,toV=6,curNum=715,hash=ed6e856d...,nodeIdx=3,
↪myNode=e39000ea...
```

The meaning of each field of the log is as follows:

- v: PBFT consensus view of current node
- toV: The view the current node is trying to switch to
- curNum: The highest block height of the node
- hash: Node highest block hash
- nodeIdx: Current consensus node request
- myNode: Node ID of the current node

Block Commit Log

If the block consensus is successful or the node is synchronizing blocks from other nodes, the log will be output.

Note: Send a transaction to the node, if the transaction is processed, the non-free node will output the log (n the node directory, you can use the command `tail -f log/* | grep "${group_id}.*Report"` to check the status of the node block), If the log is not output, it indicates that the node is in an abnormal state. Please check whether the network connection is normal and the node certificate is valid.

The following is the block commit log:

```
info|2019-06-26 18:00:07.802027|[g:1][CONSENSUS][PBFT]^^^^^^^^Report,num=716,
↪sealerIdx=2,hash=dfd75e06...,next=717,tx=8,nodeIdx=3
```

The description of each field in the log is as follows:

- `num`: Committed block height
- `sealerIdx`: The consensus node index that packages the block
- `hash`: Committed block hash
- `next`: Next block height
- `tx`: Number of transactions included in the block
- `nodeIdx`: Current consensus node index

Network connection log

Note: n the node directory, you can check the network status by using the command `tail -f log/* | grep "connected count"`, If the number of network connections output by the log does not meet expectations, please check the node connection through the `netstat -anp | grep fisco-bcos` command

---

Examples of logs are as follows:

```
info|2019-06-26 18:00:01.343480|[P2P][Service] heartBeat,connected count=3
```

The meaning of each field in the log is as follows:

- `connected count`: Number of nodes that establish a P2P network connection with the current node

6.16.3 Log module keywords

The core module keywords in the FISCO BCOS log are as follows:

CHAPTER

SEVEN

APPLICATION DEVELOPMENT

7.1 Manage blockchain accounts

FISCO BCOS uses accounts to identify each individual user. In a blockchain system each account corresponds to a pair of public and private keys. The account named by the address string calculated by the secure one-way algorithm such as sha256 hash, that is account address. For distinguishing from the address of smart contract, the account address is often referred to as the external account address. The private key only known by the user corresponds to the password in the traditional authentication model. Users need to prove that they own the private key of the corresponding account through a secure cryptographic protocol for claiming their ownership of the account, and performing some sensitive account operations.

Important: In the previous tutorials, for simplifying the operation, we operate with the account provided by the tool by default. However, in actual application deployment, users need to create their own accounts and properly save the account private key to avoid serious security problems such as account private key leakage.

In this article, we will specifically introduce the creation, storage and use of accounts. Readers are required to have a basic knowledge of Linux.

FISCO BCOS provides the get_account script to create accounts, as well as Java SDK and console to store account private keys. Users can choose to store the account private key as a file in PEM or PKCS12 format according to their needs. The PEM format uses a plaintext storage private key, and the PKCS 12 encrypts and stores the private key using a user-provided password.

7.1.1 Create your account

Use script to create account

The usage of `get_gm_account.sh` is the same as `get_account.sh`.

1. get script

```
curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/console/master-2.0/tools/
↪get_account.sh && chmod u+x get_account.sh && bash get_account.sh -h
```

Note:

- If the get_account.sh script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/get_account.sh && chmod u+x get_account.sh && bash get_account.sh -h
- Please use curl -#LO https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/FISCO-BCOS/FISCO-BCOS/tools/tassl-1.0.2/tassl.tar.gz, and place in ~/.fisco/tassl

If you use guomi version fisco, please execute below command to get `get_gm_account.sh`

```
curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/console/master-2.0/tools/
↪get_gm_account.sh && chmod u+x get_gm_account.sh && bash get_gm_account.sh -h
```

Note:

- If the get_gm_account.sh script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/get_gm_account.sh && chmod u+x get_gm_account.sh && bash get_gm_account.sh -h

execute the above command and if you see the following output, you are downloading the correct script, otherwise please try again.

```
Usage: ./get_account.sh
    default       generate account and store private key in PEM format file
    -p            generate account and store private key in PKCS12 format file
    -k [FILE]     calculate the address of PEM format [FILE]
    -P [FILE]     calculate the address of PKCS12 format [FILE]
    -h Help
```

2. Generate private key in PEM format

- generate private key and address

```
bash get_account.sh
```

Execute the above command to get output similar to the following. It includes the account address and the private key PEM file with the account address as the file name.

```
[INFO] Account Address   : 0xee5fffba2da55a763198e361c7dd627795906ead
[INFO] Private Key (pem) : accounts/0xee5fffba2da55a763198e361c7dd627795906ead.pem
```

- Specify the calculation account address of PEM format

```
bash get_account.sh -k accounts/0xee5fffba2da55a763198e361c7dd627795906ead.pem
```

Execute the above command. The result is as follows

```
[INFO] Account Address   : 0xee5fffba2da55a763198e361c7dd627795906ead
```

3. Use script to generate PKCS12 format private key

- generate private key and address

```
bash get_account.sh -p
```

Execute the above command to get output similar to the following. You can follow the prompts to enter the password and generate the corresponding p12 file.

```
Enter Export Password:
Verifying - Enter Export Password:
[INFO] Account Address   : 0x02f1b23310ac8e28cb6084763d16b25a2cc7f5e1
[INFO] Private Key (p12) : accounts/0x02f1b23310ac8e28cb6084763d16b25a2cc7f5e1.p12
```

- Specify the calculation account address of p12 private key. Enter the p12 file password as prompted

```
bash get_account.sh -P accounts/0x02f1b23310ac8e28cb6084763d16b25a2cc7f5e1.p12
```

Execute the above command. The result is as follows

```
Enter Import Password:
MAC verified OK
[INFO] Account Address   : 0x02f1b23310ac8e28cb6084763d16b25a2cc7f5e1
```

Calling Java SDK to create an account

```
//create normal account
EncryptType.encryptType = 0;
//create national cryptography account, which uses for sending transaction to␣
↪national blockchain node
// EncryptType.encryptType = 1;
Credentials credentials = GenCredential.create();
//account address
String address = credentials.getAddress();
//account private key
String privateKey = credentials.getEcKeyPair().getPrivateKey().toString(16);
//account public key
String publicKey = credentials.getEcKeyPair().getPublicKey().toString(16);
```

For more details on the operation, to see Creating and Using a Specified External Account.

7.1.2 Store your account credential

- Java SDK supports loading via private key string or file, so the private key of the account can be stored in the database or in a local file.
- Local files support two storage formats, which are PKCS12 encrypted storage and PEM plaintext storage.
- When developing a service, you can select the storage management of private key according to the actual business scenario.

7.1.3 Load your account credential

Console loads private key file

The console provides the account generation script get_account.sh. The generated account file is in the accounts directory, and the account file loaded by console must be placed in this directory.

The console startup methods are as follows:

```
./start.sh
./start.sh groupID
./start.sh groupID -pem pemName
./start.sh groupID -p12 p12Name
```

Default startup

Console randomly generates an account, startup with the group number specified in console configuration file.

```
./start.sh
```

Specify group number to startup

Console randomly generates an account, startup with the group number specified on the command line.

```
./start.sh 2
```

- Note: The specified group needs to configure bean in console configuration file.

Use PEM private key file to startup

- Startup with the account of the specified pem file. Enter the parameters: group number, -pem, and pem file path

```
./start.sh 1 -pem accounts/0xebb824a1122e587b17701ed2e512d8638dfb9c88.pem
```

Use PKCS12 private key file to startup

- Startup with the account of the specified p12 file. Enter the parameters: group number, -p12, and p12 file path

```
./start.sh 1 -p12 accounts/0x5ef4df1b156bc9f077ee992a283c2dbb0bf045c0.p12
Enter Export Password:
```

Java SDK loads private file

If the account private key file in PEM or PKCS12 format is generated by the account generation script get_accounts.sh, the account can be used by loading the PEM or PKCS12 account private key file. There are two classes of private keys to be loaded: P12Manager and PEMManager. P12Manager is used to load the private key file in PKCS12 format. PEMManager is used to load the private key file in PEM format.

- P12Manager usage example: Load p12 key.

```
// init BcosSDK
BcosSDK sdk =  BcosSDK.build(configFile);
// get Client
Client client = sdk.getClient(Integer.valueOf(1));
// get CryptoSuite
CryptoSuite cryptoSuite = client.getCryptoSuite();
// load p12 key
cryptoSuite.loadAccount("p12", p12AccountFilePath, password);
```

- PEMManager usage example:

Load key.

```
// init BcosSDK
BcosSDK sdk =  BcosSDK.build(configFile);
// get Client
Client client = sdk.getClient(Integer.valueOf(1));
// get CryptoSuite
CryptoSuite cryptoSuite = client.getCryptoSuite();
// load pem key
cryptoSuite.loadAccount("pem", pemAccountFilePath, null);
```

7.1.4 Account address calculation

The account address of FISCO BCOS is calculated by the ECDSA public key. The hexadecimal of ECDSA public key represents the calculation of keccak-256sum hash, and the hexadecimal of the last 20 bytes of the calculation result is taken as the account address. Each byte requires two hexadecimal to represent, so the length of account address is 40. FISCO BCOS's account address is compatible with Ethereum.

Note: keccak-256sum is different from `SHA3`. For details to refer to here.

Ethernet Address Generation

1. generate ECDSA private key

First, we use OpenSSL to generate an elliptic curve private key. The parameters of the elliptic curve are secp256k1. To run the following command to generate a private key in PEM format and save it in the ecprivkey.pem file.

```
openssl ecparam -name secp256k1 -genkey -noout -out ecprivkey.pem
```

Execute the following instructions to view the contents of the file.

```
cat ecprivkey.pem
```

You can see output similar to the following

```
-----BEGIN EC PRIVATE KEY-----
MHQCAQEEINHaCmLhw9S9+vD0IOSUd9IhHO9bBVJXTbbBeTyFNvesoAcGBSuBBAAK
oUQDQgAEjSUbQAZn4tzHnsbeahQ2J0AeMu0iNOxpdpyPo3j9Diq3qdljrv07wvjx
zOzLpUNRcJCC5hnU500MD+4+Zxc8zQ==
-----END EC PRIVATE KEY-----
```

Next, to calculate the public key based on the private key. To execute the following command.

```
openssl ec -in ecprivkey.pem -text -noout 2>/dev/null| sed -n '7,11p' | tr -d ": \n
↪" | awk '{print substr($0,3);}'
```

You can get output similar to the following

```
8d251b400667e2dcc79ec6de6a143627401e32ed2234ec69769c8fa378fd0e2ab7a9d963aefd3bc2f8f1cceccba543517
```

2. Calculate the address based on the public key

In this section, we calculate the corresponding account address based on the public key. The keccak-256sum tool we need to get is available for download from here.

```
openssl ec -in ecprivkey.pem -text -noout 2>/dev/null| sed -n '7,11p' | tr -d ": \n
↪" | awk '{print substr($0,3);}' | ./keccak-256sum -x -l | tr -d ' -' | tail -c 41
```

Get the output similar to the following, which is the calculated account address.

```
dcc703c0e500b653ca82273b7bfad8045d85a470
```

7.2 Smart contract development

FISCO BCOS platform currently supports two smart contract forms which are Solidity and pre-compiled.

- The Solidity contract is the same as Ethereum.

- The KVTable contract get/set interfacr and Table contract CRUD interface supporting the distributed storage pre-compilation contract in the Solidity contract, which can store the data of Solidtiy contract in the AMDB table structure, realizes the separation of contract logic and data.
- The precompiled (precompiled) contract is developed in C++ and built into the FISCO BCOS platform. It has better performance than the Solidity contract. Its contract interface that needs to be pre-determined when compiling, is suitable for the scenarios with fixed logic but consensus, such as group configuration. The development of precompiled contracts will be introduced in the next section.

7.2.1 Solidity contract development

- Solidity official file
- Remix online IDE

7.2.2 Use KVTable contract get/set interface

Note:

- To make the table created by AMDB accessible to multiple contracts, it should have a unique name that acknowledged globally. So it is unable to create tables with same name within one group on the same chain
- KVTable added in v2.3.0, the version of chain >= v2.3.0 can use this function.

KVTable contract use key/value type to get/set data of table, code is as follows:

```
pragma solidity ^0.4.24;

contract KVTableFactory {
    function openTable(string) public view returns (KVTable);
    function createTable(string, string, string) public returns (int256);
}

//one record
contract Entry {
    function getInt(string) public constant returns (int256);
    function getUInt(string) public constant returns (uint256);
    function getAddress(string) public constant returns (address);
    function getBytes64(string) public constant returns (bytes1[64]);
    function getBytes32(string) public constant returns (bytes32);
    function getString(string) public constant returns (string);

    function set(string, int256) public;
    function set(string, uint256) public;
    function set(string, string) public;
    function set(string, address) public;
}

//KVTable per permiary key has only one Entry
contract KVTable {
    function get(string) public view returns (bool, Entry);
    function set(string, Entry) public returns (int256);
    function newEntry() public view returns (Entry);
}
```

Offer a use case of `KVTableTest.sol`, code is as follows:

```
pragma solidity ^0.4.24;
import "./Table.sol";
```

(continues on next page)

(continued from previous page)

```
contract KVTableTest {
    event SetResult(int256 count);

    KVTableFactory tableFactory;
    string constant TABLE_NAME = "t_kvtest";

    constructor() public {
        //The fixed address is 0x1010 for KVTableFactory
        tableFactory = KVTableFactory(0x1010);
        // the parameters of createTable are tableName,keyField,"vlaueFiled1,
↪vlaueFiled2,vlaueFiled3,..."
        tableFactory.createTable(TABLE_NAME, "id", "item_price,item_name");
    }

    //get record
    function get(string id) public view returns (bool, int256, string) {
        KVTable table = tableFactory.openTable(TABLE_NAME);
        bool ok = false;
        Entry entry;
        (ok, entry) = table.get(id);
        int256 item_price;
        string memory item_name;
        if (ok) {
            item_price = entry.getInt("item_price");
            item_name = entry.getString("item_name");
        }
        return (ok, item_price, item_name);
    }

    //set record
    function set(string id, int256 item_price, string item_name)
        public
        returns (int256)
    {
        KVTable table = tableFactory.openTable(TABLE_NAME);
        Entry entry = table.newEntry();
        // the length of entry's field value should < 16MB
        entry.set("id", id);
        entry.set("item_price", item_price);
        entry.set("item_name", item_name);
        // the first parameter length of set should <= 255B
        int256 count = table.set(id, entry);
        emit SetResult(count);
        return count;
    }
}
```

`KVTableTest.sol` calls `KVTable` contract to create a user table `t_kvtest`. The table structure of `t_kvtest`is as follows. This table records the materials in a company's warehouse, takes the unique material id as the key, and saves the name and price of the materials.

7.2.3 To use Table contract CRUD interface

Accessing AMDB requires using the AMDB-specific smart contract interface `Table.sol` which is a database contract that can create tables and add, delete, and modify the tables.

Note: To make the table created by AMDB accessible to multiple contracts, it should have a unique name that acknowledged globally. So it is unable to create tables with same name within one group on the same chain. The

CRUD interface of Table contract can have multiple records under a key. When it is used, it will perform batch data operations, including batch writing and range query. For this feature, it is recommended to use MySQL as the back-end database. When using the get/set interface of KVTable, it is recommended to use rocksdb as the back-end database. Because rocksdb is a non relational database stored in key value, the single key operation efficiency is higher when using KVTable interface.

`Table.sol` file code is as follows:

```
pragma solidity ^0.4.24;

contract TableFactory {
    function openTable(string) public constant returns (Table);  // open table
    function createTable(string,string,string) public returns(int);  // create␣
↪table
}

// inquiry conditions
contract Condition {
    //equal to
    function EQ(string, int) public;
    function EQ(string, string) public;

    //unequal to
    function NE(string, int) public;
    function NE(string, string)  public;

    //greater than
    function GT(string, int) public;
    //greater than or equal to
    function GE(string, int) public;

    //smaller than
    function LT(string, int) public;
    //smaller than or equal to
    function LE(string, int) public;

    //limit the number of return record
    function limit(int) public;
    function limit(int, int) public;
}

// single entry data record
contract Entry {
    function getInt(string) public constant returns(int);
    function getAddress(string) public constant returns(address);
    function getBytes64(string) public constant returns(byte[64]);
    function getBytes32(string) public constant returns(bytes32);
    function getString(string) public constant returns(string);

    function set(string, int) public;
    function set(string, string) public;
    function set(string, address) public;
}

// data record set
contract Entries {
    function get(int) public constant returns(Entry);
    function size() public constant returns(int);
}

// Table main type
```

(continues on next page)

(continued from previous page)

```
contract Table {
    // select interface
    function select(string, Condition) public constant returns(Entries);
    // insert interface
    function insert(string, Entry) public returns(int);
    // update interface
    function update(string, Entry, Condition) public returns(int);
    // remove interface
    function remove(string, Condition) public returns(int);

    function newEntry() public constant returns(Entry);
    function newCondition() public constant returns(Condition);
}
```

Note:

- The type of key in insert, remove, update and select functions of Table contract is string, and the maximum length is 255 characters
- The key type of the get/set interface of the Entry is string, with the maximum length of 255 characters. The types supported by value are int256 (int), address and string, of which string cannot exceed 16MB.

To provide a contract case `TableTest.sol`. The code is as follows:

```
pragma solidity >=0.6.10 <0.8.20;
pragma experimental ABIEncoderV2;

import "./Table.sol";

contract TableTest {
    event CreateResult(int256 count);
    event InsertResult(int256 count);
    event UpdateResult(int256 count);
    event RemoveResult(int256 count);

    TableFactory tableFactory;
    string constant TABLE_NAME = "t_test";
    constructor() public {
        tableFactory = TableFactory(0x1001); //The fixed address is 0x1001 for␣
↪TableFactory
        // the parameters of createTable are tableName,keyField,"vlaueFiled1,
↪vlaueFiled2,vlaueFiled3,..."
        tableFactory.createTable(TABLE_NAME, "name", "item_id,item_name");
    }

    //select records
    function select(string memory name)
    public
    view
    returns (string[] memory, int256[] memory, string[] memory)
    {
        Table table = tableFactory.openTable(TABLE_NAME);

        Condition condition = table.newCondition();

        Entries entries = table.select(name, condition);
        string[] memory user_name_bytes_list = new string[](
            uint256(entries.size())
        );
        int256[] memory item_id_list = new int256[](uint256(entries.size()));
```

(continues on next page)

(continued from previous page)

```
        string[] memory item_name_bytes_list = new string[](
            uint256(entries.size())
        );

        for (int256 i = 0; i < entries.size(); ++i) {
            Entry entry = entries.get(i);

            user_name_bytes_list[uint256(i)] = entry.getString("name");
            item_id_list[uint256(i)] = entry.getInt("item_id");
            item_name_bytes_list[uint256(i)] = entry.getString("item_name");
        }

        return (user_name_bytes_list, item_id_list, item_name_bytes_list);
    }
    //insert records
    function insert(string memory name, int256 item_id, string memory item_name)
    public
    returns (int256)
    {
        Table table = tableFactory.openTable(TABLE_NAME);

        Entry entry = table.newEntry();
        entry.set("name", name);
        entry.set("item_id", item_id);
        entry.set("item_name", item_name);

        int256 count = table.insert(name, entry);
        emit InsertResult(count);

        return count;
    }
    //update records
    function update(string memory name, int256 item_id, string memory item_name)
    public
    returns (int256)
    {
        Table table = tableFactory.openTable(TABLE_NAME);

        Entry entry = table.newEntry();
        entry.set("item_name", item_name);

        Condition condition = table.newCondition();
        condition.EQ("name", name);
        condition.EQ("item_id", item_id);

        int256 count = table.update(name, entry, condition);
        emit UpdateResult(count);

        return count;
    }
    //remove records
    function remove(string memory name, int256 item_id) public returns (int256) {
        Table table = tableFactory.openTable(TABLE_NAME);

        Condition condition = table.newCondition();
        condition.EQ("name", name);
        condition.EQ("item_id", item_id);

        int256 count = table.remove(name, condition);
        emit RemoveResult(count);
```

(continues on next page)

(continued from previous page)

```
        return count;
    }
}
```

`TableTest.sol` has called the intelligent contract `Table.sol` of AMDB, which implements creating the user table `t_test` and the functions of adding, deleting and changing `t_test`. The `t_test` table is structured as follows. This table records the item and item's numbers used by a employees.

The client requiring to call the contract code which is converted to Java file, needs to put TableTest.sol and Table.sol into the directory contracts/solidity, and TableTest.java is generated by the compile script of sol2java.sh.

7.2.4 Precompiled contract development

1. Introduction

Precompiled contract is a natively supported feature of Ethereum: a contract that uses C++ code to implement specific functions at the underlying platform for EVM module calling. FISCO BCOS inherits and extends this feature, and has developed a powerful and easy-to-expand framework on this basis of it.precompiled design principle.

This article is an introductory to guide users on how to implement their own precompiled contracts and how to call them.

2. Implement precompiled contracts

2.1 Process

The process of implementing a pre-compiled contract:

- assign contract address

For calling a solid contract or pre-compiled contract, you need to distinguish it by the contract address and address space.

The address range of user assigned interval is `0x5001-0xffff`. Users needs to assign an unused address to the new precompiled contract. The precompiled contract addresses must be unique and not conflicting.

List of precompiled contracts and address assignments implemented in FISCO BCOS:

- define contract interface

It is similar to solidity contract. When designing a contract, you need to determine the ABI interface of the contract first. The ABI interface rules of the precompiled contract are exactly the same as the solidity. solidity ABI link.

> When defining a precompiled contract interface, you usually need to define a solidity contract with the same interface, and empty the function body of all interfaces. This contract is called interface contract of the precompiled contract. The interface contract need to be used when calling the precompiled contract.

```
pragma solidity ^0.4.24;
contract Contract_Name {
    function interface0(parameters ... ) {}
    ....
    function interfaceN(parameters ... ) {}
}
```

- design storage structure

When a precompiled contract involves a storage operation, it needs to determine the stored table information (table name and table structure. The stored data will be uniformly abstracted into a table structure in FISCO BCOS)storage structure.

Note: This process can be omitted without involving a storage operation.

- implement contract logic

For implementing the calling logic of the new contract, you need to implement a new C++ class that needs to inherit [precompiled] (https://github.com/FISCO-BCOS/FISCO-BCOS/blob/master-2.0/libprecompiled/Precompiled.h) #L37) to overload the call function for achieving the calling behaviour of each interface.

```
    // libprecompiled/Precompiled.h
    class Precompiled
    {
        virtual bytes call(std::shared_ptr<ExecutiveContext> _context,␣
↪bytesConstRef _param,
            Address const& _origin = Address()) = 0;
    };
```

The call function has three parameters:

`std::shared_ptr<ExecutiveContext> _context` : the context for the transaction execution saving

`bytesConstRef _param` : calling the parameter information of the contract. The calling corresponding contract interface and the parameters of interface can be obtained from _param parsing.

`Address const& _origin` : transaction sender for permission control

How to implement a Precompiled class will be detailed in the sample below.

- register contract

Finally, the contract address and the corresponding class need to be registered to the execution context of the contract, so that the execution logic of the contract can be correctly recognized when the precompiled contract is called by the address. To view the registered pre-compiled contract list.

Registration path:

```
    file        libblockverifier/ExecutiveContextFactory.cpp
    function    initExecutiveContext
```

2.2 sample contract development

```
// HelloWorld.sol
pragma solidity ^0.4.24;

contract HelloWorld{
    string name;

    function HelloWorld(){
       name = "Hello, World!";
    }

    function get()constant returns(string){
        return name;
    }

    function set(string n){
            name = n;
    }
}
```

The above source code is the HelloWorld contract written by solidity. This chapter will implement a precompiled contract with the same function to enable user step by step to have an visual understanding to the precompiled contract. sample c++source code path:

```
    libprecompiled/extension/HelloWorldPrecompiled.h
    libprecompiled/extension/HelloWorldPrecompiled.cpp
```

2.2.1 assign contract address

Referring to the address range, the address of the HelloWorld precompiled contract is assigned as:

```
0x5001
```

2.2.2 define contract interface

We need to implement the HelloWorld contract function. The interface is the same as the HelloWorld interface. HelloWorldPrecompiled interface contract:

```
pragma solidity ^0.4.24;

contract HelloWorldPrecompiled {
    function get() public constant returns(string) {}
    function set(string _m) {}
}
```

2.2.3 design storage structure

HelloWorldPrecompiled needs to store the string value of the set, so when it comes to storage operations, you need to design the stored table structure.

table name: `_ext_hello_world_`

table structure:

The table stores only a pair of key-value pairs. The key field is hello_key and the value field is hello_value. For storing the corresponding string value, it can be modified by the set(string) interface and obtained by the get() interface.

2.2.4 implement call logic

To add the HelloWorldPrecompiled class, overload the call function, and implement the calling behavior of all interfaces.call function source code.

The user-defined Precompiled contract needs to add a new class for defining the calling behaviour of the contract in the class. In the example, for adding the HelloWorldPrecompiled class, the following work must complete:

- interface registration

```
// define all interfaces in the class
const char* const HELLO_WORLD_METHOD_GET = "get()";
const char* const HELLO_WORLD_METHOD_SET = "set(string)";

// register interface in the constructor
HelloWorldPrecompiled::HelloWorldPrecompiled()
{
    // name2Selector is a member of the Base class Precompiled, which saves the␣
↪mapping relationship of the interface call.
    name2Selector[HELLO_WORLD_METHOD_GET] = getFuncSelector(HELLO_WORLD_METHOD_
↪GET);
```

(continues on next page)

(continued from previous page)

```
    name2Selector[HELLO_WORLD_METHOD_SET] = getFuncSelector(HELLO_WORLD_METHOD_
↪SET);
}
```

- table creation

define the table's name and field structure

```
// define the name
const std::string HELLO_WORLD_TABLE_NAME = "_ext_hello_world_";
// define the key field
const std::string HELLOWORLD_KEY_FIELD = "key";
// "field0,field1,field2" define other fields, multiple fields separated by commas,
↪ such as "field0,field1,field2"
const std::string HELLOWORLD_VALUE_FIELD = "value";
```

```
// In the call function, the table is opened when it exists, otherwise the table␣
↪is created first.
Table::Ptr table = openTable(_context, HELLO_WORLD_TABLE_NAME);
if (!table)
{
    //  table is created while it does not exist
    table = createTable(_context, HELLO_WORLD_TABLE_NAME, HELLOWORLD_KEY_FIELD,
        HELLOWORLD_VALUE_FIELD, _origin);
    if (!table)
    {
        //  fail to create and return false
    }
}
```

After getting the operation handle of the table, user can implement the specific logic of the table operation.

- call interface distinguishing

Parsing _param with getParamFunc can distinguish the call interface.

Note: the contract interface must be registered in the constructor

```
uint32_t func = getParamFunc(_param);
if (func == name2Selector[HELLO_WORLD_METHOD_GET])
{
    // get() call interface logic
}
else if (func == name2Selector[HELLO_WORLD_METHOD_SET])
{
    // set(string) call interface logic
}
else
{
    // unknown interface, call error, return error
}
```

- Parameter parsing and result return

The parameters during calling the contract are included in the _param parameter of the call function. They are encoded according to the Solidity ABI format. The `dev::eth::ContractABI` utility class can be used to parse the parameters. Similarly, when the interface returns, the return value also needs to be encoded according to the format. Solidity ABI。

In `dev::eth::ContractABI` class, we need to use two interfaces `abiIn abiOut`. The former serializes the former user parameter and the latter can parse the parameter from the serialized data.

```
// to serialize ABI data. c++ type data serialized to the format used by evm
// _id: The corresponding string of the function interface declaration, which␣
↪generally default to ""
template <class... T> bytes abiIn(std::string _id, T const&... _t)
// to parse serialized data into c++ type data
template <class... T> void  abiOut(bytesConstRef _data, T&... _t)
```

The sample code below shows how the interface works:

```
// for transfer interface: transfer(string,string,uint256)

// Parameter1
std::string str1 = "fromAccount";
// Parameter12
std::string str2 = "toAccount";
// Parameter13
uint256 transferAmoumt = 11111;

dev::eth::ContractABI abi;
// serialization, abiIn first string parameter default to ""
bytes out = abi.abiIn("", str1, str2, transferAmoumt);

std::string strOut1;
std::string strOut2;
uint256 amoumt;

// parse parameter
abi.abiOut(out, strOut1, strOut2, amount);
// parse after
// strOut1 = "fromAccount";
// strOut2 = "toAccount"
// amoumt = 11111
```

Finally, the HelloWorldPrecompiled call function is implemented completely.source code link.

```
bytes HelloWorldPrecompiled::call(dev::blockverifier::ExecutiveContext::Ptr _
↪context,
    bytesConstRef _param, Address const& _origin)
{
    // parse function interface
    uint32_t func = getParamFunc(_param);
    //
    bytesConstRef data = getParamData(_param);
    bytes out;
    dev::eth::ContractABI abi;

    // open table
    Table::Ptr table = openTable(_context, HELLO_WORLD_TABLE_NAME);
    if (!table)
    {
        // table is created while it does not exist
        table = createTable(_context, HELLO_WORLD_TABLE_NAME, HELLOWORLD_KEY_FIELD,
            HELLOWORLD_VALUE_FIELD, _origin);
        if (!table)
        {
            // fail to create table. no authority?
            out = abi.abiIn("", CODE_NO_AUTHORIZED);
            return out;
        }
    }

    // to distinguish the calling interface and specify the calling logic of each␣
↪interface
```

(continues on next page)

(continued from previous page)

```
    if (func == name2Selector[HELLO_WORLD_METHOD_GET])
    {  // get() call interface
        // default to return value
        std::string retValue = "Hello World!";
        auto entries = table->select(HELLOWORLD_KEY_FIELD_NAME, table->
↪newCondition());
        if (0u != entries->size())
        {
            auto entry = entries->get(0);
            retValue = entry->getField(HELLOWORLD_VALUE_FIELD);
        }
        out = abi.abiIn("", retValue);
    }
    else if (func == name2Selector[HELLO_WORLD_METHOD_SET])
    {  // set(string) call interface

        std::string strValue;
        abi.abiOut(data, strValue);
        auto entries = table->select(HELLOWORLD_KEY_FIELD_NAME, table->
↪newCondition());
        auto entry = table->newEntry();
        entry->setField(HELLOWORLD_KEY_FIELD, HELLOWORLD_KEY_FIELD_NAME);
        entry->setField(HELLOWORLD_VALUE_FIELD, strValue);

        int count = 0;
        if (0u != entries->size())
        {  // value exists, update
            count = table->update(HELLOWORLD_KEY_FIELD_NAME, entry, table->
↪newCondition(),
                std::make_shared<AccessOptions>(_origin));
        }
        else
        {  // value does not exist, insert
            count = table->insert(
                HELLOWORLD_KEY_FIELD_NAME, entry, std::make_shared<AccessOptions>(_
↪origin));
        }

        if (count == CODE_NO_AUTHORIZED)
        {  // no table operation authority
            PRECOMPILED_LOG(ERROR) << LOG_BADGE("HelloWorldPrecompiled") << LOG_
↪DESC("set")
                                   << LOG_DESC("non-authorized");
        }
        out = abi.abiIn("", u256(count));
    }
    else
    {  // parameter error, unknown calling interface
        PRECOMPILED_LOG(ERROR) << LOG_BADGE("HelloWorldPrecompiled") << LOG_DESC("␣
↪unknown func ")
                               << LOG_KV("func", func);
        out = abi.abiIn("", u256(CODE_UNKNOW_FUNCTION_CALL));
    }

    return out;
}
```

2.2.5 Register contract and compile source code

- Register Precompiled contract. Modify `FISCO-BCOS/cmake/templates/UserPrecompiled.h.in`, register the address of `HelloWorldPrecompiled` contract in its below function. Default to be existed, and revoke annotation.

```
void␣
↪dev::blockverifier::ExecutiveContextFactory::registerUserPrecompiled(dev::blockverifier::Executi
↪context)
{
    // Address should in [0x5001,0xffff]
    context->setAddress2Precompiled(Address(0x5001), std::make_shared
↪<dev::precompiled::HelloWorldPrecompiled>());
}
```

- Compile source code. Please read here to install dependencies and compile source code.

Note: The implemented HelloWorldPrecompiled.cpp and header files should be placed under FISCO-BCOS/libprecompiled/extension directory.

- Build FISCO BCOS consortium blockchain Given that it is stored under `FISCO-BCOS/build` directory, use the following instruction to build chain for node 4. For more options please read here.

```
bash ../manual/build_chain.sh -l 127.0.0.1:4 -e bin/fisco-bcos
```

3 Calling

From the user's viewing, the pre-compiled contract is basically the same as the solidity contract. The only difference is the solidity contract can obtain the contracted address after deployment while the per-compiled contract can be used directly without deployment because of the pre-compiled contract address is pre-allocated.

3.1 Call HelloWorld precompiled contract using console

Create HelloWorldPrecompiled.sol file under console contracts/solidity with the content of interface declaration:

```
pragma solidity ^0.4.24;
contract HelloWorldPrecompiled{
    function get() public constant returns(string);
    function set(string n);
}
```

After the nodes are built by compiled binaries, deploy console v1.0.2 and above version and execute the following statement to call contract:

```
[group:1]> call HelloWorldPrecompiled.sol 0x5001 get
Hello World!

[group:1]> call HelloWorldPrecompiled.sol 0x5001 set "Hello, FISCO BCOS"
0xb0542ffab97f93b8cebadb39d54825b1f709c2f185c093e8ed39ce74b5391b83

[group:1]> call HelloWorldPrecompiled.sol 0x5001 get
Hello, FISCO BCOS

[group:1]>
```

3.2 Call solidity

Now, we try to create precompiled contract object and call its interface in Solidity contract. Create HelloWorldHelper.sol file in console contracts/solidity with the following content:

```
pragma solidity ^0.4.24;
import "./HelloWorldPrecompiled.sol";

contract HelloWorldHelper {
    HelloWorldPrecompiled hello;
    function HelloWorldHelper() {
        // call HelloWorld precompiled contract
        hello = HelloWorldPrecompiled(0x5001);
    }
    function get() public constant returns(string) {
        return hello.get();
    }
    function set(string m) {
        hello.set(m);
    }
}
```

Deploy HelloWorldHelper contract and call the interface of HelloWorldHelper contract, you will get the following result:

```
[group:1]> deploy HelloWorldHelper.sol
0x6096966a7c06006385ec0eb774f6dc783a8ee4f0

[group:1]> call HelloWorldHelper.sol 0x6096966a7c06006385ec0eb774f6dc783a8ee4f0 get
Hello, FISCO BCOS

[group:1]> call HelloWorldHelper.sol 0x6096966a7c06006385ec0eb774f6dc783a8ee4f0 set "Hello World"
0x62b0277f4b265cb40c64a05f4c5ca52307013dcbb678ab9092c4fec512b40c79

[group:1]> call HelloWorldHelper.sol 0x6096966a7c06006385ec0eb774f6dc783a8ee4f0 get
Hello World

[group:1]>
```

7.3 Java SDK

Java SDK provides the Java API for FISCO BCOS client. You can easily and efficiently build your blockchain applications. The version only supports FISCO BCOS 2.0+.

It includes functions:

- Contract compiling.
- Interacting with FISCO BCOS JSON-RPC interface.
- constructing and sending transactions.
- Advanced Messages Onchain Protocol(AMOP) functions.
- Contract event subscription.
- Encoding and decoding data with ABI.
- Account Management.

7.4 Python SDK

7.5 Go SDK

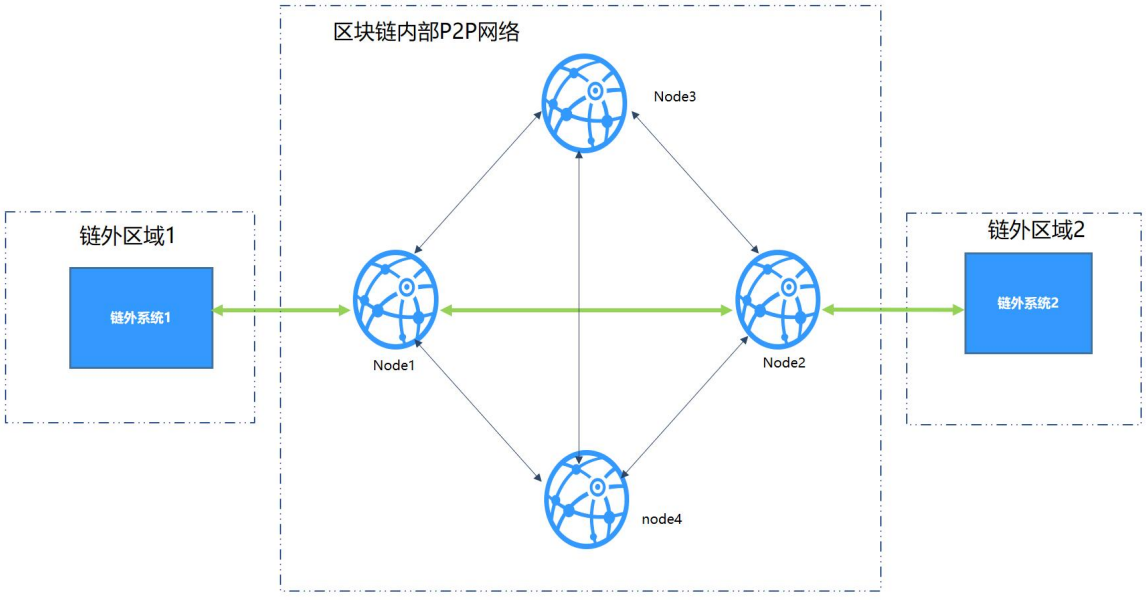
7.6 AMOP

7.6.1 Introduction

Advanced Messages Onchain Protocol (AMOP) aims to provide a secure and efficient message channel for alliance chain. Each agency in the alliance chain can use AMOP to communicate as long as they deploy blockchain nodes, whether they are Sealer or Observer. AMOP has the following advantages:

- Real-time: AMOP messages do not rely on blockchain transactions and consensus. Messages are transmitted in real time among nodes with a milliseconds delay.
- Reliability: When AMOP message is transmitting, it can automatically search all feasible link in blockchain network for communication. As long as at least one link is available, the message is guaranteed to be reachable.
- Efficiency: The AMOP message has simple structure and efficient processing logic. AMOP message's simple structure and efficient processing logic makes it to fully utilize network bandwidth and require a small amount of CPU usage only.
- Security: All communication links of AMOP use SSL encryption. The encryption algorithm is configurable,and the topic supports identity authentication mechanism.
- Easy to use: No additional configuration is required in SDK when using AMOP.

7.6.2 Logical architecture



We take the typical IDC architecture of the bank as the example to overview each region:

- SF region: The business service region inside the agency. The business subsystem in this region uses blockchain SDK. the configured SDK connects to the blockchain node.
- Blockchain P2P network: This is a logical region and deployed blockchain nodes of each agency. Blockchain nodes can also be deployed inside the agency.

7.6.3 Configuration

AMOP does not require any additional configuration. The following is a configuration case for [Java SDK]](./../configuration.md) . SDK configuration（Spring Bean）：

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.
↪springframework.org/schema/p"
  xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.
↪springframework.org/schema/aop"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
          http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

<!-- AMOP message processing pool configuration, which can be configured according␣
↪to actual needs -->
<bean id="pool" class="org.springframework.scheduling.concurrent.
↪ThreadPoolTaskExecutor">
  <property name="corePoolSize" value="50" />
  <property name="maxPoolSize" value="100" />
  <property name="queueCapacity" value="500" />
  <property name="keepAliveSeconds" value="60" />
  <property name="rejectedExecutionHandler">
    <bean class="java.util.concurrent.ThreadPoolExecutor.AbortPolicy" />
  </property>
</bean>

<!-- group information configuration -->
  <bean id="groupChannelConnectionsConfig" class="org.fisco.bcos.channel.handler.
↪GroupChannelConnectionsConfig">
      <property name="allChannelConnections">
        <list>
          <bean id="group1"  class="org.fisco.bcos.channel.handler.
↪ChannelConnections">
            <property name="groupId" value="1" />
            <property name="connectionsStr">
              <list>
                <value>127.0.0.1:20200</value> <!-- format: IP:port -->
                <value>127.0.0.1:20201</value>
              </list>
            </property>
          </bean>
        </list>
      </property>
    </bean>

  <!-- blockchain node information configuration -->
    <bean id="channelService" class="org.fisco.bcos.channel.client.Service"␣
↪depends-on="groupChannelConnectionsConfig">
      <property name="groupId" value="1" />
      <property name="orgID" value="fisco" />
      <property name="allChannelConnections" ref="groupChannelConnectionsConfig"></
↪property>
      <!-- If you want to enable topic authentication, please uncomment the␣
↪following configuration. -->
      <!-- <property name="topic2KeyInfo" ref="amopVerifyTopicToKeyInfo"></
↪property>-->
```

(continues on next page)

(continued from previous page)

```
    </bean>

  <!-- If you want to enable topic authentication, please uncomment the following␣
↪configuration. -->
  <!--
    <bean class="org.fisco.bcos.channel.handler.AMOPVerifyTopicToKeyInfo" id=
↪"amopVerifyTopicToKeyInfo">
                <property name="topicToKeyInfo">
                        <map>
                                <entry key="${topicname}" value-ref=
↪"AMOPVerifyKeyInfo_${topicname}" />
                        </map>
                </property>
        </bean>
        -->

  <!-- If you are a topic producer, you need to configure the publicKey property.
                Each authenticated consumer holds a different public-private key␣
↪pair.
                Please list the public key files of all the authenticated␣
↪consumers.
        -->
  <!--
        <bean class="org.fisco.bcos.channel.handler.AMOPVerifyKeyInfo" id=
↪"AMOPVerifyKeyInfo_${topicname}">
                <property name="publicKey">
                        <list>
                                <value>classpath:$consumer_public_key_1.pem$</
↪value>
                                <value>classpath:$consumer_public_key_2.pem$</
↪value>
                        </list>
                </property>
        </bean>
        -->

  <!-- If you are a topic consumer, you need to configure the privateKey property.
                This private key will authenticate you to the corresponding topic␣
↪producer.
        -->
  <!--
        <bean class="org.fisco.bcos.channel.handler.AMOPVerifyKeyInfo" id=
↪"AMOPVerifyKeyInfo_${topicname}">
                <property name="privateKey" value="classpath:$consumer_private_key.
↪pem$"></property>
        </bean>
        -->
```

7.6.4 SDK uses

AMOP's messaging is based on the topic mechanism. A topic is set in a server first. When a client sends a message to the topic, the server can receive soon. AMOP supports multiple topic for messaging in the same blockchain network. Topic supports any number of servers and clients. When multiple servers follow on the same topic, the topic messages are randomly sent to one of available servers.

Server code:

```
package org.fisco.bcos.channel.test.amop;

import org.fisco.bcos.channel.client.Service;
```

(continues on next page)

(continued from previous page)

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.HashSet;
import java.util.Set;

public class Channel2Server {
    static Logger logger = LoggerFactory.getLogger(Channel2Server.class);

    public static void main(String[] args) throws Exception {
        if (args.length < 1) {
          System.out.println("Param: topic");
          return;
        }

        String topic = args[0];

        logger.debug("init Server");

        ApplicationContext context = new ClassPathXmlApplicationContext(
↪"classpath:applicationContext.xml");
        Service service = context.getBean(Service.class);

        // set topic to support multiple topic
        Set<String> topics = new HashSet<String>();
        topics.add(topic);
        service.setTopics(topics);

        // PushCallback class, is used to handles messages. see the Callback code.
        PushCallback cb = new PushCallback();
        service.setPushCallback(cb);

        System.out.println("3s...");
        Thread.sleep(1000);
        System.out.println("2s...");
        Thread.sleep(1000);
        System.out.println("1s...");
        Thread.sleep(1000);

        System.out.println("start test");
        System.out.println(
↪"======================================================================");

        // launch service
        service.run();
    }
}
```

The server's PushCallback class case:

```
package org.fisco.bcos.channel.test.amop;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import org.fisco.bcos.channel.client.ChannelPushCallback;
```

(continues on next page)

(continued from previous page)

```
import org.fisco.bcos.channel.dto.ChannelPush;
import org.fisco.bcos.channel.dto.ChannelResponse;

class PushCallback extends ChannelPushCallback {
    static Logger logger = LoggerFactory.getLogger(PushCallback2.class);

    // onPush method is called when an AMOP message is received.
    @Override
    public void onPush(ChannelPush push) {
        DateTimeFormatter df = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        logger.debug("push:" + push.getContent());

        System.out.println(df.format(LocalDateTime.now()) + "server:push:" + push.
↪getContent());

        // respond message
        ChannelResponse response = new ChannelResponse();
        response.setContent("receive request seq:" + String.valueOf(push.
↪getMessageID()));
        response.setErrorCode(0);

        push.sendResponse(response);
    }
}
```

client case:

```
package org.fisco.bcos.channel.test.amop;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.fisco.bcos.channel.client.Service;
import org.fisco.bcos.channel.dto.ChannelRequest;
import org.fisco.bcos.channel.dto.ChannelResponse;

public class Channel2Client {
    static Logger logger = LoggerFactory.getLogger(Channel2Client.class);

    public static void main(String[] args) throws Exception {
        if(args.length < 2) {
            System.out.println("param: target topic total number of request");
            return;
        }

        String topic = args[0];
        Integer count = Integer.parseInt(args[1]);
        DateTimeFormatter df = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

        ApplicationContext context = new ClassPathXmlApplicationContext(
↪"classpath:applicationContext.xml");

        Service service = context.getBean(Service.class);
        service.run();

        System.out.println("3s ...");
```

(continues on next page)

(continued from previous page)

```
        Thread.sleep(1000);
        System.out.println("2s ...");
        Thread.sleep(1000);
        System.out.println("1s ...");
        Thread.sleep(1000);

        System.out.println("start test");
        System.out.println(
↪"======================================================================");
        for (Integer i = 0; i < count; ++i) {
          Thread.sleep(2000);  // It needs to take time to establish a connection.␣
↪If you send a message immediately, it will fail.

          ChannelRequest request = new ChannelRequest();
          request.setToTopic(topic); // set the message topic
          request.setMessageID(service.newSeq());  // The message sequence number.␣
↪When you need to uniquely identify a message, you can use newSeq() to generate␣
↪randomly.
          request.setTimeout(5000);  // Message timeout

          request.setContent("request seq:" + request.getMessageID());  // The␣
↪message content is sent
          System.out.println(df.format(LocalDateTime.now()) + " request seq:" +␣
↪String.valueOf(request.getMessageID())
          + ", Content:" + request.getContent());

          ChannelResponse response = service.sendChannelMessage2(request);  //␣
↪send message

          System.out.println(df.format(LocalDateTime.now()) + "response seq:" +␣
↪String.valueOf(response.getMessageID())
                  + ", ErrorCode:" + response.getErrorCode() + ", Content:" +␣
↪response.getContent());
        }
    }
}
```

7.6.5 Test

After completing the configuration as described above, user can specify a topic: topic and execute the following two commands for testing.

Unicast text

Launch AMOP server:

```
java -cp 'conf/:apps/*:lib/*' org.fisco.bcos.channel.test.amop.Channel2Server␣
↪[topic]
```

Launch AMOP client:

```
java -cp 'conf/:apps/*:lib/*' org.fisco.bcos.channel.test.amop.Channel2Client␣
↪[topic] [Number of messages]
```

In addition to supporting unicast text, AMOP also supports sending binary data, multicast, and authentication mechanisms. The corresponding test commands are as follows:

Unicast binary，multicast text，multicast binary

Launch AMOP server:

```
java -cp 'conf/:apps/*:lib/*' org.fisco.bcos.channel.test.amop.Channel2Server␣
↪[topic]
```

Launch AMOP client:

```
#unicast binary
java -cp 'conf/:apps/*:lib/*' org.fisco.bcos.channel.test.amop.Channel2ClientBin␣
↪[topic] [filename]
#multicast text
java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.channel.test.amop.Channel2ClientMulti␣
↪[topic]  [Number of messages]
#multicast binary
java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.channel.test.amop.
↪Channel2ClientMultiBin [topic]  [filename]
```

Unicast text and binary,multicast text and binary with authentication mechanisms

Launch AMOP server:

```
java -cp 'conf/:apps/*:lib/*' org.fisco.bcos.channel.test.amop.
↪Channel2ServerNeedVerify [topic]
```

启动AMOP客户端:

```
#Unicast text with authentication mechanisms
java -cp 'conf/:apps/*:lib/*' org.fisco.bcos.channel.test.amop.
↪Channel2ClientNeedVerify [topic] [Number of messages]
#Unicast binary with authentication mechanisms
java -cp 'conf/:apps/*:lib/*' org.fisco.bcos.channel.test.amop.
↪Channel2ClientBinNeedVerify [topic] [filename]
#multicast text with authentication mechanisms
java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.channel.test.amop.
↪Channel2ClientMultiNeedVerify [topic]  [Number of messages]
#multicast binary with authentication mechanisms
java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.channel.test.amop.
↪Channel2ClientMultiBinNeedVerify [topic]  [filename]
```

7.6.6 Error code

- 99:message failed to be sent. After AMOP attempts to send message by all the links, the message is not sent to the server. It is recommended to use `seq` that is generated during the transmission to check the processing status of each node on the link.
- 100：message failed to be sent. After AMOP attempts to send message by all the links, the message is not sent to the node from one node by P2P. It is recommended to use `seq` that is generated during the transmission to check the processing status of each node on the link.
- 101：message failed to be sent. After AMOP attempts to send message by all the links, the message is not sent to the sdk from node. It is recommended to use `seq` that is generated during the transmission to check the processing status of each node on the link.
- 102: message times out. It is recommended to check whether the server has processed the message correctly and the bandwidth is sufficient.
- 103: Due to the bandwidth limitation of the node, the AMOP request from the SDK to the node was rejected.

7.7 Parallel contract

FISCO BCOS provides development structure for parallel contract. Contract developed under the structure regulation can be parallelly executed by nodes of FISCO BCOS. The advantages of parallel contract include:

- high TPS: multiple independent transaction being executed at the same time can utilize the CPU resources to the most extent and reach high TPS
- scalable: improve the performance of transaction execution with better configuration of machine to support expansion of applications

The following context will introduce how to compile, deploy and execute FISCO BCOS parallel contract.

7.7.1 Basic knowledge

Parallel exclusion

Whether two transactions can be executed in parallel depends on whether they are mutually exclusive. By exclusive, it means the two transactions have intersection in their contract storage variables collection.

Taking payment transfer as an example, it involves transactions of payment transfer between users. Use transfer(X, Y) to represent the access of user X to user Y. The exclusion is as below.

Here are detailed definitions:

- exclusive parameter： parameter that is related to "read/write" of contract storage variable in contract interface. Such as the interface of payment transfer transfer(X, Y), in which X and Y are exclusive parameters.
- exclusive object： the exclusive content extracted from exclusive parameters. Such as the payment transfer interface transfer(X, Y). In a transaction that calls the interface, the parameter is transfer(A, B), then the exclusive object is [A, B]; for another transaction that calls parameter transfer(A, C), the exclusive object is [A, C].

To judge whether 2 transactions at the same moment can be executed in parallel depends on whether there is intersection between their exclusive objects. Transaction without intersection can be executed in parallel.

7.7.2 Compile parallel contract

FISCO BCOS provides parallel contract development structure. Developers only need to adhere to its regulation and define the exclusive parameter of each contract interface so as to realize parallelly executed contract. When contract is deployed, FISCO BCOS will auto-analyze exclusive object before the transaction is excuted to make non-dependent transaction execute in parallel as much as possible.

So far, FISCO BCOS offers two types of parallel contract development structure: solidity and Precompiled contract.

Solidity development structure

Parallel solidity contract shares the same development process with regular solidity contract: make `ParallelContract` as the base class of the parallel contract and call `registerParallelFunction()` to register the interface. （ParallelContract.sol can be found at here）

Here is a complete example of how ParallelOk contract realize parallel payment transfer

```
pragma solidity ^0.4.25;

import "./ParallelContract.sol";  // import ParallelContract.sol

contract ParallelOk is ParallelContract // make ParallelContract as the base class
{
```

(continues on next page)

(continued from previous page)

```
    // contract realization
    mapping (string => uint256) _balance;

    function transfer(string from, string to, uint256 num) public
    {
        // here is a simple example, please use SafeMath instead of "+/-" in real␣
↪production
        _balance[from] -= num;
        _balance[to] += num;
    }

    function set(string name, uint256 num) public
    {
        _balance[name] = num;
    }

    function balanceOf(string name) public view returns (uint256)
    {
        return _balance[name];
    }

    // register parallel contract interface
    function enableParallel() public
    {
        // function defined character string (no blank space behind ","), the␣
↪former part of parameter constitutes exclusive parameter (which should be put␣
↪ahead when designing function)
        registerParallelFunction("transfer(string,string,uint256)", 2); //␣
↪critical: string string
        registerParallelFunction("set(string,uint256)", 1); // critical: string
    }

    // revoke parallel contract interface
    function disableParallel() public
    {
        unregisterParallelFunction("transfer(string,string,uint256)");
        unregisterParallelFunction("set(string,uint256)");
    }
}
```

The detail steps are:

（1） make `ParallelContract` as the base class of contract

```
pragma solidity ^0.4.25;

import "./ParallelContract.sol"; // import ParallelContract.sol

contract ParallelOk is ParallelContract // make ParallelContract as the base class
{
   // contract realization

   // register parallel contract interface
   function enableParallel() public;

   // revoke parallel contract interface
   function disableParallel() public;
}
```

（2） Compile parallel contract interface

Public function in contract is the interface of contract. To compile a parallel contract interface is to realize the

public function of a contract according to certain rules.

Confirm whether the interface is parallelable

A parallelable contract interface has to meet following conditions:

- no call of external contract
- no call of other function interface

Confirm exclusive parameter

Before compiling interface, please confirm the exclusive parameter of interface. The exclusion of interface is the exclusion of global variables. The confirmation of exclusive parameter has following rules:

- the interface accessed global mapping, the key of mapping is the exclusive parameter
- the interface accessed global arrays, the subscript of a array is the exclusive parameter
- the interface accessed simple type of global variables, all the simple type global variables share one exclusive parameter and use different variable names as the exclusive objects.

  For example: If `setA(int x)` writes `globalA`, we need to declare it as `setA(string aflag, int x)` and call it like `setA("globalA", 10)` by using `globalA` to declare the exclusive object.

Confirm parameter type and sequence

After the exclusive parameter is confirmed, confirm parameter type and sequence according to following rules:

- interface parameter is limited to: string、address、uint256、int256 (more types coming in the future)
- exclusive parameter should all be contained in interface parameter
- all exclusive should be put in the beginning of the interface parameter

```
mapping (string => uint256) _balance; // global mapping

// exclusive variable from, to are put at the beginning of transfer()
function transfer(string from, string to, uint256 num) public
{
    _balance[from] -= num;  // from is the key of global mapping, the exclusive␣
↪parameter
    _balance[to] += num; // to is the key of global mapping, the exclusive␣
↪parameter
}

// the exclusive variable name is put at the beginning of the parameter of set()
function set(string name, uint256 num) public
{
    _balance[name] = num;
}
```

（3）Register parallelable contract interface

Implement enableParallel() function in contract, call registerParallelFunction() to register parallelable contract interface, and implement disableParallel() function to endow the contract with ability to revoke parallel execution.

```
// register parallelable contract interface
function enableParallel() public
{
    // function defined character string (no blank space behind ","), the␣
↪parameter starts with exclusive parameters
    registerParallelFunction("transfer(string,string,uint256)", 2); // transfer␣
↪interface, the former 2 is exclusive parameter
    registerParallelFunction("set(string,uint256)", 1); // set interface, the␣
↪first 1 is exclusive parameter
}
```

(continues on next page)

(continued from previous page)

```
// revoke parallel contract interface
function disableParallel() public
{
    unregisterParallelFunction("transfer(string,string,uint256)");
    unregisterParallelFunction("set(string,uint256)");
}
```

（4）Deploy/execute parallel contract

Please refer to here for the console manual for version 2.6 and above, and here for the console manual for version 1.x. Here we use console as an example.

deploy contract

```
[group:1]> deploy ParallelOk.sol
```

call `enableParallel()` interface to make ParallelOk executed parallelly

```
[group:1]> call ParallelOk.sol 0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744␣
↪enableParallel
```

send parallel transaction `set()`

```
[group:1]> call ParallelOk.sol 0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744 set
↪"jimmyshi" 100000
```

send parallel transaction `transfer()`

```
[group:1]> call ParallelOk.sol 0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744 transfer
↪"jimmyshi" "jinny" 80000
```

check transaction execution result `balanceOf()`

```
[group:1]> call ParallelOk.sol 0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744␣
↪balanceOf "jinny"
80000
```

The following context contains an example to send massive transaction through SDK.

Precompile parallel contract structure

Parallel precompiled contract has the same compilation and development process with regular precompiled contract. Regular precompiled contract uses Precompile as the base class to implement contract logical. Based on this, Precompile base class offers 2 virtual functions for parallel to enable implementation of parallel precompiled contract.

（1）Define the contract as parallel contract

```
bool isParallelPrecompiled() override { return true; }
```

（2）Define parallel interface and exclusive parameter

It needs attention that once contract is defined parallelable, all interfaces need to be defined. If an interface is returned with null, it has no exclusive object. Exclusive parameter is related to the implementation of precompiled contract, which needs understanding of FISCO BCOS storage. You can read the codes or consult experienced programmer for implementation details.

```
// take out exclusive object from parallel interface parameter, return exclusive␣
↪object
std::vector<std::string> getParallelTag(bytesConstRef param) override
```

(continues on next page)

(continued from previous page)

```
{
    // get the func and data to be called
    uint32_t func = getParamFunc(param);
    bytesConstRef data = getParamData(param);

    std::vector<std::string> results;
    if (func == name2Selector[DAG_TRANSFER_METHOD_TRS_STR2_UINT]) // function is␣
↪parallel interface
    {
        // interfaces: userTransfer(string,string,uint256)
        // take out exclusive object from data
        std::string fromUser, toUser;
        dev::u256 amount;
        abi.abiOut(data, fromUser, toUser, amount);

        if (!invalidUserName(fromUser) && !invalidUserName(toUser) && (amount > 0))
        {
            // write results to exclusive object
            results.push_back(fromUser);
            results.push_back(toUser);
        }
    }
    else if ... // all interfaces needs to offer exclusive object, returning null␣
↪means no exclusive object

         return results;  //return exclusion
}
```

（3）Compile, restart node

To manually compile nodes please check here

After compilation, close node and replace with the original node binaries, and restart node.

7.7.3 Example: parallel payment transfer

Here gives 2 parallel examples of solidity contract and precompiled contract.

Config environment

The execution environment in this case:

- java-sdk-demo client end
- a FISCO BCOS chain

java-sdk-demo is to send parallel transaction, FISCO BCOS chain is to execute parallel transaction. The related configuration are:

- java-sdk-demo configuration
- Chain building

For pressure test on maximum performance, it at least needs:

- 3 java-sdk-demo to generate enough transactions
- 4 nodes, all java-sdk-demo are configured with all information of nodes on chain to send transaction evenly to each node so that the chain can receive enough transaction

Parallel Solidity contract: ParallelOk

Payment transfer based on account model is a typical operation. ParallelOk contract is an example of account model and is capable of parallel transfer. The ParallelOk contract is given in former context.

FISCO BCOS has built-in ParallelOk contract in java-sdk-demo. Here is the operation method to send massive parallel transactions through java-sdk-demo.

（1）Deploy contract, create new user, activate parallel contract through SDK

```
# java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.sdk.demo.perf.ParallelOkPerf␣
↪[precompiled] [groupID] [add] [count] [tps] [file]
java -cp conf/:lib/*:apps/* org.fisco.bcos.sdk.demo.perf.ParallelOkPerf␣
↪precompiled 1 add 10000 2500 user
# 在group1上创建了 10000个用户，创建操作以2500TPS发送的，生成的用户信息保存在user中
```

After executed, ParallelOk contract will be deployed to blockchain, the created user information is stored in user file, and the parallel ability of ParallelOk contract is activated.

（2）Send parallel transfer transactions in batch

Note: before send transactions in batch, please adjust the SDK log level to `ERROR` to ensure capacity.

```
# java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.sdk.demo.perf.ParallelOkPerf␣
↪[precompiled] [groupID] [transfer] [count] [tps] [file]
java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.sdk.demo.perf.ParallelOkPerf␣
↪precompiled 1 transfer 100000 4000 user
# 100000 transactions have been sent to group1, the TPS limit is 4000, users are␣
↪the same in the user file created formerly.
```

（3）Verify parallel correctness

After parallel transaction is executed, java-sdk-demo will print execution result. `TPS` is the TPS executed on node in the transaction sent by SDK. `validation` is the verification of transfer transaction result.

```
Total transactions:  100000
Total time: 34412ms
TPS: 2905.9630361501804
Avg time cost: 4027ms
Error rate: 0%
Return Error rate: 0%
Time area:
0    < time <  50ms   : 0   : 0.0%
50   < time <  100ms  : 44  : 0.044000000000000004%
100  < time <  200ms  : 2617  : 2.617%
200  < time <  400ms  : 6214  : 6.214%
400  < time <  1000ms : 14190  : 14.19%
1000 < time <  2000ms : 9224  : 9.224%
2000 < time           : 67711  : 67.711%
validation:
         user count is 10000
         verify_success count is 10000
         verify_failed count is 0
```

We can see that the TPS of this transaction is 2905. No error (`verify_failed count is 0`) after execution result is verified.

（4）Count total TPS

Single java-sdk-demo cannot send enough transactions to reach the parallel execution limit of nodes. It needs multiple java-sdk-demos to send transactions at the same time. TPS by simply summing together won't be correct enough when multiple java-sdk-demos sending transactions, so it should be acquired directly from node.

count TPS from log file using script

```
cd tools
sh get_tps.sh log/log_2019031821.00.log 21:26:24 21:26:59 # parameters: <log file>
↪<count start time> <count end time>
```

get TPS（2 SDK, 4 nodes, 8 cores, 16G memory）

```
statistic_end = 21:26:58.631195
statistic_start = 21:26:24.051715
total transactions = 193332, execute_time = 34580ms, tps = 5590 (tx/s)
```

Parallel precompiled contract: DagTransferPrecompiled

Same with the function of ParallelOk contract, FISCO BCOS has built-in example of parallel precompiled contract (DagTransferPrecompiled) and realizes transfer function based on account model. The contract can manage deposits from multiple users and provides a parallel transfer interface for parallel transactions of payment transfer between users.

Note: DagTransferPrecompiled is the example of parallel transaction with simple functions, please don't use it for online transactions.

（1）Create user

Use java-sdk-demo to send transaction to create user, the user information will be stored in user file. Command parameter is the same with parallelOk, the only difference is that the object called by the command is precompile.

```
# java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.sdk.demo.perf.ParallelOkPerf␣
↪[precompiled] [groupID] [add] [count] [tps] [file]
java -cp conf/:lib/*:apps/* org.fisco.bcos.sdk.demo.perf.ParallelOkPerf␣
↪precompiled 1 add 10000 2500 user
# 在group1上创建了 10000个用户，创建操作以2500TPS发送的，生成的用户信息保存在user中
```

（2）Send parallel transfer transactions in batch

Send parallel transfer transactions through java-sdk-demo

Note: before sending transactions in batch, please adjust SDK log level to `ERROR` for enough capability to send transactions.

```
# java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.sdk.demo.perf.ParallelOkPerf␣
↪[precompiled] [groupID] [transfer] [count] [tps] [file]
java -cp 'conf/:lib/*:apps/*' org.fisco.bcos.sdk.demo.perf.ParallelOkPerf␣
↪precompiled 1 transfer 100000 4000 user

# 100000 transactions has been sent to group1, the TPS limit is 4000, users are␣
↪the same ones in the user file created formerly, 20% exclusion exists between␣
↪transactions.
```

（3）Verify parallel correctness

After parallel transactions are executed, java-sdk-demo will print execution result. `TPS` is the TPS of the transaction sent by SDK on the node. `validation` is the verification of transfer execution result.

```
Total transactions:  80000
Total time: 25451ms
TPS: 3143.2949589407094
Avg time cost: 5203ms
Error rate: 0%
Return Error rate: 0%
Time area:
0    < time <  50ms   : 0   : 0.0%
50   < time <  100ms  : 0   : 0.0%
100  < time <  200ms  : 0   : 0.0%
200  < time <  400ms  : 0   : 0.0%
400  < time <  1000ms : 403   : 0.50375%
1000 < time <  2000ms : 5274   : 6.592499999999999%
2000 < time           : 74323   : 92.90375%
validation:
         user count is 10000
```

(continues on next page)

(continued from previous page)

```
        verify_success count is 10000
        verify_failed count is 0
```

We can see that in this transaction, the TPS is 3143. No error (`verify_failed count is 0`) after execution result verification.

（4） Count total TPS

Single java-sdk-demo can send enough transactions to meet the parallel execution limit of node. It needs multiple java-sdk-demo to send tranactions. And by simply summing the TPS of each transaction won't be correct, so the TPS should be acquired from node directly.

Count TPS from log file using script

```
cd tools
sh get_tps.sh log/log_2019031311.17.log 11:25 11:30 # parameter: <log file> <count␣
↪start time> <count end time>
```

get TPS (3 SDK, 4 nodes, 8 cores, 16G memory)

```
statistic_end = 11:29:59.587145
statistic_start = 11:25:00.642866
total transactions = 3340000, execute_time = 298945ms, tps = 11172 (tx/s)
```

Result description

The performance result in the example of this chapter is tested in 3SDK, 4 nodes, 8 cores, 16G memory, 1G network. Each SDK and node are deployed in different VPS with cloud disk. The real TPS depends on the condition of your hardware configuration, operation system and bandwidth.

7.8 Privacy protection

Privacy protection is a major technical challenge for the alliance chain. In order to protect the data, keep the anonymity of alliance members, and ensure the effectiveness of supervision, FISCO BCOS integrates homomorphic encryption and group/ring signature algorithms in Precompiled Contracts, providing multiple privacy protection methods.

Note:

1. FISCO BCOS 2.3.0+ supports homomorphic encryption, group signature and ring signature
2. FISCO BCOS 2.3.0, 2.4.0 and 2.4.1, you need to manually compile the binary to enable the privacy protection module
3. FISCO BCOS 2.5.0+ enables privacy protection module by default

---

7.8.1 Homomorphic encryption

Introduction

Homomorphic Encryption is one of the jewels in the field of public key cryptosystems. It has more than 40 years of research history, and its wonderful cryptographic characteristics have attracted wide attention。

- Homomorphic encryption is essentially a public key encryption algorithm, that is, the public key is used for encryption, and the private key is used for decryption.

- Homomorphic encryption allows computation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext.
- Formula description:

$$
\begin{aligned}
C_1 &= Encrypt(m_1, pk) \\
C_2 &= Encrypt(m_2, pk) \\
C_3 &= Hom_{f()}(C_1, C_2, pk) \\
Decrypt(C_3, sk) &= f(m_1, m_2)
\end{aligned}
$$

FISCO BCOS uses the paillier encryption algorithm which supports addition homomorphism. Paillier key pairs are compatible with mainstream RSA public key encryption algorithms, and the use costs is low. At the same time, paillier, as a lightweight homomorphic encryption algorithm, has low calculation overhead and is easily accepted by business systems. Therefore, after balancing the trade-off between functionality and usability, the paillier algorithm was finally selected.

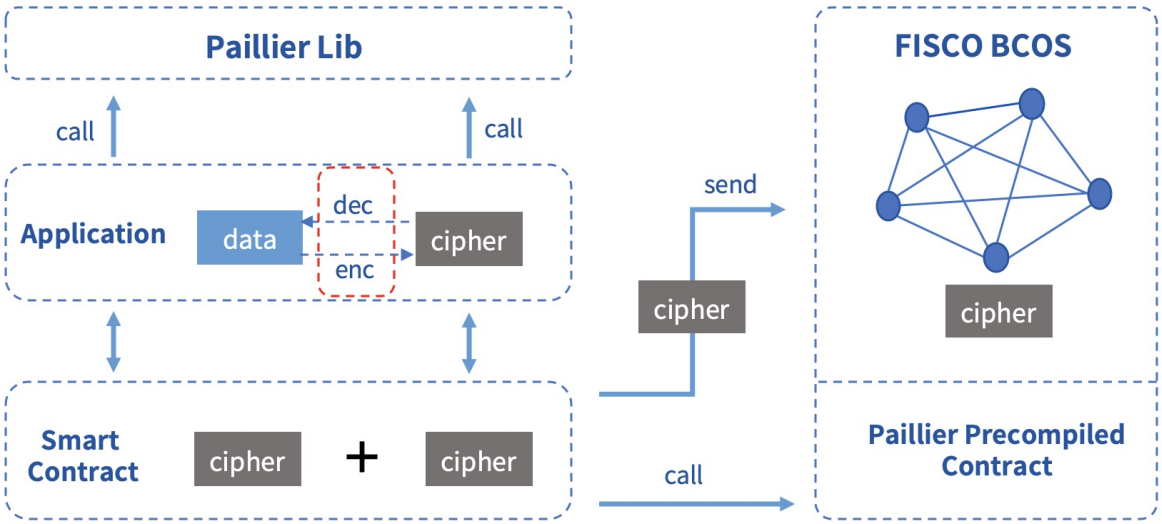
Components

Components of FISCO BCOS homomorphic encryption module include:

- paillierlibrary, provides java version library and c++ version homomorphic interface.
- paillierprecompiled contract, provides homomorphic interface for smart contracts.

Suggestion

For services that require privacy protection, if it needs simple ciphertext calculation, this module can be used to fulfill related demands. All data on the chain can be encrypted by calling the paillier library. The ciphertext data on the chain can be homomorphically added by calling the paillier precompiled contract. After the ciphertext is returned to the application layer, it can be decrypted by calling the paillier library. The specific process is shown in the following figure:



Scenario

In the alliance chain, different business scenarios need to be equipped with different privacy protection policies. For businesses with strong privacy, such as reconciliation between financial institutions, it is necessary to encrypt asset data. In FISCO BCOS, the user can call the homomorphic encryption library to encrypt the data. When the consensus node executes the transaction, the homomorphic encryption precompiled contract is called to obtain the result of the ciphertext calculation.

7.8.2 Group/Ring signature

Introduction

Group signature

Group signature is a digital signature scheme that can protect the identity of the signer. Users can sign messages instead of the group they belong to, and the verifier can verify whether the signature is valid. In this case, the verifier could not know which group member the signature belongs to. However, users cannot abuse this anonymous behavior, because the group administrator can open the signature through the group owner's private key, exposing the attribution information of the signature. Group signature features include:

- Anonymity: group members use group parameters to generate signatures, others can only verify the validity of the signature, and know the group to which the signer belongs through the signature, but cannot obtain the identity information of the signer;
- Unforgeability: only group members can generate valid and verifiable group signatures;
- Unlinkability: given two signatures, it is impossible to determine whether they are from the same signer;
- Traceability: the group owner can obtain the signer identity through signature.

Ring signature

Ring signature is a special group signature scheme, but has complete anonymity, that is, there is no administrator, users can actively join the ring, and the signature cannot be opened. Ring signature features include:

- Unforgeability: other members in the ring cannot fake the signature of the real signer;
- Full anonymity: there is no administrator, others can only verify the validity of the ring signature, but no one can obtain the identity information of the signer.

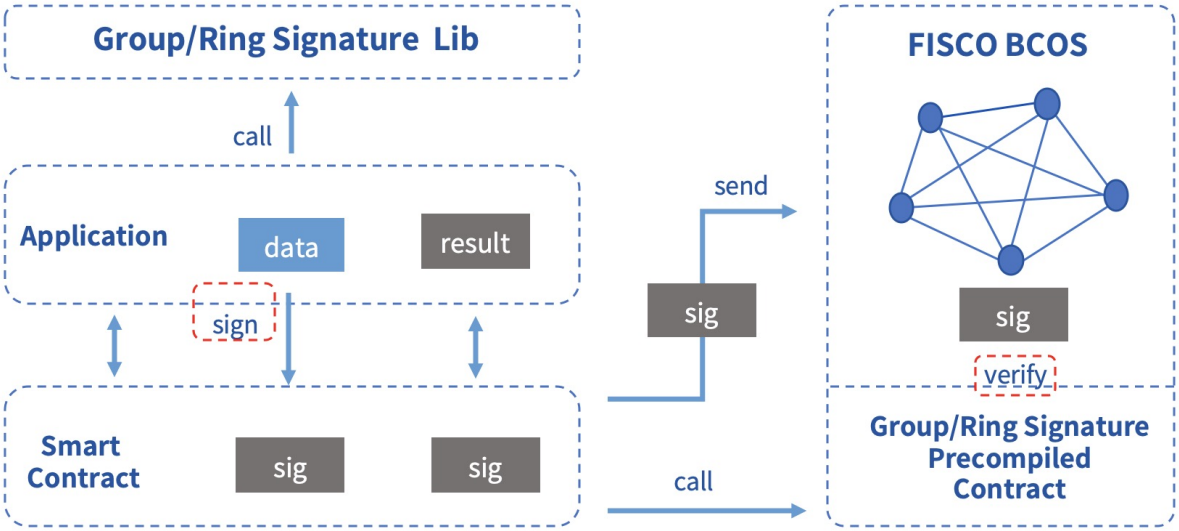
Components

Components of FISCO BCOS group/ring signature module include:

- group/ring signaturelibrary, provides c++ version library.
- group/ring signatureprecompiled contracts, provides verify interfaces for smart contracts.

Suggestion

Services that have the need to hide the identity of the signer can use this module to fulfill related demands. The user signs the data by calling the group/ring signature library, and then uploads the signature. The application contracts verifies the signature by calling the group/ring signature precompiled contracts, and returns the verification result back to the application layer. If it is a group signature, the supervisor can also open the specified signature data to obtain the identity of the signer. The specific process is shown in the following figure:



Scenario

Due to its natural anonymity, group/ring signature has broad application prospects in scenarios where participants' identities need to be hidden, such as anonymous voting, anonymous auctions, anonymous auctions, etc., and can even be used in the blockchain with UTXO model to achieve anonymous transfers. At the same time, due to the traceability of group signature, it can be used in scenarios that require regulatory intervention. The supervisor acts as the group owner or entrusts the group owner to reveal the identity of the signer.

7.8.3 How to start

Build chain

Make sure you are in the `FISCO-BCOS/build` directory, and execute the following command to build a local 4-node chain. Refer to here to get more build options.

```
bash ../manual/build_chain.sh -l 127.0.0.1:4 -e bin/fisco-bcos
```

7.8.4 Precompiled contract interface

The code of the privacy module is put together with the precompiled contracts developed by the user, located in the `FISCO-BCOS/libprecompiled/extension` directory. The calling method of the privacy module is exactly the same as the calling method of the precompiled contract developed by the user, but there are two points to note:

1. Addresses have been assigned to the precompiled contracts for the privacy module and no additional registration is required. The precompiled contract list and address allocation of the privacy module are as follows:

1. Need to declare the interfaces of the precompiled contracts through the `solidity` contract. The contract files need to be saved in the console contract directory `console/contracts/solidity`. The contract interface of each privacy function is as follows:

- Homomorphic encryption

```
// PaillierPrecompiled.sol
pragma solidity ^0.4.24;
contract PaillierPrecompiled{
```

(continues on next page)

(continued from previous page)

```
    function paillierAdd(string cipher1, string cipher2) public constant␣
↪returns(string);
}
```

- Group signature

```
// GroupSigPrecompiled.sol
pragma solidity ^0.4.24;
contract GroupSigPrecompiled{
                function groupSigVerify(string signature, string message,␣
↪string gpkInfo, string paramInfo) public constant returns(bool);
}
```

- Ring signature

```
// RingSigPrecompiled.sol
pragma solidity ^0.4.24;
contract RingSigPrecompiled{
     function ringSigVerify(string signature, string message, string␣
↪paramInfo) public constant returns(bool);
}
```

7.8.5 Called by console

After building the chain by the newly compiled binary, deploy the console (version v1.0.2 or later), and copy the interface declaration files to the console contract directory. Take calling homomorphic encryption as an example:

```
# start the console in the console directory
bash start.sh

# call contract
call PaillierPrecompiled 0x5003 paillierAdd
↪"0100E97E06A781DAAE6DBC9C094FC963D73B340D99FD934782A5D629E094D3B051FBBEA26F46BB681EB5314AE98A6A
↪"
↪"0100E97E06A781DAAE6DBC9C094FC963D73B340D99FD934782A5D629E094D3B051FBBEA26F46BB681EB5314AE98A6A
↪"

# the result
0100E97E06A781DAAE6DBC9C094FC963D73B340D99FD934782A5D629E094D3B051FBBEA26F46BB681EB5314AE98A6A638
```

Note: The inputted ciphertexts can be generated through the java library in the paillier library.

7.8.6 Called by solidity contract

Take homomorphic encryption as an example. First, create a precompiled contract object in the solidity contract and call its interface, then create a `CallPaillier.sol` file in the console`console/contracts/solidity`. The file content is as follows:

```
// CallPaillier.sol
pragma solidity ^0.4.24;
import "./PaillierPrecompiled.sol";

contract CallPaillier {
    PaillierPrecompiled paillier;
    function CallPaillier() {

        // call PaillierPrecompiled contract
        paillier = PaillierPrecompiled(0x5003);
```

(continues on next page)

(continued from previous page)

```
    }
    function add(string cipher1, string cipher2) public constant returns(string) {
        return paillier.paillierAdd(cipher1, cipher2);
    }
}
```

Deploy the `CallPaillier` contract, and then call the `add` interface. Using the above ciphertext as inputs, you will get the same result.

CHAPTER

EIGHT

FAQ (REVISION IN PROGRESS)

8.1 Version

Q: What changes have been made to FISCO BCOS version 2.0 compares to previous versions? A: Please refer to here.

Q: How do developers interact with the FISCO BCOS platform? A: FISCO BCOS provides multiple ways for developers to interact with the platform. Please refer as follows:

- FISCO BCOS 2.0 version provides JSON-RPC interface. For the detail, please refer to here.
- FISCO BCOS 2.0 version provides Java SDK to help developers quickly implement applications. For the detail, please refer to here.
- FISCO BCOS version 2.0 provides a console to help users quickly understand how to use FISCO BCOS. For the detail, Please refer to here for the console user manual of version 2.6 and above, and here for the console user manual of version 1.x

Q: How to build FISCO BCOS 2.0 version? A: FISCO BCOS supports multiple building methods. The common methods are:

- build_chain.sh: It is suitable for developer experience and testing FISCO BCOS alliance chain. For the detail, please refer to here。
- FISCO-Generator： For deploying and maintaining the FISCO BCOS Alliance Chain with enterprise users. For the detail, please refer to here。

Q: What is the difference on the smart contract between FISCO BCOS 2.0 version and the previous version? And how is the compatibility? A: FISCO BCOS version 2.0 supports the latest Solidity contract and precompile contract. For the detail, please [refer to here] (../manual/smart_contract.md).

Q: What is the difference between the national cryptographic version and the normal version? A: The national cryptography version FISCO BCOS replaces the cryptographic algorithms of underlying modules such as transaction signature verification, p2p network connection, node connection, and data disk encryption with the national cryptography algorithm. Meanwhile, in compiling version and certificate, disk encryption, and solidity compiling java, there are some difference on web3sdk using national cryptography version and normal version, please refer to [refer to here] (./manual/guomi_crypto.md).

Q: Does it support to upgrade to 2.0 version from 1.3 or 1.5? A: It does not.

8.2 Console

Q: Is the console instruction case sensitive? A: It is case-sensitive. The command will match exactly, and `tab` can be used to complete your command.

Q: When adding to the Sealer list or the Observer list, it will report error as nodeID is not in network, why? A: The nodes that adds to the Sealer list and the Observer list must be a member of the nodeID list that connects to the peer.

Q: To delete node will report error as nodeID is not in group peers, why? A: The node to be delete must be the peer of the group displayed in getGroupPeers

Q: Can the RemoveNodes (non-group nodes) synchronize group data? A: RemoveNodes does not participate in the consensus, synchronization, and block generation within the group. RemoveNodes can add the exit node as Sealer/Observer through the command of console `addSealer/addObserver`.

Q: If the node belongs to a different group, can it support querying information of multiple groups? A: Yes, when you enter the console, you can input the groupID you want to view: ./start [groupID]

8.3 FISCO BCOS using

Q: Where to get Ver 2.0 certificates? A: Please read Certificates Decsription

Q: What fields are contained in the transaction structure of Ver 2.0? A: Please read here Q: What are the system configuration, group configuration, and node configuration? A: System configuration refers to some configuration items that affect the ledger function and require the consensus of the ledger node in the node configuration. Group configuration refers to the configuration of the group which the node belongs to. Each group of nodes has an independent configuration. Node configuration refers to all configurable items.

Q: Can the group configuration be changed? A: Whether the configuration item could be changed can be measured by:

- The node that is first time to launch and has generated a genesis block can not be modified. This type of configuration is placed in the group.x.genesis file, where x is the group number, and it is unique in the entire chain.
- To implement consistence in ledger by sending the transaction modification configuration item.
- After the configuration file is modified, the node can be restarted to takes effect. This type of configuration is placed in the `group.x.ini` file. After the group configuration is changed, the restart can be changed locally, the changeable item becomes the local configuration. The `group.*.ini` file under nodeX/conf is changed and restarted to takes effect. The involved configuration items are [tx_pool].limit (transaction pool capacity) and [consensus].ttl (node forwarding number).

Q: Which configurations can the group configuration user change? A: The group can be modified and configured into consensus changeable configuration and manual changeable configuration.

- consensus changeable configuration: all nodes in the group are the same, and takes effect after consensus. [consensus].max_trans_num,[consensus].node.X,[tx].gas_limit.
- manual changeable configuration: it is in the `group.x.ini` file and restarted to take effect after modification. It only affects node. The configuration item has [tx_pool].limit.

Q: How to change and inquire the consensus changeable configuration? A: Consensus changeable configuration can be changed through console. It can be inquired through console and RPC interface. For detail, please [refer to here] (../design/rpc.md).

- [consensus].max_trans_num,[tx].gas_limit is changed by using the interface setSystemConfigByKey, and the corresponding configuration items are tx_count_limit, tx_gas_limit. See setSystemConfigByKey -h for details.
- [consensus].node.X's change refers to node management. The console interface refer to addSealer, addObserver, removeNode. For detail, please refer to Node Management.

Q: What is the difference between Observer node and Sealer node in group? A: Observer node can synchronize the group data, but cannot participate in consensus. Consensus nodes have the Observer permission and participate in consensus.

Q: How to incorporate contract into CNS management? A: When contract is deployed, to call the CNS contract interface, and to compile the information of contract name, version, and address information into the CNS list.

Q: How to query the contract CNS list? A: To query CNS list through the command of web3sdk console, and the query command is queried according to the contract name.

Q: Why can't local SDK connect to FISCO BCOS nodes on cloud servers? A:

1. Check the node configuration on the cloud server to see if Channel is listening for IP over the extranet, rather than `127.0.0.1`. Port Description Refer here
2. Check the console provided by the cloud manufacturer of the cloud server, check whether the security group is configured, and open the channel port used by FISCO BCOS nodes in the security group.
3. Check that the generated certificate is correct, refer to here

Q: Why connection to other peer nodes can not be established after launching the node, and the log report there are network exceptions? A:

1. Please check that certificate files of the node are correct
2. Please check that type (SM_CRYPTO or not) of the node is consistent with other peer nodes

Q: Why "invalid group status" error information appears in the log of node?

A: Due to possible fault of file system, group status recorded by node on local disk may be invalid. Users can check `.group_status` file in group's data folder, and change the content to the one of following items:

- STOPPED
- DELETED
- RUNNING

8.4 Java SDK

Q: What does Web3SDK require to Java version? A: It requires JDK8 version or above

The OpenJDK of yum repository of CentOS lacks JCE (Java Cryptography Extension), which causes Java SDK to fail to connect to blockchain node. When using the CentOS operation system, it is recommended to download it from the OpenJDK website. [Installation Guide] (https://openjdk.java.net/install/index.html)

Q: After the Java SDK configuration is completed, what is the reason for the failed transaction? A: The ip, port, group number in applicationContext.xml are incorrectly filled or the node files of ca.crt, sdk.crt, and sdk.key files are missing.

8.5 Enterprise deployment tool

Q: There is pip cannot be found appears when using enterprise deployment tools. A: The enterprise deployment tool relies on python pip. To install it with the following command:

```
$ bash ./scripts/install.sh
```

Q: When using enterprise deployment tools, the following information appears:

```
Traceback (most recent call last):
   File "./generator", line 19, in <module>
    from pys.build import config
   File "/data/asherli/generator/pys/build/config.py", line 25, in <module>
     import configparse
```

A: The python configparser module is missing from the system. Please follow the command below to install:

```
  $ pip install configparser
```

Q: downloading occurred certificate verify failed 答: vim ./pys/tool/utils.py, add this code in first line.

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

CHAPTER

NINE

CHAIN BUILDING SCRIPT

Important: The goal of the script is to let users apply FISCO BCOS as quickly as possible. For the enterprise applications deploying FISCO BCOS, please refer to Enterprise Deployment Tools .

FISCO BCOS has provided `build_chain` script to help users quickly build FISCO BCOS alliance chain. By default, the script downloads `master` branch of the latest version pre-compiles executable program from GitHubfor building related environment.

9.1 Script introduction

- `build_chain.sh` is used to quickly generate configuration files of a chain node. For the script that depends on `openssl`, please according your own operating system to install `openssl 1.0.2` version and above. The source code of script is located at here.
- For quick experience can use the `-l` option to specify the node IP and number. `-f` option supports the creation of FISCO BCOS chains for complex business scenarios by using a configuration file in a specified format. `-l` and `-f` options must be specified uniquely and cannot coexist.
- It is recommended to use `-T` option for testing. `-T` enables log level to DEBUG, p2p module listens for `0.0.0.0` by default.

Note: In order to facilitate development and experience, the default listening IP of the P2P module is 0.0.0.0. For security reasons, please modify it to a safe listening address according to the actual business network situation, such as the internal IP or a specific external IP.

9.2 Help

```
Usage:
    -l <IP list>                        [Required] "ip1:nodeNum1,ip2:nodeNum2" e.g:
↪"192.168.0.1:2,192.168.0.2:3"
    -f <IP list file>                   [Optional] split by line, every line␣
↪should be "ip:nodeNum agencyName groupList p2p_port,channel_port,jsonrpc_port".␣
↪eg "127.0.0.1:4 agency1 1,2 30300,20200,8545"
    -v <FISCO-BCOS binary version>      Default is the latest v${default_version}
    -e <FISCO-BCOS binary path>         Default download fisco-bcos from GitHub.␣
↪If set -e, use the binary at the specified location
    -o <Output Dir>                     Default ./nodes/
    -p <Start Port>                     Default 30300,20200,8545 means p2p_port␣
↪start from 30300, channel_port from 20200, jsonrpc_port from 8545
    -q <List FISCO-BCOS releases>       List FISCO-BCOS released versions
    -i <Host ip>                        Default 127.0.0.1. If set -i, listen 0.0.0.
↪0
```

(continues on next page)

(continued from previous page)

```
    -s <DB type>                        Default RocksDB. Options can be RocksDB /␣
↪mysql / Scalable, RocksDB is recommended
    -d <docker mode>                    Default off. If set -d, build with docker
    -c <Consensus Algorithm>            Default PBFT. Options can be pbft / raft /
↪rpbft, pbft is recommended
    -C <Chain id>                       Default 1. Can set uint.
    -g <Generate guomi nodes>           Default no
    -z <Generate tar packet>            Default no
    -t <Cert config file>               Default auto generate
    -6 <Use ipv6>                       Default no. If set -6, treat IP as IPv6
    -k <The path of ca root>            Default auto generate, the ca.crt and ca.
↪key must in the path, if use intermediate the root.crt must in the path
    -K <The path of sm crypto ca root>  Default auto generate, the gmca.crt and␣
↪gmca.key must in the path, if use intermediate the gmroot.crt must in the path
    -D <Use Deployment mode>            Default false, If set -D, use deploy mode␣
↪directory struct and make tar
    -G <channel use sm crypto ssl>      Default false, only works for guomi mode
    -X <Certificate expiration time>    Default 36500 days
    -T <Enable debug log>               Default off. If set -T, enable debug log
    -R <Channel use ecdsa crypto ssl>   Default false. If -R is set, use the ecdsa␣
↪cert for channel ssl, Otherwise the rsa cert will be used
    -S <Enable statistics>              Default off. If set -S, enable statistics
    -F <Disable log auto flush>         Default on. If set -F, disable log auto␣
↪flush
    -E <Enable free_storage_evm>        Default off. If set -E, enable free_
↪storage_evm
    -h Help
e.g
    ./manual/build_chain.sh -l 127.0.0.1:4
```

9.3 Option introduction

9.3.1 `l`option:

Use to specify the chain to be generated and the number of nodes under each IP, separated by commas. The script generates configuration file of corresponding node according to the input parameters. The port number of each node is incremented from 30300 by default. All nodes belong to the same organization and Group.

9.3.2 `f`option

```
+ Use to generate node according to configuration file. It supports more␣
↪customization than `l` option.
+ Split by row. Each row represents a server, in the format of `IP:NUM AgencyName␣
↪GroupList`. Items in each line are separated by spaces, and there must be **no␣
↪blank lines**.
+ `IP:NUM` represents the IP address of the machine and the number of nodes on the␣
↪machine.`AgencyName`represents the name of the institution to specifies the␣
↪institution certificate to use. `GroupList` represents the group that the␣
↪generated node belong to, split by`,`. For example, `192.168.0.1:2 agency1 1,2`␣
↪represents that a machine with `ip` which is `192.168.0.1` exists two nodes. For␣
↪example, 192.168.0.1:2 agency1 1,2 represents that there are two nodes on the␣
↪machine with ip 192.168.0.1. These two nodes belong to agency `agency1` and␣
↪belong to group1 and group2.
```

The following is an example of a configuration file. Each configuration item separated by a space, where `GroupList` represents the group that the server belongs to.

```
192.168.0.1:2 agency1 1,2
192.168.0.1:2 agency1 1,3
192.168.0.2:3 agency2 1
192.168.0.3:5 agency3 2,3
192.168.0.4:2 agency2 3
```

Suppose the above file is named `ipconf`, using the following command to build a chain, which indicates to use configuration file, to set the log level to `DEBUG`.

```
$ bash build_chain.sh -f ipconf -T
```

9.3.3 eoption[Optional]

is used to specify full path where `fisco-bcos` binary is located.Script will cope `fisco-bcos` to the directory named by IP number. If no path to be specified, the latest binary program of `master` branch is downloaded from GitHub by default.

```
# download the latest release binary from GitHub to generate native 4 nodes
$ bash build_chain.sh -l 127.0.0.1:4
# use bin/fisco-bcos binary to generate native 4 nodes
$ bash build_chain.sh -l 127.0.0.1:4 -e bin/fisco-bcos
```

9.3.4 ooption[Optional]

specifies the directory where the generated configuration is located.

9.3.5 poption[Optional]

specifies the starting port of the node. Each node occupies three ports which are p2p, channel, and jsonrpc, respectively. The ports are split by, and three ports must be specified. The ports used by different nodes under the same IP address are incremented from the starting port.

```
# Two nodes occupies `30300,20200,8545` and `30301,20201,8546` respectively.
$ bash build_chain -l 127.0.0.1:2 -p 30300,20200,8545
```

9.3.6 q选项[Optional]

List FISCO BCOS released version numbers.

9.3.7 voption[Optional]

Used to specify the binary version used when building FISCO BCOS. build_chain downloads the latest version of [Release Page] (https://github.com/FISCO-BCOS/FISCO-BCOS/releases) by default. When setting this option, the download parameter specifies the `version` version and sets `[compatibility].supported_version=${version}` in the configuration file `config.ini`. If you specify the binary with the `-e` option, to use the binary and configure `[compatibility].supported_version=${version}` as the latest version number of Release page.

9.3.8 doption[Optional]

Use the docker mode to build FISCO BCOS. When using this option, the binary is no longer extracted, but users are required to start the node machine to install docker, and their accounts have docker permission, which means their accounts should in the docker group. Use following command to start node at node home.

```
$ ./start.sh
```

The command to start the node in script start.sh is as follows

```
$ docker run -d --rm --name ${nodePath} -v ${nodePath}:/data --network=host -w=/
↪data fiscoorg/fiscobcos:latest -c config.ini
```

9.3.9 soption[Optional]

There are parameter options. The parameter is the name of db. Currently it supports three modes: RocksDB, mysql and Scalable. RocksDB is used by default.

- RocksDB use RocksDB as backend database.
- mysql needs to configure the information relates to mysql in the group ini file.
- Scalable mode, block data and state data are stored in different RocksDB databases, and block data is stored in RocksDB instance named after block height. The RocksDB instance used to store block data is scroll according to the configuration `scroll_threshold_multiple`*1000 and block height. If chain data need to be tailored, the Scalable mode must be used.

9.3.10 coption[Optional]

There are parameter options. The parameter is the consensus algorithm type, and currently supports PBFT, Raft, rPBFT. The default consensus algorithm is PBFT.

- `PBFT`: Set the node consensus algorithm to PBFT.
- `Raft`: Set the node consensus algorithm to Raft.
- `rPBFT`: Set the node consensus algorithm to rPBFT.

9.3.11 Coption[Optional]

Used to specify the chain identifier when building FISCO BCOS. When this option is set, using parameter to set `[chain].id` in the configuration file `config.ini`. The parameter range is a positive integer and the default setting is 1.

```
# The chain is identified as 2
$ bash build_chain.sh -l 127.0.0.1:2 -C 2
```

9.3.12 goption[Optional]

No parameter option. When setting this option, to build the national cryptography version of FISCO BCOS.

9.3.13 zoption[Optional]

No parameter option. When setting this option, the tar package of node is generated.

9.3.14 toption[Optional]

This option is used to specify the certificate configuration file when certificate is generated.

```
[ca]
default_ca=default_ca
[default_ca]
default_days = 365
default_md = sha256

[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
[req_distinguished_name]
countryName = CN
countryName_default = CN
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default =GuangDong
localityName = Locality Name (eg, city)
localityName_default = ShenZhen
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = fisco-bcos
commonName =  Organizational  commonName (eg, fisco-bcos)
commonName_default = fisco-bcos
commonName_max = 64

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment

[ v4_req ]
basicConstraints = CA:TRUE
```

9.3.15 6选项[Optional]

Use IPv6 mode, listen ::

9.3.16 Toption[Optional]

No parameter option. When setting this option, set the log level of node to DEBUG. The related configuration of log reference here.

9.3.17 koption[Optional]

Use the private key specified by the user and the certificate issued the agency and node certification. The parameter is the path of ca.crt/ca.key. If the specified private key and certificate are intermediate Ca, root.crt should also be included in this folder to store the upper certificate chain.

9.3.18 Koption[Optional]

Use the private key specified by the user and the certificate issued the agency and node certification in guomi mode. The parameter is the path of gmca.crt/gmca.key. If the specified private key and certificate are intermediate Ca, gmroot.crt should also be included in this folder to store the upper certificate chain.

9.3.19 G选项[Optional]

From 2.5.0, when use smcrypto mode, user can config to use GM SSL between node and sdk, the option set `chain.sm_crypto_channel=true`.

9.3.20 Doption[Optional]

No parameter option. When this option is set, the directory name of the generated node is IP_P2P-port.

9.3.21 Eoption[Optional]

No parameter option, when setting this option, [Free Storage] (../design/virtual_machine/gas.html#evm-gas) Gas mode is enabled, and `Free Storage` Gas mode is disabled by default.

9.4 Node file organization

- cert folder stores root certificate and organization certificate of the chain.
- The folder named by IP address stores the certificate configuration file required by related configuration of all nodes , `fisco-bcos` executable program, and SDK in the server.
- The `node*` folder under each IP folder stores configuration file required by the node. `config.ini` is the main configuration of node. In `conf` directory, to store certificate files and group related configurations. For the configuration detail, please refer to here. Each node provides two scripts which are used to start and stop the node.
- Under each IP folder, two scripts providing `start_all.sh` and `stop_all.sh` are used to start and stop all nodes.

```
nodes/
├── 127.0.0.1
│   ├── fisco-bcos # binary program
│   ├── node0 # node0 folder
│   │   ├── conf # configuration folder
│   │   │   ├── ca.crt # chain root certificate
│   │   │   ├── group.1.genesis # the initialized configuration of group1, the␣
↪file cannot be changed
│   │   │   ├── group.1.ini # the configuration file of group1
│   │   │   ├── node.crt # node certificate
│   │   │   ├── node.key # node private key
│   │   │   └── node.nodeid # node id, represented by hexadecimal of public key
│   │   ├── config.ini # node main configuration file, to configure listening IP,␣
↪port, etc.
│   │   ├── start.sh # start script, uses for starting node
│   │   └── stop.sh # stop script, uses for stopping node
│   ├── node1 # node1 folder
│   .....
│   ├── node2 # node2 folder
│   .....
│   ├── node3 # node3 folder
│   .....
│   ├── sdk # SDK needs to be used
│   │   ├── ca.crt # chain root certificate
│   │   ├── sdk.crt # The certificate file required by SKD, to use when␣
↪establishing a connection
│   │   └── sdk.key # The private key file required by SKD, to use when␣
↪establishing a connection
|   |   ├── gm # SDK sm ssl connection with nodes configuration, note: this␣
↪directory is only generated when sm blockchain environment is generated for the␣
↪node to make SSL connection with the SDK
|   |   |   ├── gmca.crt # sm ssl connection root certificate
|   |   |   ├── gmensdk.crt # sm ssl connection encrypt certificate
|   |   |   ├── gmensdk.key # sm ssl connection encrypt certificate key
|   |   |   ├── gmsdk.crt # sm ssl connection sign certificate
```

(continues on next page)

(continued from previous page)

```
|   |   |   └── gmsdk.key # sm ssl connection sign certificate key
├── cert # certificate folder
|   ├── agency # agency certificate folder
|   |   ├── agency.crt # agency certificate
|   |   ├── agency.key # agency private key
|   |   ├── agency.srl
|   |   ├── ca-agency.crt
|   |   ├── ca.crt
|   |   └── cert.cnf
|   ├── ca.crt # chain certificate
|   ├── ca.key # chain private key
|   ├── ca.srl
|   └── cert.cnf
```

9.5 Scripts generated by build_chain

9.5.1 start_all.sh

start all nodes in current directory

9.5.2 stop_all.sh

stop all nodes in current directory

9.5.3 download_console.sh

download console

- `v` specific version of console
- `f` automatically configure console

9.5.4 download_bin.sh

download fisco-bcos precompiled binary

```
Usage:
    -v <Version>            Download binary of spectfic version, default latest
    -b <Branch>             Download binary of spectfic branch
    -o <Output Dir>         Default ./bin
    -l                      List FISCO-BCOS released versions
    -m                      Download mini binary, only works with -b option
    -h Help
e.g
    ./download_bin.sh -v 2.7.1
```

9.6 Example

9.6.1 Four nodes of group 1 on a local server

To build a 4-node FISCO BCOS alliance chain on native machine for using the default start port `30300,20200,8545` (4 nodes will occupy `30300-30303,20200-20203,8545-8548`) and listening to the external network `Channel` and `jsonrpc` ports while allowing the external network interacts with node through SDK or API.

```
# to build FISCO BCOS alliance chain
$ bash build_chain.sh -l 127.0.0.1:4
# after generating successes, to output `All completed` to mention
Generating CA key...
================================================================
Generating keys ...
Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1
================================================================
Generating configurations...
Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1
================================================================
[INFO] FISCO-BCOS Path   : bin/fisco-bcos
[INFO] Start Port        : 30300 20200 8545
[INFO] Server IP         : 127.0.0.1:4
[INFO] State Type        : storage
[INFO] RPC listen IP     : 127.0.0.1
[INFO] Output Dir        : /Users/fisco/WorkSpace/FISCO-BCOS/tools/nodes
[INFO] CA Key Path       : /Users/fisco/WorkSpace/FISCO-BCOS/tools/nodes/cert/ca.
↪key
================================================================
[INFO] All completed. Files in /Users/fisco/WorkSpace/FISCO-BCOS/tools/nodes
```

9.6.2 Add new node into Groups

This section takes Group1 generated in the previous section as an example to add a consensus node.

Generate private key certificates for new node

The next operation is done under the `nodes/127.0.0.1` directory generated in the previous section.

1. Acquisition certificate generation script

```
curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/FISCO-BCOS/master-2.0/tools/
↪gen_node_cert.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/gen_node_cert.sh

1. Generating new node private key certificates

```
# -c specify the path where the certificate and private key are located
# -o Output to the specified folder, where new certificates and private keys␣
↪issued by agency agency1 exist in newNode/conf

bash gen_node_cert.sh -c ../cert/agency -o newNode
```

If you use guomi version of fisco, please execute below command to generate cert.

```
bash gen_node_cert.sh -c ../cert/agency -o newNodeGm -g ../gmcert/agency/
```

Preparing configuration files

1. Copy Node 0 Profile and Tool Script in Group 1

```
cp node0/config.ini newNode/config.ini
cp node0/conf/group.1.genesis newNode/conf/group.1.genesis
cp node0/conf/group.1.ini newNode/conf/group.1.ini
cp node0/*.sh newNode/
cp -r node0/scripts newNode/
```

2. Update IP and ports monitored in `newNode/config.ini`, include IP and Port in `[rpc]` and `[p2p]`。
3. Add IP and Port in the new node's P2P configuration to the [p2p] field in the original node's config.ini. Assuming that the new node IP: Port is 127.0.0.1:30304, the modified [P2P] configuration is

```
[p2p]
    listen_ip=0.0.0.0
    listen_port=30304
    ;enable_compress=true
    ; nodes to connect
    node.0=127.0.0.1:30300
    node.1=127.0.0.1:30301
    node.2=127.0.0.1:30302
    node.3=127.0.0.1:30303
    node.4=127.0.0.1:30304
```

4. Start node, use `newNode/start.sh`
5. Add new nodes to group 1 through console, refer to here and here

Start a new node, check links and consensus

9.6.3 generate new SDK certificate of agency

The next operation is done under the `nodes/127.0.0.1` directory generated in the previous section.

1. Acquisition certificate generation script

```
curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/FISCO-BCOS/master-2.0/tools/
↪gen_node_cert.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/gen_node_cert.sh

---

1. Generating new node private key certificates

```
# -c specify the path where the certificate and private key are located
# -o Output to the specified folder, where new certificates and private keys␣
↪issued by agency exist in newSDK

bash gen_node_cert.sh -c ../cert/agency -o newSDK -s
```

If you use guomi version of fisco, please execute below command to generate cert.

```
bash gen_node_cert.sh -c ../cert/agency -o newSDK -g ../gmcert/agency/ -s
```

9.6.4 Generating new agency private key certificates

1. Acquisition agency certificate generation script

```
curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/FISCO-BCOS/master-2.0/tools/
↪gen_agency_cert.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/gen_agency_cert.sh

1. Generating new agency private key certificates

```
# -c path must have ca.crt and ca.key,  if use intermediate ca, then root.crt is␣
↪needed
# -g path must have gmca.crt and gmca.key,  if use intermediate ca, then gmroot.crt␣
↪is needed
# -a newAgencyName
bash gen_agency_cert.sh -c nodes/cert/ -a newAgencyName
```

国密版本请执行下面的指令。

```
bash gen_agency_cert.sh -c nodes/cert/ -a newAgencyName -g nodes/gmcert/
```

9.6.5 Multi-server and multi-group

Using the build_chain script to build a multi-server and multi-group FISCO BCOS alliance chain requires the script configuration file. For details, please refer to here.

CHAPTER

TEN

CONSOLE

10.1 Console

Important:

- `Console 1.x` series is based on Web3SDK implementation, after `console 2.6+` is based on Java SDK implementation, this tutorial is aimed at 2.6 and above version console, for 1.x version console usage documentation please refer to here
- You can view the current console version through the command `./start.sh --version`

console is an important interactive client tool of FISCO BCOS 2.0. It establishes a connection with blockchain node through Java SDK to request read and write access for blockchain node data. Console has a wealth of commands, including blockchain status inquiry, blockchain nodes management, contracts deployment and calling. In addition, console provides a contract compilation tool that allows users to easily and quickly compile Solidity contract files into Java contract files.

10.1.1 Console command

Console command consists of two parts, the instructions and the parameters related to the instruction:

- Instruction: instruction is an executed operation command, including blockchain status inquiry and contracts deployment and calling. And some of the instructions call the JSON-RPC interface, so they have same name as the JSON-RPC interface. Use suggestions: instructions can be completed using the tab key, and support for displaying historical input commands by pressing the up and down keys.
- Parameters related to the instruction: parameters required by instruction call interface. Instructions to parameters and parameters to parameters are separated by spaces. The parameters name same as JSON-RPC interface and the explanation of getting information field can be referred to JSON-RPC API.

10.1.2 Common command link:

Contract related commands

- useCNS to deploy and call contract (recommend)
  - deploy contract: deployByCNS
  - call contract: callByCNS
  - query CNS deployment contract information: queryCNS
- deploy and call contract normally
  - deploy contract: deploy
  - call contract: call

Other commands

- query block number:getBlockNumber
- query Sealer list:getSealerList
- query the information of transaction receipt: getTransactionReceipt
- switch group: switch

10.1.3 Shortcut key

- `Ctrl+A`: move cursor to the beginning of line
- `Ctrl+D`: exit console
- `Ctrl+E`: move cursor to the end of line
- `Ctrl+R`: search for the history commands have been entered
- ↑: browse history commands forward
- ↓: browse history commands backward

10.1.4 Console response

When a console command is launched, the console will obtain the result of the command execution and displays the result at the terminal. The execution result is divided into two categories:

- True: The command returns to the true execution result as a string or json.
- False: The command returns to the false execution result as a string or json.
  - When console command call the JSON-RPC interface, error code reference here.
  - When console command call the Precompiled Service interface, error code reference here.

10.1.5 Console configuration and operation

Important: Precondition：to build FISCO BCOS blockchain, please refer to Building Chain Script or Enterprise Tools.

Get console

```
$ cd ~ && mkdir fisco && cd fisco
# get console
$ curl -#LO https://github.com/FISCO-BCOS/console/releases/download/v2.9.2/
↪download_console.sh && bash download_console.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/download_console.sh && bash download_console.sh

---

The directory structure is as follows:

```
|-- apps # console jar package directory
|   -- console.jar
|-- lib # related dependent jar package directory
|-- conf
|   ├── config-example.toml # configuration file
|   ├── group-generate-config.toml # group generation configuration file, please␣
↪refer to command genrateGroupFromFile for details
|   └── log4j.properties # log configuration file
|-- contracts # directory where contract locates
|   -- solidity  # directory where solidity contract locates
|       -- HelloWorld.sol # normal contract: HelloWorld contract, is deployable␣
↪and callable
|       -- TableTest.sol # the contracts by using CRUD interface: TableTest␣
↪contract, is deployable and callable
|       -- Table.sol # CRUD interfac contract
|   -- console  # The file directory of contract abi, bin, java compiled when␣
↪console deploys the contract
|   -- sdk      # The file directory of contract abi, bin, java compiled by␣
↪sol2java.sh script
|-- start.sh # console start script
|-- get_account.sh # account generate script
|-- sol2java.sh # development tool script for compiling solidity contract file as␣
↪java contract file
|-- replace_solc_jar.sh # a script for replacing the compiling jar package
```

Configure console

- Blockchain node and certificate configuration:
  - To copy the `ca.crt`, `sdk.crt`, and `sdk.key` files in the sdk node directory to the `conf` directory.
  - To rename the `config-example.toml` file in the `conf` directory to the `config.toml` file. To configure the `config.toml` file, where the remark content is modified according to the blockchain node configuration. **Hint: If the channel_listen_ip(If the node version is earlier than v2.3.0, check the configuration item listen_ip) set through chain building is 127.0.0.1 or 0.0.0.0 and the channel_port is 20200, the `config.toml` configuration is not modified. **

```
[cryptoMaterial]

certPath = "conf"                           # The certification path

# The following configurations take the certPath by default if commented
# caCert = "conf/ca.crt"                    # CA cert file path
                                            # If connect to the GM node, default␣
↪CA cert path is ${certPath}/gm/gmca.crt

# sslCert = "conf/sdk.crt"                  # SSL cert file path
                                            # If connect to the GM node, the␣
↪default SDK cert path is ${certPath}/gm/gmsdk.crt

# sslKey = "conf/sdk.key"                   # SSL key file path
                                            # If connect to the GM node, the␣
↪default SDK privateKey path is ${certPath}/gm/gmsdk.key

# enSslCert = "conf/gm/gmensdk.crt"         # GM encryption cert file path
                                            # default load the GM SSL encryption␣
↪cert from ${certPath}/gm/gmensdk.crt

# enSslKey = "conf/gm/gmensdk.key"          # GM ssl cert file path
                                            # default load the GM SSL encryption␣
↪privateKey from ${certPath}/gm/gmensdk.key
```

(continues on next page)

(continued from previous page)

```
[network]
peers=["127.0.0.1:20200", "127.0.0.1:20201"]    # The peer list to connect

# Configure a private topic as a topic message sender.
# [[amop]]
# topicName = "PrivateTopic1"
# publicKeys = [ "conf/amop/consumer_public_key_1.pem" ]    # Public keys of the␣
↪nodes that you want to send AMOP message of this topic to.

# Configure a private topic as a topic subscriber.
# [[amop]]
# topicName = "PrivateTopic2"
# privateKey = "conf/amop/consumer_private_key.p12"         # Your private key␣
↪that used to subscriber verification.
# password = "123456"

[account]
keyStoreDir = "account"         # The directory to load/store the account file,␣
↪default is "account"
# accountFilePath = ""          # The account file path (default load from the␣
↪path specified by the keyStoreDir)
accountFileFormat = "pem"       # The storage format of account file (Default is
↪"pem", "p12" as an option)

# accountAddress = ""           # The transactions sending account address
                                # Default is a randomly generated account
                                # The randomly generated account is stored in the␣
↪path specified by the keyStoreDir

# password = ""                 # The password used to load the account file

[threadPool]
# channelProcessorThreadSize = "16"         # The size of the thread pool to␣
↪process channel callback
                                            # Default is the number of cpu cores

# receiptProcessorThreadSize = "16"         # The size of the thread pool to␣
↪process transaction receipt notification
                                            # Default is the number of cpu cores

maxBlockingQueueSize = "102400"             # The max blocking queue size of the␣
↪thread pool
```

Configuration detail reference here.

Important: Console configuration instructions

When the console configuration file configures multiple node connections in a group, some nodes in the group may leave the group during operation. Therefore, it shows a norm which is when the console is polling, the return information may be inconsistent. It is recommended to configure a node or ensure that the configured nodes are always in the group when using the console, so that the inquired information in the group will keep consistent during the synchronization time.

Contract compilation tool

Console provides a special compilation contract tool that allows developers to compile Solidity contract files into Java contract files. Two steps for using the tool:

- To place the Solidity contract file in the `contracts/solidity` directory.
- Complete the task of compiling contract by running the `sol2java.sh` script (requires specifying a java package name). For example, there are `HelloWorld.sol`, `TableTest.sol`, and `Table.sol` contracts in the `contracts/solidity` directory, and we specify the package name as `org.com.fisco`. The command is as follows:

```
$ cd ~/fisco/console
$ ./sol2java.sh org.com.fisco
```

After running successfully, the directories of Java, ABI and bin will be generated in the `console/contracts/sdk` directory as shown below.

```
```bash
|-- abi # to compile the generated abi directory and to store the abi file␣
↪compiled by solidity contract
|   |-- HelloWorld.abi
|   |-- Table.abi
|   |-- TableTest.abi
|-- bin # to compile the generated bin directory and to store the bin file␣
↪compiled by solidity contract
|   |-- HelloWorld.bin
|   |-- Table.bin
|   |-- TableTest.bin
|-- java  # to store compiled package path and Java contract file
|   |-- org
|       |-- com
|           |-- fisco
|               |-- HelloWorld.java # the target Java file which is compiled␣
↪successfully
|               |-- Table.java  # the CRUD interface Java file which is compiled␣
↪successfully
|               |-- TableTest.java  # the TableTest Java file which is compiled␣
↪successfully
```
```

In the java directory, `org/com/fisco/` package path directory is generated. In the package path directory, the java contract files `HelloWorld.java`, `TableTest.java` and `Table.java` will be generated. `HelloWorld.java` and `TableTest.java` are the java contract files required by the java application.

Launch console

Start the console while the node is running:

```
$ ./start.sh
# To output the following information to indicate successful launch
================================================================================================
Welcome to FISCO BCOS console(2.0.0)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
 ________ _______  _______   _______  _______       ________   _______   _______   _______
|        |       \/       \ /       \ /       \     |        \ /       \ /       \ /       ␣
↪\
| $$$$$$$$\$$$$$|  $$$$$$|  $$$$$$|  $$$$$$\    | $$$$$$$|  $$$$$$|  $$$$$$|  $$$$$
↪$\
| $$__      | $$ | $$___\$| $$   \$| $$  | $$    | $$__/ $| $$   \$| $$  | $| $$___\
↪$$
| $$  \     | $$  \$$    \| $$      | $$  | $$    | $$    $| $$      | $$  | $$\$$   ␣
↪\
| $$$$$     | $$  _\$$$$$$| $$   __| $$  | $$    | $$$$$$$| $$   __| $$  | $$_\$$$$$
↪$\
| $$       _| $$_|  \__| $| $$__/  | $$__/ $$    | $$__/ $| $$__/  | $$__/ $|  \__|
↪$$
```

(continues on next page)

(continued from previous page)

```
| $$      |   $$ \\$$    $$\$$    $$\$$    $$    | $$    $$\$$    $$\$$    $$\$$
↪$$
 \$$       \$$$$$$ \$$$$$$  \$$$$$$  \$$$$$$      \$$$$$$$  \$$$$$$  \$$$$$$  \$$$$$
↪$

================================================================================
```

Launch script description

To view the current console version:

```
./start.sh --version
console version: 2.0.0
```

Account using method

Console loads private key

The console provides the account generation script get_account.sh (for the script tutorial, please refer to Account Management Document. The generated account file is in the accounts directory, and the account file loaded by console must be placed in the directory.

The console startup methods are as follows:

```
./start.sh
./start.sh groupID
./start.sh groupID -pem pemName
./start.sh groupID -p12 p12Name
```

Default start

The console randomly generates an account that is started with the group number specified in the console configuration file.

```
./start.sh
```

Specify group number to start

The console randomly generates an account that is started with the group number specified on the command line.

```
./start.sh 2
```

- Note: The specified group needs to configure 'bean' in the console configuration file.

Start with PEM format private key file

- Start with the account of the specified pem file, enter the parameters: group number, -pem, and pem file path

```
./start.sh 1 -pem accounts/0xebb824a1122e587b17701ed2e512d8638dfb9c88.pem
```

Start with PKCS12 format private key file

- Start with the account of the specified p12 file, enter the parameters: group number, -p12, and p12 file path

```
./start.sh 1 -p12 accounts/0x5ef4df1b156bc9f077ee992a283c2dbb0bf045c0.p12
Enter Export Password:
```

10.1.6 Console command

help

Enter help or h to see all the commands on the console.

```
[group:1]> help
* help([-h, -help, --h, --H, --help, -H, h])  Provide help information
* addObserver                                 Add an observer node
* addSealer                                   Add a sealer node
* call                                        Call a contract by a function and␣
↪parameters
* callByCNS                                   Call a contract by a function and␣
↪parameters by CNS
* create                                      Create table by sql
* delete                                      Remove records by sql
* deploy                                      Deploy a contract on blockchain
* deployByCNS                                 Deploy a contract on blockchain by CNS
* desc                                        Description table information
* quit([quit, q, exit])                       Quit console
* freezeAccount                               Freeze the account
* freezeContract                              Freeze the contract
* generateGroup                               Generate a group for the specified node
* generateGroupFromFile                       Generate group according to the␣
↪specified file
* getAccountStatus                            GetAccountStatus of the account
* getAvailableConnections                     Get the connection information of the␣
↪nodes connected with the sdk
* getBatchReceiptsByBlockHashAndRange         Get batched transaction receipts␣
↪according to block hash and the transaction range
* getBatchReceiptsByBlockNumberAndRange       Get batched transaction receipts␣
↪according to block number and the transaction range
* getBlockByHash                              Query information about a block by hash
* getBlockByNumber                            Query information about a block by␣
↪number
* getBlockHashByNumber                        Query block hash by block number
* getBlockHeaderByHash                        Query information about a block header␣
↪by hash
* getBlockHeaderByNumber                      Query information about a block header␣
↪by block number
* getBlockNumber                              Query the number of most recent block
* getCode                                     Query code at a given address
* getConsensusStatus                          Query consensus status
* getContractStatus                           Get the status of the contract
* getCryptoType                               Get the current crypto type
* getCurrentAccount                           Get the current account info
* getDeployLog                                Query the log of deployed contracts
* getGroupConnections                         Get the node information of the group␣
↪connected to the SDK
* getGroupList                                Query group list
* getGroupPeers                               Query nodeId list for sealer and␣
↪observer nodes
* getNodeIDList                               Query nodeId list for all connected␣
↪nodes
```

(continues on next page)

(continued from previous page)

```
* getNodeInfo                                Query the specified node information.
* getNodeVersion                             Query the current node version
* getObserverList                            Query nodeId list for observer nodes.
* getPbftView                                Query the pbft view of node
* getPeers                                   Query peers currently connected to the␣
↪client
* getPendingTransactions                     Query pending transactions
* getPendingTxSize                           Query pending transactions size
* getSealerList                              Query nodeId list for sealer nodes
* getSyncStatus                              Query sync status
* getSystemConfigByKey                       Query a system config value by key
* getTotalTransactionCount                   Query total transaction count
* getTransactionByBlockHashAndIndex          Query information about a transaction␣
↪by block hash and transaction index position
* getTransactionByBlockNumberAndIndex        Query information about a transaction␣
↪by block number and transaction index position
* getTransactionByHash                       Query information about a transaction␣
↪requested by transaction hash
* getTransactionByHashWithProof              Query the transaction and transaction␣
↪proof by transaction hash
* getTransactionReceipt                      Query the receipt of a transaction by␣
↪transaction hash
* getTransactionReceiptByHashWithProof       Query the receipt and transaction␣
↪receipt proof by transaction hash
* grantCNSManager                            Grant permission for CNS by address
* grantCommitteeMember                       Grant the account committee member
* grantContractStatusManager                 Grant contract authorization to the␣
↪user
* grantContractWritePermission               Grant the account the contract write␣
↪permission.
* grantDeployAndCreateManager                Grant permission for deploy contract␣
↪and create user table by address
* grantNodeManager                           Grant permission for node␣
↪configuration by address
* grantOperator                              Grant the account operator
* grantSysConfigManager                      Grant permission for system␣
↪configuration by address
* grantUserTableManager                      Grant permission for user table by␣
↪table name and address
* insert                                     Insert records by sql
* listAbi                                    List functions and events info of the␣
↪contract.
* listAccount                                List the current saved account list
* listCNSManager                             Query permission information for CNS
* listCommitteeMembers                       List all committee members
* listContractStatusManager                  List the authorization of the contract
* listContractWritePermission                Query the account list which have␣
↪write permission of the contract.
* listDeployAndCreateManager                 Query permission information for␣
↪deploy contract and create user table
* listDeployContractAddress                  List the contractAddress for the␣
↪specified contract
* listNodeManager                            Query permission information for node␣
↪configuration
* listOperators                              List all operators
* listSysConfigManager                       Query permission information for␣
↪system configuration
* listUserTableManager                       Query permission for user table␣
↪information
* loadAccount                                Load account for the transaction␣
↪signature
```

(continues on next page)

(continued from previous page)

```
* newAccount                                    Create account
* queryCNS                                      Query CNS information by contract name␣
↪and contract version
* queryCommitteeMemberWeight                    Query the committee member weight
* queryGroupStatus                              Query the status of the specified␣
↪group of the specified node
* queryThreshold                                Query the threshold
* queryVotesOfMember                            Query votes of a committee member.
* queryVotesOfThreshold                         Query votes of updateThreshold␣
↪operation
* recoverGroup                                  Recover the specified group of the␣
↪specified node
* registerCNS                                   RegisterCNS information for the given␣
↪contract
* removeGroup                                   Remove the specified group of the␣
↪specified node
* removeNode                                    Remove a node
* revokeCNSManager                              Revoke permission for CNS by address
* revokeCommitteeMember                         Revoke the account from committee␣
↪member
* revokeContractStatusManager                   Revoke contract authorization to the␣
↪user
* revokeContractWritePermission                 Revoke the account the contract write␣
↪permission
* revokeDeployAndCreateManager                  Revoke permission for deploy contract␣
↪and create user table by address
* revokeNodeManager                             Revoke permission for node␣
↪configuration by address
* revokeOperator                                Revoke the operator
* revokeSysConfigManager                        Revoke permission for system␣
↪configuration by address
* revokeUserTableManager                        Revoke permission for user table by␣
↪table name and address
* switch([s])                                   Switch to a specific group by group ID
* select                                        Select records by sql
* setSystemConfigByKey                          Set a system config value by key
* startGroup                                    Start the specified group of the␣
↪specified node
* stopGroup                                     Stop the specified group of the␣
↪specified node
* unfreezeAccount                               Unfreeze the account
* unfreezeContract                              Unfreeze the contract
* update                                        Update records by sql
* updateCommitteeMemberWeight                   Update the committee member weight
* updateThreshold                               Update the threshold
-------------------------------------------------------------------------------
↪----------
```

**Note: **

- help shows the meaning of each command: command and command description
- for instructions on how to use specific commands, enter the command -h or –help to view them. E.g:

```
[group:1]> getBlockByNumber -h
Query information about a block by block number.
Usage: getBlockByNumber blockNumber [boolean]
blockNumber -- Integer of a block number, from 0 to 2147483647.
boolean -- (optional) If true it returns the full transaction objects, if false␣
↪only the hashes of the transactions.
```

switch

To run switch or s to switch to the specified group. The group number is displayed in front of the command prompt.

```
[group:1]> switch 2
Switched to group 2.

[group:2]>
```

**Note: ** For the group that needs to be switched, make sure that the information of the group is configured in `applicationContext.xml` (the initial state of this configuration file only provides the group 1 configuration) in the `console/conf` directory, the configured node ID and port in the group are correct, and the node is running normally.

getBlockNumber

To run getBlockNumber to view block number.

```
[group:1]> getBlockNumber
90
```

getSealerList

To run getSealerList to view the list of consensus nodes.

```
[group:1]> getSealerList
[
    ↪0c0bbd25152d40969d3d3cee3431fa28287e07cff2330df3258782d3008b876d146ddab97eab42796495bfbb281591f
    ↪
    ↪10b3a2d4b775ec7f3c2c9e8dc97fa52beb8caab9c34d026db9b95a72ac1d1c1ad551c67c2b7fdc34177857eada75836
    ↪
    ↪622af37b2bd29c60ae8f15d467b67c0a7fe5eb3e5c63fdc27a0ee8066707a25afa3aa0eb5a3b802d3a8e5e26de9d5af
]
```

getObserverList

To run getSealerList to view the list of observer nodes.

```
[group:1]> getObserverList
[
    ↪037c255c06161711b6234b8c0960a6979ef039374ccc8b723afea2107cba3432dbbc837a714b7da20111f74d5a24e91
]
```

getNodeIDList

To run getNodeIDList to view the nodes and the list of nodeIds connected to p2p nodes.

```
[group:1]> getNodeIDList
[
    ↪41285429582cbfe6eed501806391d2825894b3696f801e945176c7eb2379a1ecf03b36b027d72f480e89d15bacd4346
    ↪
```

(continues on next page)

(continued from previous page)

```
↪87774114e4a496c68f2482b30d221fa2f7b5278876da72f3d0a75695b81e2591c1939fc0d3fadb15cc359c997bafc9ea
↪
↪29c34347a190c1ec0c4507c6eed6a5bcd4d7a8f9f54ef26da616e81185c0af11a8cea4eacb74cf6f61820292b24bc5d9
↪
↪d5b3a9782c6aca271c9642aea391415d8b258e3a6d92082e59cc5b813ca123745440792ae0b29f4962df568f8ad58b75
]
```

getPbftView

To run getPbftView to view the pbft viewgraph.

```
[group:1]> getPbftView
2730
```

getConsensusStatus

To run getConsensusStatus to view the consensus status.

```
[group:1]> getConsensusStatus
ConsensusInfo{
    baseConsensusInfo=BasicConsensusInfo{
        nodeNum='4',
        nodeIndex='3',
        maxFaultyNodeNum='1',
        sealerList=[
↪11e1be251ca08bb44f36fdeedfaeca40894ff80dfd80084607a75509edeaf2a9c6fee914f1e9efda571611cf4575a157
↪
↪78a313b426c3de3267d72b53c044fa9fe70c2a27a00af7fea4a549a7d65210ed90512fc92b6194c14766366d434235c7
↪
↪95b7ff064f91de76598f90bc059bec1834f0d9eeb0d05e1086d49af1f9c2f321062d011ee8b0df7644bd54c4f9ca3d85
↪
↪b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a995847
        ],
        consensusedBlockNumber='1',
        highestblockNumber='0',
        groupId='1',
        protocolId='65544',
        accountType='1',
        cfgErr='false',
        omitEmptyBlock='true',
        nodeId=
↪'b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a99584
↪',
        allowFutureBlocks='true',
        connectedNodes='3',
        currentView='1735',
        toView='1735',
        leaderFailed='false',
        highestblockHash=
↪'0x4f6394763c33c1709e5a72b202ad4d7a3b8152de3dc698cef6f675ecdaf20a3b'
    },
```

(continues on next page)

(continued from previous page)

```
    viewInfos=[
        ViewInfo{
            nodeId=
↪'11e1be251ca08bb44f36fdeedfaeca40894ff80dfd80084607a75509edeaf2a9c6fee914f1e9efda571611cf4575a1
↪',
            view='1732'
        },
        ViewInfo{
            nodeId=
↪'78a313b426c3de3267d72b53c044fa9fe70c2a27a00af7fea4a549a7d65210ed90512fc92b6194c14766366d434235
↪',
            view='1733'
        },
        ViewInfo{
            nodeId=
↪'95b7ff064f91de76598f90bc059bec1834f0d9eeb0d05e1086d49af1f9c2f321062d011ee8b0df7644bd54c4f9ca3d
↪',
            view='1734'
        },
        ViewInfo{
            nodeId=
↪'b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a9958
↪',
            view='1735'
        }
    ]
}
```

getSyncStatus

To run getSyncStatus to view the synchronization status.

```
[group:1]> getSyncStatus
{
    "blockNumber":5,
    "genesisHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
    "isSyncing":false,
    "latestHash":
↪"0xb99703130e24702d3b580111b0cf4e39ff60ac530561dd9eb0678d03d7acce1d",
    "nodeId":
↪"cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bc
↪",
    "peers":[
        {
            "blockNumber":5,
            "genesisHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
            "latestHash":
↪"0xb99703130e24702d3b580111b0cf4e39ff60ac530561dd9eb0678d03d7acce1d",
            "nodeId":
↪"0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bc
↪"
        },
        {
            "blockNumber":5,
            "genesisHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
            "latestHash":
↪"0xb99703130e24702d3b580111b0cf4e39ff60ac530561dd9eb0678d03d7acce1d",
```

(continues on next page)

(continued from previous page)

```
            "nodeId":
↪"2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966a
↪"
        },
        {
            "blockNumber":5,
            "genesisHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
            "latestHash":
↪"0xb99703130e24702d3b580111b0cf4e39ff60ac530561dd9eb0678d03d7acce1d",
            "nodeId":
↪"ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54
↪"
        }
    ],
    "protocolId":265,
    "txPoolSize":"0"
}
```

getNodeVersion

To run getNodeVersion to view the node version.

```
[group:1]> getNodeVersion
{
    "Build Time":"20200619 06:32:10",
    "Build Type":"Linux/clang/Release",
    "Chain Id":"1",
    "FISCO-BCOS Version":"2.5.0",
    "Git Branch":"HEAD",
    "Git Commit Hash":"72c6d770e5cf0f4197162d0e26005ec03d30fcfe",
    "Supported Version":"2.5.0"
}
```

getPeers

To run getPeers to view the peers of node.

```
[group:1]> getPeers
[
        {
                "IPAndPort":"127.0.0.1:50723",
                "nodeId":
↪"8718579e9a6fee647b3d7404d59d66749862aeddef22e6b5abaafe1af6fc128fc33ed5a9a105abddab51e12004c6bf
↪",
                "Topic":[

                ]
        },
        {
                "IPAndPort":"127.0.0.1:50719",
                "nodeId":
↪"697e81e512cffc55fc9c506104fb888a9ecf4e29eabfef6bb334b0ebb6fc4ef8fab60eb614a0f2be178d0b5993464c
↪",
                "Topic":[

                ]
        },
```

(continues on next page)

(continued from previous page)

```
        {
                "IPAndPort":"127.0.0.1:30304",
                "nodeId":
↪"8fc9661baa057034f10efacfd8be3b7984e2f2e902f83c5c4e0e8a60804341426ace51492ffae087d96c0b968bd5e92
↪",
                "Topic":[

                ]
        }
]
```

getGroupPeers

To run getGroupPeers to view the list of consensus and observer node of the group where the node is located.

```
[group:1]> getGroupPeers
[
    ␣
↪cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bcc
↪
    ␣
↪ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54b
↪
    ␣
↪0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bca
↪
    ␣
↪2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966a
]
```

getGroupList

To run getGroupList to view the list of group:

```
[group:1]> getGroupList
[1]
```

getBlockHeaderByHash

Run getBlockHeaderByHash to query the block header information based on the block hash.

parameter:

- Block hash: the hash value of the block starting with 0x
- Signature list flag: The default is false, that is, the block signature list information is not displayed in the block header information, and if set to true, the block signature list is displayed.

```
[group:1]> getBlockHeaderByHash␣
↪0x99576e7567d258bd6426ddaf953ec0c953778b2f09a078423103c6555aa4362d
{
    "dbHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0x99576e7567d258bd6426ddaf953ec0c953778b2f09a078423103c6555aa4362d",
```

(continues on next page)

(continued from previous page)

```
    "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":1,
    "parentHash":
↪"0x4f6394763c33c1709e5a72b202ad4d7a3b8152de3dc698cef6f675ecdaf20a3b",
    "receiptsRoot":
↪"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
    "sealer":"0x2",
    "sealerList":[

↪"11e1be251ca08bb44f36fdeedfaeca40894ff80dfd80084607a75509edeaf2a9c6fee914f1e9efda571611cf4575a15
↪",

↪"78a313b426c3de3267d72b53c044fa9fe70c2a27a00af7fea4a549a7d65210ed90512fc92b6194c14766366d434235
↪",

↪"95b7ff064f91de76598f90bc059bec1834f0d9eeb0d05e1086d49af1f9c2f321062d011ee8b0df7644bd54c4f9ca3d8
↪",

↪"b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a99584
↪"
    ],
    "stateRoot":"0x0000000000000000000000000000000000000000000000000000000000000000
↪",
    "timestamp":"0x173ad8703d6",
    "transactionsRoot":
↪"0xb563f70188512a085b5607cac0c35480336a566de736c83410a062c9acc785ad"
}
```

getBlockHeaderByNumber

Run getBlockHeaderByNumber to query the block header information according to the block height. parameter:

- Block height
- Signature list flag: The default is false, that is, the block signature list information is not displayed in the block header information, and if set to true, the block signature list is displayed.

```
[group:1]> getBlockHeaderByNumber 1 true
{
    "dbHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0x99576e7567d258bd6426ddaf953ec0c953778b2f09a078423103c6555aa4362d",
    "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":1,
    "parentHash":
↪"0x4f6394763c33c1709e5a72b202ad4d7a3b8152de3dc698cef6f675ecdaf20a3b",
    "receiptsRoot":
↪"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
    "sealer":"0x2",
    "sealerList":[

↪"11e1be251ca08bb44f36fdeedfaeca40894ff80dfd80084607a75509edeaf2a9c6fee914f1e9efda571611cf4575a15
↪",
```

(continues on next page)

(continued from previous page)

```
↪"78a313b426c3de3267d72b53c044fa9fe70c2a27a00af7fea4a549a7d65210ed90512fc92b6194c14766366d434235
↪",

↪"95b7ff064f91de76598f90bc059bec1834f0d9eeb0d05e1086d49af1f9c2f321062d011ee8b0df7644bd54c4f9ca3d
↪",

↪"b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a9958
↪"
    ],
    "signatureList":[
        {
            "index":"0x3",
            "signature":
↪"0xb5b41e49c0b2bf758322ecb5c86dc3a3a0f9b98891b5bbf50c8613a241f05f595ce40d0bb212b6faa32e98546754
↪"
        },
        {
            "index":"0x0",
            "signature":
↪"0x411cb93f816549eba82c3bf8c03fa637036dcdee65667b541d0da06a6eaea80d16e6ca52bf1b08f77b59a834bffb
↪"
        },
        {
            "index":"0x1",
            "signature":
↪"0xea3c27c2a1486c7942c41c4dc8f15fbf9a668aff2ca40f00701d73fa659a14317d45d74372d69d43ced8e81f789e
↪"
        }
    ],
    "stateRoot":"0x0000000000000000000000000000000000000000000000000000000000000000
↪",
    "timestamp":"0x173ad8703d6",
    "transactionsRoot":
↪"0xb563f70188512a085b5607cac0c35480336a566de736c83410a062c9acc785ad"
}
```

getBlockByHash

To run getBlockByHash to view block information according to the block hash. Parameter:

- Block hash: The hash starting with 0x.
- Transaction sign: to set it false by default, the transaction in the block only displays the hash. To set it true, it displays the transaction specific information.

```
[group:1]> getBlockByHash␣
↪0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855
{
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855",
    "logsBloom":
↪"0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":"0x1",
    "parentHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
```

(continues on next page)

(continued from previous page)

```
    "sealer":"0x0",
    "sealerList":[

↪"0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bca
↪",

↪"2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966a
↪",

↪"cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bc
↪",

↪"ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54
↪"
    ],
    "stateRoot":"0x9711819153f7397ec66a78b02624f70a343b49c60bc2f21a77b977b0ed91cef9
↪",
    "timestamp":"0x1692f119c84",
    "transactions":[
        "0xa14638d47cc679cf6eeb7f36a6d2a30ea56cb8dcf0938719ff45023a7a8edb5d"
    ],
    "transactionsRoot":
↪"0x516787f85980a86fd04b0e9ce82a1a75950db866e8cdf543c2cae3e4a51d91b7"
}
[group:1]> getBlockByHash␣
↪0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855 true
{
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855",
    "logsBloom":
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":"0x1",
    "parentHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
    "sealer":"0x0",
    "sealerList":[

↪"0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bca
↪",

↪"2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966a
↪",

↪"cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bc
↪",

↪"ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54
↪"
    ],
    "stateRoot":"0x9711819153f7397ec66a78b02624f70a343b49c60bc2f21a77b977b0ed91cef9
↪",
    "timestamp":"0x1692f119c84",
    "transactions":[
        {
            "blockHash":
↪"0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855",
```

(continues on next page)

(continued from previous page)

```
            "blockNumber":"0x1",
            "from":"0x7234c32327795e4e612164e3442cfae0d445b9ad",
            "gas":"0x1c9c380",
            "gasPrice":"0x1",
            "hash":
↪"0xa14638d47cc679cf6eeb7f36a6d2a30ea56cb8dcf0938719ff45023a7a8edb5d",
            "input":
↪"0x608060405234801561001057600080fd5b506040805190810160405280600d81526020017f48656c6c6f2c20576f
↪",
            "nonce":
↪"0x3443a1391c9c29f751e8350304efb310850b8afbaa7738f5e89ddfce79b1d6",
            "to":null,
            "transactionIndex":"0x0",
            "value":"0x0"
        }
    ],
    "transactionsRoot":
↪"0x516787f85980a86fd04b0e9ce82a1a75950db866e8cdf543c2cae3e4a51d91b7"
}
```

getBlockByNumber

To run getBlockByNumber to view block information according to the block number. Parameter:

- Block number: decimal integer.
- Transaction sign: to set it false by default, the transaction in the block only displays the hash. To set it true, it displays the transaction specific information.

```
[group:1]> getBlockByNumber 1
{
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855",
    "logsBloom":
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":"0x1",
    "parentHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
    "sealer":"0x0",
    "sealerList":[

↪"0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bca
↪",

↪"2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966a
↪",

↪"cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bc
↪",

↪"ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54
↪"
    ],
    "stateRoot":"0x9711819153f7397ec66a78b02624f70a343b49c60bc2f21a77b977b0ed91cef9
↪",
```

(continues on next page)

(continued from previous page)

```
    "timestamp":"0x1692f119c84",
    "transactions":[
        "0xa14638d47cc679cf6eeb7f36a6d2a30ea56cb8dcf0938719ff45023a7a8edb5d"
    ],
    "transactionsRoot":
↪"0x516787f85980a86fd04b0e9ce82a1a75950db866e8cdf543c2cae3e4a51d91b7"
}
```

getBlockHashByNumber

To run getBlockHashByNumber to get hash through block number Parameter:

- Block number: decimal integer.

```
[group:1]> getBlockHashByNumber 1
0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855
```

getTransactionByHash

To run getTransactionByHash to check the transaction information through transaction hash. Parameter:

- Transaction hash: the transaction hash starting with 0x.

```
[group:1]> getTransactionByHash␣
↪0xed82e2cda98db8614677aba1fa8a795820bd7f68a5919a2f85018ba8c10952ac
{
        "blockHash":
↪"0x77e5b6d799edabaeae654ac5cea9baacd6f8e7ace33531d40c7ed65192de1f02",
        "blockNumber":"0x5a",
        "from":"0x7a5b31b49c6e944e9e1768785b1bc9a96cea0c17",
        "gas":"0x1c9c380",
        "gasPrice":"0x1",
        "hash":"0xed82e2cda98db8614677aba1fa8a795820bd7f68a5919a2f85018ba8c10952ac
↪",
        "input":
↪"0x100095626 16c6963650000000000000000000000000000000000000000000000000000000000000006a6f726500000000000000
↪",
        "nonce":"0x18711fff2ea68dc8b8dce4a3d3845c62a0630766a448bd9453a9127efe6f9e5
↪",
        "to":"0x738eedd873bb9722173194ab990c5b9a6c0beb25",
        "transactionIndex":"0x0",
        "value":"0x0"
}
```

getTransactionByBlockHashAndIndex

To run getTransactionByBlockHashAndIndex to inquire transaction information through block hash and transaction index. Parameter:

- Block hash: the transaction hash starting with 0x.
- Transaction index: decimal integer.

```
[group:1]> getTransactionByBlockHashAndIndex␣
↪0x77e5b6d799edabaeae654ac5cea9baacd6f8e7ace33531d40c7ed65192de1f02 0
{
        "blockHash":
↪"0x77e5b6d799edabaeae654ac5cea9baacd6f8e7ace33531d40c7ed65192de1f02",
```

(continues on next page)

(continued from previous page)

```
        "blockNumber":"0x5a",
        "from":"0x7a5b31b49c6e944e9e1768785b1bc9a96cea0c17",
        "gas":"0x1c9c380",
        "gasPrice":"0x1",
        "hash":"0xed82e2cda98db8614677aba1fa8a795820bd7f68a5919a2f85018ba8c10952ac
↪",
        "input":
↪"0x10009562616c6963650000000000000000000000000000000000000000000000000000000000000006a6f726500000000000000
↪",
        "nonce":"0x18711fff2ea68dc8b8dce4a3d3845c62a0630766a448bd9453a9127efe6f9e5
↪",
        "to":"0x738eedd873bb9722173194ab990c5b9a6c0beb25",
        "transactionIndex":"0x0",
        "value":"0x0"
}
```

getTransactionByBlockNumberAndIndex

To run getTransactionByBlockNumberAndIndex to inquire transaction information through block number and transaction index. Parameter:

- Block number: decimal integer.
- Transaction index: decimal integer.

```
[group:1]> getTransactionByBlockNumberAndIndex 2 0
{
        "blockHash":
↪"0x77e5b6d799edabaeae654ac5cea9baacd6f8e7ace33531d40c7ed65192de1f02",
        "blockNumber":"0x5a",
        "from":"0x7a5b31b49c6e944e9e1768785b1bc9a96cea0c17",
        "gas":"0x1c9c380",
        "gasPrice":"0x1",
        "hash":"0xed82e2cda98db8614677aba1fa8a795820bd7f68a5919a2f85018ba8c10952ac
↪",
        "input":
↪"0x10009562616c6963650000000000000000000000000000000000000000000000000000000000000006a6f726500000000000000
↪",
        "nonce":"0x18711fff2ea68dc8b8dce4a3d3845c62a0630766a448bd9453a9127efe6f9e5
↪",
        "to":"0x738eedd873bb9722173194ab990c5b9a6c0beb25",
        "transactionIndex":"0x0",
        "value":"0x0"
}
```

getTransactionReceipt

To run getTransactionReceipt to inquire transaction receipt through transaction hash. Parameter:

- Transaction hash: the transaction hash starting with 0x.

```
[group:1]> getTransactionReceipt␣
↪0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1
{
    "blockHash":"0x68a1f47ca465acc89edbc24115d1b435cb39fa0def53e8d0ad8090cf1827cafd
↪",
    "blockNumber":"0x5",
    "contractAddress":"0x0000000000000000000000000000000000000000",
    "from":"0xc44e7a8a4ae20d6afaa43221c6120b5e1e9f9a72",
```

(continues on next page)

(continued from previous page)

```
    "gasUsed":"0x8be5",
    "logs":[
        {
            "address":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
            "data":
↪"0x0000000000000000000000000000000000000000000000000000000000000001",
            "topics":[
                "0x66f7705280112a4d1145399e0414adc43a2d6974b487710f417edcf7d4a39d71
↪"
            ]
        }
    ],
    "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000400000000000000000000000000
↪",
    "output":"0x0000000000000000000000000000000000000000000000000000000000000001",
    "status":"0x0",
    "to":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
    "transactionHash":
↪"0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1",
    "transactionIndex":"0x0"
}
```

getPendingTransactions

To run getPendingTransactions to inquire the transactions waiting to be processed.

```
[group:1]> getPendingTransactions
[]
```

getPendingTxSize

To run getPendingTxSize to inquire the number of transactions waiting to be processed.

```
[group:1]> getPendingTxSize
0
```

getCode

To run getCode to inquire contract code according contract address. Parameter:

- Contract address: The contract address starting with 0x(To deploy contract can get contract address).

```
[group:1]> getCode 0x97b8c19598fd781aaeb53a1957ef9c8acc59f705
0x60606040526000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806
```

getTotalTransactionCount

To run getTotalTransactionCount to inquire the current block number and the total number of transaction.

```
[group:1]> getTotalTransactionCount
{
        "blockNumber":1,
        "txSum":1,
        "failedTxSum":0
}
```

deploy

To run deploy to deploy contract. (By default, HelloWorld contract and TableTest.sol are provided for example) Parameter:

- Contract name: deployment contract name (can be suffixed with .sol). It can name as either HelloWorld or HelloWorld.sol.

```
# To deploy HelloWorld contract
[group:1]> deploy HelloWorld
contract address:0xb3c223fc0bf6646959f254ac4e4a7e355b50a344

# To deploy TableTest contract
[group:1]> deploy TableTest
contract address:0x3554a56ea2905f366c345bd44fa374757fb4696a
```

Note:

- For deploying a user-written contract, we just need to place the solidity contract file in the `contracts/solidity/` directory of the console root, and then deploy it. Press the tab key to search for the contract name in the `contracts/solidity` directory.
- If the contract need to be deployed refers to other contracts or libraries, the reference format is `import "./XXX.sol";`. The related contracts and libraries are placed in the `contracts/solidity/` directory.
- If contract references the library library, the name of library file must start with `Lib` string to distinguish between the normal contract and the library file. Library files cannot be deployed and called separately.
- **Because FISCO BCOS has removed the transfer payment logic of Ethereum, the solidity contract does not support using `payable` keyword. This keyword will cause the Java contract file converted by solidity contract to fail at compilation. **

getDeployLog

Run getDeployLog to query the log information of the contract deployed by current console in the group. The log information includes the time of deployment contract, the group ID, the contract name, and the contract address. parameter:

- Log number: optional. To return the latest log information according to the expected value entered. When the actual number is less than the expected value, it returns by the actual number. When the expected value is not given, it returns by the latest 20 log information by default.

```
[group:1]> getDeployLog 2

2019-05-26 08:37:03  [group:1]  HelloWorld           ␣
↪0xc0ce097a5757e2b6e189aa70c7d55770ace47767
2019-05-26 08:37:45  [group:1]  TableTest           ␣
↪0xd653139b9abffc3fe07573e7bacdfd35210b5576

[group:1]> getDeployLog 1

2019-05-26 08:37:45  [group:1]  TableTest           ␣
↪0xd653139b9abffc3fe07573e7bacdfd35210b5576
```

Note: If you want to see all the deployment contract log information, please check the `deploylog.txt` file in the `console` directory. The file only stores the log records of the last 10,000 deployment contracts.

call

To run call to call contract. Parameter:

- Contract name: the contract name of the deployment (can be suffixed with .sol).

- Contract address: the address obtained by the deployment contract. The contract address can omit the prefix 0. For example, 0x000ac78 can be abbreviated as 0xac78.
- Contract interface name: the called interface name.
- Parameter: determined by contract interface parameters.

**Parameters are separated by spaces; array parameters need to be enclosed in brackets, such as [1,2,3]; array is a string or byte type and needs to be enclosed in double quotation marks, such as ["alice", "bob"]. Note that there are no spaces in the array parameters; boolean types are true or false. **

```text
# To call the get interface of HelloWorld to get the name string
[group:1]> call HelloWorld 0x175b16a1299c7af3e2e49b97e68a44734257a35e get
-------------------------------------------------------------------------------------------
Return code: 0
description: transaction executed successfully
Return message: Success
-------------------------------------------------------------------------------------------
Return values:
[
    "Hello,World!"
]
-------------------------------------------------------------------------------------------

# To call the set interface of HelloWorld to set the name string
[group:1]> call HelloWorld 0x175b16a1299c7af3e2e49b97e68a44734257a35e set "Hello,␣
↪FISCO BCOS"
transaction hash:␣
↪0x54b7bc73e3b57f684a6b49d2fad41bd8decac55ce021d24a1f298269e56f1ce1
-------------------------------------------------------------------------------------------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------------
Output
Receipt message: Success
Return message: Success
-------------------------------------------------------------------------------------------
Event logs
Event: {}

# To call the get interface of HelloWorld to get the name string for checking␣
↪whether the settings take effect
[group:1]> call HelloWorld 0x175b16a1299c7af3e2e49b97e68a44734257a35e get
-------------------------------------------------------------------------------------------
Return code: 0
description: transaction executed successfully
Return message: Success
-------------------------------------------------------------------------------------------
Return values:
[
    "Hello,FISCO BCOS"
]
-------------------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
# To call the insert interface of TableTest to insert the record, the fields are␣
↪name, item_id, item_name
[group:1]> call TableTest 0x5f248ad7e917cddc5a4d408cf18169d19c0990e5 insert "fruit
↪" 1 "apple"
transaction hash:␣
↪0x64bfab495dc1f50c58d219b331df5a47577aa8afc16be926260238a9b0ec0fbb
-------------------------------------------------------------------------------------
↪----------
transaction status: 0x0
description: transaction executed successfully
-------------------------------------------------------------------------------------
↪----------
Output
Receipt message: Success
Return message: Success
-------------------------------------------------------------------------------------
↪----------
Event logs
Event: {"InsertResult":[1]}

# To call TableTest's select interface to inquiry records
[group:1]> [group:1]> call TableTest 0x5f248ad7e917cddc5a4d408cf18169d19c0990e5␣
↪select "fruit"
-------------------------------------------------------------------------------------
↪----------
Return code: 0
description: transaction executed successfully
Return message: Success
-------------------------------------------------------------------------------------
↪----------
Return values:
[
    [
        "fruit"
    ],
    [
        1
    ],
    [
        "apple"
    ]
]
-------------------------------------------------------------------------------------
↪----------
```

Note: TableTest.sol contract codeReference here。

deployByCNS

Run deployByCNS and deploy the contract with CNS. Contracts deployed with CNS can be called directly with the contract name.

Parameter:

- Contract name: deployable contract name.
- Contract version number: deployable contract version number.

```
# To deploy HelloWorld contract 1.0 version
[group:1]> deployByCNS HelloWorld 1.0
contract address:0x3554a56ea2905f366c345bd44fa374757fb4696a

# To deploy HelloWorld contract 2.0 version
[group:1]> deployByCNS HelloWorld 2.0
contract address:0x07625453fb4a6459cbf14f5aa4d574cae0f17d92

# To deploy TableTest contract
[group:1]> deployByCNS TableTest 1.0
contract address:0x0b33d383e8e93c7c8083963a4ac4a58b214684a8
```

Note:

- For deploying the contracts compiled by users only needs to place the solidity contract file in the `contracts/solidity/` directory of the console root and to deploy it. Press tab key to search for the contract name in the `contracts/solidity/` directory.
- If the contract to be deployed references other contracts or libraries, the reference format is `import "./XXX.sol";`. The related contract and library are placed in the `contracts/solidity/` directory.
- **Because FISCO BCOS has removed the transfer payment logic of Ethereum, the solidity contract does not support using `payable` keyword. This keyword will cause the Java contract file converted by solidity contract to fail at compilation. **

queryCNS

Run queryCNS and query the CNS table record information (the mapping of contract name and contract address) according to the contract name and contract version number (optional parameter).

Parameter:

- Contract name: deployable contract name.
- Contract version number: (optional) deployable contract version number.

```
[group:1]> queryCNS HelloWorld.sol
----------------------------------------------------------------------------------------
↪----------
|                    version                   |                    address
↪          |
|                      1.0                     |
↪0x3554a56ea2905f366c345bd44fa374757fb4696a  |
----------------------------------------------------------------------------------------
↪----------

[group:1]> queryCNS HelloWorld 1.0
----------------------------------------------------------------------------------------
↪----------
|                    version                   |                    address
↪          |
|                      1.0                     |
↪0x3554a56ea2905f366c345bd44fa374757fb4696a  |
----------------------------------------------------------------------------------------
↪----------
```

callByCNS

To run deployByCNS and deploy the contract with CNS. Parameter:

- Contract name and contract version number: The contract name and contract version number are separated by colon, such as `HelloWorld:1.0` or `HelloWorld.sol:1.0`. When the contract version number is omitted like `HelloWorld` or `HelloWorld.sol`, the latest version of the contract is called.
- Contract interface name: The called contract interface name.
- Parameter: is determined by the parameter of contract interface. The parameters are separated by spaces, where the string and byte type parameters need to be enclosed in double quotation marks; the array parameters need to be enclosed in brackets, such as [1, 2, 3]. The array is a string or byte type with double quotation marks such as ["alice", "bob"]; the boolean type is true or false.

```
# To call the HelloWorld contract 1.0 version to set the name string by the set␣
↪interface
[group:1]> callByCNS HelloWorld:1.0 set "Hello,CNS"
transaction hash:0x80bb37cc8de2e25f6a1cdcb6b4a01ab5b5628082f8da4c48ef1bbc1fb1d28b2d

# To call the HelloWorld contract 2.0 version to set the name string by the set␣
↪interface
[group:1]> callByCNS HelloWorld:2.0 set "Hello,CNS2"
transaction hash:0x43000d14040f0c67ac080d0179b9499b6885d4a1495d3cfd1a79ffb5f2945f64

# To call the HelloWorld contract 1.0 version to get the name string by the get␣
↪interface
[group:1]> callByCNS HelloWorld:1.0 get
Hello,CNS

# To call the HelloWorld contract 2.0 version to get the name string by the get␣
↪interface
[group:1]> callByCNS HelloWorld get
Hello,CNS2
```

addSealer

To run addSealer to add the node as a consensus node. Parameter:

- node's nodeId

```
[group:1]> addSealer␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
{
        "code":0,
        "msg":"success"
}
```

addObserver

To run addObserver to add the node as an observed node. Parameter:

- node's nodeId

```
[group:1]> addObserver␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
{
        "code":0,
        "msg":"success"
}
```

removeNode

To run removeNode to exit the node. The exit node can be added as a consensus node by the addSealer command or can be added as an observation node by the addObserver command. Parameter:

- node's nodeId

```
[group:1]> removeNode␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
{
        "code":0,
        "msg":"success"
}
```

setSystemConfigByKey

To run setSystemConfigByKey to set the system configuration in key-value pairs. The currently system configuration supports `tx_count_limit`, `tx_gas_limit`, `rpbft_epoch_sealer_num` and `rpbft_epoch_block_num`. The key name of these two configuration can be complemented by the tab key:

- tx_count_limit: block maximum number of packaged transactions
- tx_gas_limit: The maximum number of gas allowed to be consumed
- rpbft_epoch_sealer_num: rPBFT system configuration, the number of consensus nodes selected in a consensus epoch
- rpbft_epoch_block_num: rPBFT system configuration, number of blocks generated in one consensus epoch
- consensus_timeout: During the PBFT consensus process, the timeout period for each block execution, the default is 3s, the unit is seconds

Parameters:

- key
- value

```
[group:1]> setSystemConfigByKey tx_count_limit 100
{
        "code":0,
        "msg":"success"
}
```

getSystemConfigByKey

To run getSystemConfigByKe to inquire the value of the system configuration according to the key. Parameter:

- key

```
[group:1]> getSystemConfigByKey tx_count_limit
100
```

grantUserTableManager

Run grantUserTableManager to grant the account to write to the user table.

parameter:

- table name
- account address

```
[group:1]> grantUserTableManager t_test 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listUserTableManager

Run listUserTableManager to query the account's table that has writing permission to the user table.

parameter:

- table name

```
[group:1]> listUserTableManager t_test
----------------------------------------------------------------------------------------
↪----------
|                    address                    |                 enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                      2
↪          |
----------------------------------------------------------------------------------------
↪----------
```

revokeUserTableManager

Run revokeUserTableManager to revoke the account's writing permission from the user table.

parameter:

- table name
- account address

```
[group:1]> revokeUserTableManager t_test 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantDeployAndCreateManager

Run grantDeployAndCreateManager to grant the account's permission of deployment contract and user table creation.

parameter:

- account address

```
[group:1]> grantDeployAndCreateManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listDeployAndCreateManager

Run listDeployAndCreateManager to query the account's permission of deployment contract and user table creation.

```
[group:1]> listDeployAndCreateManager
-------------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                       2
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

revokeDeployAndCreateManager

Run revokeDeployAndCreateManager to revoke the account's permission of deployment contract and user table creation.

parameter:

- account address

```
[group:1]> revokeDeployAndCreateManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantNodeManager

Run grantNodeManager to grant the account's node management permission.

parameter:

- account address

```
[group:1]> grantNodeManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listNodeManager

Run the listNodeManager to query the list of accounts that have node management.

```
[group:1]> listNodeManager
-------------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                       2
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

revokeNodeManager

Run revokeNodeManager to revoke the account's node management permission.

parameter:

- account address

```
[group:1]> revokeNodeManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantCNSManager

Run grantCNSManager to grant the account's permission of using CNS. parameter:

- account address

```
[group:1]> grantCNSManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listCNSManager

Run listCNSManager to query the list of accounts that have CNS.

```
[group:1]> listCNSManager
--------------------------------------------------------------------------------------
↪----------
|                    address                    |                enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                     2
↪          |
--------------------------------------------------------------------------------------
↪----------
```

revokeCNSManager

Run revokeCNSManager to revoke the account's permission of using CNS. parameter:

- account address

```
[group:1]> revokeCNSManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantSysConfigManager

Run grantSysConfigManager to grant the account's permission of modifying system parameter. parameter:

- account address

```
[group:1]> grantSysConfigManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listSysConfigManager

Run listSysConfigManager to query the list of accounts that have modified system parameters.

```
[group:1]> listSysConfigManager
-------------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                       2
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

revokeSysConfigManager

Run revokeSysConfigManager to revoke the account's permission of modifying system parameter. parameter:

- account address

```
[group:1]> revokeSysConfigManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantContractWritePermission

Run grantContractWritePermissio to grant the account the contract write permission. parameters:

- contract address
- account address

```
[group:1]> grantContractWritePermission 0xc0ce097a5757e2b6e189aa70c7d55770ace47767
↪0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listContractWritePermission

Run listContractWritePermission to query the account list which have write permission of the contract. parameters:

- contract address

```
[group:1]> listContractWritePermission 0xc0ce097a5757e2b6e189aa70c7d55770ace47767
-------------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                      11
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

revokeContractWritePermission

Run revokeContractWritePermission to Revoke the account the contract write permission. parameters:

- 合约地址
- account address

```
[group:1]> revokeContractWritePermission␣
↪0xc0ce097a5757e2b6e189aa70c7d55770ace47767␣
↪0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

quit

To run quit, q or exit to exit the console.

```
quit
```

[create sql]

Run create sql statement to create a user table in mysql statement form.

```
# Create user table t_demo whose primary key is name and other fields are item_id␣
↪and item_name
[group:1]> create table t_demo(name varchar, item_id varchar, item_name varchar,␣
↪primary key(name))
Create 't_demo' Ok.
```

Note:

- The field types for creating table are all string types. Even if other field types of the database are provided, the field types have to be set according to the string type.
- The primary key field must be specified. For example, to create a t_demo table with the primary key field as name.
- The primary key of the table has different concept from the primary key in the relational database. Here, the value of the primary key is not unique, and the primary key value needs to be passed when the blockchain underlying platform is handling records.
- You can specify the field as the primary key, but the setting fields such as self-incrementing, non-empty, indexing, etc do not work.

desc

Run desc statement to query the field information of the table in mysql statement form.

```
# query the field information of the t_demo table. you can view the primary key␣
↪name and other field names of the table.

[group:1]> desc t_demo
{
    "key":"name",
    "valueFields":"item_id,item_name"
}
```

[insert sql]

Run insert sql statement to insert the record in the mysql statement form.

```
[group:1]> insert into t_demo (name, item_id, item_name) values (fruit, 1, apple1)
Insert OK, 1 row affected.
```

Note:

- must insert a record sql statement with the primary key field value of the table.
- The enter values with punctuation, spaces, or strings containing letters starting with a number requires double quotation marks, and no more double quotation marks are allowed inside.

[select sql]

Run select sql statement to query the record in mysql statement form.

```
# query the records contain all fields
select * from t_demo where name = fruit
{item_id=1, item_name=apple1, name=fruit}
1 row in set.

# query the records contain the specified fields
[group:1]> select name, item_id, item_name from t_demo where name = fruit
{name=fruit, item_id=1, item_name=apple1}
1 row in set.

# insert a new record
[group:1]> insert into t_demo values (fruit, 2, apple2)
Insert OK, 1 row affected.

# use the keyword 'and' to connect multiple query condition
[group:1]> select * from t_demo where name = fruit and item_name = apple2
{item_id=2, item_name=apple2, name=fruit}
1 row in set.

# use limit field to query the first line of records. If the offset is not␣
↪provided, it is 0 by default.
[group:1]> select * from t_demo where name = fruit limit 1
{item_id=1, item_name=apple1, name=fruit}
1 row in set.

# use limit field to query the second line record. The offset is 1
[group:1]> select * from t_demo where name = fruit limit 1,1
{item_id=2, item_name=apple2, name=fruit}
1 rows in set.
```

Note:

- For querying the statement recording sql, the primary key field value of the table in the where clause must be provided.
- The limit field in the relational database can be used. Providing two parameters which are offset and count.
- The where clause only supports the keyword 'and'. Other keywords like 'or', 'in', 'like', 'inner', 'join', 'union', subquery, multi-table joint query, and etc. are not supported.
- The enter values with punctuation, spaces, or strings containing letters starting with a number requires double quotation marks, and no more double quotation marks are allowed inside.

[update sql]

Run update sql statement to update the record in mysql statement form.

```
[group:1]> update t_demo set item_name = orange where name = fruit and item_id = 1
Update OK, 1 row affected.
```

Note:

- For updating the where clause of recording sql statement, the primary key field value of the table in the where clause must be provided.
- The enter values with punctuation, spaces, or strings containing letters starting with a number requires double quotation marks, and no more double quotation marks are allowed inside.

[delete sql]

Run delete sql statement to delete the record in mysql statement form.

```
[group:1]> delete from t_demo where name = fruit and item_id = 1
Remove OK, 1 row affected.
```

Note:

- For deleting the where clause of recording sql statement, the primary key field value of the table in the where clause must be provided.
- The enter values with punctuation, spaces, or strings containing letters starting with a number requires double quotation marks, and no more double quotation marks are allowed inside.

Important: The executing of the freezeContract/unfreezeContract/grantContractStatusManager commands for contract management should specify the private key to start the console for permission.This private key is also the account private key used to deploy the specified contract. So a private key should be specified to launch the console when deploying the contract.

---

freezeContract

Run freezeContract to freeze contract according contract address. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.

```
[group:1]> freezeContract 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
{
    "code":0,
    "msg":"success"
}
```

unfreezeContract

Run unfreezeContract to unfreeze contract according contract address. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.

```
[group:1]> unfreezeContract 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
{
    "code":0,
    "msg":"success"
}
```

grantContractStatusManager

Run grantCNSManager to grant the account's permission of contract status managememt. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.
- Account address: tx.origin. The prefix of 0x is not necessary.

```
[group:1]> grantContractStatusManager 0x30d2a17b6819f0d77f26dd3a9711ae75c291f7f1␣
↪0x965ebffc38b309fa706b809017f360d4f6de909a
{
    "code":0,
    "msg":"success"
}
```

revokeContractStatusManager

Run revokeContractStatusManager to revoke the contract management authority of the specified authority account for the specified contract. parameter:

- Contract address: The contract address can be obtained by deploying the contract, and the 0x prefix is not required.
- Account address: tx.origin, where 0x prefix is optional.

```
[group:1]> revokeContractStatusManager 0x30d2a17b6819f0d77f26dd3a9711ae75c291f7f1␣
↪0x965ebffc38b309fa706b809017f360d4f6de909a
{
    "code":1,
    "msg":"success"
}
```

getContractStatus

To run getContractStatus to query contract status according contract address. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.

```
[group:1]> getContractStatus 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
The contract is available.
```

listContractStatusManager

To run listContractStatusManager to query a list of authorized accounts that can manage a specified contract. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.

```
[group:1]> listContractStatusManager 0x30d2a17b6819f0d77f26dd3a9711ae75c291f7f1
[
    "0x0cc9b73b960323816ac5f52806257184c08b5ac0",
    "0x965ebffc38b309fa706b809017f360d4f6de909a"
]
```

grantCommitteeMember

grant account with Committee Member permission. Parameters:

- account address

```
[group:1]> grantCommitteeMember 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a
{
    "code":0,
    "msg":"success"
}
```

revokeCommitteeMember

revoke account's Committee Member permission, parameters:

- account address

```
[group:1]> revokeCommitteeMember 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a
{
    "code":0,
    "msg":"success"
}
```

listCommitteeMembers

```
[group:1]> listCommitteeMembers
-------------------------------------------------------------------------------------------
↪----------
|                   address                    |                  enable_num
↪          |
| 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a   |                       1
↪          |
| 0x85961172229aec21694d742a5bd577bedffcfec3   |                       2
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

updateThreshold

vote to modify the votes threshold, Parameters:

- threshold:[0,99]

```
[group:1]> updateThreshold 75
{
    "code":0,
    "msg":"success"
}
```

queryThreshold

query votes threshold

```
[group:1]> queryThreshold
Effective threshold : 50%
```

queryCommitteeMemberWeight

```
[group:1]> queryCommitteeMemberWeight 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a
Account: 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a Weight: 1
```

updateCommitteeMemberWeight

update Committee Member's votes. Parameters:

- account address
- votes

```
[group:1]> updateCommitteeMemberWeight 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a 2
{
    "code":0,
    "msg":"success"
}
```

grantOperator

grantOperator, committee member's permission, parameters:

- account address

```
[group:1]> grantOperator 0x283f5b859e34f7fd2cf136c07579dcc72423b1b2
{
    "code":0,
    "msg":"success"
}
```

revokeOperator

revokeOperator, committee member's permission, parameters:

- account address

```
[group:1]> revokeOperator 0x283f5b859e34f7fd2cf136c07579dcc72423b1b2
{
    "code":0,
    "msg":"success"
}
```

listOperators

list address who has operator permission。

```
[group:1]> listOperators
---------------------------------------------------------------------------------------
↪----------
|                     address                     |                  enable_num        ␣
↪           |
| 0x283f5b859e34f7fd2cf136c07579dcc72423b1b2  |                       1               ␣
↪           |
---------------------------------------------------------------------------------------
↪----------
```

queryVotesOfThreshold

Query the voting status of updateThreshold:

```
[group:1]> queryVotesOfThreshold
The votes of the updateThreshold operation : {"0.100000":[{"block_limit":"10002",
↪"origin":"0x2eb1be0f52c0d00f9594a021240ea7fb027d7485"}]}
```

(continues on next page)

(continued from previous page)

queryVotesOfMember

Query the voting status of the designated account being elected as a committee member. If no committee member votes, it will return `null`:

- Account address: The account address being queried

```
[group:1]> queryVotesOfMember 0xc398d318662aa19487c405a45267ecd60115adec
queried account: 0xc398d318662aa19487c405a45267ecd60115adec
votes:{"grant":[{"block_limit":"10003","origin":
↪"0x2eb1be0f52c0d00f9594a021240ea7fb027d7485"}]}
```

freezeAccount

Run freezeAccount to freeze account according account address. Parameter:

- account address: tx.origin. The prefix of 0x is necessary.

```
[group:1]> freezeAccount 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
{
    "code":0,
    "msg":"success"
}
```

unfreezeAccount

Run unfreezeAccount to unfreeze account according account address. Parameter:

- account address: tx.origin. The prefix of 0x is necessary.

```
[group:1]> unfreezeAccount 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
{
    "code":0,
    "msg":"success"
}
```

getAccountStatus

Run getAccountStatus to get status of the account according account address. Parameter:

- account address: tx.origin. The prefix of 0x is necessary.

```
[group:1]> getAccountStatus 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
The account is available.
```

getCurrentAccount

Get the current account address.

```
[group:1]> getCurrentAccount
0x6fad87071f790c3234108f41b76bb99874a6d813
```

getCryptoType

Get the type of the node ledger and SSL protocol that the current console is connected to.

Note:

- The OSCCA-approved cryptography ledger type is `ECDSA`, the OSCCA-approved cryptography ledger type is `SM`
- The OpenSSL protocol type is `ECDSA`, and the OSCCA-approved cryptography SSL protocol type is `SM`

```
[group:1]> getCryptoType
ledger crypto type: ECDSA
ssl crypto type: ECDSA
```

getAvailableConnections

Get the node connection information of the SDK connection.

```
[group:1]> getAvailableConnections
[
    127.0.0.1:20200,
    127.0.0.1:20201
]
```

getGroupConnections

From the list of nodes connected to the SDK, filter out the node list information that starts the console currently logged in to the group.

```
[group:1]> getGroupConnections
[
    127.0.0.1:20200,
    127.0.0.1:20201
]
```

generateGroup

Dynamically create a new group for the specified node, parameters:

- `endPoint`: The `IP:Port` of the blockchain node receiving the request for creating a new group. The information of all the nodes `IP:Port` connected by the SDK can be obtained through the command `getAvailableConnections`;
- `groupId`: The newly created group ID;
- `timestamp`: The timestamp of the genesis block of the newly created group, can be obtained by the command `echo $(($(date'+%s')*1000));`
- `sealerList`: The consensus node list of the newly created group, separated by spaces between multiple consensus node IDs。

An example of creating a new group 2 for the blockchain nodes listening on the port `20200` of this machine is as follows:

```
# Get timestamp
$ echo $(($(date '+%s')*1000))
1590586645000
```

```
[group:1]> generateGroup  127.0.0.1:20200 2 1590586645000␣
↪b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a99584
GroupStatus{
    code='0x0',
    message='Group 2 generated successfully',
    status='null'
}
```

generateGroupFromFile

Create a new group for the specified node list through the new group configuration file. The configuration file specifies the need to create the group node list, the consensus list of the new group, and the creation block timestamp. The group configuration example is `group-generate-config. toml` is as follows:

```
# The peers to generate the group
[groupPeers]
peers=["127.0.0.1:20200", "127.0.0.1:20201"]

# The consensus configuration of the generated group
[consensus]
# The sealerList
sealerList=[
↪"b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a9958
↪",
↪"11e1be251ca08bb44f36fdeedfaeca40894ff80dfd80084607a75509edeaf2a9c6fee914f1e9efda571611cf4575a1
↪"]

[genesis]
# The genesis timestamp, It is recommended to set to the current utcTime, which␣
↪must be greater than 0
timestamp = "1590586645000"
```

The parameters of the `generateGroupFromFile` command include:

`-groupConfigFilePath`: group configuration file path, console `conf/group-generate-config. toml` is the provided group configuration file template, users can copy and modify the configuration template according to the actual scene, and load the modified Group configuration file;

`-groupId`: The newly created group ID.

```
[group:1]> generateGroupFromFile conf/group-generate-config.toml 3
* Result of 127.0.0.1:20200:
GroupStatus{
    code='0x0',
    message='Group 3 generated successfully',
    status='null'
}
* Result of 127.0.0.1:20201:
GroupStatus{
    code='0x0',
    message='Group 3 generated successfully',
    status='null'
}
```

startGroup

Start the group for the specified node, parameters:

- `endPoint`: The `IP:Port` of the blockchain node receiving the request for start a new group. The information of all the nodes `IP:Port` connected by the SDK can be obtained through the command

getAvailableConnections;

- groupId: The ID of the group to start.

An example of the console command to start group 2 is as follows:

```
# Get 127.0.0.1:20200 current group list
[group:1]> getGroupList 127.0.0.1:20200
[1]
[group:1]> startGroup 127.0.0.1:20200 2
GroupStatus{
    code='0x0',
    message='Group 2 started successfully',
    status='null'
}
# 127.0.0.1: After group 2 of 20200 is successfully started, group 2 is added to␣
↪the group list
[group:1]> getGroupList 127.0.0.1:20200
[1, 2]
```

stopGroup

Stop the group for the specified node, parameters:

- endPoint: The IP:Port of the blockchain node receiving the request for stop a new group. The information of all the nodes IP:Port connected by the SDK can be obtained through the command getAvailableConnections;
- groupId: The ID of the group to stop.

An example of the console command to stop group 2 is as follows:

```
# Get 127.0.0.1:20200 current group list
[group:1]> getGroupList 127.0.0.1:20200
[1, 2]
[group:1]> stopGroup 127.0.0.1:20200 2
GroupStatus{
    code='0x0',
    message='Group 2 stopped successfully',
    status='null'
}
# 127.0.0.1: After group 2 of 20200 stops successfully, group 2 is removed from␣
↪the group list
[group:1]> getGroupList 127.0.0.1:20200
[1]
```

removeGroup

Remove the group for the specified node, parameters:

- endPoint: The IP:Port of the blockchain node receiving the request for remove a new group. The information of all the nodes IP:Port connected by the SDK can be obtained through the command getAvailableConnections;
- groupId: The ID of the group to remove.

An example of the console command to delete group 2 is as follows:

```
# Delete group 2 of 127.0.0.1:20200
[group:1]> removeGroup 127.0.0.1:20200 2
GroupStatus{
    code='0x0',
```

(continues on next page)

(continued from previous page)

```
    message='Group 2 deleted successfully',
    status='null'
}
# After deleting group 2 of 127.0.0.1:20200, try to activate the deleted group,␣
↪but the activation fails
[group:1]> startGroup 127.0.0.1:20200 2
GroupStatus{
    code='0x5',
    message='Group 2 has been deleted',
    status='null'
}
[group:1]> getGroupList 127.0.0.1:20200
[1]
```

recoverGroup

Recover the group for the specified node, parameters:

- `endPoint`: The `IP:Port` of the blockchain node receiving the request for recover a new group. The information of all the nodes `IP:Port` connected by the SDK can be obtained through the command `getAvailableConnections`;
- `groupId`: The ID of the group to recover.

```
# Get the current group list of 127.0.0.1:20200
[group:1]> getGroupList 127.0.0.1:20200
[1]
# Recover group 2 at 127.0.0.1:20200
[group:1]> recoverGroup 127.0.0.1:20200 2
GroupStatus{
    code='0x0',
    message='Group 2 recovered successfully',
    status='null'
}
# Start group 2 at 127.0.0.1:20200
[group:1]> startGroup 127.0.0.1:20200 2
GroupStatus{
    code='0x0',
    message='Group 2 started successfully',
    status='null'
}
# Get the current group list of 127.0.0.1:20200, add group 2
[group:1]> getGroupList 127.0.0.1:20200
[1, 2]
```

getBatchReceiptsByBlockNumberAndRange

Get the batched transaction receipts according to the specified block number and transaction range

- `blockNumber`: (required) The number of the block that contains the required transaction receipts
- `from`: (optional) The start index of the required transaction receipts (default is 0)
- `count`: the count of the required transaction receipts (default fetch all the receipts), when set to -1, return all receipts of the block

```
[group:1]> getBatchReceiptsByBlockNumberAndRange 1
TransactionReceiptsInfo{
    blockInfo=BlockInfo{
        receiptRoot=
↪'0x67182babfe1500a8ec442a8b9548e7d0d912af4943c3d549bdf9ed0c76fe8c11'
```

(continues on next page)

(continued from previous page)

```
        blockNumber='0x1',
        blockHash=
↪'0x5eb495f6fa457dbcaf6323630a257a65a7085e01087421b7191d8efec69da0c0',
        receiptsCount='0x1'
    },
    transactionReceipts=[
        TransactionReceipt{
            transactionHash=
↪'0xa3ce50e3f03d3282e21248172efd1345b9eb15b281791b499f2e6c7bbe464667',
            transactionIndex='0x0',
            root='null',
            blockNumber='null',
            blockHash='null',
            from='0x2eb1be0f52c0d00f9594a021240ea7fb027d7485',
            to='0x0000000000000000000000000000000000001008',
            gasUsed='0x589b',
            contractAddress='0x0000000000000000000000000000000000000000',
            logs=[

            ],
            logsBloom='null',
            status='0x0',
            statusMsg='null',
            input='null',
            output=
↪'0x0000000000000000000000000000000000000000000000000000000000000001',
            txProof=null,
            receiptProof=null
        }
    ]
}
```

getBatchReceiptsByBlockHashAndRange

Get the batched transaction receipts according to the specified block hash and transaction range:

- `blockHash`: (required) The hash of the block that contains the required transaction receipts
- `from`: (optional) The start index of the required transaction receipts (default is 0)
- `count`: (optional) The count of the required transaction receipts (default fetch all the receipts), when set to -1, return all receipts of the block

```
[group:1]> getBatchReceiptsByBlockHashAndRange␣
↪0x5eb495f6fa457dbcaf6323630a257a65a7085e01087421b7191d8efec69da0c0
TransactionReceiptsInfo{
    blockInfo=BlockInfo{
        receiptRoot=
↪'0x67182babfe1500a8ec442a8b9548e7d0d912af4943c3d549bdf9ed0c76fe8c11',
        blockNumber='0x1',
        blockHash=
↪'0x5eb495f6fa457dbcaf6323630a257a65a7085e01087421b7191d8efec69da0c0',
        receiptsCount='0x1'
    },
    transactionReceipts=[
        TransactionReceipt{
            transactionHash=
↪'0xa3ce50e3f03d3282e21248172efd1345b9eb15b281791b499f2e6c7bbe464667',
            transactionIndex='0x0',
            root='null',
            blockNumber='null',
```

(continues on next page)

(continued from previous page)

```
            blockHash='null',
            from='0x2eb1be0f52c0d00f9594a021240ea7fb027d7485',
            to='0x0000000000000000000000000000000000001008',
            gasUsed='0x589b',
            contractAddress='0x0000000000000000000000000000000000000000',
            logs=[

            ],
            logsBloom='null',
            status='0x0',
            statusMsg='null',
            input='null',
            output=
↪'0x0000000000000000000000000000000000000000000000000000000000000001',
            txProof=null,
            receiptProof=null
        }
    ]
}
```

10.2 Console

Important:

- `Console 1.x` series is based on Web3SDK implementation, after `Console 2.6` is based on 'Java SDK <../sdk/java_sdk/index.html >'_ implementation, this tutorial is aimed at 1.x version console, for 2.6 and above version console usage documentation please refer to here
- You can view the current console version through the command `./start.sh --version`

Console is an important interactive client tool of FISCO BCOS 2.0. It establishes a connection with blockchain node through Java SDK to request read and write access for blockchain node data. Console has a wealth of commands, including blockchain status inquiry, blockchain nodes management, contracts deployment and calling. In addition, console provides a contract compilation tool that allows users to easily and quickly compile Solidity contract files into Java contract files.

10.2.1 Console command

Console command consists of two parts, the instructions and the parameters related to the instruction:

- Instruction: instruction is an executed operation command, including blockchain status inquiry and contracts deployment and calling. And some of the instructions call the JSON-RPC interface, so they have same name as the JSON-RPC interface. Use suggestions: instructions can be completed using the tab key, and support for displaying historical input commands by pressing the up and down keys.
- Parameters related to the instruction: parameters required by instruction call interface. Instructions to parameters and parameters to parameters are separated by spaces. The parameters name same as JSON-RPC interface and the explanation of getting information field can be referred to JSON-RPC API.

10.2.2 Common command link:

Contract related commands

- useCNS to deploy and call contract (recommend)

  - deploy contract: deployByCNS
  - call contract: callByCNS
  - query CNS deployment contract information: queryCNS
- deploy and call contract normally
  - deploy contract: deploy
  - call contract: call

Other commands

- query block number:getBlockNumber
- query Sealer list:getSealerList
- query the information of transaction receipt: getTransactionReceipt
- switch group: switch

10.2.3 Shortcut key

- `Ctrl+A`: move cursor to the beginning of line
- `Ctrl+D`: exit console
- `Ctrl+E`: move cursor to the end of line
- `Ctrl+R`: search for the history commands have been entered
- ↑: browse history commands forward
- ↓: browse history commands backward

10.2.4 Console response

When a console command is launched, the console will obtain the result of the command execution and displays the result at the terminal. The execution result is divided into two categories:

- True: The command returns to the true execution result as a string or json.
- False: The command returns to the false execution result as a string or json.
  - When console command call the JSON-RPC interface, error code reference here.
  - When console command call the Precompiled Service interface, error code reference here.

10.2.5 Console configuration and operation

Important: Precondition： to build FISCO BCOS blockchain, please refer to Building Chain Script or Enterprise Tools.

Get console

```
$ cd ~ && mkdir fisco && cd fisco
# get console
$ curl -#LO https://github.com/FISCO-BCOS/console/releases/download/v2.9.2/
↪download_console.sh && bash download_console.sh -c 1.2.0
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/console/raw/master-2.0/tools/download_console.sh && bash download_console.sh -c 1.2.0

The directory structure is as follows:

```
|-- apps # console jar package directory
|   -- console.jar
|-- lib # related dependent jar package directory
|-- conf
|   |-- applicationContext-sample.xml # configuration file
|   |-- log4j.properties  # log configuration file
|-- contracts # directory where contract locates
|   -- solidity  # directory where solidity contract locates
|       -- HelloWorld.sol # normal contract: HelloWorld contract, is deployable␣
↪and callable
|       -- TableTest.sol # the contracts by using CRUD interface: TableTest␣
↪contract, is deployable and callable
|       -- Table.sol # CRUD interfac contract
|   -- console  # The file directory of contract abi, bin, java compiled when␣
↪console deploys the contract
|   -- sdk      # The file directory of contract abi, bin, java compiled by␣
↪sol2java.sh script
|-- start.sh # console start script
|-- get_account.sh # account generate script
|-- sol2java.sh # development tool script for compiling solidity contract file as␣
↪java contract file
|-- replace_solc_jar.sh # a script for replacing the compiling jar package
```

Configure console

- Blockchain node and certificate configuration:
  - To copy the `ca.crt`, `sdk.crt`, and `sdk.key` files in the sdk node directory to the `conf` directory.
  - To rename the `applicationContext-sample.xml` file in the `conf` directory to the `applicationContext.xml` file. To configure the `applicationContext.xml` file, where the remark content is modified according to the blockchain node configuration. **Hint: If the channel_listen_ip(If the node version is earlier than v2.3.0, check the configuration item listen_ip) set through chain building is 127.0.0.1 or 0.0.0.0 and the channel_port is 20200, the `applicationContext.xml` configuration is not modified. **

```
<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://
↪www.springframework.org/schema/p"
           xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://
↪www.springframework.org/schema/aop"
           xmlns:context="http://www.springframework.org/schema/context"
           xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
        http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
```

(continues on next page)

(continued from previous page)

```
        <bean id="encryptType" class="org.fisco.bcos.web3j.crypto.EncryptType">
                <constructor-arg value="0"/> <!-- 0:standard 1:guomi -->
        </bean>

        <bean id="groupChannelConnectionsConfig" class="org.fisco.bcos.channel.
↪handler.GroupChannelConnectionsConfig">
                <property name="allChannelConnections">
                        <list>  <!-- each group need to configure a bean -->
                                <bean id="group1"  class="org.fisco.bcos.channel.
↪handler.ChannelConnections">
                                        <property name="groupId" value="1" /> <!--␣
↪groupID -->
                                        <property name="connectionsStr">
                                                <list>
                                                        <value>127.0.0.1:20200</
↪value>  <!-- IP:channel_port -->
                                                </list>
                                        </property>
                                </bean>
                        </list>
                </property>
        </bean>

        <bean id="channelService" class="org.fisco.bcos.channel.client.Service"␣
↪depends-on="groupChannelConnectionsConfig">
                <property name="groupId" value="1" /> <!-- to connect to the group␣
↪with ID 1 -->
                <property name="orgID" value="fisco" />
                <property name="allChannelConnections" ref=
↪"groupChannelConnectionsConfig"></property>
        </bean>

</beans>
```

Configuration detail reference here.

Important: Console configuration instructions

- If the console is configured correctly, but when it is launched on Cent0S system, the following error occurs:

  Failed to connect to the node. Please check the node status and the console configuration.

  It is because the JDK version that comes with the CentOS system is used (it will cause the console and blockchain node's authentication to fail). Please download Java 8 version or above from OpenJDK official website or Oracle official website and install (specific installation steps refer to Appendix). To launch the console after installation.

- When the console configuration file configures multiple node connections in a group, some nodes in the group may leave the group during operation. Therefore, it shows a norm which is when the console is polling, the return information may be inconsistent. It is recommended to configure a node or ensure that the configured nodes are always in the group when using the console, so that the inquired information in the group will keep consistent during the synchronization time.

Configure OSCCA-approved cryptography console

- Blockchain node and certificate configuration:
  - To copy the `ca.crt`, `sdk.crt`, and `sdk.key` files in the sdk node directory to the `conf` directory.

  - To rename the `applicationContext-sample.xml` file in the `conf` directory to the `applicationContext.xml` file. To configure the `applicationContext.xml` file, where the remark content is modified according to the blockchain node configuration. **Hint: If the channel_listen_ip(If the node version is earlier than v2.3.0, check the configuration item listen_ip) set through chain building is 127.0.0.1 or 0.0.0.0 and the channel_port is 20200, the `applicationContext.xml` configuration is not modified. **

Contract compilation tool

Console provides a special compilation contract tool that allows developers to compile Solidity contract files into Java contract files. Two steps for using the tool:

- To place the Solidity contract file in the `contracts/solidity` directory.
- Complete the task of compiling contract by running the `sol2java.sh` script (requires specifying a java package name). For example, there are `HelloWorld.sol`, `TableTest.sol`, and `Table.sol` contracts in the `contracts/solidity` directory, and we specify the package name as `org.com.fisco`. The command is as follows:

```
$ cd ~/fisco/console
$ ./sol2java.sh org.com.fisco
```

After running successfully, the directories of Java, ABI and bin will be generated in the `console/contracts/sdk` directory as shown below.

```
```bash
|-- abi # to compile the generated abi directory and to store the abi file␣
↪compiled by solidity contract
|   |-- HelloWorld.abi
|   |-- Table.abi
|   |-- TableTest.abi
|-- bin # to compile the generated bin directory and to store the bin file␣
↪compiled by solidity contract
|   |-- HelloWorld.bin
|   |-- Table.bin
|   |-- TableTest.bin
|-- java  # to store compiled package path and Java contract file
|   |-- org
|       |-- com
|           |-- fisco
|               |-- HelloWorld.java # the target Java file which is compiled␣
↪successfully
|               |-- Table.java  # the CRUD interface Java file which is compiled␣
↪successfully
|               |-- TableTest.java  # the TableTest Java file which is compiled␣
↪successfully
```
```

In the java directory, `org/com/fisco/` package path directory is generated. In the package path directory, the java contract files `HelloWorld.java`, `TableTest.java` and `Table.java` will be generated. `HelloWorld.java` and `TableTest.java` are the java contract files required by the java application.

Launch console

Start the console while the node is running:

```
$ ./start.sh
# To output the following information to indicate successful launch
================================================================================
Welcome to FISCO BCOS console(1.0.3)!
```

(continues on next page)

(continued from previous page)

```
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
 ________ _______  _______   _______   _______       ________   _______   _______   _______
|        |       \/       \ /       \ /       \     |        \ /       \ /       \ /       \
↪\
| $$$$$$$$\$$$$$|  $$$$$$|  $$$$$$|  $$$$$$\    | $$$$$$$|  $$$$$$|  $$$$$$|  $$$$$
↪$\
| $$__      | $$ | $$___\$| $$   \$| $$  | $$    | $$__/ $| $$   \$| $$  | $| $$___\
↪$$
| $$  \     | $$  \$$    \| $$      | $$  | $$    | $$    $| $$      | $$  | $$\$$   \
↪\
| $$$$$     | $$  _\$$$$$$| $$   __| $$  | $$    | $$$$$$$| $$   __| $$  | $$_\$$$$$
↪$\
| $$       _| $$_|  \__| $| $$__/  | $$__/ $$    | $$__/ $| $$__/  | $$__/ $|  \__|
↪$$
| $$      |   $$ \\$$    $$\$$    $$\$$    $$    | $$    $$\$$    $$\$$    $$\$$
↪$$
 \$$       \$$$$$$ \$$$$$$  \$$$$$$  \$$$$$$      \$$$$$$$  \$$$$$$  \$$$$$$  \$$$$$
↪$

================================================================================
```

Launch script description

To view the current console version:

```
./start.sh --version
console version: 1.0.3
```

Account using method

Console loads private key

The console provides the account generation script get_account.sh (for the script tutorial, please refer to Account Management Document. The generated account file is in the accounts directory, and the account file loaded by console must be placed in the directory.

The console startup methods are as follows:

```
./start.sh
./start.sh groupID
./start.sh groupID -pem pemName
./start.sh groupID -p12 p12Name
```

Default start

The console randomly generates an account that is started with the group number specified in the console configuration file.

```
./start.sh
```

Specify group number to start

The console randomly generates an account that is started with the group number specified on the command line.

```
./start.sh 2
```

- Note: The specified group needs to configure 'bean' in the console configuration file.

Start with PEM format private key file

- Start with the account of the specified pem file, enter the parameters: group number, -pem, and pem file path

```
./start.sh 1 -pem accounts/0xebb824a1122e587b17701ed2e512d8638dfb9c88.pem
```

Start with PKCS12 format private key file

- Start with the account of the specified p12 file, enter the parameters: group number, -p12, and p12 file path

```
./start.sh 1 -p12 accounts/0x5ef4df1b156bc9f077ee992a283c2dbb0bf045c0.p12
Enter Export Password:
```

10.2.6 Console command

help

Enter help or h to see all the commands on the console.

```
[group:1]> help
--------------------------------------------------------------------------------
↪--
addObserver                              Add an observer node.
addSealer                                Add a sealer node.
call                                     Call a contract by a function and␣
↪Parameters.
callByCNS                                Call a contract by a function and␣
↪Parameters by CNS.
deploy                                   Deploy a contract on blockchain.
deployByCNS                              Deploy a contract on blockchain by CNS.
desc                                     Description table information.
exit                                     Quit console.
getBlockHeaderByHash                     Query information about a block header by␣
↪hash.
getBlockHeaderByNumber                   Query information about a block header by␣
↪block number.
getBlockByHash                           Query information about a block by hash.
getBlockByNumber                         Query information about a block by block␣
↪number.
getBlockHashByNumber                     Query block hash by block number.
getBlockNumber                           Query the number of most recent block.
getCode                                  Query code at a given address.
getConsensusStatus                       Query consensus status.
getDeployLog                             Query the log of deployed contracts.
getGroupList                             Query group list.
getGroupPeers                            Query nodeId list for sealer and observer␣
↪nodes.
getNodeIDList                            Query nodeId list for all connected nodes.
getNodeVersion                           Query the current node version.
getNodeInfo                              Query the specified node information.
getObserverList                          Query nodeId list for observer nodes.
```

(continues on next page)

(continued from previous page)

```
getPbftView                                 Query the pbft view of node.
getPeers                                    Query peers currently connected to the␣
↪client.
getPendingTransactions                      Query pending transactions.
getPendingTxSize                            Query pending transactions size.
getSealerList                               Query nodeId list for sealer nodes.
getSyncStatus                               Query sync status.
getSystemConfigByKey                        Query a system config value by key.
setSystemConfigByKey                        Set a system config value by key.
getTotalTransactionCount                    Query total transaction count.
getTransactionByBlockHashAndIndex           Query information about a transaction by␣
↪block hash and transaction index position.
getTransactionByBlockNumberAndIndex         Query information about a transaction by␣
↪block number and transaction index position.
getTransactionByHash                        Query information about a transaction␣
↪requested by transaction hash.
getTransactionReceipt                       Query the receipt of a transaction by␣
↪transaction hash.
getTransactionByHashWithProof               Query the transaction and transaction␣
↪proof by transaction hash.
getTransactionReceiptByHashWithProof        Query the receipt and transaction receipt␣
↪proof by transaction hash.
grantCNSManager                             Grant permission for CNS by address.
grantDeployAndCreateManager                 Grant permission for deploy contract and␣
↪create user table by address.
grantNodeManager                            Grant permission for node configuration␣
↪by address.
grantSysConfigManager                       Grant permission for system configuration␣
↪by address.
grantUserTableManager                       Grant permission for user table by table␣
↪name and address.
help(h)                                     Provide help information.
listCNSManager                              Query permission information for CNS.
listDeployAndCreateManager                  Query permission information for deploy␣
↪contract and create user table.
listNodeManager                             Query permission information for node␣
↪configuration.
listSysConfigManager                        Query permission information for system␣
↪configuration.
listUserTableManager                        Query permission for user table␣
↪information.
queryCNS                                    Query CNS information by contract name␣
↪and contract version.
quit(q)                                     Quit console.
removeNode                                  Remove a node.
revokeCNSManager                            Revoke permission for CNS by address.
revokeDeployAndCreateManager                Revoke permission for deploy contract and␣
↪create user table by address.
revokeNodeManager                           Revoke permission for node configuration␣
↪by address.
revokeSysConfigManager                      Revoke permission for system␣
↪configuration by address.
revokeUserTableManager                      Revoke permission for user table by table␣
↪name and address.
listContractWritePermission                 Query the account list which have write␣
↪permission of the contract.
grantContractWritePermission                Grant the account the contract write␣
↪permission.
revokeContractWritePermission               Revoke the account the contract write␣
↪permission.
grantContractStatusManager                  Grant contract authorization to the user.
```

(continues on next page)

(continued from previous page)

```
getContractStatus                       Get the status of the contract.
listContractStatusManager               List the authorization of the contract.
grantCommitteeMember                    Grant the account committee member
revokeCommitteeMember                   Revoke the account from committee member
listCommitteeMembers                    List all committee members
grantOperator                           Grant the account operator
revokeOperator                          Revoke the operator
listOperators                           List all operators
updateThreshold                         Update the threshold
queryThreshold                          Query the threshold
updateCommitteeMemberWeight             Update the committee member weight
queryCommitteeMemberWeight              Query the committee member weight
freezeAccount                           Freeze the account.
unfreezeAccount                         Unfreeze the account.
getAccountStatus                        GetAccountStatus of the account.
freezeContract                          Freeze the contract.
unfreezeContract                        Unfreeze the contract.
switch(s)                               Switch to a specific group by group ID.
[create sql]                            Create table by sql.
[delete sql]                            Remove records by sql.
[insert sql]                            Insert records by sql.
[select sql]                            Select records by sql.
[update sql]                            Update records by sql.
--------------------------------------------------------------------------------
↪--
```

**Note: **

- help shows the meaning of each command: command and command description
- for instructions on how to use specific commands, enter the command -h or –help to view them. E.g:

```
[group:1]> getBlockByNumber -h
Query information about a block by block number.
Usage: getBlockByNumber blockNumber [boolean]
blockNumber -- Integer of a block number, from 0 to 2147483647.
boolean -- (optional) If true it returns the full transaction objects, if false␣
↪only the hashes of the transactions.
```

switch

To run switch or s to switch to the specified group. The group number is displayed in front of the command prompt.

```
[group:1]> switch 2
Switched to group 2.

[group:2]>
```

**Note: ** For the group that needs to be switched, make sure that the information of the group is configured in `applicationContext.xml` (the initial state of this configuration file only provides the group 1 configuration) in the `console/conf` directory, the configured node ID and port in the group are correct, and the node is running normally.

getBlockNumber

To run getBlockNumber to view block number.

```
[group:1]> getBlockNumber
90
```

getSealerList

To run getSealerList to view the list of consensus nodes.

```
[group:1]> getSealerList
[
    ↵
↪0c0bbd25152d40969d3d3cee3431fa28287e07cff2330df3258782d3008b876d146ddab97eab42796495bfbb281591f
↪
    ↵
↪10b3a2d4b775ec7f3c2c9e8dc97fa52beb8caab9c34d026db9b95a72ac1d1c1ad551c67c2b7fdc34177857eada75836
↪
    ↵
↪622af37b2bd29c60ae8f15d467b67c0a7fe5eb3e5c63fdc27a0ee8066707a25afa3aa0eb5a3b802d3a8e5e26de9d5af
]
```

getObserverList

To run getSealerList to view the list of observer nodes.

```
[group:1]> getObserverList
[
    ↵
↪037c255c06161711b6234b8c0960a6979ef039374ccc8b723afea2107cba3432dbbc837a714b7da20111f74d5a24e91
]
```

getNodeIDList

To run getNodeIDList to view the nodes and the list of nodeIds connected to p2p nodes.

```
[group:1]> getNodeIDList
[
    ↵
↪41285429582cbfe6eed501806391d2825894b3696f801e945176c7eb2379a1ecf03b36b027d72f480e89d15bacd4346
↪
    ↵
↪87774114e4a496c68f2482b30d221fa2f7b5278876da72f3d0a75695b81e2591c1939fc0d3fadb15cc359c997bafc9e
↪
    ↵
↪29c34347a190c1ec0c4507c6eed6a5bcd4d7a8f9f54ef26da616e81185c0af11a8cea4eacb74cf6f61820292b24bc5d
↪
    ↵
↪d5b3a9782c6aca271c9642aea391415d8b258e3a6d92082e59cc5b813ca123745440792ae0b29f4962df568f8ad58b7
]
```

getPbftView

To run getPbftView to view the pbft viewgraph.

```
[group:1]> getPbftView
2730
```

getConsensusStatus

To run getConsensusStatus to view the consensus status.

```
[group:1]> getConsensusStatus
[
    {
  "id": 1,
  "jsonrpc": "2.0",
  "result": [
    {
      "accountType": 1,
      "allowFutureBlocks": true,
      "cfgErr": false,
      "connectedNodes": 3,
      "consensusedBlockNumber": 38207,
      "currentView": 54477,
      "groupId": 1,
      "highestblockHash":
↪"0x19a16e8833e671aa11431de589c866a6442ca6c8548ba40a44f50889cd785069",
      "highestblockNumber": 38206,
      "leaderFailed": false,
      "max_faulty_leader": 1,
      "nodeId":
↪"f72648fe165da17a889bece08ca0e57862cb979c4e3661d6a77bcc2de85cb766af5d299fec8a4337eedd142dca026a
↪",
      "nodeNum": 4,
      "node_index": 3,
      "omitEmptyBlock": true,
      "protocolId": 65544,
      "sealer.0":
↪"6a99f357ecf8a001e03b68aba66f68398ee08f3ce0f0147e777ec77995369aac470b8c9f0f85f91ebb58a98475764b
↪",
      "sealer.1":
↪"8a453f1328c80b908b2d02ba25adca6341b16b16846d84f903c4f4912728c6aae1050ce4f24cd9c13e010ce922d339
↪",
      "sealer.2":
↪"ed483837e73ee1b56073b178f5ac0896fa328fc0ed418ae3e268d9e9109721421ec48d68f28d6525642868b40dd265
↪",
      "sealer.3":
↪"f72648fe165da17a889bece08ca0e57862cb979c4e3661d6a77bcc2de85cb766af5d299fec8a4337eedd142dca026a
↪",
      "toView": 54477
    },
    [
      {
        "nodeId":
↪"6a99f357ecf8a001e03b68aba66f68398ee08f3ce0f0147e777ec77995369aac470b8c9f0f85f91ebb58a98475764b
↪",
        "view": 54474
      },
      {
        "nodeId":
↪"8a453f1328c80b908b2d02ba25adca6341b16b16846d84f903c4f4912728c6aae1050ce4f24cd9c13e010ce922d339
↪",
        "view": 54475
      },
      {
        "nodeId":
↪"ed483837e73ee1b56073b178f5ac0896fa328fc0ed418ae3e268d9e9109721421ec48d68f28d6525642868b40dd265
↪",
        "view": 54476
```

(continues on next page)

(continued from previous page)

```
      },
      {
        "nodeId":
↪"f72648fe165da17a889bece08ca0e57862cb979c4e3661d6a77bcc2de85cb766af5d299fec8a4337eedd142dca026a
↪",
        "view": 54477
      }
    ]
  ]
}
]
```

getSyncStatus

To run getSyncStatus to view the synchronization status.

```
[group:1]> getSyncStatus
{
    "blockNumber":5,
    "genesisHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
    "isSyncing":false,
    "latestHash":
↪"0xb99703130e24702d3b580111b0cf4e39ff60ac530561dd9eb0678d03d7acce1d",
    "nodeId":
↪"cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bc
↪",
    "peers":[
        {
            "blockNumber":5,
            "genesisHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
            "latestHash":
↪"0xb99703130e24702d3b580111b0cf4e39ff60ac530561dd9eb0678d03d7acce1d",
            "nodeId":
↪"0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bc
↪"
        },
        {
            "blockNumber":5,
            "genesisHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
            "latestHash":
↪"0xb99703130e24702d3b580111b0cf4e39ff60ac530561dd9eb0678d03d7acce1d",
            "nodeId":
↪"2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966
↪"
        },
        {
            "blockNumber":5,
            "genesisHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
            "latestHash":
↪"0xb99703130e24702d3b580111b0cf4e39ff60ac530561dd9eb0678d03d7acce1d",
            "nodeId":
↪"ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54
↪"
        }
    ],
    "protocolId":265,
```

(continues on next page)

(continued from previous page)

```
    "txPoolSize":"0"
}
```

getNodeVersion

To run getNodeVersion to view the node version.

```
[group:1]> getNodeVersion
{
    "Build Time":"20200619 06:32:10",
    "Build Type":"Linux/clang/Release",
    "Chain Id":"1",
    "FISCO-BCOS Version":"2.5.0",
    "Git Branch":"HEAD",
    "Git Commit Hash":"72c6d770e5cf0f4197162d0e26005ec03d30fcfe",
    "Supported Version":"2.5.0"
}
```

getPeers

To run getPeers to view the peers of node.

```
[group:1]> getPeers
[
        {
                "IPAndPort":"127.0.0.1:50723",
                "nodeId":
↪"8718579e9a6fee647b3d7404d59d66749862aeddef22e6b5abaafe1af6fc128fc33ed5a9a105abddab51e12004c6bf
↪",
                "Topic":[

                ]
        },
        {
                "IPAndPort":"127.0.0.1:50719",
                "nodeId":
↪"697e81e512cffc55fc9c506104fb888a9ecf4e29eabfef6bb334b0ebb6fc4ef8fab60eb614a0f2be178d0b5993464c
↪",
                "Topic":[

                ]
        },
        {
                "IPAndPort":"127.0.0.1:30304",
                "nodeId":
↪"8fc9661baa057034f10efacfd8be3b7984e2f2e902f83c5c4e0e8a60804341426ace51492ffae087d96c0b968bd5e9
↪",
                "Topic":[

                ]
        }
]
```

getGroupPeers

To run getGroupPeers to view the list of consensus and observer node of the group where the node is located.

```
[group:1]> getGroupPeers
[
    "cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bcc
    "ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54b
    "0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bca
    "2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966a
]
```

getGroupList

To run getGroupList to view the list of group:

```
[group:1]> getGroupList
[1]
```

getBlockHeaderByHash

Run getBlockHeaderByHash to query the block header information based on the block hash.

parameter:

- Block hash: the hash value of the block starting with 0x
- Signature list flag: The default is false, that is, the block signature list information is not displayed in the block header information, and if set to true, the block signature list is displayed.

```
[group:1]> getBlockHeaderByHash
0x99576e7567d258bd6426ddaf953ec0c953778b2f09a078423103c6555aa4362d
{
    "dbHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0x99576e7567d258bd6426ddaf953ec0c953778b2f09a078423103c6555aa4362d",
    "logsBloom":
"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
",
    "number":1,
    "parentHash":
"0x4f6394763c33c1709e5a72b202ad4d7a3b8152de3dc698cef6f675ecdaf20a3b",
    "receiptsRoot":
"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
    "sealer":"0x2",
    "sealerList":[

"11e1be251ca08bb44f36fdeedfaeca40894ff80dfd80084607a75509edeaf2a9c6fee914f1e9efda571611cf4575a1
",

"78a313b426c3de3267d72b53c044fa9fe70c2a27a00af7fea4a549a7d65210ed90512fc92b6194c14766366d434235
",

"95b7ff064f91de76598f90bc059bec1834f0d9eeb0d05e1086d49af1f9c2f321062d011ee8b0df7644bd54c4f9ca3d8
",
```

(continues on next page)

(continued from previous page)

```
↪"b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a9958
↪"
    ],
    "stateRoot":"0x0000000000000000000000000000000000000000000000000000000000000000
↪",
    "timestamp":"0x173ad8703d6",
    "transactionsRoot":
↪"0xb563f70188512a085b5607cac0c35480336a566de736c83410a062c9acc785ad"
}
```

getBlockHeaderByNumber

Run getBlockHeaderByNumber to query the block header information according to the block height. parameter:

- Block height
- Signature list flag: The default is false, that is, the block signature list information is not displayed in the block header information, and if set to true, the block signature list is displayed.

```
[group:1]> getBlockHeaderByNumber 1 true
{
    "dbHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0x99576e7567d258bd6426ddaf953ec0c953778b2f09a078423103c6555aa4362d",
    "logsBloom":
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":1,
    "parentHash":
↪"0x4f6394763c33c1709e5a72b202ad4d7a3b8152de3dc698cef6f675ecdaf20a3b",
    "receiptsRoot":
↪"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
    "sealer":"0x2",
    "sealerList":[

↪"11e1be251ca08bb44f36fdeedfaeca40894ff80dfd80084607a75509edeaf2a9c6fee914f1e9efda571611cf4575a1
↪",

↪"78a313b426c3de3267d72b53c044fa9fe70c2a27a00af7fea4a549a7d65210ed90512fc92b6194c14766366d434235
↪",

↪"95b7ff064f91de76598f90bc059bec1834f0d9eeb0d05e1086d49af1f9c2f321062d011ee8b0df7644bd54c4f9ca3d
↪",

↪"b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a9958
↪"
    ],
    "signatureList":[
        {
            "index":"0x3",
            "signature":
↪"0xb5b41e49c0b2bf758322ecb5c86dc3a3a0f9b98891b5bbf50c8613a241f05f595ce40d0bb212b6faa32e98546754
↪"
        },
        {
```

(continues on next page)

(continued from previous page)

```
            "index":"0x0",
            "signature":
↪"0x411cb93f816549eba82c3bf8c03fa637036dcdee65667b541d0da06a6eaea80d16e6ca52bf1b08f77b59a834bffb
↪"
        },
        {
            "index":"0x1",
            "signature":
↪"0xea3c27c2a1486c7942c41c4dc8f15fbf9a668aff2ca40f00701d73fa659a14317d45d74372d69d43ced8e81f789e
↪"
        }
    ],
    "stateRoot":"0x0000000000000000000000000000000000000000000000000000000000000000
↪",
    "timestamp":"0x173ad8703d6",
    "transactionsRoot":
↪"0xb563f70188512a085b5607cac0c35480336a566de736c83410a062c9acc785ad"
}
```

getBlockByHash

To run getBlockByHash to view block information according to the block hash. Parameter:

- Block hash: The hash starting with 0x.
- Transaction sign: to set it false by default, the transaction in the block only displays the hash. To set it true, it displays the transaction specific information.

```
[group:1]> getBlockByHash␣
↪0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855
{
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855",
    "logsBloom":
↪"0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":"0x1",
    "parentHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
    "sealer":"0x0",
    "sealerList":[

↪"0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bc
↪",

↪"2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966
↪",

↪"cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bc
↪",

↪"ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54
↪"
    ],
    "stateRoot":"0x9711819153f7397ec66a78b02624f70a343b49c60bc2f21a77b977b0ed91cef9
↪",
```

(continues on next page)

(continued from previous page)

```
    "timestamp":"0x1692f119c84",
    "transactions":[
        "0xa14638d47cc679cf6eeb7f36a6d2a30ea56cb8dcf0938719ff45023a7a8edb5d"
    ],
    "transactionsRoot":
↪"0x516787f85980a86fd04b0e9ce82a1a75950db866e8cdf543c2cae3e4a51d91b7"
}
[group:1]> getBlockByHash␣
↪0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855 true
{
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855",
    "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":"0x1",
    "parentHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
    "sealer":"0x0",
    "sealerList":[

↪"0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bc
↪",

↪"2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966
↪",

↪"cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bc
↪",

↪"ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54
↪"
    ],
    "stateRoot":"0x9711819153f7397ec66a78b02624f70a343b49c60bc2f21a77b977b0ed91cef9
↪",
    "timestamp":"0x1692f119c84",
    "transactions":[
        {
            "blockHash":
↪"0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855",
            "blockNumber":"0x1",
            "from":"0x7234c32327795e4e612164e3442cfae0d445b9ad",
            "gas":"0x1c9c380",
            "gasPrice":"0x1",
            "hash":
↪"0xa14638d47cc679cf6eeb7f36a6d2a30ea56cb8dcf0938719ff45023a7a8edb5d",
            "input":
↪"0x6080604052348015610010576000080fd5b506040805190810160405280600d81526020017f48656c6c6f2c20576f
↪",
            "nonce":
↪"0x3443a1391c9c29f751e8350304efb310850b8afbaa7738f5e89ddfce79b1d6",
            "to":null,
            "transactionIndex":"0x0",
            "value":"0x0"
        }
    ],
    "transactionsRoot":
↪"0x516787f85980a86fd04b0e9ce82a1a75950db866e8cdf543c2cae3e4a51d91b7"
```

(continues on next page)

(continued from previous page)

```
}
```

getBlockByNumber

To run getBlockByNumber to view block information according to the block number. Parameter:

- Block number: decimal integer.
- Transaction sign: to set it false by default, the transaction in the block only displays the hash. To set it true, it displays the transaction specific information.

```
[group:1]> getBlockByNumber 1
{
    "extraData":[

    ],
    "gasLimit":"0x0",
    "gasUsed":"0x0",
    "hash":"0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855",
    "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number":"0x1",
    "parentHash":
↪"0xeccad5274949b9d25996f7a96b89c0ac5c099eb9b72cc00d65bc6ef09f7bd10b",
    "sealer":"0x0",
    "sealerList":[

↪"0471101bcf033cd9e0cbd6eef76c144e6eff90a7a0b1847b5976f8ba32b2516c0528338060a4599fc5e3bafee188bc
↪",

↪"2b08375e6f876241b2a1d495cd560bd8e43265f57dc9ed07254616ea88e371dfa6d40d9a702eadfd5e025180f9d966
↪",

↪"cf93054cf524f51c9fe4e9a76a50218aaa7a2ca6e58f6f5634f9c2884d2e972486c7fe1d244d4b49c6148c1cb524bc
↪",

↪"ed1c85b815164b31e895d3f4fc0b6e3f0a0622561ec58a10cc8f3757a73621292d88072bf853ac52f0a9a9bbb10a54
↪"
    ],
    "stateRoot":"0x9711819153f7397ec66a78b02624f70a343b49c60bc2f21a77b977b0ed91cef9
↪",
    "timestamp":"0x1692f119c84",
    "transactions":[
        "0xa14638d47cc679cf6eeb7f36a6d2a30ea56cb8dcf0938719ff45023a7a8edb5d"
    ],
    "transactionsRoot":
↪"0x516787f85980a86fd04b0e9ce82a1a75950db866e8cdf543c2cae3e4a51d91b7"
}
```

getBlockHashByNumber

To run getBlockHashByNumber to get hash through block number Parameter:

- Block number: decimal integer.

```
[group:1]> getBlockHashByNumber 1
0xf6afbcc3ec9eb4ac2c2829c2607e95ea0fa1be914ca1157436b2d3c5f1842855
```

getTransactionByHash

To run getTransactionByHash to check the transaction information through transaction hash. Parameter:

- Transaction hash: the transaction hash starting with 0x.

```
[group:1]> getTransactionByHash␣
↪0xed82e2cda98db8614677aba1fa8a795820bd7f68a5919a2f85018ba8c10952ac
{
        "blockHash":
↪"0x77e5b6d799edabaeae654ac5cea9baacd6f8e7ace33531d40c7ed65192de1f02",
        "blockNumber":"0x5a",
        "from":"0x7a5b31b49c6e944e9e1768785b1bc9a96cea0c17",
        "gas":"0x1c9c380",
        "gasPrice":"0x1",
        "hash":"0xed82e2cda98db8614677aba1fa8a795820bd7f68a5919a2f85018ba8c10952ac
↪",
        "input":
↪"0x10009562616c6963650000000000000000000000000000000000000000000000000000000000000006a6f726500000000000000
↪",
        "nonce":"0x18711fff2ea68dc8b8dce4a3d3845c62a0630766a448bd9453a9127efe6f9e5
↪",
        "to":"0x738eedd873bb9722173194ab990c5b9a6c0beb25",
        "transactionIndex":"0x0",
        "value":"0x0"
}
```

getTransactionByBlockHashAndIndex

To run getTransactionByBlockHashAndIndex to inquire transaction information through block hash and transaction index. Parameter:

- Block hash: the transaction hash starting with 0x.
- Transaction index: decimal integer.

```
[group:1]> getTransactionByBlockHashAndIndex␣
↪0x77e5b6d799edabaeae654ac5cea9baacd6f8e7ace33531d40c7ed65192de1f02 0
{
        "blockHash":
↪"0x77e5b6d799edabaeae654ac5cea9baacd6f8e7ace33531d40c7ed65192de1f02",
        "blockNumber":"0x5a",
        "from":"0x7a5b31b49c6e944e9e1768785b1bc9a96cea0c17",
        "gas":"0x1c9c380",
        "gasPrice":"0x1",
        "hash":"0xed82e2cda98db8614677aba1fa8a795820bd7f68a5919a2f85018ba8c10952ac
↪",
        "input":
↪"0x10009562616c6963650000000000000000000000000000000000000000000000000000000000000006a6f726500000000000000
↪",
        "nonce":"0x18711fff2ea68dc8b8dce4a3d3845c62a0630766a448bd9453a9127efe6f9e5
↪",
        "to":"0x738eedd873bb9722173194ab990c5b9a6c0beb25",
        "transactionIndex":"0x0",
        "value":"0x0"
}
```

getTransactionByBlockNumberAndIndex

To run getTransactionByBlockNumberAndIndex to inquire transaction information through block number and transaction index. Parameter:

- Block number: decimal integer.
- Transaction index: decimal integer.

```
[group:1]> getTransactionByBlockNumberAndIndex 2 0
{
        "blockHash":
↪"0x77e5b6d799edabaeae654ac5cea9baacd6f8e7ace33531d40c7ed65192de1f02",
        "blockNumber":"0x5a",
        "from":"0x7a5b31b49c6e944e9e1768785b1bc9a96cea0c17",
        "gas":"0x1c9c380",
        "gasPrice":"0x1",
        "hash":"0xed82e2cda98db8614677aba1fa8a795820bd7f68a5919a2f85018ba8c10952ac
↪",
        "input":
↪"0x10009562616c6963650000000000000000000000000000000000000000000000000000006a6f726500000000000000
↪",
        "nonce":"0x18711fff2ea68dc8b8dce4a3d3845c62a0630766a448bd9453a9127efe6f9e5
↪",
        "to":"0x738eedd873bb9722173194ab990c5b9a6c0beb25",
        "transactionIndex":"0x0",
        "value":"0x0"
}
```

getTransactionReceipt

To run getTransactionReceipt to inquire transaction receipt through transaction hash. Parameter:

- Transaction hash: the transaction hash starting with 0x.
- contract name: Optional. The contract name generated by transaction receipt. To use this parameter can parse and output the event log in the transaction receipt.
- event name: optional. Event Name. To specify this parameter to output the specified event log information.
- event index number: optional. Event index. To specify this parameter to output the event log information of the specified event index location.

```
[group:1]> getTransactionReceipt␣
↪0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1
{
    "blockHash":"0x68a1f47ca465acc89edbc24115d1b435cb39fa0def53e8d0ad8090cf1827cafd
↪",
    "blockNumber":"0x5",
    "contractAddress":"0x0000000000000000000000000000000000000000",
    "from":"0xc44e7a8a4ae20d6afaa43221c6120b5e1e9f9a72",
    "gasUsed":"0x8be5",
    "logs":[
        {
            "address":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
            "data":
↪"0x0000000000000000000000000000000000000000000000000000000000000001",
            "topics":[
                "0x66f7705280112a4d1145399e0414adc43a2d6974b487710f417edcf7d4a39d71
↪"
            ]
        }
    ],
    "logsBloom":
↪"0x000000000000000000000000000000000000000000000000000000000000000000004000000000000000000000000
↪",
    "output":"0x0000000000000000000000000000000000000000000000000000000000000001",
    "status":"0x0",
```

(continues on next page)

(continued from previous page)

```
    "to":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
    "transactionHash":
↪"0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1",
    "transactionIndex":"0x0"
}

[group:1]> getTransactionReceipt␣
↪0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1 TableTest
{
    "blockHash":"0x68a1f47ca465acc89edbc24115d1b435cb39fa0def53e8d0ad8090cf1827cafd
↪",
    "blockNumber":"0x5",
    "contractAddress":"0x0000000000000000000000000000000000000000",
    "from":"0xc44e7a8a4ae20d6afaa43221c6120b5e1e9f9a72",
    "gasUsed":"0x8be5",
    "logs":[
        {
            "address":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
            "data":
↪"0x0000000000000000000000000000000000000000000000000000000000000001",
            "topics":[
                "0x66f7705280112a4d1145399e0414adc43a2d6974b487710f417edcf7d4a39d71
↪"
            ]
        }
    ],
    "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000004000000000000000000000000
↪",
    "output":"0x0000000000000000000000000000000000000000000000000000000000000001",
    "status":"0x0",
    "to":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
    "transactionHash":
↪"0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1",
    "transactionIndex":"0x0"
}
---------------------------------------------------------------------------------
↪----------
Event logs
---------------------------------------------------------------------------------
↪----------
InsertResult index: 0
count = 1
---------------------------------------------------------------------------------
↪----------

[group:1]> getTransactionReceipt␣
↪0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1 TableTest␣
↪InsertResult
{
    "blockHash":"0x68a1f47ca465acc89edbc24115d1b435cb39fa0def53e8d0ad8090cf1827cafd
↪",
    "blockNumber":"0x5",
    "contractAddress":"0x0000000000000000000000000000000000000000",
    "from":"0xc44e7a8a4ae20d6afaa43221c6120b5e1e9f9a72",
    "gasUsed":"0x8be5",
    "logs":[
        {
            "address":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
            "data":
↪"0x0000000000000000000000000000000000000000000000000000000000000001",
```

(continues on next page)

(continued from previous page)

```
            "topics":[
                "0x66f7705280112a4d1145399e0414adc43a2d6974b487710f417edcf7d4a39d71
↪"
            ]
        }
    ],
    "logsBloom":
↪"0x000000000000000000000000000000000000000000000000000000000000000000004000000000000000000000000
↪",
    "output":"0x0000000000000000000000000000000000000000000000000000000000000001",
    "status":"0x0",
    "to":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
    "transactionHash":
↪"0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1",
    "transactionIndex":"0x0"
}
---------------------------------------------------------------------------------
↪----------
Event logs
---------------------------------------------------------------------------------
↪----------
InsertResult index: 0
count = 1
---------------------------------------------------------------------------------
↪----------

[group:1]> getTransactionReceipt␣
↪0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1 TableTest␣
↪InsertResult 0
{
    "blockHash":"0x68a1f47ca465acc89edbc24115d1b435cb39fa0def53e8d0ad8090cf1827cafd
↪",
    "blockNumber":"0x5",
    "contractAddress":"0x0000000000000000000000000000000000000000",
    "from":"0xc44e7a8a4ae20d6afaa43221c6120b5e1e9f9a72",
    "gasUsed":"0x8be5",
    "logs":[
        {
            "address":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
            "data":
↪"0x0000000000000000000000000000000000000000000000000000000000000001",
            "topics":[
                "0x66f7705280112a4d1145399e0414adc43a2d6974b487710f417edcf7d4a39d71
↪"
            ]
        }
    ],
    "logsBloom":
↪"0x000000000000000000000000000000000000000000000000000000000000000000004000000000000000000000000
↪",
    "output":"0x0000000000000000000000000000000000000000000000000000000000000001",
    "status":"0x0",
    "to":"0xd653139b9abffc3fe07573e7bacdfd35210b5576",
    "transactionHash":
↪"0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1",
    "transactionIndex":"0x0"
}
---------------------------------------------------------------------------------
↪----------
Event logs
---------------------------------------------------------------------------------
↪----------
```

(continues on next page)

(continued from previous page)

```
InsertResult index: 0
count = 1
--------------------------------------------------------------------------------
↪----------
```

getPendingTransactions

To run getPendingTransactions to inquire the transactions waiting to be processed.

```
[group:1]> getPendingTransactions
[]
```

getPendingTxSize

To run getPendingTxSize to inquire the number of transactions waiting to be processed.

```
[group:1]> getPendingTxSize
0
```

getCode

To run getCode to inquire contract code according contract address. Parameter:

- Contract address: The contract address starting with 0x(To deploy contract can get contract address).

```
[group:1]> getCode 0x97b8c19598fd781aaeb53a1957ef9c8acc59f705
0x60606040526000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806
```

getTotalTransactionCount

To run getTotalTransactionCount to inquire the current block number and the total number of transaction.

```
[group:1]> getTotalTransactionCount
{
        "blockNumber":1,
        "txSum":1,
        "failedTxSum":0
}
```

deploy

To run deploy to deploy contract. (By default, HelloWorld contract and TableTest.sol are provided for example) Parameter:

- Contract name: deployment contract name (can be suffixed with .sol). It can name as either HelloWorld or HelloWorld.sol.

```
# To deploy HelloWorld contract
[group:1]> deploy HelloWorld.sol
contract address:0xb3c223fc0bf6646959f254ac4e4a7e355b50a344

# To deploy TableTest contract
[group:1]> deploy TableTest.sol
contract address:0x3554a56ea2905f366c345bd44fa374757fb4696a
```

Note:

- For deploying a user-written contract, we just need to place the solidity contract file in the `contracts/solidity/` directory of the console root, and then deploy it. Press the tab key to search for the contract name in the `contracts/solidity` directory.
- If the contract need to be deployed refers to other contracts or libraries, the reference format is `import "./XXX.sol";`. The related contracts and libraries are placed in the `contracts/solidity/` directory.
- If contract references the library library, the name of library file must start with `Lib` string to distinguish between the normal contract and the library file. Library files cannot be deployed and called separately.
- **Because FISCO BCOS has removed the transfer payment logic of Ethereum, the solidity contract does not support using `payable` keyword. This keyword will cause the Java contract file converted by solidity contract to fail at compilation. **

getDeployLog

Run getDeployLog to query the log information of the contract deployed by current console in the group. The log information includes the time of deployment contract, the group ID, the contract name, and the contract address. parameter:

- Log number: optional. To return the latest log information according to the expected value entered. When the actual number is less than the expected value, it returns by the actual number. When the expected value is not given, it returns by the latest 20 log information by default.

```
[group:1]> getDeployLog 2

2019-05-26 08:37:03  [group:1]  HelloWorld            ␣
↪0xc0ce097a5757e2b6e189aa70c7d55770ace47767
2019-05-26 08:37:45  [group:1]  TableTest             ␣
↪0xd653139b9abffc3fe07573e7bacdfd35210b5576

[group:1]> getDeployLog 1

2019-05-26 08:37:45  [group:1]  TableTest             ␣
↪0xd653139b9abffc3fe07573e7bacdfd35210b5576
```

Note: If you want to see all the deployment contract log information, please check the `deploylog.txt` file in the `console` directory. The file only stores the log records of the last 10,000 deployment contracts.

call

To run call to call contract. Parameter:

- Contract name: the contract name of the deployment (can be suffixed with .sol).
- Contract address: the address obtained by the deployment contract. The contract address can omit the prefix 0. For example, 0x000ac78 can be abbreviated as 0xac78.
- Contract interface name: the called interface name.
- Parameter: determined by contract interface parameters.

**Parameters are separated by spaces. The string and byte type parameters need to be enclosed in double quotes; array parameters need to be enclosed in brackets, such as [1,2,3]; array is a string or byte type and needs to be enclosed in double quotation marks, such as [“alice”, “bob”]. Note that there are no spaces in the array parameters; boolean types are true or false. **

```
```text
# To call the get interface of HelloWorld to get the name string
[group:1]> call HelloWorld.sol 0xc0ce097a5757e2b6e189aa70c7d55770ace47767 get
```

(continues on next page)
```

(continued from previous page)

```
Hello, World!

# To call the set interface of HelloWorld to set the name string
[group:1]> call HelloWorld.sol 0xc0ce097a5757e2b6e189aa70c7d55770ace47767 set
↪"Hello, FISCO BCOS"
transaction hash:0xa7c7d5ef8d9205ce1b228be1fe90f8ad70eeb6a5d93d3f526f30d8f431cb1e70

# To call the get interface of HelloWorld to get the name string for checking␣
↪whether the settings take effect
[group:1]> call HelloWorld.sol 0xc0ce097a5757e2b6e189aa70c7d55770ace47767 get
Hello, FISCO BCOS

# To call the insert interface of TableTest to insert the record, the fields are␣
↪name, item_id, item_name
[group:1]> call TableTest.sol 0xd653139b9abffc3fe07573e7bacdfd35210b5576 insert
↪"fruit" 1 "apple"
transaction hash:0x6393c74681f14ca3972575188c2d2c60d7f3fb08623315dbf6820fc9dcc119c1
----------------------------------------------------------------------------------
↪----------
Event logs
----------------------------------------------------------------------------------
↪----------
InsertResult index: 0
count = 1
----------------------------------------------------------------------------------
↪----------

# To call TableTest's select interface to inquiry records
[group:1]> call TableTest.sol 0xd653139b9abffc3fe07573e7bacdfd35210b5576 select
↪"fruit"
[[fruit], [1], [apple]]
```

Note: TableTest.sol contract codeReference here。

deployByCNS

Run deployByCNS and deploy the contract with CNS. Contracts deployed with CNS can be called directly with the contract name.

Parameter:

- Contract name: deployable contract name.
- Contract version number: deployable contract version number(the length cannot exceed 40).

```
# To deploy HellowWorld contract 1.0 version
[group:1]> deployByCNS HelloWorld.sol 1.0
contract address:0x3554a56ea2905f366c345bd44fa374757fb4696a

# To deploy HellowWorld contract 2.0 version
[group:1]> deployByCNS HelloWorld.sol 2.0
contract address:0x07625453fb4a6459cbf14f5aa4d574cae0f17d92

# To deploy TableTest contract
[group:1]> deployByCNS TableTest.sol 1.0
contract address:0x0b33d383e8e93c7c8083963a4ac4a58b214684a8
```

Note:

- For deploying the contracts compiled by users only needs to place the solidity contract file in the `contracts/solidity/` directory of the console root and to deploy it. Press tab key to search for the contract name in the `contracts/solidity/` directory.

- If the contract to be deployed references other contracts or libraries, the reference format is `import "./XXX.sol";`. The related contract and library are placed in the `contracts/solidity/` directory.
- **Because FISCO BCOS has removed the transfer payment logic of Ethereum, the solidity contract does not support using `payable` keyword. This keyword will cause the Java contract file converted by solidity contract to fail at compilation. **

queryCNS

Run queryCNS and query the CNS table record information (the mapping of contract name and contract address) according to the contract name and contract version number (optional parameter).

Parameter:

- Contract name: deployable contract name.
- Contract version number: (optional) deployable contract version number.

```
[group:1]> queryCNS HelloWorld.sol
---------------------------------------------------------------------------------
↪----------
|                   version                   |                   address        
↪          |
|                     1.0                     |
↪0x3554a56ea2905f366c345bd44fa374757fb4696a   |
---------------------------------------------------------------------------------
↪----------

[group:1]> queryCNS HelloWorld 1.0
---------------------------------------------------------------------------------
↪----------
|                   version                   |                   address        
↪          |
|                     1.0                     |
↪0x3554a56ea2905f366c345bd44fa374757fb4696a   |
---------------------------------------------------------------------------------
↪----------
```

callByCNS

To run deployByCNS and deploy the contract with CNS. Parameter:

- Contract name and contract version number: The contract name and contract version number are separated by colon, such as `HelloWorld:1.0` or `HelloWorld.sol:1.0`. When the contract version number is omitted like `HelloWorld` or `HelloWorld.sol`, the latest version of the contract is called.
- Contract interface name: The called contract interface name.
- Parameter: is determined by the parameter of contract interface. The parameters are separated by spaces, where the string and byte type parameters need to be enclosed in double quotation marks; the array parameters need to be enclosed in brackets, such as [1, 2, 3]. The array is a string or byte type with double quotation marks such as ["alice", "bob"]; the boolean type is true or false.

```
# To call the HelloWorld contract 1.0 version to set the name string by the set
↪interface
[group:1]> callByCNS HelloWorld:1.0 set "Hello,CNS"
transaction hash:0x80bb37cc8de2e25f6a1cdcb6b4a01ab5b5628082f8da4c48ef1bbc1fb1d28b2d

# To call the HelloWorld contract 2.0 version to set the name string by the set
↪interface
[group:1]> callByCNS HelloWorld:2.0 set "Hello,CNS2"
transaction hash:0x43000d14040f0c67ac080d0179b9499b6885d4a1495d3cfd1a79ffb5f2945f64
```

(continues on next page)

(continued from previous page)

```
# To call the HelloWorld contract 1.0 version to get the name string by the get␣
↪interface
[group:1]> callByCNS HelloWorld:1.0 get
Hello,CNS

# To call the HelloWorld contract 2.0 version to get the name string by the get␣
↪interface
[group:1]> callByCNS HelloWorld get
Hello,CNS2
```

addSealer

To run addSealer to add the node as a consensus node. Parameter:

- node's nodeId

```
[group:1]> addSealer␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
{
        "code":0,
        "msg":"success"
}
```

addObserver

To run addObserver to add the node as an observed node. Parameter:

- node's nodeId

```
[group:1]> addObserver␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
{
        "code":0,
        "msg":"success"
}
```

removeNode

To run removeNode to exit the node. The exit node can be added as a consensus node by the addSealer command or can be added as an observation node by the addObserver command. Parameter:

- node's nodeId

```
[group:1]> removeNode␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
{
        "code":0,
        "msg":"success"
}
```

setSystemConfigByKey

To run setSystemConfigByKey to set the system configuration in key-value pairs. The currently system configuration supports `tx_count_limit`, `tx_gas_limit`, `rpbft_epoch_sealer_num` and `rpbft_epoch_block_num`. The key name of these two configuration can be complemented by the tab key:

- tx_count_limit: block maximum number of packaged transactions
- tx_gas_limit: The maximum number of gas allowed to be consumed
- rpbft_epoch_sealer_num: rPBFT system configuration, the number of consensus nodes selected in a consensus epoch
- rpbft_epoch_block_num: rPBFT system configuration, number of blocks generated in one consensus epoch
- consensus_timeout: During the PBFT consensus process, the timeout period for each block execution, the default is 3s, the unit is seconds

Parameters:

- key
- value

```
[group:1]> setSystemConfigByKey tx_count_limit 100
{
        "code":0,
        "msg":"success"
}
```

getSystemConfigByKey

To run getSystemConfigByKe to inquire the value of the system configuration according to the key. Parameter:

- key

```
[group:1]> getSystemConfigByKey tx_count_limit
100
```

grantPermissionManager

Run grantPermissionManager to grant the account's chain administrator privileges. parameter:

- account address

```
[group:1]> grantPermissionManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listPermissionManager

To run listPermissionManager to inquire the list of permission records with administrative privileges.

```
[group:1]> listPermissionManager
-------------------------------------------------------------------------------------------
----------
|                    address                    |                 enable_num
|
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                      2
|
-------------------------------------------------------------------------------------------
----------
```

revokePermissionManager

To run revokePermissionManager to revoke the permission management of the external account address. parameter:

- account address

```
[group:1]> revokePermissionManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantUserTableManager

Run grantUserTableManager to grant the account to write to the user table.

parameter:

- table name
- account address

```
[group:1]> grantUserTableManager t_test 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listUserTableManager

Run listUserTableManager to query the account's table that has writing permission to the user table.

parameter:

- table name

```
[group:1]> listUserTableManager t_test
-------------------------------------------------------------------------------------------
↪----------
|                    address                    |                 enable_num          ␣
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                      2              ␣
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

revokeUserTableManager

Run revokeUserTableManager to revoke the account's writing permission from the user table.

parameter:

- table name
- account address

```
[group:1]> revokeUserTableManager t_test 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantDeployAndCreateManager

Run grantDeployAndCreateManager to grant the account's permission of deployment contract and user table creation.

parameter:

- account address

```
[group:1]> grantDeployAndCreateManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listDeployAndCreateManager

Run listDeployAndCreateManager to query the account's permission of deployment contract and user table creation.

```
[group:1]> listDeployAndCreateManager
-------------------------------------------------------------------------------------
↪----------
|                    address                    |                 enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                      2
↪          |
-------------------------------------------------------------------------------------
↪----------
```

revokeDeployAndCreateManager

Run revokeDeployAndCreateManager to revoke the account's permission of deployment contract and user table creation.

parameter:

- account address

```
[group:1]> revokeDeployAndCreateManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantNodeManager

Run grantNodeManager to grant the account's node management permission.

parameter:

- account address

```
[group:1]> grantNodeManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listNodeManager

Run the listNodeManager to query the list of accounts that have node management.

```
[group:1]> listNodeManager
-------------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                       2
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

revokeNodeManager

Run revokeNodeManager to revoke the account's node management permission.

parameter:

- account address

```
[group:1]> revokeNodeManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantCNSManager

Run grantCNSManager to grant the account's permission of using CNS. parameter:

- account address

```
[group:1]> grantCNSManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listCNSManager

Run listCNSManager to query the list of accounts that have CNS.

```
[group:1]> listCNSManager
-------------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                       2
↪          |
-------------------------------------------------------------------------------------------
↪----------
```

revokeCNSManager

Run revokeCNSManager to revoke the account's permission of using CNS. parameter:

- account address

```
[group:1]> revokeCNSManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantSysConfigManager

Run grantSysConfigManager to grant the account's permission of modifying system parameter. parameter:

- account address

```
[group:1]> grantSysConfigManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

listSysConfigManager

Run listSysConfigManager to query the list of accounts that have modified system parameters.

```
[group:1]> listSysConfigManager
-------------------------------------------------------------------------------------
↪----------
|                    address                    |                enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                     2
↪          |
-------------------------------------------------------------------------------------
↪----------
```

revokeSysConfigManager

Run revokeSysConfigManager to revoke the account's permission of modifying system parameter. parameter:

- account address

```
[group:1]> revokeSysConfigManager 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

grantContractWritePermission

Run grantContractWritePermissio to grant the account the contract write permission. parameters:

- contract address
- account address

```
[group:1]> grantContractWritePermission 0xc0ce097a5757e2b6e189aa70c7d55770ace47767
↪0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
```

(continues on next page)

(continued from previous page)

```
        "msg":"success"
}
```

listContractWritePermission

Run listContractWritePermission to query the account list which have write permission of the contract. parameters:

- contract address

```
[group:1]> listContractWritePermission 0xc0ce097a5757e2b6e189aa70c7d55770ace47767
----------------------------------------------------------------------------------------
↪----------
|                    address                     |                 enable_num
↪          |
| 0xc0d0e6ccc0b44c12196266548bec4a3616160e7d  |                    11
↪          |
----------------------------------------------------------------------------------------
↪----------
```

revokeContractWritePermission

Run revokeContractWritePermission to Revoke the account the contract write permission. parameters:

- 合约地址
- account address

```
[group:1]> revokeContractWritePermission␣
↪0xc0ce097a5757e2b6e189aa70c7d55770ace47767␣
↪0xc0d0e6ccc0b44c12196266548bec4a3616160e7d
{
        "code":0,
        "msg":"success"
}
```

quit

To run quit, q or exit to exit the console.

```
quit
```

[create sql]

Run create sql statement to create a user table in mysql statement form.

```
# Create user table t_demo whose primary key is name and other fields are item_id␣
↪and item_name
[group:1]> create table t_demo(name varchar, item_id varchar, item_name varchar,␣
↪primary key(name))
Create 't_demo' Ok.
```

Note:

- The field types for creating table are all string types. Even if other field types of the database are provided, the field types have to be set according to the string type.

- The primary key field must be specified. For example, to create a t_demo table with the primary key field as name.
- The primary key of the table has different concept from the primary key in the relational database. Here, the value of the primary key is not unique, and the primary key value needs to be passed when the blockchain underlying platform is handling records.
- You can specify the field as the primary key, but the setting fields such as self-incrementing, non-empty, indexing, etc do not work.

desc

Run desc statement to query the field information of the table in mysql statement form.

```
# query the field information of the t_demo table. you can view the primary key␣
↪name and other field names of the table.

[group:1]> desc t_demo
{
    "key":"name",
    "valueFields":"item_id,item_name"
}
```

[insert sql]

Run insert sql statement to insert the record in the mysql statement form.

```
[group:1]> insert into t_demo (name, item_id, item_name) values (fruit, 1, apple1)
Insert OK, 1 row affected.
```

Note:

- must insert a record sql statement with the primary key field value of the table.
- The enter values with punctuation, spaces, or strings containing letters starting with a number requires double quotation marks, and no more double quotation marks are allowed inside.

[select sql]

Run select sql statement to query the record in mysql statement form.

```
# query the records contain all fields
select * from t_demo where name = fruit
{item_id=1, item_name=apple1, name=fruit}
1 row in set.

# query the records contain the specified fields
[group:1]> select name, item_id, item_name from t_demo where name = fruit
{name=fruit, item_id=1, item_name=apple1}
1 row in set.

# insert a new record
[group:1]> insert into t_demo values (fruit, 2, apple2)
Insert OK, 1 row affected.

# use the keyword 'and' to connect multiple query condition
[group:1]> select * from t_demo where name = fruit and item_name = apple2
{item_id=2, item_name=apple2, name=fruit}
1 row in set.
```

(continues on next page)

(continued from previous page)

```
# use limit field to query the first line of records. If the offset is not␣
↪provided, it is 0 by default.
[group:1]> select * from t_demo where name = fruit limit 1
{item_id=1, item_name=apple1, name=fruit}
1 row in set.

# use limit field to query the second line record. The offset is 1
[group:1]> select * from t_demo where name = fruit limit 1,1
{item_id=2, item_name=apple2, name=fruit}
1 rows in set.
```

Note:

- For querying the statement recording sql, the primary key field value of the table in the where clause must be provided.
- The limit field in the relational database can be used. Providing two parameters which are offset and count.
- The where clause only supports the keyword 'and'. Other keywords like 'or', 'in', 'like', 'inner', 'join', 'union', subquery, multi-table joint query, and etc. are not supported.
- The enter values with punctuation, spaces, or strings containing letters starting with a number requires double quotation marks, and no more double quotation marks are allowed inside.

[update sql]

Run update sql statement to update the record in mysql statement form.

```
[group:1]> update t_demo set item_name = orange where name = fruit and item_id = 1
Update OK, 1 row affected.
```

Note:

- For updating the where clause of recording sql statement, the primary key field value of the table in the where clause must be provided.
- The enter values with punctuation, spaces, or strings containing letters starting with a number requires double quotation marks, and no more double quotation marks are allowed inside.

[delete sql]

Run delete sql statement to delete the record in mysql statement form.

```
[group:1]> delete from t_demo where name = fruit and item_id = 1
Remove OK, 1 row affected.
```

Note:

- For deleting the where clause of recording sql statement, the primary key field value of the table in the where clause must be provided.
- The enter values with punctuation, spaces, or strings containing letters starting with a number requires double quotation marks, and no more double quotation marks are allowed inside.

Important: The executing of the freezeContract/unfreezeContract/grantContractStatusManager commands for contract management should specify the private key to start the console for permission.This private key is also the account private key used to deploy the specified contract. So a private key should be specified to launch the console when deploying the contract.

---

freezeContract

Run freezeContract to freeze contract according contract address. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.

```
[group:1]> freezeContract 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
{
    "code":0,
    "msg":"success"
}
```

unfreezeContract

Run unfreezeContract to unfreeze contract according contract address. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.

```
[group:1]> unfreezeContract 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
{
    "code":0,
    "msg":"success"
}
```

grantContractStatusManager

Run grantCNSManager to grant the account's permission of contract status managememt. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.
- Account address: tx.origin. The prefix of 0x is not necessary.

```
[group:1]> grantContractStatusManager 0x30d2a17b6819f0d77f26dd3a9711ae75c291f7f1␣
↪0x965ebffc38b309fa706b809017f360d4f6de909a
{
    "code":0,
    "msg":"success"
}
```

getContractStatus

To run getContractStatus to query contract status according contract address. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.

```
[group:1]> getContractStatus 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
The contract is available.
```

listContractStatusManager

To run listContractStatusManager to query a list of authorized accounts that can manage a specified contract. Parameter:

- Contract address: To deploy contract can get contract address. The prefix of 0x is not necessary.

```
[group:1]> listContractStatusManager 0x30d2a17b6819f0d77f26dd3a9711ae75c291f7f1
[
    "0x0cc9b73b960323816ac5f52806257184c08b5ac0",
    "0x965ebffc38b309fa706b809017f360d4f6de909a"
]
```

grantCommitteeMember

grant account with Committee Member permission. Parameters:

- account address

```
[group:1]> grantCommitteeMember 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a
{
    "code":0,
    "msg":"success"
}
```

revokeCommitteeMember

revoke account's Committee Member permission, parameters:

- account address

```
[group:1]> revokeCommitteeMember 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a
{
    "code":0,
    "msg":"success"
}
```

listCommitteeMembers

```
[group:1]> listCommitteeMembers
-------------------------------------------------------------------------------------
↪----------
|                   address                    |                 enable_num
↪          |
| 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a   |                       1
↪          |
| 0x85961172229aec21694d742a5bd577bedffcfec3   |                       2
↪          |
-------------------------------------------------------------------------------------
↪----------
```

updateThreshold

vote to modify the votes threshold, Parameters:

- threshold:[0,99]

```
[group:1]> updateThreshold 75
{
    "code":0,
    "msg":"success"
}
```

queryThreshold

query votes threshold

```
[group:1]> queryThreshold
Effective threshold : 50%
```

queryCommitteeMemberWeight

```
[group:1]> queryCommitteeMemberWeight 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a
Account: 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a Weight: 1
```

updateCommitteeMemberWeight

update Committee Member's votes. Parameters：

- account address
- votes

```
[group:1]> updateCommitteeMemberWeight 0x61d88abf7ce4a7f8479cff9cc1422bef2dac9b9a 2
{
    "code":0,
    "msg":"success"
}
```

grantOperator

grantOperator, committee member's permission, parameters:

- account address

```
[group:1]> grantOperator 0x283f5b859e34f7fd2cf136c07579dcc72423b1b2
{
    "code":0,
    "msg":"success"
}
```

revokeOperator

revokeOperator, committee member's permission, parameters:

- account address

```
[group:1]> revokeOperator 0x283f5b859e34f7fd2cf136c07579dcc72423b1b2
{
    "code":0,
    "msg":"success"
}
```

listOperators

list address who has operator permission。

```
[group:1]> listOperators
--------------------------------------------------------------------------------------
↪----------
|                    address                    |                  enable_num
↪          |
| 0x283f5b859e34f7fd2cf136c07579dcc72423b1b2  |                       1
↪          |
--------------------------------------------------------------------------------------
↪----------
```

freezeAccount

Run freezeAccount to freeze account according account address. Parameter:

- account address: tx.origin. The prefix of 0x is necessary.

```
[group:1]> freezeAccount 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
{
    "code":0,
    "msg":"success"
}
```

unfreezeAccount

Run unfreezeAccount to unfreeze account according account address. Parameter:

- account address: tx.origin. The prefix of 0x is necessary.

```
[group:1]> unfreezeAccount 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
{
    "code":0,
    "msg":"success"
}
```

getAccountStatus

Run getAccountStatus to get status of the account according account address. Parameter:

- account address: tx.origin. The prefix of 0x is necessary.

```
[group:1]> getAccountStatus 0xcc5fc5abe347b7f81d9833f4d84a356e34488845
The account is available.
```

10.2.7 Appendix: Java environment configuration

Install Java in ubuntu environment

```
# Install the default Java version (Java 8 version or above)
sudo apt install -y default-jdk
# query Java version
java -version
```

Install Java in CentOS environment

Note: the OpenJDK under CentOS does not work properly and needs to be replaced with the OracleJDK.

```
# To create new folder to install Java 8 version or above. To put the downloaded␣
↪jdk in the software directory
# Download Java 8 version or above from Oracle official website (https://www.
↪oracle.com/technetwork/java/javase/downloads/index.html). For example, to␣
↪download jdk-8u201-linux-x64.tar.gz
$ mkdir /software
# To unzip jdk
$ tar -zxvf jdk-8u201-linux-x64.tar.gz
# To configure the Java environment and edit the /etc/profile file.
$ vim /etc/profile
# After opening the file, to enter the following three sentences into the file and␣
↪exit
export JAVA_HOME=/software/jdk-8u201-linux-x64.tar.gz
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
# profile takes effect
$ source /etc/profile
# To inquire the Java version. If the result shows the version you just downloaded,
↪ the installation is successful.
java -version
```

CHAPTER

ELEVEN

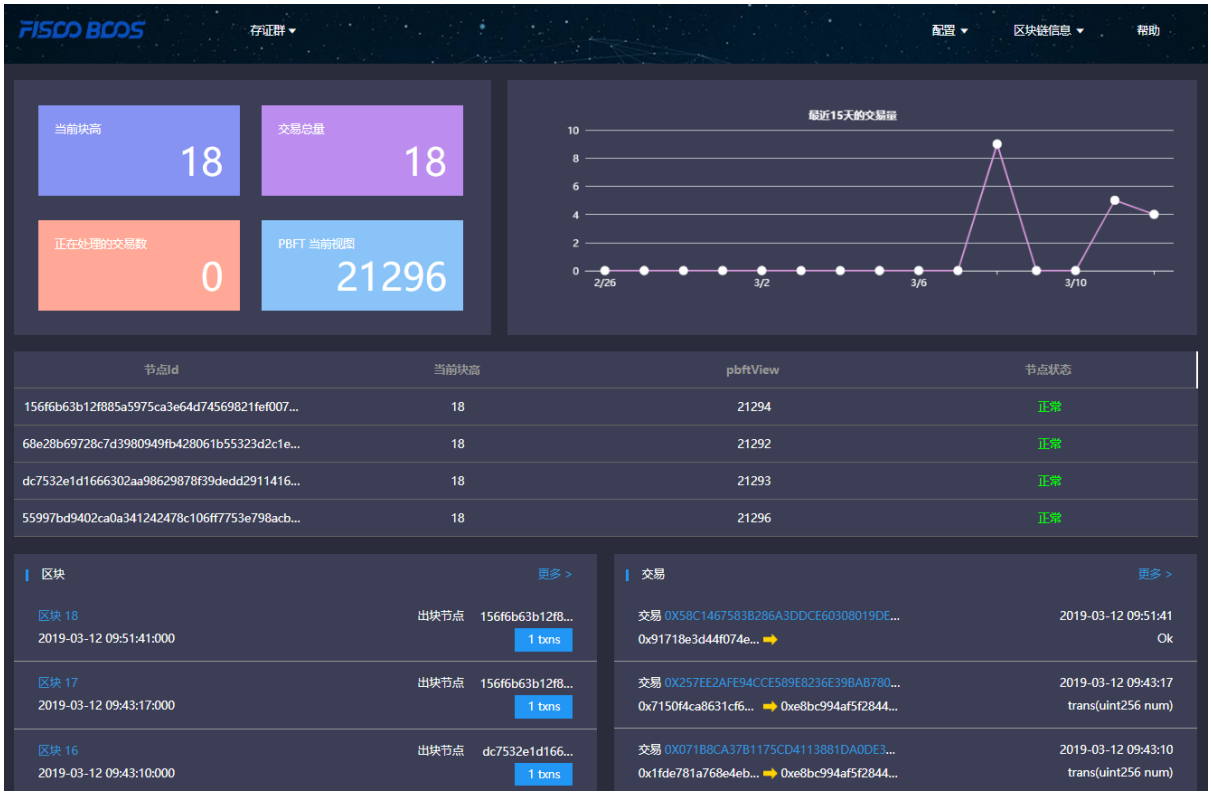
BLOCKCHAIN BROWSER

11.1 1. Description

11.1.1 1.1 Introduction

This blockchain browser is compatible with FISCO BCOS 2.0.0. FISCO BCOS 1.2 or 1.3 users please check v1.2.1.

Blockchain browser is capable of blockchain visualization and real-time presentation. Users can get the information of the blockchain through web pages. This browser is only compatible with FISCO BCOS 2.0+. You can learn the newest features in here. Before using this browser, you may need to learn the groups feature of FISCO BCOS 2.0+.

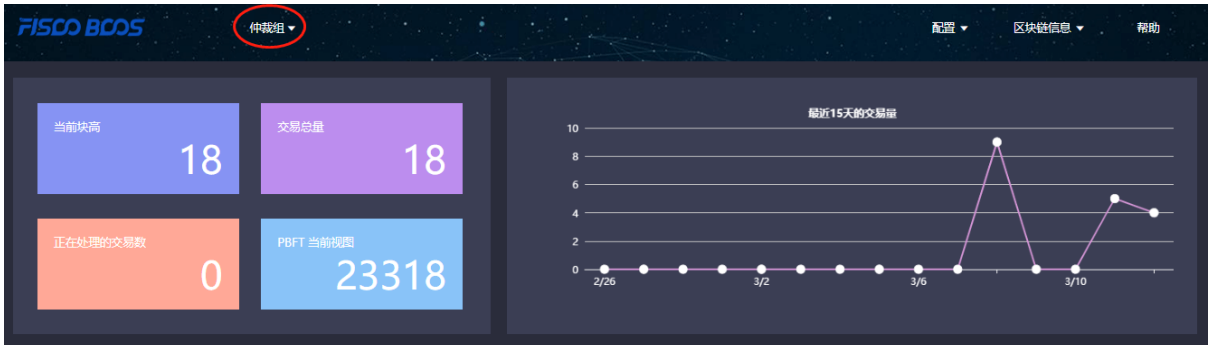


1.2 Main functional modules

This chapter will give a brief introduction on each module of the browser for all-round understanding. Main functional modules of the blockchain browser includes: group switch module, configuration module and data visualization module.

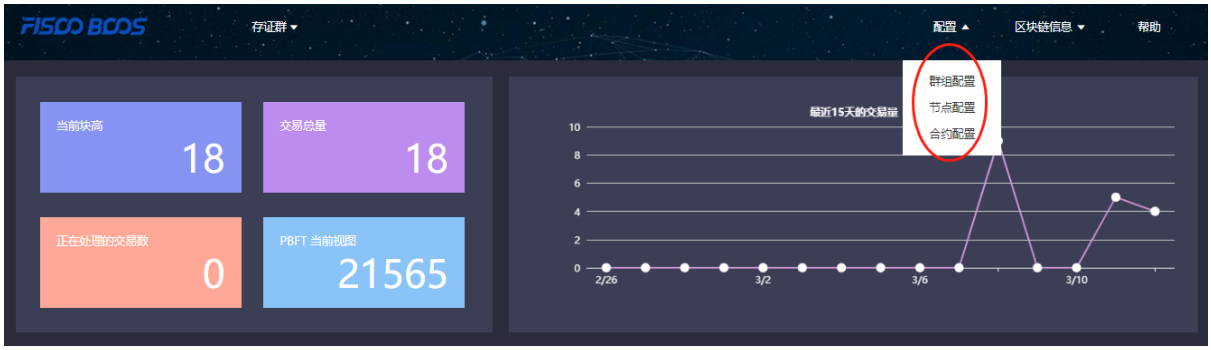
1.2.1 Group switch module

Group switch module is adopted to access blockchain data when switching to different groups in multi-groups case.



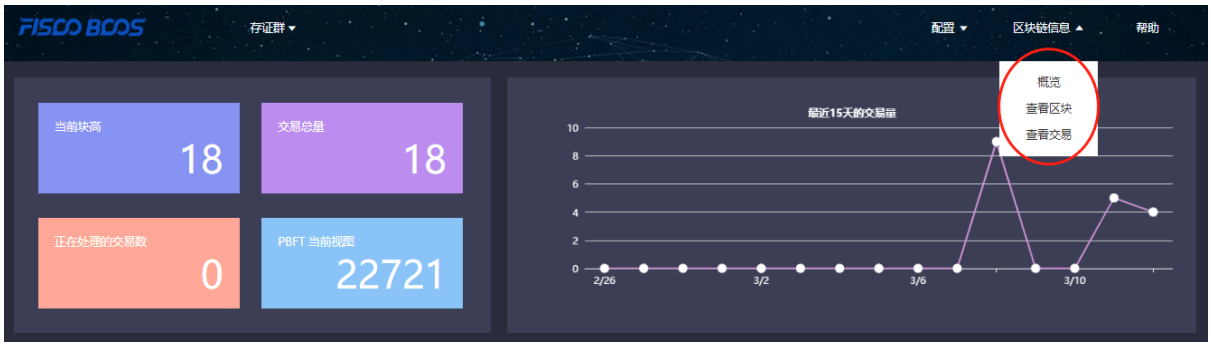
1.2.2 Configuration module

Configuration module includes group configuration, node configuration and contract configuration.



1.2.3 Data visualization module

Blockchain browser demonstrates the detail information of specific group on the chain including: overview, block information and transaction information.



11.2 2. Premises of use

11.2.1 2.1 Group building

Data shown in the blockchain browser is synchronized with blockchain. To synchronize data, initialization configuration (adding group information and node information) is needed. So users have to run a FISCO BCOS instance

and build groups before data synchronizing. FISCO BCOS 2.0+ has provided multiple convenient group building methods.

1. For developers to experience and debug quickly, we recommend the script build_chain.
2. For enterprise applications, FISCO BCOS generator is a more considerable deployment tool.

The distinguish of the above methods lie in that the script `build_chain` is for better and quicker building experience and it helps developers generate private key of each node in groups; deployment tool doesn't automatically generate private key for safety consideration, and business users need to generate and set by themselves.

11.3 3. Building of blockchain browser

Blockchain browser can be divided into two parts: the back-end service "fisco-bcos-browser" and the front-end web page "fisco-bcos-browser-front".

We also provide two ways for browser building in the current version: one-key setup and manual setup.

11.3.1 3.1.1 One-click setup

One-click setup is suitable for single-machine deployment of front-end and back-end to experience quickly. The detail process is introduced in Installation document.

11.3.2 3.1.2 Manual setup

Back-end service building

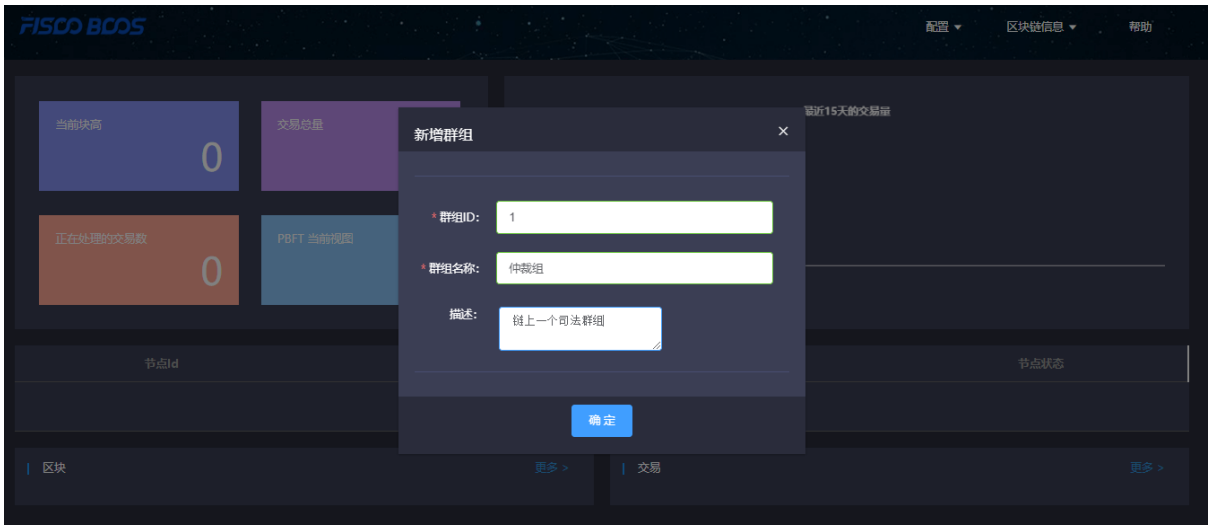
The back-end service of blockchain browser adopts JAVA back-end service, i.e., Spring Boot. The exactly building process can be referred in Installation document.

Front-end web page service building

Front-end conducts `vue-cli`. Also, the tutorial can be found in Installation document.

11.4 4. Initialization environment

11.4.1 4.1 Adding group



Once it is set up, users can access the front-end by typing IP and its port configured by nginx through web browser. Browser without group initialization will lead to new group configuration page, where the group ID, group name and group description are needed.

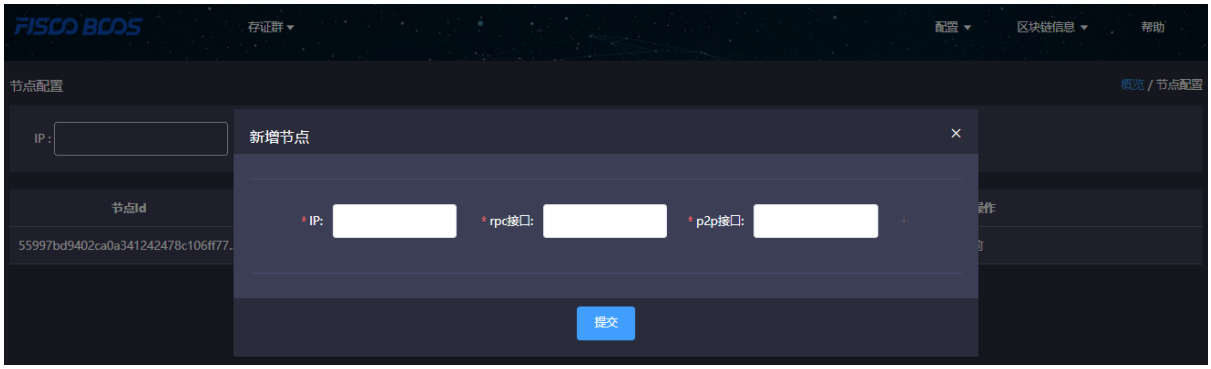
Group ID should be consistent with the specific blockchain. There are many methods to check group ID:

- acquire rpc interface.
- console commands: Please refer to here for the console user manual of version 2.6 and above, and here for the console user manual of version 1.x

Group name should be meaningful and better understandable as a explanation of group ID.

Group description is the further illustration of the name.

11.4.2 4.2 Adding node



The next step, you need to add the node information belong to the group to obtain relative information shown in blockchain browser. RPC port and P2P port of nodes can be acquired from the file config.ini in the directory of a specific node.

For easy use, the newly added group will synchronize the information of shared node which configured by other groups before.

11.4.3 4.3 Adding contract

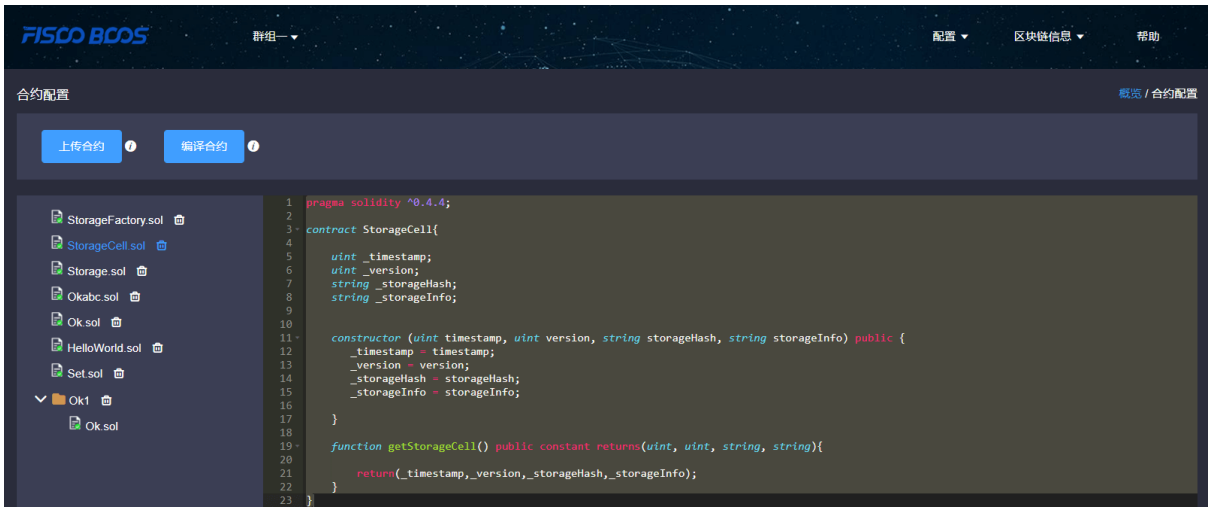
On this version, the browser provides the function of contract analysis, which requires users to import all contracts the group had deployed before. User can upload zip package (only support one-level directory) to solve namesake contract issues.

Steps of import

4.3.1 Import contract

1. Contract is required to be uploaded as `*.sol` file or zip package.
2. Zip package is compatible with one-level directory at most and defaulted to be uploaded to root directory. Zip package can only contain `*.sol` files.

4.3.2 Compile contract



11.5 5. Functions introduction

11.5.1 5.1 Blockchain overview

5.1.1 Overall overview

Overall overview includes block number of the group, transaction volume, processing transaction amount and the PBFT view.

5.1.2 Transaction volume in 15 days

The transactions of the group in 15 days are shown in the line chart.

5.1.3 Node overview

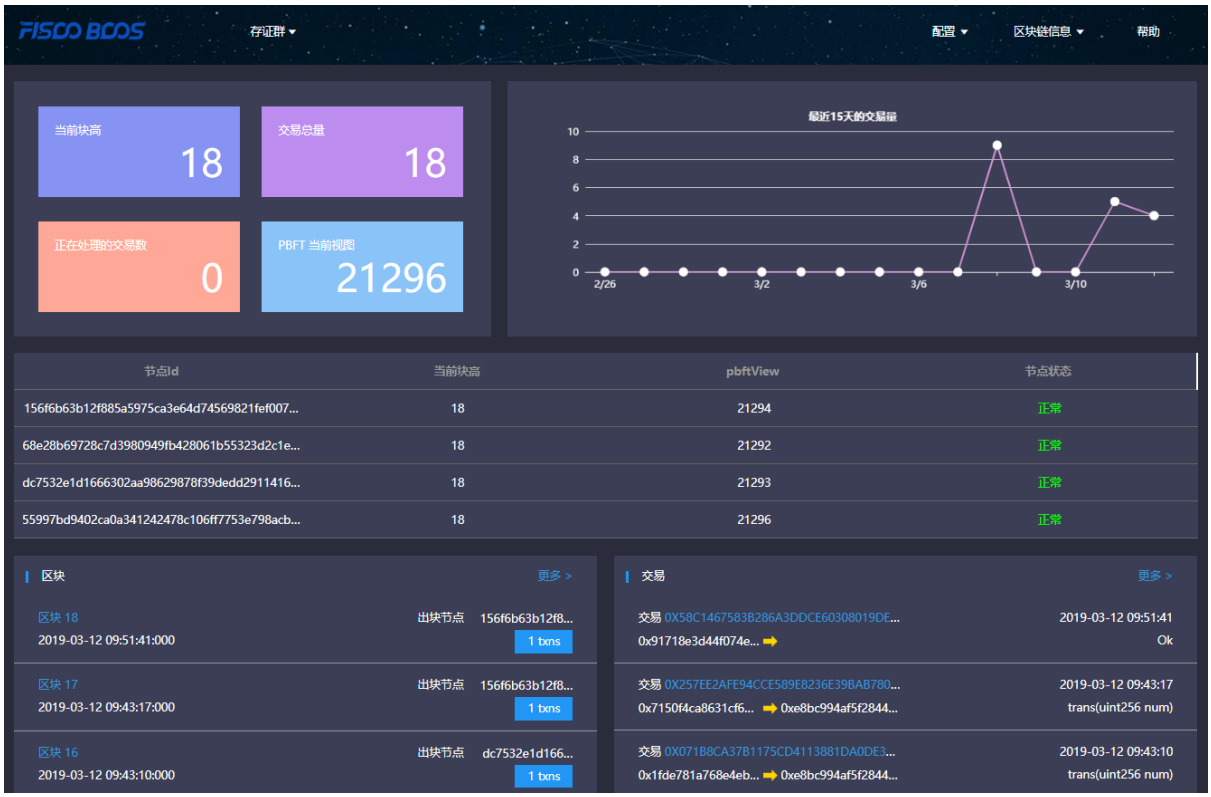
Node overview includes node ID, current block height, the PBFT view and node status.

5.1.4 Block overview

Block overview includes the information of the latest four blocks, including block height, block generator, generation time and transaction volume on the block.

5.1.5 Transaction overview

Transaction overview includes the latest four transactions, including transaction hash, transaction time, transaction sender & receiver. The information invoked by transactions can also be shown if the related contract is imported correctly.



11.5.2 5.2 Block information

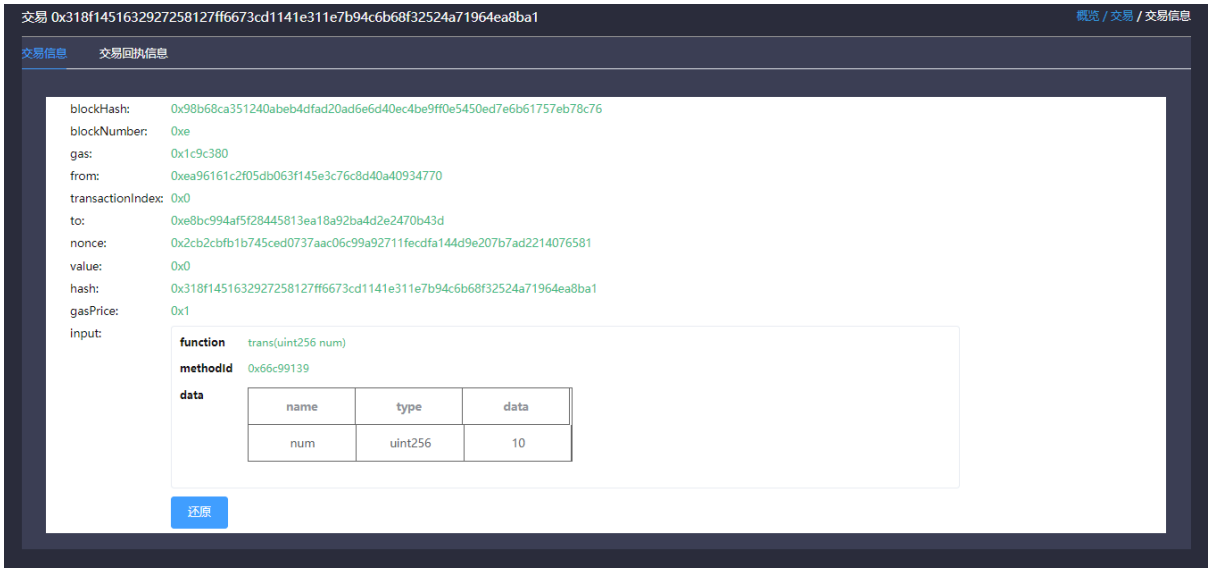
Block information includes pages of block list and block details.

11.5.3 5.3 Transaction information

Transaction information includes pages of transaction list and transaction details.

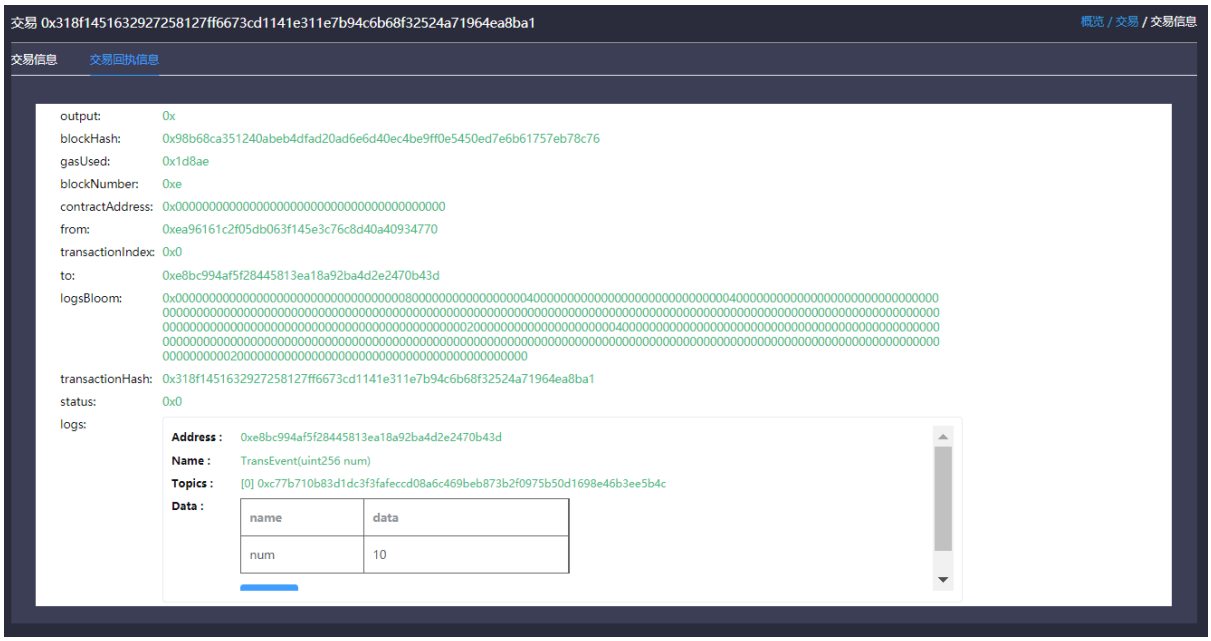
5.3.1 Transaction analysis

After contract is uploaded and compiled, blockchain browser can analyze the transaction method names and parameters. The analysis of the browser is based on correct import of contract. Therefore, when using JAVA or JS to call contract, please save the correct version of contract.



5.3.2 Event analysis

After contract is uploaded and compiled, blockchain browser can analyze event method names and parameters in the transaction receipts.



CHAPTER

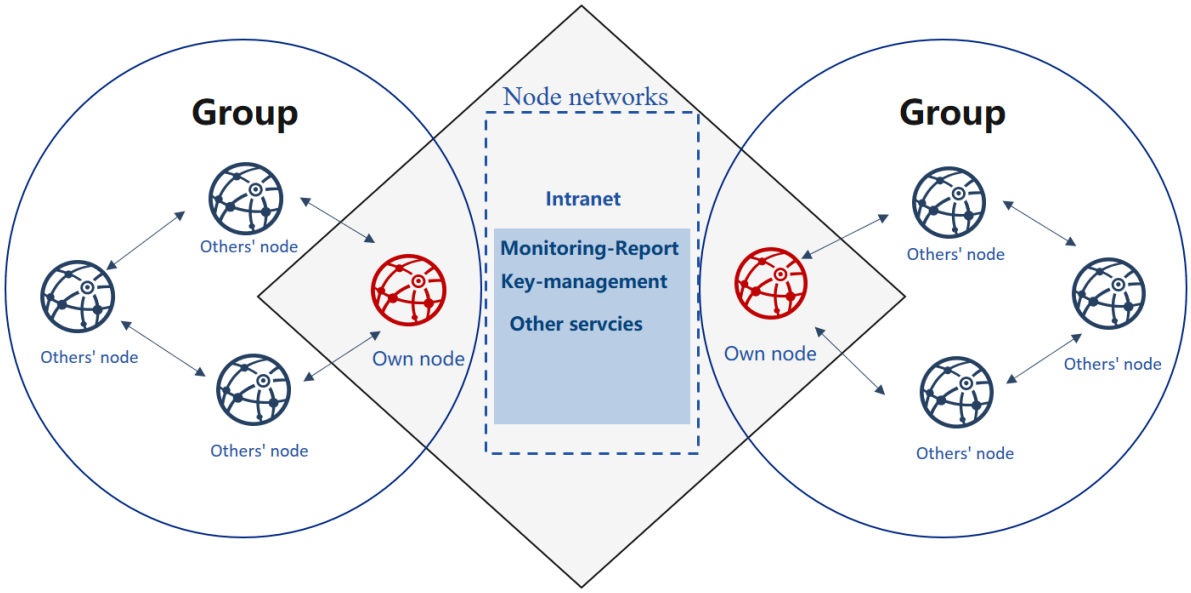
TWELVE

ENTERPRISE DEPLOYMENT TOOL

Introduction

FISCO BCOS generator provides companies with an easy toolkit for deployment, administration, and monitoring of multi-group consortium chain.

- It eliminates the complexity of generating and maintaining the blockchain and offers alternative deployment methods.
- It requires agencies to share node credentials and manage their private key but not exposed to the outsider, maintaining the security of all nodes.
- It helps agencies deploy nodes safely through e-certificate trading, supporting equality of all nodes.



Design background

There cannot be exhaustive trust between equal agencies in consortium chain, where e-certificate will be needed for nodes to authenticate each other's identity.

The certificate is the identity documentation for each agency. And the generation of the certificate depends on its public & private key pair. The private key represents its identity information that is private and strictly confidential. In the process of activation and operation, node signs on the data packet with the private key to fulfilling identity authentication. Provided that an agency's private key is revealed, anyone else can pretend as the owner and get authorized without the affirmation of this agency.

When initializing the group of FISCO BCOS, nodes should reach an agreement to create a Genesis Block. Genesis Block, unique and only within one group, bears the identity information of the initial nodes, which is formed through e-credential exchanging.

Current IT administration tools for consortium chain usually ignore the requirement for equality and security of companies during initialization. And initialization needs agencies to agree on identity information on Genesis

Block. So, who should be the information generator is crucial. Firstly, an agency generates its node information first and then activate blockchain for other nodes to join in. Secondly, a third-party authority makes information for all nodes and sends the node configuration files to each agency.

Additionally, FISCO BCOS 2.0+ adapts more private and scalable multi-group architecture. It is an architecture where data and transactions between groups are separated by running independent consensus algorithm, a way to maintain privacy and security in blockchain scenarios.

In the above models, there is always one agency who gains priority to join the consortium chain or acquires private keys of all nodes.

How to make sure the group is formed in a balanced, safe, and private way? How to guarantee reliable and effective operation of nodes? The privacy and security of group ledgers, as well as the confidentiality of group formation and maintenance, need to be achieved efficiently.

Design concept

FISCO BCOS generator is a solution designed for problems described above. It takes into consideration the equal deployment and group formation of different agencies based on flexibility, security, ease-of-use, and equality.

Flexibility:

- No installation, ready to use
- Alternative deployment methods
- Allow multiple changes in architecture

Safety:

- Allow multiple changes in architecture
- The private key is kept internally
- Negotiation between agencies is based on certificates only

Ease-to-use:

- Support multiple networking models
- Alternative commands for various needs
- Monitor audit script

Equality:

- The equal authority of agencies
- All agencies co-generate Genesis Block
- Same administrative power within groups

For consortium chain based on existed root credential, it can fast configure multiple groups on-chain to adapt for different business needs.

Each agency can generate a configuration file folder locally that includes no private key Agencies can keep their private keys internally and prevent malicious attackers in disguise of nodes or any information leakage, even if the configuration files are lost. In this way, security and usability of nodes can be achieved at the same time.

Users agree to generate the Genesis Block and node configuration file folder and then activate nodes so that they will conduct multi-group networking according to the configuration files.

12.1 Tutorial of one_click_generator.sh

The `one_click_generator.sh` script is a script that deploys a federated chain with one click based on the node configuration filled out by the user. The script will generate the corresponding node under the folder according to the `node_deployment.ini` configured under the user-specified folder.

This chapter mainly uses the networking mode of deploying 3 organizations 2 groups 6 nodes to explain the use of enterprise-level deployment tools for single-button one-button deployment.

This tutorial is suitable for single-node deployment of all nodes. The enterprise-level deployment tool multi-agent deployment tutorial can refer to Using Enterprise Deployment Tools.

Important: When using the one-click deployment script, you need to ensure that the current meta folder does not contain node certificate information. You can clean the meta folder.

12.1.1 Download and install

download

```
cd ~/
git clone https://github.com/FISCO-BCOS/generator.git

# If you have network issue for exec the command above, please try:
git clone https://gitee.com/FISCO-BCOS/generator.git
```

Installation

This action requires the user to have sudo privileges.

```
cd ~/generator && bash ./scripts/install.sh
```

Check if the installation is successful. If successful, output usage: generator xxx

```
./generator -h
```

Get node binary

Pull the latest fisco-bcos binary to the meta, you can try `--cdn` to improve your download speed.

```
./generator --download_fisco ./meta
```

Check the binary version

If successful, output FISCO-BCOS Version : x.x.x-x

```
./meta/fisco-bcos -v
```

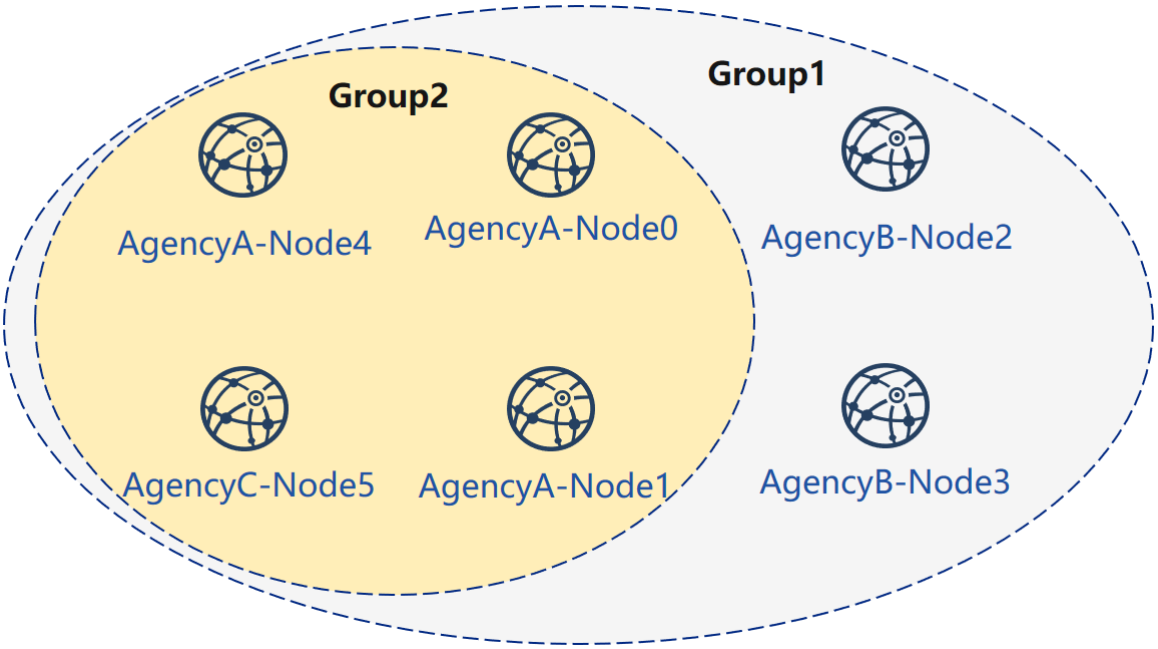
PS: source compile Node binary user, you only need to replace the binary in the `meta` folder with the compiled binary.

12.1.2 Typical example

This section demonstrates how to use the one-click deployment function of the enterprise-level deployment tool to build a blockchain.

Node Network Topology

A networking model of a 6-node 3-institution 2 group as shown. agency B and agency C are located in Group 1 and Group 2, respectively. agency A belongs to both Group 1 and Group 2.



Machine Environment

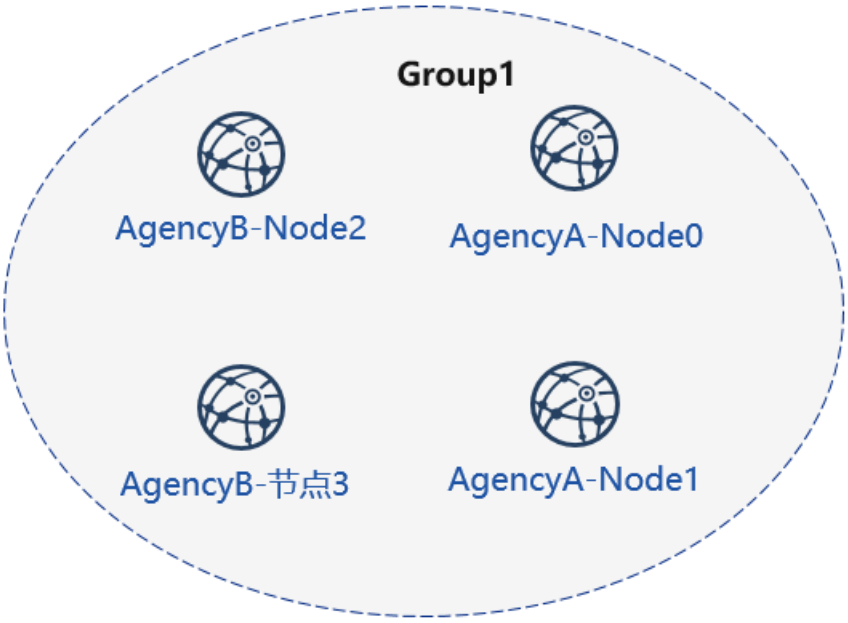
The IP of each node, the port number is as follows:

Note:

- The public IP of the cloud host is a virtual IP. If you enter the external IP in rpc_ip/p2p_ip/channel_ip, the binding will fail. You must fill in 0.0.0.0
- The RPC/P2P/Channel listening port must be in the range of 1024-65535, and must not conflict with other application listening ports on the machine
- For security and ease of use consideration, FISCO BCOS v2.3.0 latest node config.ini configuration splits listen_ip into jsonrpc_listen_ip and channel_listen_ip, but still retains the parsing function of listen_ip, please refer to here
- In order to facilitate development and experience, the reference configuration of channel_listen_ip is 0.0.0.0. For security reasons, please modify it to a safe listening address according to the actual business network situation, such as: intranet IP or specific external IP

12.1.3 Generate group 1 node

First, complete the operation of group A and B to set up group 1, as shown in the figure:



Before use, the user needs to prepare a folder such as `tmp_one_click`, which has a directory of different organizations under the folder. Each agency directory needs to have a corresponding configuration file `node_deployment.ini`. Before use, you need to ensure that the generator's meta folder has not been used.

View the one-click deployment template folder:

```
cd ~/generator
ls ./tmp_one_click
```

```
#Parameter explanation
# For multiple organizations, you need to create this folder manually.
Tmp_one_click # user specifies the folder for a one-click deployment operation
├── agencyA #agencyA directory, after the command is executed, the node of the␣
↪agency A and related files will be generated in the list.
│   └── node_deployment.ini # Institution A node configuration file, one-click␣
↪deployment command will create the corresponding node according to the data
└── agencyB #agencyB directory, after the command is executed, the node of the␣
↪agency B and related files will be generated in the list.
    └── node_deployment.ini # Institution B node configuration file, one-click␣
↪deployment command will generate the corresponding node according to the data
```

Institution to fill in node information

The configuration file is placed in the tutorial with agencyA under the tmp_one_click folder, under agencyB

```
cat > ./tmp_one_click/agencyA/node_deployment.ini << EOF
[group]
group_id=1

[node0]
; host IP for the communication among peers.
; Please use your ssh login IP.
p2p_ip=127.0.0.1
; listening IP for the communication between SDK clients.
; This IP is the same as p2p_ip for the physical host.
; But for virtual host e.g., VPS servers, it is usually different from p2p_ip.
; You can check accessible addresses of your network card.
; Please see https://tecadmin.net/check-ip-address-ubuntu-18-04-desktop/
```

(continues on next page)

(continued from previous page)

```
; for more instructions.
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30300
channel_listen_port=20200
jsonrpc_listen_port=8545

[node1]
p2p_ip=127.0.0.1
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30301
channel_listen_port=20201
jsonrpc_listen_port=8546
EOF
```

```
cat > ./tmp_one_click/agencyB/node_deployment.ini << EOF
[group]
group_id=1

[node0]
; Host IP for the communication among peers.
; Please use your ssh login IP.
p2p_ip=127.0.0.1
; listening IP for the communication between SDK clients.
; This IP is the same as p2p_ip for the physical host.
; But for virtual host e.g., VPS servers, it is usually different from p2p_ip.
; You can check accessible addresses of your network card.
; Please see https://tecadmin.net/check-ip-address-ubuntu-18-04-desktop/
; for more instructions.
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30302
channel_listen_port=20202
jsonrpc_listen_port=8547

[node1]
p2p_ip=127.0.0.1
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30303
channel_listen_port=20203
jsonrpc_listen_port=8548
EOF
```

Generate node

```
bash ./one_click_generator.sh -b ./tmp_one_click
```

After the execution is completed, the ./tmp_one_click folder structure is as follows:

View the one-click deployment template folder after execution:

```
ls ./tmp_one_click
```

```
├── agencyA # A agency folder
│   ├── agency_cert # A agency certificate and private key
│   ├── generator-agency # Automatically replaces the generator folder operated by␣
↪the A mechanism
```

(continues on next page)

(continued from previous page)

```
| ├── node #A node generated by the agency, when the multi-machine is deployed, it␣
↪can be pushed to the corresponding server.
| ├── node_deployment.ini # Node configuration information of A agency
| └── SDK # A SDK or console configuration file
├── agencyB
|   ├── agency_cert
|   ├── generator-agency
|   ├── node
|   ├── node_deployment.ini
|   └── sdk
|── ca.crt # chain certificate
|── ca.key # chain private key
|── group.1.genesis # group one's genesis block
|── peers.txt # node's peers.txt
```

Start node

Call the script to start the node:

```
bash ./tmp_one_click/agencyA/node/start_all.sh
```

```
bash ./tmp_one_click/agencyB/node/start_all.sh
```

View the node process:

```
ps -ef | grep fisco
```

```
#Command explanation
# can see the following process
fisco 15347 1 0 17:22 pts/2 00:00:00 ~/generator/tmp_one_click/agencyA/node/node_
↪127.0.0.1_30300/fisco-bcos -c config.ini
fisco 15402 1 0 17:22 pts/2 00:00:00 ~/generator/tmp_one_click/agencyA/node/node_
↪127.0.0.1_30301/fisco-bcos -c config.ini
fisco 15442 1 0 17:22 pts/2 00:00:00 ~/generator/tmp_one_click/agencyB/node/node_
↪127.0.0.1_30302/fisco-bcos -c config.ini
fisco 15456 1 0 17:22 pts/2 00:00:00 ~/generator/tmp_one_click/agencyB/node/node_
↪127.0.0.1_30303/fisco-bcos -c config.ini
```

View node running status

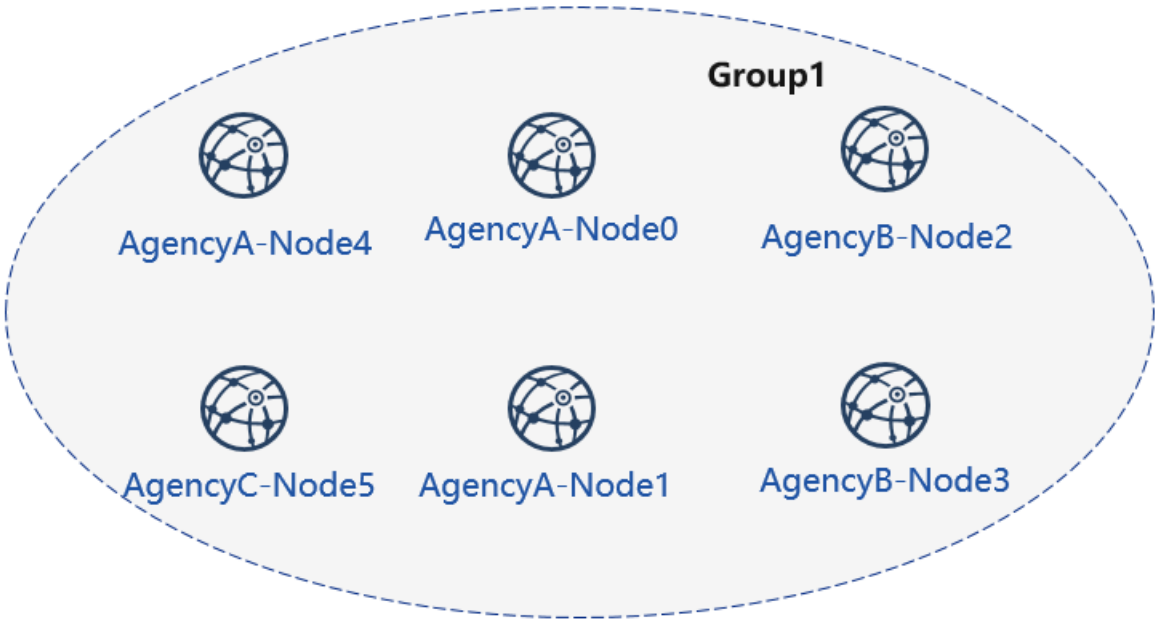
View the node log:

```
tail -f ~/generator/tmp_one_click/agency*/node/node*/log/log* | grep +++
```

```
#Command explanation
# +++ is the normal consensus of the node
Info|2019-02-25 17:25:56.028692| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum =1, tx=0, myIdx=0, hash=833bd983...
Info|2019-02-25 17:25:59.058625| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum =1, tx=0, myIdx=0, hash=343b1141...
Info|2019-02-25 17:25:57.038284| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum =1, tx=0, myIdx=1, hash=ea85c27b...
```

12.1.4 Add node to group 1

Next, we add new nodes for agency A and agency C, and complete the networking shown in the following figure:



Initialize the expansion configuration

Important: When using the one-click deployment script, you need to ensure that the current meta folder does not contain node certificate information. You can clean the meta folder.

Create an expansion folder, don't use the same foler

```
mkdir ~/generator/tmp_one_click_expand/
```

Copy the chain certificate and private key to the expansion folder

```
cp ~/generator/tmp_one_click/ca.* ~/generator/tmp_one_click_expand/
```

Copy group 1 genesis block `group.1.genesis` to expansion folder

```
cp ~/generator/tmp_one_click/group.1.genesis ~/generator/tmp_one_click_expand/
```

Copy group 1 node P2P connection file `peers.txt` to expansion folder

```
cp ~/generator/tmp_one_click/peers.txt ~/generator/tmp_one_click_expand/
```

agency A configuration node information

Create the directory where the agency A expansion node is located.

```
mkdir ~/generator/tmp_one_click_expand/agencyA
```

At this time, the agency A already exists in the alliance chain. So it is necessary to copy the agency A certificate and the private key to the corresponding folder.

```
cp -r ~/generator/tmp_one_click/agencyA/agency_cert ~/generator/tmp_one_click_
↪expand/agencyA
```

agency A fills in the node configuration information

```
cat > ./tmp_one_click_expand/agencyA/node_deployment.ini << EOF
[group]
group_id=1

[node0]
; Host IP for the communication among peers.
; Please use your ssh login IP.
p2p_ip=127.0.0.1
; listening IP for the communication between SDK clients.
; This IP is the same as p2p_ip for the physical host.
; But for virtual host e.g., VPS servers, it is usually different from p2p_ip.
; You can check accessible addresses of your network card.
; Please see https://tecadmin.net/check-ip-address-ubuntu-18-04-desktop/
; for more instructions.
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30304
channel_listen_port=20204
jsonrpc_listen_port=8549
EOF
```

Institution C configuration node information

Create a directory where the agency C expansion node is located.

```
mkdir ~/generator/tmp_one_click_expand/agencyC
```

agency C fills in the node configuration information

```
cat > ./tmp_one_click_expand/agencyC/node_deployment.ini << EOF
[group]
group_id=1

[node0]
; Host IP for the communication among peers.
; Please use your ssh login IP.
p2p_ip=127.0.0.1
; listening IP for the communication between SDK clients.
; This IP is the same as p2p_ip for the physical host.
; But for virtual host e.g., VPS servers, it is usually different from p2p_ip.
; You can check accessible addresses of your network card.
; Please see https://tecadmin.net/check-ip-address-ubuntu-18-04-desktop/
; for more instructions.
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30305
channel_listen_port=20205
jsonrpc_listen_port=8550
EOF
```

Generate expansion nodes

```
bash ./one_click_generator.sh -e ./tmp_one_click_expand
```

Starting a new node

Call the script to start the node:

```
bash ./tmp_one_click_expand/agencyA/node/start_all.sh
```

```
bash ./tmp_one_click_expand/agencyC/node/start_all.sh
```

View the node process:

```
ps -ef | grep fisco
```

```
#Command explanation
# can see the following process
fisco  15347     1  0 17:22 pts/2    00:00:00 ~/generator/tmp_one_click/agencyA/
↪node/node_127.0.0.1_30300/fisco-bcos -c config.ini
fisco  15402     1  0 17:22 pts/2    00:00:00 ~/generator/tmp_one_click/agencyA/
↪node/node_127.0.0.1_30301/fisco-bcos -c config.ini
fisco  15403     1  0 17:22 pts/2    00:00:00 ~/generator/tmp_one_click_expand/
↪agencyA/node/node_127.0.0.1_30304/fisco-bcos -c config.ini
fisco  15442     1  0 17:22 pts/2    00:00:00 ~/generator/tmp_one_click/agencyB/
↪node/node_127.0.0.1_30302/fisco-bcos -c config.ini
fisco  15456     1  0 17:22 pts/2    00:00:00 ~/generator/tmp_one_click/agencyB/
↪node/node_127.0.0.1_30303/fisco-bcos -c config.ini
fisco  15466     1  0 17:22 pts/2    00:00:00 ~/generator/tmp_one_click_expand/
↪agencyC/node/node_127.0.0.1_30305/fisco-bcos -c config.ini
```

Important: New nodes that are expanded for group 1 need to be added to the group using SDK or console.

Registering a node with the console

Due to the large size of the console, there is no direct integration in a one-click deployment. Users can use the following command to get the console.

Getting the console may take a long time, and domestic users can use the `--cdn` command:

For example, if the agency A uses the console, this step needs to be switched to the `generator-agency` folder corresponding to the agency A.

```
cd ~/generator/tmp_one_click/agencyA/generator-agency
```

```
./generator --download_console ./ --cdn
```

Viewing agency A node-4 information

agency A uses the console to join the agency A node 4 as the consensus node, where the second parameter needs to be replaced with the nodeid of the joining node, and the nodeid is the `node.nodeid` of the conf of the node folder.

View the agency A node nodeid:

```
cat ~/generator/tmp_one_click_expand/agencyA/node/node_127.0.0.1_30304/conf/node.
↪nodeid
```

```
ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c9715
```

Registering Consensus Nodes Using the Console

Start the console:

```
cd ~/generator/tmp_one_click/agencyA/generator-agency/console && bash ./start.sh 1
```

Use the console `addSealer` command to register the node as a consensus node. In this step, you need to use the `cat` command to view the node `node.nodeid` of the agency A node:

```
addSealer␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
```

```
$ [group:1]> addSealer␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
{
        "code":0,
        "msg":"success"
}
```

exit console:

```
exit
```

Viewing agency C Node 5

agency A uses the console to join node 5 of agency C as the observation node, where the second parameter needs to be replaced with the nodeid of the joining node, and the nodeid is the node.nodeid file of the conf of the node folder.

View the agency C node nodeid:

```
cat ~/generator/tmp_one_click_expand/agencyC/node/node_127.0.0.1_30305/conf/node.
↪nodeid
```

```
5d70e046047e15a68aff8e32f2d68d1f8d4471953496fd97b26f1fbdc18a76720613a34e3743194bd78aa7acb59b9fa9a
```

Registering an observation node using the console

Start the console:

```
cd ~/generator/tmp_one_click/agencyA/generator-agency/console && bash ./start.sh 1
```

Use the console `addObserver` command to register the node as a watch node. In this step, you need to use the `cat` command to view the node #node.nodeid‘ of the agency C node:

```
addObserver␣
↪5d70e046047e15a68aff8e32f2d68d1f8d4471953496fd97b26f1fbdc18a76720613a34e3743194bd78aa7acb59b9fa
```

```
$ [group:1]> addObserver␣
↪5d70e046047e15a68aff8e32f2d68d1f8d4471953496fd97b26f1fbdc18a76720613a34e3743194bd78aa7acb59b9fa
{
        "code":0,
        "msg":"success"
}
```

Exit the console:

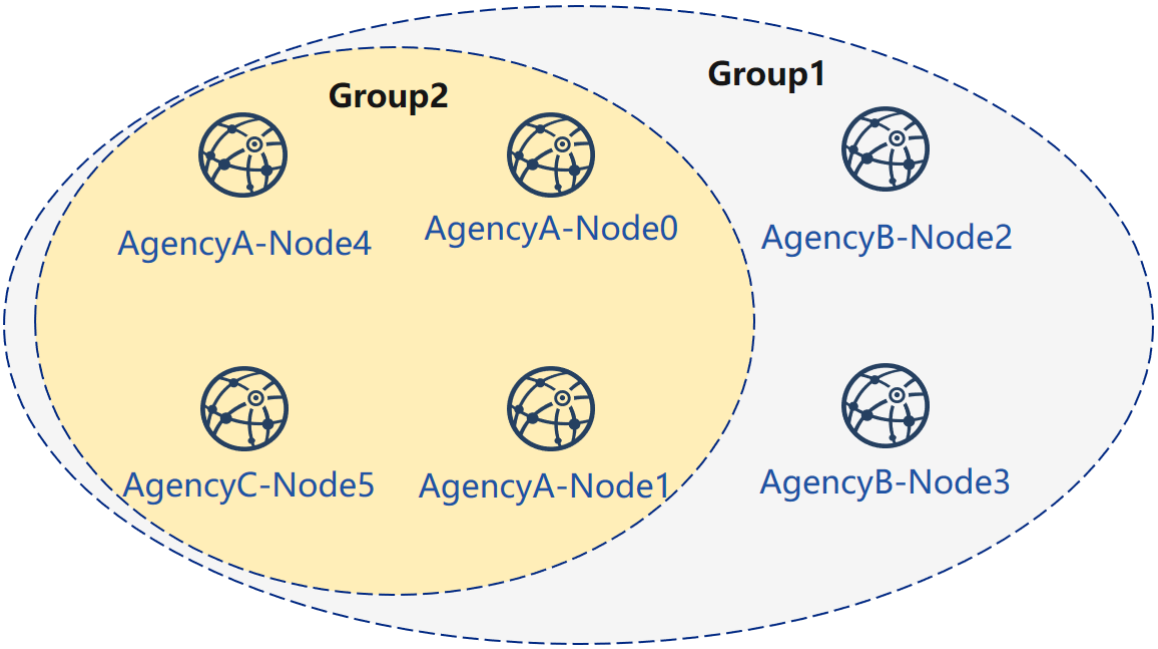
```
exit
```

At this point, we have completed the operation of adding a new node to an existing group.

12.1.5 Existing Node New Group 2

The operation of the new group can be done by modifying the `./conf/group_genesis.ini` file in the directory where the `one_click_generator.sh` is executed and executing the `--create_group_genesis` command.

Generate group 2 as shown in Figure 4



Configure Group 2 genesis block

This operation needs to be performed under the above operation generator.

```
cd ~/generator
```

Configuring the group genesis block file. Add `nodeX=ip:port` under `[nodes]` which belong to the members of new group. For example they are: AgencyA-Node0, AgencyA-Node1, AgencyA-Node4 and AgencyC-Node5.

```
cat > ./conf/group_genesis.ini << EOF
[group]
group_id=2

[nodes]
node0=127.0.0.1:30300
node1=127.0.0.1:30301
node2=127.0.0.1:30304
node3=127.0.0.1:30305
EOF
```

Get the corresponding node certificate

AgencyA-Node0（`node0=127.0.0.1:30300`）

```
cp ~/generator/tmp_one_click/agencyA/generator-agency/meta/cert_127.0.0.1_30300.
↪crt ~/generator/meta
```

AgencyA-Node1（`node1=127.0.0.1:30301`）

```
cp ~/generator/tmp_one_click/agencyA/generator-agency/meta/cert_127.0.0.1_30301.
↪crt ~/generator/meta
```

AgencyA-Node4（node2=127.0.0.1:30304）

```
cp ~/generator/tmp_one_click_expand/agencyA/generator-agency/meta/cert_127.0.0.1_
↪30304.crt ~/generator/meta
```

AgencyC-Node5（node3=127.0.0.1:30305）

```
cp ~/generator/tmp_one_click_expand/agencyC/generator-agency/meta/cert_127.0.0.1_
↪30305.crt ~/generator/meta
```

Generating genesis block and group configure file

```
./generator --create_group_genesis ./group2
```

将群组创世区块加入现有节点:

AgencyA-Node0（node0=127.0.0.1:30300）

```
./generator --add_group ./group2/group.2.genesis ./tmp_one_click/agencyA/node/node_
↪127.0.0.1_30300
```

AgencyA-Node1（node1=127.0.0.1:30301）

```
./generator --add_group ./group2/group.2.genesis ./tmp_one_click/agencyA/node/node_
↪127.0.0.1_30301
```

AgencyA-Node4（node2=127.0.0.1:30304）

```
 ./generator --add_group ./group2/group.2.genesis ./tmp_one_click_expand/agencyA/
↪node/node_127.0.0.1_30304
```

AgencyC-Node5（node3=127.0.0.1:30305）

```
./generator --add_group ./group2/group.2.genesis ./tmp_one_click_expand/agencyC/
↪node/node_127.0.0.1_30305
```

Load and start new group

Use `load_new_groups.sh` to load configuration of new group, and call `startGroup` RPC interface to start new group

AgencyA-Node0（node0=127.0.0.1:30300）

```
bash ./tmp_one_click/agencyA/node/node_127.0.0.1_30300/scripts/load_new_groups.sh
curl -X POST --data '{"jsonrpc":"2.0","method":"startGroup","params":[2],"id":1}'␣
↪http://127.0.0.1:8545
```

AgencyA-Node1（node1=127.0.0.1:30301）

```
bash ./tmp_one_click/agencyA/node/node_127.0.0.1_30301/scripts/load_new_groups.sh
curl -X POST --data '{"jsonrpc":"2.0","method":"startGroup","params":[2],"id":1}'␣
↪http://127.0.0.1:8546
```

AgencyA-Node4（node2=127.0.0.1:30304）

```
bash ./tmp_one_click_expand/agencyA/node/node_127.0.0.1_30304/scripts/load_new_
↪groups.sh
curl -X POST --data '{"jsonrpc":"2.0","method":"startGroup","params":[2],"id":1}'␣
↪http://127.0.0.1:8549
```

AgencyA-Node5（node3=127.0.0.1:30305）

```
bash ./tmp_one_click_expand/agencyC/node/node_127.0.0.1_30305/scripts/load_new_
↪groups.sh
curl -X POST --data '{"jsonrpc":"2.0","method":"startGroup","params":[2],"id":1}'␣
↪http://127.0.0.1:8550
```

Check node

View the group1 information in the node log:

```
tail -f ~/generator/tmp_one_click/agency*/node/node*/log/log* | grep g:2 | grep +++
```

```
info|2019-02-25 17:25:56.028692| [g:2][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=1,tx=0,myIdx=0,hash=833bd983...
info|2019-02-25 17:25:59.058625| [g:2][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=1,tx=0,myIdx=0,hash=343b1141...
info|2019-02-25 17:25:57.038284| [g:2][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=1,tx=0,myIdx=1,hash=ea85c27b...
```

So far, we have completed all the operations in the build tutorial shown.

After using it, it is recommended to clean the meta folder with the following command:

- rm ./meta/cert_* - rm ./meta/group*

12.1.6 More operations

For more operations, refer to the Operation Manual or Enterprise Tools Peer-to-Peer Deployment Tutorial.

If you have problems with this tutorial, please check FAQ

12.2 enterprise deployment tools

FISCO BCOS enterprise deployment tools are designed for multi-agency production environments. For ensure the security of the agency's private keys, enterprise deployment tools provide agencies' collaboration to deploy an alliance chain.

This chapter will demonstrate how to use enterprise deployment tools by deploying a 6 nodes 3 agencies 2 groups alliance chain. For more parameter options, please refer to here.

This chapter is a process that multi-agency peer-to-peer deployment and a situation that the private key of the agency does not come out of intranet. The tutorial for generating configuration files of all agency nodes through a single agency's click start can refer to FISCO BCOS Enterprise Deployment Tool ClickStart deployment.

12.2.1 Download and install

download

```
cd ~/
git clone https://github.com/FISCO-BCOS/generator.git

# If you have network issue for exec the command above, please try:
git clone https://gitee.com/FISCO-BCOS/generator.git
```

install

This operation requires sudo permission.

```
cd generator && bash ./scripts/install.sh
```

Check whether the installation is successful. If it is, output usage: generator xxx

```
./generator -h
```

download fisco-bcos binary

download the latest fisco-bcos binary to `./meta`, you can try `--cdn` to improve your download speed.

```
./generator --download_fisco ./meta
```

check fisco-bcos version

Output will be: FISCO-BCOS Version : x.x.x-x

```
./meta/fisco-bcos -v
```

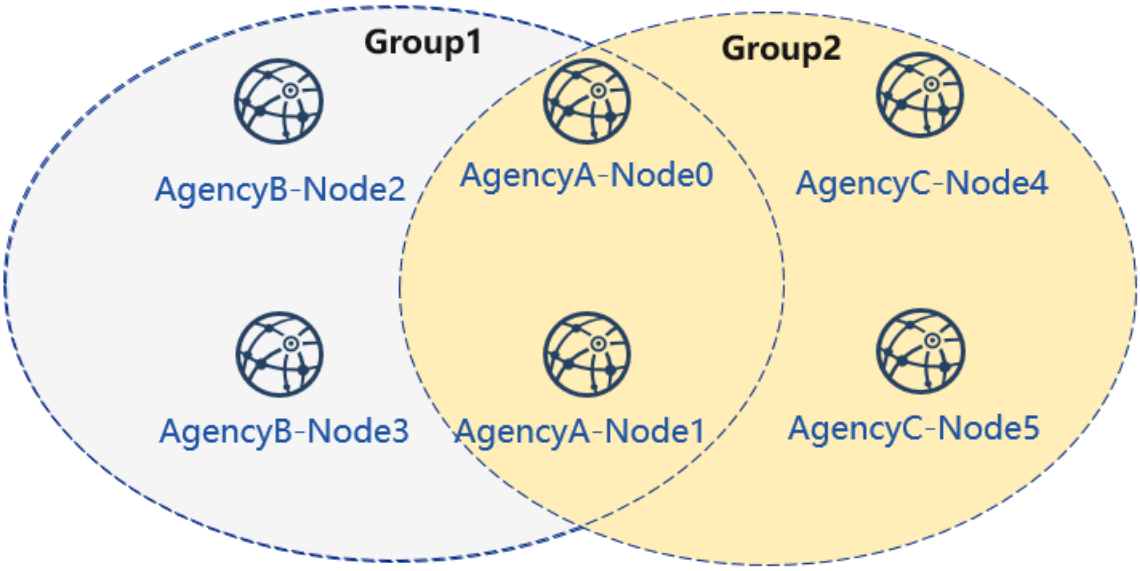
PS: If someone wants to use Source Code Compile fisco-bcos binary, they need to replace the binary in the `meta` folder with the compiled binary.

12.2.2 Typical example

For ensure the security of the agency's private keys, enterprise deployment tools provide a secure way to build chain between agencies. This chapter will demonstrate how to make a chain between agencies in a deployment model of 6 nodes 3 agencies 2 groups.

Node networking overview

A networking model of 6 nodes 3 agencies 2 groups is shown as follows. Agency B and agency C is located in Group 1 and Group 2, and agency A belongs to both Group 1 and Group 2.



Machine address

The IP address of each node and port are as follows:

Note:

- The public IP of the cloud host is a virtual IP. If you enter the external IP in rpc_ip/p2p_ip/channel_ip, the binding will fail. You must fill in 0.0.0.0
- The RPC/P2P/Channel listening port must be in the range of 1024-65535, and must not conflict with other application listening ports on the machine
- For security and ease of use consideration, FISCO BCOS v2.3.0 latest node config.ini configuration splits listen_ip into jsonrpc_listen_ip and channel_listen_ip, but still retains the parsing function of listen_ip, please refer to here
- In order to facilitate development and experience, the reference configuration of channel_listen_ip is 0.0.0.0. For security reasons, please modify it to a safe listening address according to the actual business network situation, such as: intranet IP or specific external IP

---

cooperate agencies

Building chain involves the cooperation between multiple agencies, including:

- Certificate authority agency
- alliance chain member agency(next named “agency”)

Key process

In this section, we briefly provide How Certificate authority agency and alliance chain member agency cooperate to build a blockchain.

1. Initialize chain certificate

1. Certificate authority agency operation:

- Generate chain certificate

2. Generate group 1

1. Certificate authority agency operations
   - generate agency certificate
   - send the certificate to agencies
2. Operation between agencies
   - modify the configuration file `node_deployment.ini`
   - generate node certificate and node P2P port address file `peers.txt`
3. Select one of the agencies to create `group.genesis`
   - collect all node certificates in the group
   - modify configuration file `group_genesis.ini`
   - generate genesis block files for the group
   - distribute genesis block files to other agencies
4. Operation between agencies: generating nodes
   - collect P2P port address files of other nodes in the group
   - generate node
   - start node

3. Initialize a new institution

1. Certificate authority agency operations
   - generate agency certificate
   - send the certificate to the new agency

4. Generate group2

1. New agency operation
   - modify the configuration file `node_deployment.ini`
   - generate node certificate and node P2P port address file
2. Select one of the agencies as a group to create genesis block
   - collect all node certificates in the group
   - modify configuration file `group_genesis.ini`
   - generate genesis block files for the group
   - distribute genesis block files to other agency
3. New agency independent operation: create nodes
   - collect P2P port address files of other nodes in the group
   - generate nodes
   - start nodes
4. Existing agency's operations: configure new groups

- collect P2P port address files of other nodes in the group
- configure P2P port address of the new group and the new nodes
- restart nodes

5. Existing nodes join group 1

1. Group 1 original agency operation:
   - send group 1 genesis block to the existing node
   - configure console
   - get the entering node nodeid
   - add nodes to group1 by using console

12.2.3 Chain initialization

All the operations in this example are performed on the local machine. We use different catalogs to simulate various agencies' environment and use the file copy operation to simulate the sending in the network. After performing `Download and Install` in the tutorial, please copy the generator to the corresponding agency's generator directory.

Institutional initialization

We use generator downloaded from the tutorial as certificate agency.

Initialize agencyA

```
cp -r ~/generator ~/generator-A
```

Initialize agencyB

```
cp -r ~/generator ~/generator-B
```

Initialize chain certificate

A single chain has a unique ca.crt.

use `--generate_chain_certificate` to generate chain certificate

Operate in the certificate agency directory:

```
cd ~/generator
```

```
./generator --generate_chain_certificate ./dir_chain_ca
```

view the chain certificate and the private key:

```
ls ./dir_chain_ca
```

```
# the above order has explained
# From left to right,  they are chain's certificate, and chain's private key.
ca.crt  ca.key
```

12.2.4 AgencyA, B to build group 1

Initialize agencyA

In the tutorial, for operating simply, the certificate of agency and the private key are directly generated. In actual application, the private key `agency.key` should be created locally by agency first, and then the certificate request file is made, and the certificate `agency.crt` is obtained from the certificate agency

Operate in the certificate directory:

```
cd ~/generator
```

Generate agencyA certificate:

```
./generator --generate_agency_certificate ./dir_agency_ca ./dir_chain_ca agencyA
```

View agency certificate and the private key:

```
ls dir_agency_ca/agencyA/
```

```
# From left to right, they are agency's certificate, and agency's private key
agency.crt  agency.key  ca.crt
```

For sending the chain certificate, agency certificate, and agency private key to agencyA, we use an example is to send the certificate from the certificate agency to the corresponding agency through the file copy, and put the certificate in the subdirectory of meta which is agency's working directory.

```
cp ./dir_agency_ca/agencyA/* ~/generator-A/meta/
```

Initialize agencyB

Operate in the certificate directory:

```
cd ~/generator
```

Generate agencyB certificate:

```
./generator --generate_agency_certificate ./dir_agency_ca ./dir_chain_ca agencyB
```

For sending the chain certificate, agency certificate, and agency private key to agencyB, we use an example is to send the certificate from the certificate agency to the corresponding agency through the file copy, and put the certificate in the subdirectory of meta which is agency's working directory.

```
cp ./dir_agency_ca/agencyB/* ~/generator-B/meta/
```

Important: Only one root certificate, ca.crt, can be used in an alliance chain. Do not generate multiple root certificates and private keys when deploying various servers. A group can only have one genesis block group.x.genesis.

AgencyA modifies configuration file

node_deployment.ini is the node configuration file. Enterprise deployment tool generates the corresponding node certificate according to the configuration of `node_deployment.ini` and the node configuration folder etc..

AgencyA modifies the `node_deployment.ini` in the conf folder, as shown below:

Execute the following command in the ~/generator-A directory

```
cd ~/generator-A
```

```
cat > ./conf/node_deployment.ini << EOF
[group]
group_id=1

[node0]
; host ip for the communication among peers.
; Please use your ssh login ip.
p2p_ip=127.0.0.1
; listen ip for the communication between sdk clients.
; This ip is the same as p2p_ip for physical host.
; But for virtual host e.g. vps servers, it is usually different from p2p_ip.
; You can check accessible addresses of your network card.
; Please see https://tecadmin.net/check-ip-address-ubuntu-18-04-desktop/
; for more instructions.
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30300
channel_listen_port=20200
jsonrpc_listen_port=8545

[node1]
p2p_ip=127.0.0.1
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30301
channel_listen_port=20201
jsonrpc_listen_port=8546
EOF
```

AgencyA modifies configuration file

AgencyB modifies the `node_deployment.ini` in the conf folder, as shown below:

Execute the following command in the ~/generator-B directory

```
cd ~/generator-B
```

```
cat > ./conf/node_deployment.ini << EOF
[group]
group_id=1

[node0]
; host ip for the communication among peers.
; Please use your ssh login ip.
p2p_ip=127.0.0.1
; listen ip for the communication between sdk clients.
; This ip is the same as p2p_ip for physical host.
; But for virtual host e.g. vps servers, it is usually different from p2p_ip.
; You can check accessible addresses of your network card.
; Please see https://tecadmin.net/check-ip-address-ubuntu-18-04-desktop/
; for more instructions.
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30302
channel_listen_port=20202
jsonrpc_listen_port=8547

[node1]
```

(continues on next page)

(continued from previous page)

```
p2p_ip=127.0.0.1
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30303
channel_listen_port=20203
jsonrpc_listen_port=8548
EOF
```

AgencyA generates and sends node information

Execute the following command in the ~/generator-A directory

```
cd ~/generator-A
```

AgencyA generates the node certificate and the P2P connection information file. In this step, we need to use the above configuration `node_deployment.ini` and the agency certificate and private key in the agency meta folder.

```
./generator --generate_all_certificates ./agencyA_node_info
```

view generated files:

```
ls ./agencyA_node_info
```

```
# From left to right, they are the node certificate that needs to have interacted␣
↪with the agencyA and the file that node P2P connects to the address (the node␣
↪information of agency generated by the node_deployment.ini)
cert_127.0.0.1_30300.crt cert_127.0.0.1_30301.crt peers.txt
```

When the agency generates a node, it needs to specify the node P2P connection address of other nodes. Therefore, AgencyA needs to send the node P2P connection address file to AgencyB.

```
cp ./agencyA_node_info/peers.txt ~/generator-B/meta/peersA.txt
```

AgencyB generates and sends node information

Execute the following command in the ~/generator-B directory

```
cd ~/generator-B
```

AgencyB generates the node certificate and the P2P connection information file:

```
./generator --generate_all_certificates ./agencyB_node_info
```

The agency that generates the genesis block needs the node certificate. In the example, the agencyA generates the genesis block. Therefore, in addition to sending the node P2P connection address file, the agencyB needs to send the node certificate to agencyA.

Send certificate to agencyA

```
cp ./agencyB_node_info/cert*.crt ~/generator-A/meta/
```

Send the node P2P connection address file to agencyA

```
cp ./agencyB_node_info/peers.txt ~/generator-A/meta/peersB.txt
```

AgencyA generates the genesis block of group1

Execute the following command in the ~/generator-A directory

```
cd ~/generator-A
```

AgencyA modifies `group_genesis.ini` in the conf folder. For configuration items, refer to Manuals:

```
cat > ./conf/group_genesis.ini << EOF
[group]
group_id=1

[nodes]
node0=127.0.0.1:30300
node1=127.0.0.1:30301
node2=127.0.0.1:30302
node3=127.0.0.1:30303
EOF
```

After the command is executed, the ./conf/group_genesis.ini file will be modified:

```
;command interpretation
[group]
;group id
group_id=1

[nodes]
;AgencyA node p2p address
node0=127.0.0.1:30300
;AgencyA node p2p address
node1=127.0.0.1:30301
;AgencyB node p2p address
node2=127.0.0.1:30302
;AgencyB node p2p address
node3=127.0.0.1:30303
```

In the tutorial, we choose agencyA to generate genesis block of the group. But in the actual production, you can negotiate with alliance chain committee to make a choice.

This step will generate the genesis block of group_genesis.ini according to the node certificate configured in the meta folder of agencyA. In the tutorial, the agencyA's meta is required to have the node certificates name as `cert_127.0.0.1_30300.crt`, `cert_127.0.0.1_30301.crt`, `cert_127.0.0.1_30302.crt`, `cert_127.0.0.1_30303.crt`. This step requires the node certificate of agencyB.

```
./generator --create_group_genesis ./group
```

Send group.1.genesis to AgencyB:

```
cp ./group/group.1.genesis ~/generator-B/meta
```

AgencyA generates the node to which it belongs

Execute the following command in the ~/generator-A directory

```
cd ~/generator-A
```

AgencyA generates the node to which it belongs. This command generates the corresponding node configuration folder according to the user-configured file `node_deployment.ini`:

Note: The node P2P connection information `peers.txt` specified in this step is the connect information of other nodes in the group. In the case of multiple agencies networking, they need to be merged.

```
./generator --build_install_package ./meta/peersB.txt ./nodeA
```

View the generated node configuration folder:

```
ls ./nodeA
```

```
# command interpretation, displayed in tree style here
# The generated folder nodeA information is as follows
├── monitor # monitor script
├── node_127.0.0.1_30300 # node configuration folder with server address 127.0.0.1
↪and port number 30300
├── node_127.0.0.1_30301
├── scripts # node related tool script
├── start_all.sh # node startups script in batch
└── stop_all.sh # node stops script in batch
```

AgencyA startups node:

```
bash ./nodeA/start_all.sh
```

View node process:

```
ps -ef | grep fisco
```

```
# command interpretation
# you can see the following process
fisco  15347     1  0 17:22 pts/2    00:00:00 ~/generator-A/nodeA/node_127.0.0.1_
↪30300/fisco-bcos -c config.ini
fisco  15402     1  0 17:22 pts/2    00:00:00 ~/generator-A/nodeA/node_127.0.0.1_
↪30301/fisco-bcos -c config.ini
```

AgencyB generates the node to which it belongs

Execute the following command in the ~/generator-B directory

```
cd ~/generator-B
```

AgencyB generates the node to which it belongs. This command generates the corresponding node configuration folder according to the user-configured file `node_deployment.ini`:

```
./generator --build_install_package ./meta/peersA.txt ./nodeB
```

AgencyB startups node:

```
bash ./nodeB/start_all.sh
```

Note: Startup node only needs to send the node folder corresponding to IP address. For example, the server of 127.0.0.1 only needs the node configuration folder corresponding to node_127.0.0.1_port. When deploying multiple machines, you only need to send the generated node folder to the corresponding server.

View group1 node running status

View process:

```
ps -ef | grep fisco
```

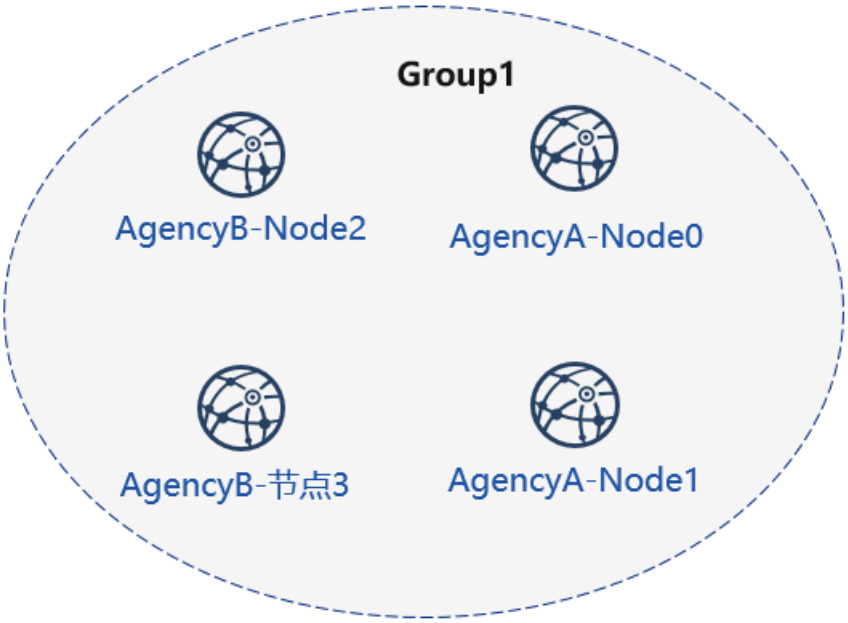
```
# command interpretation
# you can see the following process
fisco  15347     1  0 17:22 pts/2    00:00:00 ~/generator-A/nodeA/node_127.0.0.1_
↪30300/fisco-bcos -c config.ini
fisco  15402     1  0 17:22 pts/2    00:00:00 ~/generator-A/nodeA/node_127.0.0.1_
↪30301/fisco-bcos -c config.ini
fisco  15457     1  0 17:22 pts/2    00:00:00 ~/generator-B/nodeB/node_127.0.0.1_
↪30302/fisco-bcos -c config.ini
fisco  15498     1  0 17:22 pts/2    00:00:00 ~/generator-B/nodeB/node_127.0.0.1_
↪30303/fisco-bcos -c config.ini
```

view node log:

```
tail -f ./node*/node*/log/log*  | grep +++
```

```
# command interpretation
# +++ printed in log is the normal consensus of the node
info|2019-02-25 17:25:56.028692| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=1,tx=0,myIdx=0,hash=833bd983...
info|2019-02-25 17:25:59.058625| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=1,tx=0,myIdx=0,hash=343b1141...
info|2019-02-25 17:25:57.038284| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++␣
↪Generating seal on,blkNum=1,tx=0,myIdx=1,hash=ea85c27b...
```

By now, we have completed the operation of agencyA,B to build group1 as shown:



12.2.5 Certificate authority initialize agencyC

Operate in the certificate generator directory:

```
cd ~/generator
```

Initialize agencyC Note. Now there is a chain certificate and a private key in the generator directory. In the actual environment, agencyC cannot obtain the chain certificate and the private key.

```
cp -r ~/generator ~/generator-C
```

Generate agencyC certificate:

```
./generator --generate_agency_certificate ./dir_agency_ca ./dir_chain_ca agencyC
```

View agency certificate and private key:

```
ls dir_agency_ca/agencyC/
```

```
# command interpretation
# From left to right, they are agency's certificate, agency's private key, and␣
↪chain's certificate
agency.crt  agency.key  ca.crt
```

For sending the chain certificate, agency certificate, and agency private key to agencyA, we use an example is to send the certificate from the certificate agency to the corresponding agency through the file copy, and put the certificate in the subdirectory of meta which is agency's working directory.

```
cp ./dir_agency_ca/agencyC/* ~/generator-C/meta/
```

12.2.6 AgencyA,C build group2

Next, agencyC needs to perform a new group establishment operation with agencyA. We take an example of agencyC generating genesis block.

AgencyA sends node information

Since agencyA has generated the node certificate and the peers file, we only need to send the previous generated node P2P connection information and the node certificate to agencyC as follows:

Execute the following command in the ~/generator-A directory

```
cd ~/generator-A
```

In the example, the genesis block of the group is generated by agencyC. Therefore the node certificate of agencyA and the node P2P connection address file are required, and the above file is sent to agencyC.

Send certificate to agencyC

```
cp ./agencyA_node_info/cert*.crt ~/generator-C/meta/
```

Send node P2P connection address file to agencyC

```
cp ./agencyA_node_info/peers.txt ~/generator-C/meta/peersA.txt
```

AgencyC modifies configuration file

AgencyC modifies `node_deployment.ini` in the conf folder as shown below:

Execute the following command in the ~/generator-C directory.

```
cd ~/generator-C
```

```
cat > ./conf/node_deployment.ini << EOF
[group]
group_id=2

[node0]
; host ip for the communication among peers.
; Please use your ssh login ip.
```

(continues on next page)

(continued from previous page)

```
p2p_ip=127.0.0.1
; listen ip for the communication between sdk clients.
; This ip is the same as p2p_ip for physical host.
; But for virtual host e.g. vps servers, it is usually different from p2p_ip.
; You can check accessible addresses of your network card.
; Please see https://tecadmin.net/check-ip-address-ubuntu-18-04-desktop/
; for more instructions.
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30304
channel_listen_port=20204
jsonrpc_listen_port=8549

[node1]
p2p_ip=127.0.0.1
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30305
channel_listen_port=20205
jsonrpc_listen_port=8550
EOF
```

AgencyC generates and sends node information

Execute the following command in the ~/generator-C directory.

```
cd ~/generator-C
```

AgencyC generates node certificate and P2P connection information file:

```
./generator --generate_all_certificates ./agencyC_node_info
```

View generated file:

```
ls ./agencyC_node_info
```

```
# command interpretation
# From left to right, they are the node certificate that needs to have interacted␣
↪with the agencyA and the file that node P2P connects to the address (the node␣
↪information of agency generated by the node_deployment.ini)
cert_127.0.0.1_30304.crt cert_127.0.0.1_30305.crt peers.txt
```

When the agency generates a node, it needs to specify the node P2P connection address of other nodes. Therefore, agencyC needs to send the node P2P connection address file to agencyA.

```
cp ./agencyC_node_info/peers.txt ~/generator-A/meta/peersC.txt
```

AgencyC generates genesis block of group2

Execute the following command in the ~/generator-C directory.

```
cd ~/generator-C
```

AgencyC modifies `group_genesis.ini` in the conf folder as shown below:

```
cat > ./conf/group_genesis.ini << EOF
[group]
```

(continues on next page)

(continued from previous page)

```
group_id=2

[nodes]
node0=127.0.0.1:30300
node1=127.0.0.1:30301
node2=127.0.0.1:30304
node3=127.0.0.1:30305
EOF
```

./conf/group_genesis.ini file will be modified after the command is executed:

```
;command interpretation
[group]
group_id=2

[nodes]
node0=127.0.0.1:30300
;agencyA node p2p address
node1=127.0.0.1:30301
;agencyA node p2p address
node2=127.0.0.1:30304
;agencyC node p2p address
node3=127.0.0.1:30305
;agencyC node p2p address
```

In the tutorial, agency C is chosen to generate a genesis block of the group. In the actual production, you can negotiate with the alliance chain committee to determine.

This step generates a genesis block of group_genesis.ini configuration according to the node certificate configured in the meta folder of agencyC.

```
./generator --create_group_genesis ./group
```

Distribute genesis block of group2 to agencyA:

```
cp ./group/group.2.genesis ~/generator-A/meta/
```

AgencyC generates the node to which it belongs

Execute the following command in the ~/generator-C directory

```
cd ~/generator-C
```

```
./generator --build_install_package ./meta/peersA.txt ./nodeC
```

AgencyC startups node:

```
bash ./nodeC/start_all.sh
```

```
ps -ef | grep fisco
```

```
# command interpretation
# you can see the following process
fisco  15347     1  0 17:22 pts/2    00:00:00 ~/generator-A/nodeA/node_127.0.0.1_
↪30300/fisco-bcos -c config.ini
fisco  15402     1  0 17:22 pts/2    00:00:00 ~/generator-A/nodeA/node_127.0.0.1_
↪30301/fisco-bcos -c config.ini
fisco  15457     1  0 17:22 pts/2    00:00:00 ~/generator-B/nodeB/node_127.0.0.1_
↪30302/fisco-bcos -c config.ini
```

(continues on next page)

(continued from previous page)

```
fisco  15498     1  0 17:22 pts/2    00:00:00 ~/generator-B/nodeB/node_127.0.0.1_
↪30303/fisco-bcos -c config.ini
fisco  15550     1  0 17:22 pts/2    00:00:00 ~/generator-C/nodeC/node_127.0.0.1_
↪30304/fisco-bcos -c config.ini
fisco  15589     1  0 17:22 pts/2    00:00:00 ~/generator-C/nodeC/node_127.0.0.1_
↪30305/fisco-bcos -c config.ini
```

AgencyA initializes group2 for existing nodes

Execute the following command in the ~/generator-A directory

```
cd ~/generator-A
```

Add the group2 configuration file to the existing node. This step adds the genesis block of group2 `group.2.genesis` to all nodes under ./nodeA:

```
./generator --add_group ./meta/group.2.genesis ./nodeA
```

Add the agencyC node connect file peers to the existing node. This step adds the node P2P connection address of `peersC.txt` to all nodes under ./nodeA:

```
./generator --add_peers ./meta/peersC.txt ./nodeA
```

Restart agencyA node:

```
bash ./nodeA/stop_all.sh
```

```
bash ./nodeA/start_all.sh
```

View group2 node running status

View node's process:

```
ps -ef | grep fisco
```

```
# command interpretation
# you can see the following process
fisco  15347     1  0 17:22 pts/2    00:00:00 ~/generator-A/nodeA/node_127.0.0.1_
↪30300/fisco-bcos -c config.ini
fisco  15402     1  0 17:22 pts/2    00:00:00 ~/generator-A/nodeA/node_127.0.0.1_
↪30301/fisco-bcos -c config.ini
fisco  15457     1  0 17:22 pts/2    00:00:00 ~/generator-B/nodeB/node_127.0.0.1_
↪30302/fisco-bcos -c config.ini
fisco  15498     1  0 17:22 pts/2    00:00:00 ~/generator-B/nodeB/node_127.0.0.1_
↪30303/fisco-bcos -c config.ini
fisco  15550     1  0 17:22 pts/2    00:00:00 ~/generator-C/nodeC/node_127.0.0.1_
↪30304/fisco-bcos -c config.ini
fisco  15589     1  0 17:22 pts/2    00:00:00 ~/generator-C/nodeC/node_127.0.0.1_
↪30305/fisco-bcos -c config.ini
```

View node log：

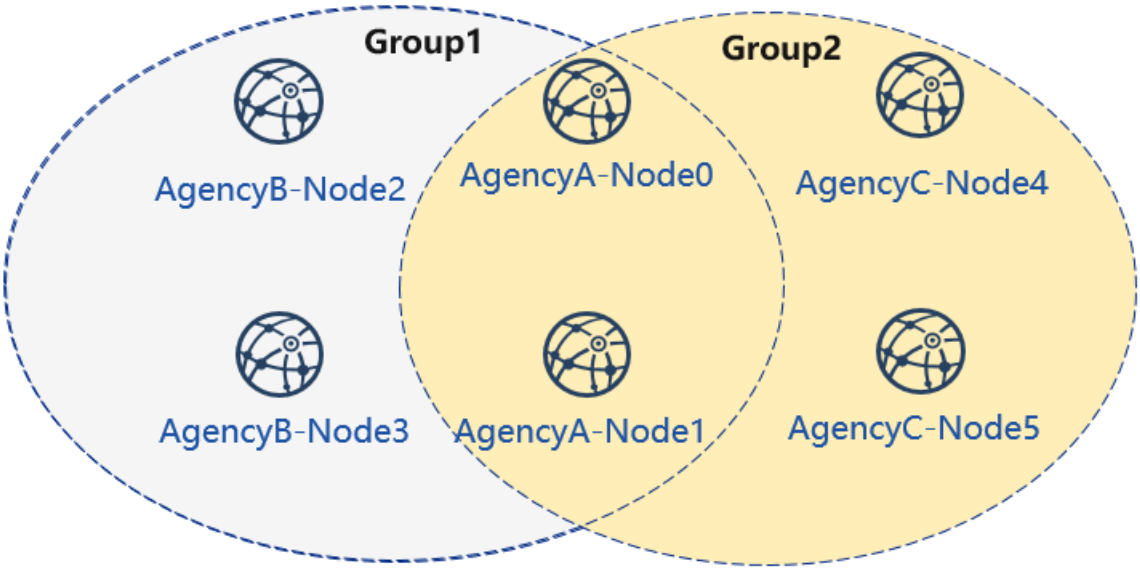
Execute the following command in the ~/generator-C directory

```
cd ~/generator-C
```

```
tail -f ./node*/node*/log/log*  | grep +++
```

```
# command interpretation
#+++ rinted in log is the normal consensus of the node
info|2019-02-25 17:25:56.028692| [g:2][p:264][CONSENSUS][SEALER]++++++++++++++++
↪Generating seal on,blkNum=1,tx=0,myIdx=0,hash=833bd983...
info|2019-02-25 17:25:59.058625| [g:2][p:264][CONSENSUS][SEALER]++++++++++++++++
↪Generating seal on,blkNum=1,tx=0,myIdx=0,hash=343b1141...
info|2019-02-25 17:25:57.038284| [g:2][p:264][CONSENSUS][SEALER]++++++++++++++++
↪Generating seal on,blkNum=1,tx=0,myIdx=1,hash=ea85c27b...
```

By now, we have completed the construction of agencyA, C to build group2 as shown:



12.2.7 Extended Tutorial–agencyC node joins group1

Adding a node to an existing group requires users to send command by console. The example of adding nodes to the group is as follows:

Now there are nodes of agencyA, B and B in group1. Adding the node of agencyC to group1 needs to get the permission of the nodes in the group. To take the node of agencyA as an example:

Execute the following command in the ~/generator-A directory.

```
cd ~/generator-A
```

Send gensis block of group1 to agencyC

Send the configuration file of group1 to agencyC.

```
./generator --add_group ./group/group.1.genesis  ~/generator-C/nodeC
```

Restart agencyC's node:

```
bash ~/generator-C/nodeC/stop_all.sh
```

```
bash ~/generator-C/nodeC/start_all.sh
```

Configure console

agencyA's configure console or Java SDK. In the tutorial, console is used as an example:

Note: This command will complete the console configuration according to the node and group in the user-configured `node_deployment.ini`. User can directly start the console. Please ensure that java is installed before starting.

```
./generator --download_console ./
```

View agencyC node4 information

AgencyA uses the console to join agencyC node4 as observation node. The second parameter needs to be replaced with the joining node 'nodeid', which locates in the `node.nodeid` file of the node folder conf.

View the agencyC node nodeid:

```
cat ~/generator-C/nodeC/node_127.0.0.1_30304/conf/node.nodeid
```

```
# command interpretation
# you can see a nodeid similar to the following. When using the console, you need␣
↪to pass this parameter.
ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c9715
```

Register observation node by using console

start console:

```
cd ~/generator-A/console && bash ./start.sh 1
```

Use the console `addObserver` command to register the node as an observation node. In this step, you need to use the `cat` command to view `node.nodeid` of agencyC node.

```
addObserver␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
```

```
# command interpretation
# Successful execution will prompt success
$ [group:1]> addObserver␣
↪ea2ca519148cafc3e92c8d9a8572b41ea2f62d0d19e99273ee18cccd34ab50079b4ec82fe5f4ae51bd95dd788811c971
{
        "code":0,
        "msg":"success"
}
```

exit console:

```
exit
```

View agencyC node 5 information

AgencyA uses console to join node 5 of agencyC as the consensus node. The second parameter needs to be replaced with the joining node 'nodeid', which locates in the `node.nodeid` file of the node folder conf.

View the agencyC node nodeid:

```
cat ~/generator-C/nodeC/node_127.0.0.1_30305/conf/node.nodeid
```

```
# command interpretation
# you can see a nodeid similar to the following. When using the console, you need␣
↪to pass this parameter.
5d70e046047e15a68aff8e32f2d68d1f8d4471953496fd97b26f1fbdc18a76720613a34e3743194bd78aa7acb59b9fa9a
```

Register consensus node by using console

Start console:

```
cd ~/generator-A/console && bash ./start.sh 1
```

Use the console's `addSealer` command to register the node as a consensus node. In this step, you need to use the `cat` command to view the `node.nodeid` of agencyC node.

```
addSealer␣
↪5d70e046047e15a68aff8e32f2d68d1f8d4471953496fd97b26f1fbdc18a76720613a34e3743194bd78aa7acb59b9fa
```

```
# command interpretation
# Successful execution will prompt success
$ [group:1]> addSealer␣
↪5d70e046047e15a68aff8e32f2d68d1f8d4471953496fd97b26f1fbdc18a76720613a34e3743194bd78aa7acb59b9fa
{
        "code":0,
        "msg":"success"
}
```

Exit console:

```
exit
```

View node

Execute the following command in the ~/generator-C directory

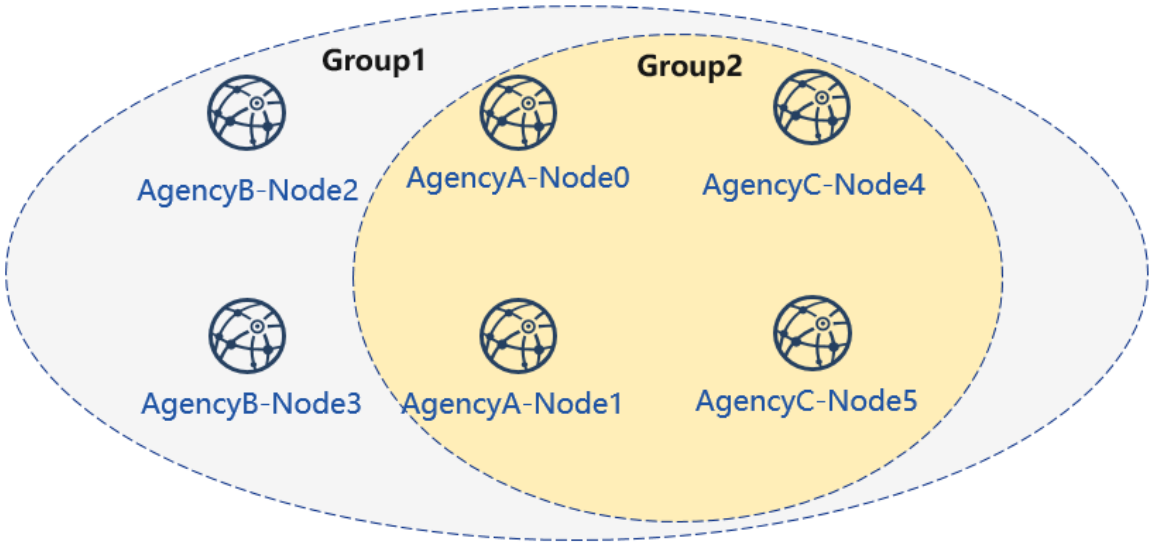
```
cd ~/generator-C
```

View the group1 information in node log:

```
cat node*/node_127.0.0.1_3030*/log/log* | grep g:1  | grep Report
```

```
# command interpretation
# Observation node will only synchronize transaction data, and will not␣
↪synchronize the consensus information in non-transaction status
# ^^^ is the transaction information of the node, and g:1 is the information␣
↪printed by group1
info|2019-02-26 16:01:39.914367| [g:1][p:65544][CONSENSUS][PBFT]^^^^^^^^Report,
↪num=0,sealerIdx=0,hash=9b76de5d...,next=1,tx=0,nodeIdx=65535
info|2019-02-26 16:01:40.121075| [g:1][p:65544][CONSENSUS][PBFT]^^^^^^^^Report,
↪num=1,sealerIdx=3,hash=46b7f17c...,next=2,tx=1,nodeIdx=65535
info|2019-02-26 16:03:44.282927| [g:1][p:65544][CONSENSUS][PBFT]^^^^^^^^Report,
↪num=2,sealerIdx=2,hash=fb982013...,next=3,tx=1,nodeIdx=65535
info|2019-02-26 16:01:39.914367| [g:1][p:65544][CONSENSUS][PBFT]^^^^^^^^Report,
↪num=0,sealerIdx=0,hash=9b76de5d...,next=1,tx=0,nodeIdx=4
info|2019-02-26 16:01:40.121075| [g:1][p:65544][CONSENSUS][PBFT]^^^^^^^^Report,
↪num=1,sealerIdx=3,hash=46b7f17c...,next=2,tx=1,nodeIdx=4
info|2019-02-26 16:03:44.282927| [g:1][p:65544][CONSENSUS][PBFT]^^^^^^^^Report,
↪num=2,sealerIdx=2,hash=fb982013...,next=3,tx=1,nodeIdx=4
```

By now, we have completed all the operations in the tutorial shown.



In this tutorial, we have generated a multi-group architecture alliance chain with a network topology of 3 agencies, 2 groups, and 6 nodes.

If you have problems with this tutorial, please view FAQ.

12.3 Download & install

12.3.1 Environment dependency

Dependency of FISCO BCOS generator:

12.3.2 Download & install

Download

```
$ git clone https://github.com/FISCO-BCOS/generator.git

# If you have network issue for exec the command above, please try:
$ git clone https://gitee.com/FISCO-BCOS/generator.git
```

Install

```
$ cd generator
$ bash ./scripts/install.sh
```

Check if it is installed successfully.

```
$ ./generator -h
# if succeed, output usage: generator xxx
```

12.3.3 Pull node binaries

Pull the latest fisco-bcos binary files to meta.

```
$ ./generator --download_fisco ./meta
```

Check binaries version.

```
$ ./meta/fisco-bcos -v
# if succeed, output FISCO-BCOS Version : x.x.x-x
```

PS: Source code compilation node binaries user need only to put the compiled binaries to `meta` directory.

12.4 Config file

The config files of FISCO BCOS generator are placed under ./conf folder, including group's genesis block config file `group_genesis.ini`, node config file `node_deployment.ini`.

The user configures node config file folder by operations on files under the conf folder.

12.4.1 Metadata folder meta

The meta folder of FISCO BCOS generator is metadata folder to store `fisco-bcos` binaries, chain certificate `ca.crt`, agency certificate `agency.crt`, agency private key and node certificate, group genesis block file, and so on.

The format of stored certificates should be cert_p2pip_port.crt. For example: cert_127.0.0.1_30300.crt.

FISCO BCOS generator will generate nodes config file according to the certificates under the meta folder and the config files under the conf folder.

12.4.2 group_genesis.ini

Through modifying the configuration of `group_genesis.ini`, the user generates a configuration of new group genesis block under the specific directory and meta folder. Such as `group.1.genesis`.

```
[group]
group_id=1

[nodes]
;node p2p address of group genesis block
node0=127.0.0.1:30300
node1=127.0.0.1:30301
node2=127.0.0.1:30302
node3=127.0.0.1:30303
```

Important: Node certificate is needed during generating genesis block. In the above case, the config file needs 4 nodes' certificates, which are: cert_127.0.0.1_30301.crt, cert_127.0.0.1_30302.crt, cert_127.0.0.1_30303.crt and cert_127.0.0.1_30304.crt.

12.4.3 node_deployment.ini

Through modifying `node_deployment.ini` configuration, user can use –build_install_package command to generate node config file containing no private key under a specific folder. Each `section[node]` configured by the user is the needed node config file folder. `section[peers]` is the p2p information for connection with other nodes.

For example:

```
[group]
group_id=1

# Owned nodes
[node0]
p2p_ip=127.0.0.1
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30300
channel_listen_port=20200
jsonrpc_listen_port=8545

[node1]
p2p_ip=127.0.0.1
rpc_ip=127.0.0.1
channel_ip=0.0.0.0
p2p_listen_port=30301
channel_listen_port=20201
jsonrpc_listen_port=8546
```

Read the node config command. To generate node certificate and node config file folder will need to read the config file.

12.4.4 Template folder tpl

The template folder of the generator is as below:

```
├── config.toml # sdk config file
├── config.ini # node config file template
├── config.ini.gm # OSCCA node config file template
├── group.i.genesis # group genesis block template
├── group.i.ini # group block configuration template
├── start.sh  # start node script template
├── start_all.sh # start nodes in batch script template
├── stop.sh # stop node script template
└── stop_all.sh # stop nodes in batch template
```

To modify the consensus algorithm of node and the configured default DB, the user only needs to alter the configuration of `config.ini`, re-execute the commands to set relative node information.

For details of FISCO BCOS configuration please check FISCO BCOS config file

12.4.5 P2p node connection file peers.txt

P2P node connection file `peers.txt` is the node connection information of the other agencies specified when generating node config file folder. When executing `build_install_package` command, it's needed to determine the p2p node connection file `peers.txt`, according to which node config file folder will start communication with other nodes.

User that executes `generate_all_certificates` command generates `peers.txt` according to the `node_deployment.ini` filled under `conf` directory. The user that adopts other ways to generate certificate needs to generate p2p node connection file manually and send to peers. The format of the p2p node connection file is:

```
127.0.0.1:30300
127.0.0.1:30301
```

Format like this: node ip:p2p_listen_port

- for multi-agency node communication, the files need to be combined

12.5 Operation Tutorial

FISCO BCOS generator contains multiple operations about node generation, expansion, group division, and certificate, which are introduced as below:

12.5.1 create_group_genesis (-c)

Operation Tutorial

```
$ cp node0/node.crt ./meta/cert_127.0.0.1_3030n.crt
...
$ vim ./conf/group_genesis.ini
$ ./generator --create_group_genesis ~/mydata
```

After the program is executed, `group.i.genesis` of mgroup.ini will be generated under ~/mydata folder.

The generated `group.i.genesis` is the Genesis Block of the new group.

Note: In FISCO BCOS 2.0+, each group has one Genesis Block.

12.5.2 build_install_package (-b)

Operation Tutorial

```
$ vim ./conf/node_deployment.ini
$ ./generator --build_install_package ./peers.txt ~/mydata
```

After the program is executed, a few file folders named node_hostip_port will be generated under ~/mydata folder and pushed to the relative server to activate the node.

12.5.3 generate_chain_certificate

```
$ ./generator --generate_chain_certificate ./dir_chain_ca
```

Now, user can find root certificate `ca.crt` and private key `ca.key` under ./dir_chain_ca folder.

12.5.4 generate_agency_certificate

```
$ ./generator --generate_agency_certificate ./dir_agency_ca ./chain_ca_dir The_
↪Agency_Name
```

Now, user can locate The_Agency_Name folder containing agency certificate `agency.crt` and private key `agency.key` through route ./dir_agency_ca.

12.5.5 generate_node_certificate

```
$ ./generator --generate_node_certificate node_dir(SET) ./agency_dir  node_p2pip_
↪port
```

Then user can locate node certificate `node.crt` and private key `node.key` through route node_dir.

12.5.6 generate_sdk_certificate

```
$ ./generator --generate_sdk_certificate ./dir_sdk_ca ./dir_agency_ca
```

Now user can locate SDK file folder containing SDK certificate `sdk.crt` and private key `sdk.key` through route ./sdk.

12.5.7 generate_all_certificates

```
$ ./generator --generate_all_certificates ./cert
```

Note: the above command will create node certificate according to ca.crt, agency.crt and agency.key of meta folder.

- absence of any of the above 3 files will fail the creation of node certificate, and the program will throw an exception

---

Once the execution is done, user can find node certificate and private key under ./cert folder, and node certificate under ./meta folder.

12.5.8 merge_config (-m)

–merge_config command can merge p2p sections of 2 config.ini

For instance, the p2p section in config.ini of A is:

```
[p2p]
listen_ip = 127.0.0.1
listen_port = 30300
node.0 = 127.0.0.1:30300
node.1 = 127.0.0.1:30301
node.2 = 127.0.0.1:30302
node.3 = 127.0.0.1:30303
```

The p2p section in config.ini of B is:

```
[p2p]
listen_ip = 127.0.0.1
listen_port = 30303
node.0 = 127.0.0.1:30300
node.1 = 127.0.0.1:30303
node.2 = 127.0.0.1:30300
node.3 = 127.0.0.1:30301
```

The command will result in:

```
[p2p]
listen_ip = 127.0.0.1
listen_port = 30304
node.0 = 127.0.0.1:30300
node.1 = 127.0.0.1:30301
node.2 = 127.0.0.1:30302
node.3 = 127.0.0.1:30303
node.4 = 127.0.0.1:30300
node.5 = 127.0.0.1:30301
```

For example:

```
$ ./generator --merge_config ~/mydata/node_A/config.ini  ~/mydata/node_B/config.ini
```

When it works, the p2p sections in config.ini of node_A and node_B will be merged to ~/mydata/node_B/config.ini

12.5.9 deploy_private_key (-d)

–deploy_private_key command will import the private key of nodes with same name to the generated configuration file folder

For example:

```
$./generator --deploy_private_key ./cert ./data
```

If ./cert contains file folders named node_127.0.0.1_30300 and node_127.0.0.1_30301, which are placed with node private key node.key,

./data contains file folders named node_127.0.0.1_30300 and node_127.0.0.1_30301,

then this command would import private key under./cert to ./data folder

12.5.10 add_peers (-p)

–add_peers command can import the files of assigned peers to the generated node configuration file folder.

For example:

```
$./generator --add_peers ./meta/peers.txt ./data
```

If ./data contains configuration file folder named node_127.0.0.1_30300 and node_127.0.0.1_30301, then this command will import peers file with connection information to the node configuration files `config.ini` under ./data.

add_group (-a)

–add_group command will import assigned peers file to the generated node configuration file folder.

For example:

```
$./generator --add_group ./meta/group.2.genesis ./data
```

If ./data contains configuration file folder named node_127.0.0.1_30300 and node_127.0.0.1_30301, then this command will import connection information of Group 2 to `conf` folder of all nodes under ./data.

12.5.11 download_fisco

–download_fisco can download `fisco-bcos` binary file under assigned section.

For example:

```
$./generator --download_fisco ./meta
```

This command will download `fisco-bcos` executable binary file under ./meta folder.

12.5.12 download_console

–download_consolecan download and control console under assigned section.

For example:

```
$./generator --download_console ./meta
```

This command will configure the console under ./meta folder according to `node_deployment.ini`.

12.5.13 get_sdk_file

–get_sdk_file command can acquire `sdk.crt`, `sdk.key`, `ca.crt` that are needed in configuration of console and sdk under assigned section.

For example:

```
$./generator --get_sdk_file ./sdk
```

This command will generate the above configuration file according to `node_deployment.ini` under ./sdk

12.5.14 version (-v)

–version command can help view the version code of current deployment tool.

```
$ ./generator --version
```

12.5.15 help (-h)

User can use -h or –help command to check help list

For example:

```
$ ./generator -h
usage: generator [-h] [-v] [-b peer_path data_dir] [-c data_dir]
                 [--generate_chain_certificate chain_dir]
                 [--generate_agency_certificate agency_dir chain_dir agency_name]
                 [--generate_node_certificate node_dir agency_dir node_name]
                 [--generate_sdk_certificate sdk_dir agency_dir] [-g]
                 [--generate_all_certificates cert_dir] [-d cert_dir pkg_dir]
                 [-m config.ini config.ini] [-p peers config.ini]
                 [-a group genesis config.ini]
```

12.5.16 Operation in OSCCA standard (China)

All the commands in FISCO BCOS generator are adaptable for OSCCA version of `fisco-bcos` (oscca-approved encryption). When using this version, every certificate and private key should be prefixed with `gm`. The description reads below:

On-off switch (-g command)

Once -g command is executed, all the operations concerning certificates, nodes, and group Genesis Block will generate the above files of OSCCA standard.

generate certificate

When executing generate_*_certificate together with -g command, the generated certificate will be in OSCCA standard.

For example:

```
./generator --generate_all_certificates ./cert -g
```

Note: The above command will generate node certificate according to gmca.crt, agency certificate gmagency.crt, and agency private key gmagency.key placed under meta folder.

- Absence of any one of the three files will fail the generation of node certificate, and the program will throw an exception.

generate group Genesis Block in OSCCA standard

For example:

```
$ cp node0/gmnode.crt ./meta/gmcert_127.0.0.1_3030n.crt
...
$ vim ./conf/group_genesis.ini
$ ./generator --create_group_genesis ~/mydata -g
```

After executing this program, the user can locate `group.i.genesis` in mgroup.ini under ~/mydata folder.

`group.i.genesis` is the Genesis Block of the new group.

generate node configuration file folder in OSCCA standard

For example:

```
$ vim ./conf/node_deployment.ini
$ ./generator --build_install_package ./peers.txt ~/mydata -g
```

After executing the program, multiple folders named node_hostip_port will be generated under ~/mydata folder and pushed to the relative server to activate the node.

12.5.17 Monitoring design

FISCO BCOS generator has a preset monitoring script in all generated node configuration folders. Warning information of node will be sent to the IP address assigned by the user after configuration. FISCO BCOS generator places the monitoring script under the assigned section for node configuration files. If the user assigns the folder named "data", then the user can locate it under the monitor folder of the data section.

For example:

```
$ cd ./data/monitor
```

Purpose of use:

1. to monitor the status of nodes, to reactive node
2. to acquire block number of node and view information, to ensure the consensus of nodes
3. to analyze printing of node logs in last 1 minutes, to collect critical errors information of printed logs, to monitor the status of nodes in real-time
4. to assign log file or period, to analyze data of consensus message management, block generation, and transaction volume and node's health.

Warning service configuration

Before using this service, the user needs to configure the warning service. Here we take the WeChat notification of server酱 as an example. You can read the configuration as reference: server酱

The user associates it with personal GitHub account and WeChat account, then uses -s command of this script to send a warning message to the associated WeChat.

If the user wants to try different services, the user can personalize the service by modifying alarm() {

```
# change HTTP server
```

} function of monitor.sh

Help command

The way to check the usage of the script by the help command

```
$ ./monitor.sh -h
Usage : bash monitor.sh
   -s : send alert to your address
   -m : monitor, statistics.  default : monitor .
   -f : log file to be analyzed.
   -o : dirpath
   -p : name of the monitored program , default is fisco-bcos
   -g : specified the group list to be analyzed
   -d : log analyze time range. default : 10(min), it should not bigger than max␣
↪value : 60(min).
   -r : setting alert receiver
   -h : help.
 example :
   bash  monitor.sh -s YourHttpAddr -o nodes -r your_name
   bash  monitor.sh -s YourHttpAddr -m statistics -o nodes -r your_name
   bash  monitor.sh -s YourHttpAddr -m statistics -f node0/log/log_2019021314.log -
↪g 1 2 -r your_name
```

Command description:

- -s assign the configuration address of warning service. It can be the ip of warning notification.
- -m set monitor mode to statistics or monitor. Monitor is the default mode.
- -f analyze node log
- -o assign location of node
- -p set notification name. “fisco-bcos” is the default one.
- -g assign group to be monitored. All group is defaulted to be monitored.
- -d time scope of log analysis. Default to be within 10 minutes and 60 minutes as the maximum
- -r assign the receiver of warning notification
- -h help command

For example

- use script to monitor status of assigned nodes, send message to receiver Alice:

```
$ bash monitor.sh -s https://sc.ftqq.com/[SCKEY(登入后可见)].send -o alice/nodes -r␣
↪Alice
```

- use script to collect node information and send message to Alice:

```
$ bash monitor.sh -s https://sc.ftqq.com/[SCKEY(登入后可见)].send -m statistics -o␣
↪alice/nodes -r Alice
```

- use script to collect information of group 1 and group 2 of specific node log, send message to Alice:

```
$ bash monitor.sh -s https://sc.ftqq.com/[SCKEY(replace by your code)].send -m␣
↪statistics -f node0/log/log_2019021314.log -g 1 2 -o alice/nodes -r Alice
```

12.5.18 handshake failed checked

The `check_certificates.sh` script in the scripts folder of FISCO BCOS generator contains anomaly detection with error message `handshake failed` in node log.

Acquire test script

If the user needs to test node generated by `build_chain.sh`, the following command can help acquire test script:

```
curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/generator/master/scripts/
↪check_certificates.sh && chmod u+x check_certificates.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/generator/raw/master/scripts/check_certificates.sh && chmod u+x check_certificates.sh

---

The user that deployed node with a generator can acquire script from scripts/check_certificates.sh under the root directory of the generator.

Test valid date of certificate

-t command of `check_certificates.sh` can test certificate by comparing the valid date with the current system time.

Example:

```
$ ./check_certificates.sh -t ~/certificates.crt
```

The second parameter can be any x509 certificate that prompt `check certificates time successfully` after passing verification or exception if it is failing.

Certificate verification

-v command of `check_certificates.sh` will verify node certificate according to the root certificate set by user.

```
$ ./check_certificates.sh -v ~/ca.crt ~/node.crt
```

After successful verification, it will prompt `use ~/ca.crt verify ~/node.crt successful`, or prompt exception if it is failing.

12.5.19 Node configuration error checking

Acquire script

```
curl -#LO https://raw.githubusercontent.com/FISCO-BCOS/FISCO-BCOS/master-2.0/tools/
↪check_node_config.sh && chmod u+x check_node_config.sh
```

Note:

- If the script cannot be downloaded for a long time due to network problems, try curl -#LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/master-2.0/tools/check_node_config.sh

使用

Using the following command, the script `-p` option specifies the node path, and the script will analyze the configuration errors based on config. ini under the path.

```
bash check_node_config.sh -p node_127.0.0.1_30300
```

CHAPTER

THIRTEEN

JSON-RPC API

The following examples in this chapter adopt the curl command, which is a data transfer tool that runs the command line by the URL language. JSON-RPC API of FISCO BCOS can be accessed by sending HTTP post request through curl commands. The URL address of the curl command is set as `[jsonrpc_listen_ip]`(If the node is less than v2.3.0, set as the configuration item `listen_ip`) and `[jsonrpc listen port]` of `[rpc]` in a node config file. To format the json data, jq is used as a formatter. For the error codes, please check the RPC Design Documentation. For the transaction receipt status, please check here.

13.1 Error codes

13.1.1 RPC error code

When a rpc call is made, the Server will reply with a response, which contains error result field, which includes as follows:

- code: A Number that indicates the error type that occurred.
- message: A String providing a short description of the error.
- data: A Primitive or Structured value that contains additional information about the error. This may be omitted.

There are 2 types of error code: JSON-RPC standard error code and FISCO BCOS RPC error code.

JSON-RPC standard error code

Standard error codes and their corresponding meanings are as follows:

FISCO BCOS RPC error code

FISCO BCOS RPC error codes and their corresponding meanings are as follows:

13.2 Transaction receipt status list

13.2.1 Precompiled Service API error code

13.2.2 Dynamice group management API status code

13.3 getClientVersion

Returns the current node version.

13.3.1 Parameters

none

13.3.2 Returns

- `object` - An object with version data:
  - `Build Time`: `string` - compile time
  - `Build Type`: `string` - compile machine environment
  - `Chain Id`: `string` - blockchain id
  - `FISCO-BCOS Version`: `string` - The version of the node
  - `Git Branch`: `string` - version branch
  - `Git Commit Hash`: `string` - latest commit hash
  - `Supported Version`: `string` - The supported version of the node
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getClientVersion","params":[],"id
↪":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 83,
  "jsonrpc": "2.0",
  "result": {
    "Build Time": "20190106 20:49:10",
    "Build Type": "Linux/g++/RelWithDebInfo",
    "Chain Id": "1",
    "FISCO-BCOS Version": "2.0.0",
    "Git Branch": "master",
    "Git Commit Hash": "693a709ddab39965d9c39da0104836cfb4a72054",
    "Supported Version": "2.0.0-rc3"
  }
}
```

13.4 getBlockNumber

Returns the number of most recent block in the specific group.

13.4.1 Parameters

- `groupID`: `unsigned int` - group ID

13.4.2 Returns

- `string` - the highest block number (a hexadecimal string start with 0x)
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getBlockNumber","params":[1],"id
↪":1}' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": "0x1"
}
```

13.5 getPbftView

Returns the latest PBFT View in the specific group.

13.5.1 Parameters

- `groupID`: `unsigned int` - group ID

13.5.2 Returns

- `string` - PBFT view (a hexadecimal string start with 0x)
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getPbftView","params":[1],"id":1}'␣
↪http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": "0x1a0"
}
```

Note: FISCO BCOS supports PBFT Consensus and Raft Consensus. When the blockchain adopts Raft consensus, the error is:

```
{
  "error": {
    "code": 7,
    "data": null,
    "message": "Only pbft consensus supports the view property"
  },
  "id": 1,
  "jsonrpc": "2.0"
}
```

13.6 getSealerList

Returns the consensus node list in the specific group.

13.6.1 Parameters

- `groupID`: `unsigned int` - group ID

13.6.2 Returns

- `array` - consensus node ID list
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getSealerList","params":[1],"id":1}
↪' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": [

↪"037c255c06161711b6234b8c0960a6979ef039374ccc8b723afea2107cba3432dbbc837a714b7da20111f74d5a24e9
↪",

↪"0c0bbd25152d40969d3d3cee3431fa28287e07cff2330df3258782d3008b876d146ddab97eab42796495bfbb281591
↪",

↪"622af37b2bd29c60ae8f15d467b67c0a7fe5eb3e5c63fdc27a0ee8066707a25afa3aa0eb5a3b802d3a8e5e26de9d5a
↪"
    ]
}
```

13.7 getObserverList

Returns the observer node list in the specific group.

13.7.1 Parameters

- `groupID`: `unsigned int` - group ID

13.7.2 Returns

- `array` - observer node ID list
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getObserverList","params":[1],"id
↪":1}' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": [

↪"10b3a2d4b775ec7f3c2c9e8dc97fa52beb8caab9c34d026db9b95a72ac1d1c1ad551c67c2b7fdc34177857eada7583
↪"
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

13.8 getConsensusStatus

Returns the consensus status data in the specific group.

13.8.1 Parameters

- `groupID`: `unsigned int` - group ID

13.8.2 Returns

- `object` - An object with consensus status data.
- When PBFT consensus mechanism is used,（PBFT design is introduced in PBFT Design Documentation), the fields as follows:
  - `accountType`: `unsigned int` - account type
  - `allowFutureBlocks`: `bool` - allow future blocks
  - `cfgErr`: `bool` - configure errors
  - `connectedNodes`: `unsigned int` - connected nodes
  - `consensusedBlockNumber`: `unsigned int` - the latest consensus block number
  - `currentView`: `unsigned int` - the current view
  - `groupId`: `unsigned int` - group ID
  - `highestblockHash`: `string` - the hash of the highest block
  - `highestblockNumber`: `unsigned int` - the highest block number
  - `leaderFailed`: `bool` - leader failed
  - `max_faulty_leader`: `unsigned int` - the max number of faulty nodes
  - `sealer.index`: `string` - node ID with sequence number "index"
  - `node index`: `unsigned int` - sequence number of node
  - `nodeId`: `string` - node ID
  - `nodeNum`: `unsigned int` - number of nodes
  - `omitEmptyBlock`: `bool` - omit empty block
  - `protocolId`: `unsigned int` - protocol ID
  - `toView`: `unsigned int` - current view value
- When Raft consensus mechanism is adopted (Raft design is introduced in Raft Design Documentation), the fields as follows:
  - `accountType`: `unsigned int` - account type
  - `allowFutureBlocks`: `bool` - allow future blocks
  - `cfgErr`: `bool` - configure error
  - `consensusedBlockNumber`: `unsigned int` - the latest consensus block number
  - `groupId`: `unsigned int` - group ID

  - highestblockHash: string - hash of the latest block
  - highestblockNumber: unsigned int - the highest block number
  - leaderId: string - leader node ID
  - leaderIdx: unsigned int - leader node sequence number
  - max_faulty_leader: unsigned int - the max number of faulty nodes
  - sealer.index: string - node ID with sequence number "index"
  - node index: unsigned int - index of node
  - nodeId: string - node ID
  - nodeNum: unsigned int - number of nodes
  - omitEmptyBlock: bool - omit empty block
  - protocolId: unsigned int - protocol ID

- Example

```
// Request PBFT
curl -X POST --data '{"jsonrpc":"2.0","method":"getConsensusStatus","params":[1],
↪"id":1}' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": [
        {
  "id": 1,
  "jsonrpc": "2.0",
  "result": [
    {
      "accountType": 1,
      "allowFutureBlocks": true,
      "cfgErr": false,
      "connectedNodes": 3,
      "consensusedBlockNumber": 38207,
      "currentView": 54477,
      "groupId": 1,
      "highestblockHash":
↪"0x19a16e8833e671aa11431de589c866a6442ca6c8548ba40a44f50889cd785069",
      "highestblockNumber": 38206,
      "leaderFailed": false,
      "max_faulty_leader": 1,
      "nodeId":
↪"f72648fe165da17a889bece08ca0e57862cb979c4e3661d6a77bcc2de85cb766af5d299fec8a4337eedd142dca026a
↪",
      "nodeNum": 4,
      "node_index": 3,
      "omitEmptyBlock": true,
      "protocolId": 65544,
      "sealer.0":
↪"6a99f357ecf8a001e03b68aba66f68398ee08f3ce0f0147e777ec77995369aac470b8c9f0f85f91ebb58a98475764b
↪",
      "sealer.1":
↪"8a453f1328c80b908b2d02ba25adca6341b16b16846d84f903c4f4912728c6aae1050ce4f24cd9c13e010ce922d339
↪",
      "sealer.2":
↪"ed483837e73ee1b56073b178f5ac0896fa328fc0ed418ae3e268d9e9109721421ec48d68f28d6525642868b40dd265
↪",
      "sealer.3":
↪"f72648fe165da17a889bece08ca0e57862cb979c4e3661d6a77bcc2de85cb766af5d299fec8a4337eedd142dca026a
↪",
```

(continues on next page)

(continued from previous page)

```
      "toView": 54477
    },
    [
      {
        "nodeId":
↪"6a99f357ecf8a001e03b68aba66f68398ee08f3ce0f0147e777ec77995369aac470b8c9f0f85f91ebb58a98475764b
↪",
        "view": 54474
      },
      {
        "nodeId":
↪"8a453f1328c80b908b2d02ba25adca6341b16b16846d84f903c4f4912728c6aae1050ce4f24cd9c13e010ce922d339
↪",
        "view": 54475
      },
      {
        "nodeId":
↪"ed483837e73ee1b56073b178f5ac0896fa328fc0ed418ae3e268d9e9109721421ec48d68f28d6525642868b40dd265
↪",
        "view": 54476
      },
      {
        "nodeId":
↪"f72648fe165da17a889bece08ca0e57862cb979c4e3661d6a77bcc2de85cb766af5d299fec8a4337eedd142dca026a
↪",
        "view": 54477
      }
    ]
  ]
}
}

// Request Raft
curl -X POST --data '{"jsonrpc":"2.0","method":"getConsensusStatus","params":[1],
↪"id":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": [
    {
      "accountType": 1,
      "allowFutureBlocks": true,
      "cfgErr": false,
      "consensusedBlockNumber": 1,
      "groupId": 1,
      "highestblockHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
      "highestblockNumber": 0,
      "leaderId":
↪"d5b3a9782c6aca271c9642aea391415d8b258e3a6d92082e59cc5b813ca123745440792ae0b29f4962df568f8ad58b
↪",
      "leaderIdx": 3,
      "max_faulty_leader": 1,
      "sealer.0":
↪"29c34347a190c1ec0c4507c6eed6a5bcd4d7a8f9f54ef26da616e81185c0af11a8cea4eacb74cf6f61820292b24bc5
↪",
      "sealer.1":
↪"41285429582cbfe6eed501806391d2825894b3696f801e945176c7eb2379a1ecf03b36b027d72f480e89d15bacd434
↪",
```

(continues on next page)

(continued from previous page)


```
      "sealer.2":
↪"87774114e4a496c68f2482b30d221fa2f7b5278876da72f3d0a75695b81e2591c1939fc0d3fadb15cc359c997bafc9e
↪",
      "sealer.3":
↪"d5b3a9782c6aca271c9642aea391415d8b258e3a6d92082e59cc5b813ca123745440792ae0b29f4962df568f8ad58b7
↪",
      "node index": 1,
      "nodeId":
↪"41285429582cbfe6eed501806391d2825894b3696f801e945176c7eb2379a1ecf03b36b027d72f480e89d15bacd434
↪",
      "nodeNum": 4,
      "omitEmptyBlock": true,
      "protocolId": 267
    }
  ]
}
```


13.9 getSyncStatus

Returns the sync status data in the specific group.

13.9.1 Parameters

- `groupID`: `unsigned int` - group ID

13.9.2 Returns

- `object` - An object with sync status information:
  - `blockNumber`: `unsigned int` - the highest block number
  - `genesisHash`: `string` - hash of genesis block
  - `isSyncing`: `bool` - syncing
  - `knownHighestNumber`: `unsigned int` - the highest number of the blockchain known by the node
  - `knownLatestHash`: `string` - the latest hash of the blockchain known by the node
  - `latestHash`: `string` - hash of the latest block
  - `nodeId`: `string` - node ID
  - `protocolId`: `unsigned int` - protocol ID
  - `txPoolSize`: `string` - transaction volume in txPool
  - `peers`: `array` - connected p2p nodes in the specific group, fields of node information are:
    - `blockNumber`: `unsigned int` - the latest block number
    - `genesisHash`: `string` - hash of genesis block
    - `latestHash`: `string` - hash of the latest block
    - `nodeId`: `string` - node ID
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getSyncStatus","params":[1],"id":1}
↪' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "blockNumber": 0,
    "genesisHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
    "isSyncing": false,
    "knownHighestNumber":0,
    "knownLatestHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
    "latestHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
    "nodeId":
↪"41285429582cbfe6eed501806391d2825894b3696f801e945176c7eb2379a1ecf03b36b027d72f480e89d15bacd434
↪",
    "peers": [
      {
        "blockNumber": 0,
        "genesisHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
        "latestHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
        "nodeId":
↪"29c34347a190c1ec0c4507c6eed6a5bcd4d7a8f9f54ef26da616e81185c0af11a8cea4eacb74cf6f61820292b24bc5
↪"
      },
      {
        "blockNumber": 0,
        "genesisHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
        "latestHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
        "nodeId":
↪"87774114e4a496c68f2482b30d221fa2f7b5278876da72f3d0a75695b81e2591c1939fc0d3fadb15cc359c997bafc9
↪"
      },
      {
        "blockNumber": 0,
        "genesisHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
        "latestHash":
↪"0x4765a126a9de8d876b87f01119208be507ec28495bef09c1e30a8ab240cf00f2",
        "nodeId":
↪"d5b3a9782c6aca271c9642aea391415d8b258e3a6d92082e59cc5b813ca123745440792ae0b29f4962df568f8ad58b
↪"
      }
    ],
    "protocolId": 265,
    "txPoolSize": "0"
  }
}
```

13.10 getPeers

Returns the connected p2p node data.

13.10.1 Parameters

- `groupID`: `unsigned int` - group ID

13.10.2 Returns

- `array` - The connected p2p node data:
  - `IPAndPort`: `string` - The IP and port of a node
  - `nodeId`: `string` - node ID
  - `Topic`: `array` - The topic data followed by a node
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getPeers","params":[1],"id":1}'␣
↪http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": [
        {
            "IPAndPort": "127.0.0.1:30308",
            "nodeId":
↪"0701cc9f05716690437b78db5b7c9c97c4f8f6dd05794ba4648b42b9267ae07cfcd589447ac36c491e760424214960
↪",
            "Topic": [ ]
        },
        {
            "IPAndPort": "127.0.0.1:58348",
            "nodeId":
↪"353ab5990997956f21b75ff5d2f11ab2c6971391c73585963e96fe2769891c4bc5d8b7c3d0d04f50ad6e04c4445c09
↪",
            "Topic": [ ]
        },
        {
            "IPAndPort": "127.0.0.1:30300",
            "nodeId":
↪"73aebaea2baa9640df416d0e879d6e0a6859a221dad7c2d34d345d5dc1fe9c4cda0ab79a7a3f921dfc9bdea4a49bb3
↪",
            "Topic": [ ]
        }
    ]
}
```

13.11 getGroupPeers

Returns the consensus node and observer node list in specific group.

13.11.1 Parameters

- `groupID`: `unsigned int` - group ID

13.11.2 Returns

- `array` - The consensus node and observer node ID list
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getGroupPeers","params":[1],"id":1}
↪' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": [

↪"0c0bbd25152d40969d3d3cee3431fa28287e07cff2330df3258782d3008b876d146ddab97eab42796495bfbb281591
↪",

↪"037c255c06161711b6234b8c0960a6979ef039374ccc8b723afea2107cba3432dbbc837a714b7da20111f74d5a24e9
↪",

↪"622af37b2bd29c60ae8f15d467b67c0a7fe5eb3e5c63fdc27a0ee8066707a25afa3aa0eb5a3b802d3a8e5e26de9d5a
↪",

↪"10b3a2d4b775ec7f3c2c9e8dc97fa52beb8caab9c34d026db9b95a72ac1d1c1ad551c67c2b7fdc34177857eada7583
↪"
    ]
}
```

13.12 getNodeIDList

Returns the node and connected p2p node list.

13.12.1 Parameters

- `groupID`: `unsigned int` - group ID

13.12.2 Returns

- `array` - The node and connected p2p node ID list
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getNodeIDList","params":[1],"id":1}
↪' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": [
```

(continues on next page)

(continued from previous page)

```
↪"0c0bbd25152d40969d3d3cee3431fa28287e07cff2330df3258782d3008b876d146ddab97eab42796495bfbb281591
↪",

↪"037c255c06161711b6234b8c0960a6979ef039374ccc8b723afea2107cba3432dbbc837a714b7da20111f74d5a24e9
↪",

↪"622af37b2bd29c60ae8f15d467b67c0a7fe5eb3e5c63fdc27a0ee8066707a25afa3aa0eb5a3b802d3a8e5e26de9d5a
↪",

↪"10b3a2d4b775ec7f3c2c9e8dc97fa52beb8caab9c34d026db9b95a72ac1d1c1ad551c67c2b7fdc34177857eada7583
↪"
    ]
}
```

13.13 getGroupList

Returns the group ID list where the node belongs.

13.13.1 Parameters

none

13.13.2 Returns

- `array` - group ID list
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getGroupList","params":[],"id":1}'␣
↪http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": [1]
}
```

13.14 getBlockByHash

Returns information about a block by hash.

13.14.1 Parameters

- `groupID`: `unsigned int` - group ID
- `blockHash`: `string` - hash of a block
- `includeTransactions`: `bool` - If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

13.14.2 Returns

- `object` - A block object:
  - `dbHash`: `string` - hash that records transaction data changes
  - `extraData`: `array` - extra data
  - `gasLimit`: `string` - the maximum gas allowed in block
  - `gasUsed`: `string` - gas used by all transactions
  - `hash`: `string` - block hash
  - `logsBloom`: `string` - bloom filter value of log
  - `number`: `string` - block height
  - `parentHash`: `string` - hash of parent block
  - `receiptsRoot`: `string` - the merkle root of all transaction receipts in the block
  - `sealer`: `string` - consensus node sequence number
  - `sealerList`: `array` - consensus nodes list
  - `stateRoot`: `string` - hash of state root
  - `timestamp`: `string` - time stamp
  - `signatureList`: `string` - PBFT consensus signature list
  - `transactions`: `array` - transaction list, when `includeTransactions` is `false`, it shows the hash of transaction; when `includeTransactions` is `true`, it shows transaction details (detail fields please check getTransactionByHash)
  - `transactionsRoot`: `string` - The merkle root of all transactions in the block
- Example

```
// Request
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getBlockByHash","params":[1,
↪"0xfa639d1454362a8cdfcab1ca1948a5defaf7048b28f67e80780ab1e24e8f8c59",true],"id
↪":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "dbHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData": [],
    "gasLimit": "0x0",
    "gasUsed": "0x0",
    "hash": "0xfa639d1454362a8cdfcab1ca1948a5defaf7048b28f67e80780ab1e24e8f8c59",
    "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number": "0x1",
    "parentHash":
↪"0x249f59e00beac8424a7821c4750fdd70c128f4ce795afbab53f345e9fce95d1a",
    "receiptsRoot":
↪"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
    "sealer": "0x0",
    "sealerList": [

↪"4ca3a91a4937355dba6a2e5fe76141479a1fc44e9caa86750092dab64e0b8382f6b8476749c2d2de414350a5449162
↪"
```

(continues on next page)

(continued from previous page)

```
    ],
    "signatureList": [
      {
        "index": "0x0",
        "signature":
↪"0x4602135870d9a4846e2536d4a48e831825a5d95768dd0d4f08544a0bd4c2af41242dec1751a05c07d7572027f8d6a
↪"
      }
    ],
    "stateRoot":
↪"0x0000000000000000000000000000000000000000000000000000000000000000",
    "timestamp": "0x175ac38cf10",
    "transactions": [
      {
        "blockHash":
↪"0xfa639d1454362a8cdfcab1ca1948a5defaf7048b28f67e80780ab1e24e8f8c59",
        "blockLimit": "0x100",
        "blockNumber": "0x1",
        "chainId": "0x1",
        "extraData": "0x",
        "from": "0x57c7be32cbfb3bfed4fddc87efcc735b4e945fb3",
        "gas": "0x2faf080",
        "gasPrice": "0xa",
        "groupId": "0x1",
        "hash": "0x3961fac263d8e640b148ddcfafd71d2069e93a006abc937c32fb16cfa96e661d
↪",
        "input":
↪"0x4ed3885e0000000000000000000000000000000000000000000000000000000000000020000000000000000000000
↪",
        "nonce": "0x3eb675ec791c2d19858c91d0046821c27d815e2e9c151604912205000002968
↪",
        "signature": {
          "r": "0x9edf7c0cb63645442aff11323916d51ec5440de979950747c0189f338afdcefd
↪",
          "s": "0x2f3473184513c6a3516e066ea98b7cfb55a79481c9db98e658dd016c37f03dcf
↪",
          "signature":
↪"0x9edf7c0cb63645442aff11323916d51ec5440de979950747c0189f338afdcefd2f3473184513c6a3516e066ea98b
↪",
          "v": "0x0"
        },
        "to": "0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744",
        "transactionIndex": "0x0",
        "value": "0x0"
      }
    ],
    "transactionsRoot":
↪"0xb880b08df3b43a9ffc334d7a526522b33e004ef95403d61d76454b6085b9b2f1"
  }
}

// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getBlockByHash","params":[1,
↪"0xfa639d1454362a8cdfcab1ca1948a5defaf7048b28f67e80780ab1e24e8f8c59",false],"id
↪":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
```

(continues on next page)

(continued from previous page)


```
    "dbHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData": [],
    "gasLimit": "0x0",
    "gasUsed": "0x0",
    "hash": "0xfa639d1454362a8cdfcab1ca1948a5defaf7048b28f67e80780ab1e24e8f8c59",
    "logsBloom":
↪"0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number": "0x1",
    "parentHash":
↪"0x249f59e00beac8424a7821c4750fdd70c128f4ce795afbab53f345e9fce95d1a",
    "receiptsRoot":
↪"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
    "sealer": "0x0",
    "sealerList": [

↪"4ca3a91a4937355dba6a2e5fe76141479a1fc44e9caa86750092dab64e0b8382f6b8476749c2d2de414350a5449162
↪"
    ],
    "stateRoot":
↪"0x0000000000000000000000000000000000000000000000000000000000000000",
    "timestamp": "0x175ac38cf10",
    "transactions": [
      "0x3961fac263d8e640b148ddcfafd71d2069e93a006abc937c32fb16cfa96e661d"
    ],
    "transactionsRoot":
↪"0xb880b08df3b43a9ffc334d7a526522b33e004ef95403d61d76454b6085b9b2f1"
  }
}
```


13.15 getBlockByNumber

Returns information about a block by block number.

13.15.1 Parameters

- `groupID`: `unsigned int` - group ID
- `blockNumber`: `string` - block number (hexadecimal string starts with 0x)
- `includeTransactions`: `bool` - If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

13.15.2 Returns

please check getBlockByHash

- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getBlockByNumber","params":[1,"0x0
↪",true],"id":1}' http://127.0.0.1:8545 |jq
```

See the result in getBlockByHash

13.16 getBlockHeaderByHash

Obtain block header information based on block hash.

13.16.1 Parameters

- `groupID`: `unsigned int` - group ID
- `blockHash`: `string` - block hash
- `includeSignatures`: `bool` - Contains signature list or not (true displays signature list).

13.16.2 Returns

```
//Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getBlockHeaderByHash","params":[1,
↪"0x99576e7567d258bd6426ddaf953ec0c953778b2f09a078423103c6555aa4362d",true],"id
↪":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "dbHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData": [],
    "gasLimit": "0x0",
    "gasUsed": "0x0",
    "hash": "0x99576e7567d258bd6426ddaf953ec0c953778b2f09a078423103c6555aa4362d",
    "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
    "number": "0x1",
    "parentHash":
↪"0x4f6394763c33c1709e5a72b202ad4d7a3b8152de3dc698cef6f675ecdaf20a3b",
    "receiptsRoot":
↪"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
    "sealer": "0x2",
    "sealerList": [

↪"11e1be251ca08bb44f36fdeedfaeca40894ff80dfd80084607a75509edeaf2a9c6fee914f1e9efda571611cf4575a1
↪",

↪"78a313b426c3de3267d72b53c044fa9fe70c2a27a00af7fea4a549a7d65210ed90512fc92b6194c14766366d434235
↪",

↪"95b7ff064f91de76598f90bc059bec1834f0d9eeb0d05e1086d49af1f9c2f321062d011ee8b0df7644bd54c4f9ca3d
↪",

↪"b8acb51b9fe84f88d670646be36f31c52e67544ce56faf3dc8ea4cf1b0ebff0864c6b218fdcd9cf9891ebd414a9958
↪"
    ],
    "signatureList": [
      {
        "index": "0x2",
        "signature":
↪"0xae098aabc63a53b8dcb57da9a87f13aebf231bfe1704da88f125cee6b4b30ee0609d0720a97bed1900b96bc3e7a6
↪"
      },
```

(continues on next page)

(continued from previous page)

```
      {
        "index": "0x0",
        "signature":
↪"0x411cb93f816549eba82c3bf8c03fa637036dcdee65667b541d0da06a6eaea80d16e6ca52bf1b08f77b59a834bffb
↪"
      },
      {
        "index": "0x3",
        "signature":
↪"0xb5b41e49c0b2bf758322ecb5c86dc3a3a0f9b98891b5bbf50c8613a241f05f595ce40d0bb212b6faa32e98546754
↪"
      }
    ],
    "stateRoot":
↪"0x0000000000000000000000000000000000000000000000000000000000000000",
    "timestamp": "0x173ad8703d6",
    "transactionsRoot":
↪"0xb563f70188512a085b5607cac0c35480336a566de736c83410a062c9acc785ad"
  }
}
```

13.17 getBlockHeaderByNumber

Return the block header queried according to the block number

13.17.1 Parameters

- `groupID`: `unsigned int` - group ID
- `blockNumber`: `string` - Block number (decimal string or hexadecimal string starting with 0x)
- `includeSignatures`: `bool` - Contains signature list or not (true displays signature list)

13.17.2 Returns

- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getBlockHeaderByNumber","params
↪":[1,"0x0",true],"id":1}' http://127.0.0.1:8545 |jq
```

Result reference getBlockHeaderByHash

13.18 getBlockHashByNumber

Returns a block hash by a block number.

13.18.1 Parameters

- `groupID`: `unsigned int` - group ID
- `blockNumber`: `string` - block number (hexadecimal string starts with 0x)

13.18.2 Returns

- `blockHash`: `string` - hash of block
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getBlockHashByNumber","params":[1,
↪"0x1"],"id":1}' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": "0x10bfdc1e97901ed22cc18a126d3ebb8125717c2438f61d84602f997959c631fa"
}
```

13.19 getTransactionByHash

Returns the information about a transaction by transaction hash.

13.19.1 Parameters

- `groupID`: `unsigned int` - group ID
- `transactionHash`: `string` - transaction hash

13.19.2 Returns

- `object`: - A transaction object:
  - `blockHash`: `string` - hash of the block where this transaction was in.
  - `blockLimit`: `string` - the `blockLimit` of the transaction, used to prevent duplication of transactions
  - `chainId`: `string` - Chain ID where the transaction is located
  - `extraData`: `string` - ExtraData in the transaction
  - `groupId`: `string` - Group ID where the transaction is located
  - `blockNumber`: `string` - block number where this transaction was in.
  - `from`: `string` - address of the sender
  - `gas`: `string` - gas provided by the sender
  - `gasPrice`: `string` - gas price provided by the sender
  - `hash`: `string` - hash of the transaction
  - `input`: `string` - the data send along with the transaction
  - `nonce`: `string` - the number of transactions made by the sender prior to this one
  - `signature`: transaction signature, including `r`, `s`, `v` and serialized transaction signature `signature`
  - `to`: `string` - address of the receiver, `0x0000000000000000000000000000000000000000` when its a contract creation transaction‘
  - `transactionIndex`: `string` - integer of the transaction’s index position in the block

  - `value`: `string` - value transferred

- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getTransactionByHash","params":[1,
↪"0x3961fac263d8e640b148ddcfafd71d2069e93a006abc937c32fb16cfa96e661d"],"id":1}'␣
↪http://127.0.0.1:8545 |jq

// Result
 {
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "blockHash":
↪"0xfa639d1454362a8cdfcab1ca1948a5defaf7048b28f67e80780ab1e24e8f8c59",
    "blockLimit": "0x100",
    "blockNumber": "0x1",
    "chainId": "0x1",
    "extraData": "0x",
    "from": "0x57c7be32cbfb3bfed4fddc87efcc735b4e945fb3",
    "gas": "0x2faf080",
    "gasPrice": "0xa",
    "groupId": "0x1",
    "hash": "0x3961fac263d8e640b148ddcfafd71d2069e93a006abc937c32fb16cfa96e661d",
    "input":
↪"0x4ed3885e0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000
↪",
    "nonce": "0x3eb675ec791c2d19858c91d0046821c27d815e2e9c151604912205000002968",
    "signature": {
      "r": "0x9edf7c0cb63645442aff11323916d51ec5440de979950747c0189f338afdcefd",
      "s": "0x2f3473184513c6a3516e066ea98b7cfb55a79481c9db98e658dd016c37f03dcf",
      "signature":
↪"0x9edf7c0cb63645442aff11323916d51ec5440de979950747c0189f338afdcefd2f3473184513c6a3516e066ea98b
↪",
      "v": "0x0"
    },
    "to": "0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744",
    "transactionIndex": "0x0",
    "value": "0x0"
  }
}
```

13.20 getTransactionByBlockHashAndIndex

Returns information about a transaction by block hash and transaction index position.

13.20.1 Parameters

- `groupID`: `unsigned int` - group ID
- `blockHash`: `string` - hash of a block
- `transactionIndex`: `string` - integer of the transaction index position

13.20.2 Returns

please see getTransactionByHash

- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getTransactionByBlockHashAndIndex",
↪"params":[1,"0x10bfdc1e97901ed22cc18a126d3ebb8125717c2438f61d84602f997959c631fa",
↪"0x0"],"id":1}' http://127.0.0.1:8545 |jq
```

see result in getTransactionByHash

13.21 getTransactionByBlockNumberAndIndex

Returns information about a transaction by block number and transaction index position.

13.21.1 Parameters

- `groupID`: `unsigned int` - group ID
- `blockNumber`: `string` - a block number (hexadecimal string starts with 0x)
- `transactionIndex`: `string` - the transaction index position (hexadecimal string starts with 0x)

13.21.2 Returns

please see getTransactionByHash

- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getTransactionByBlockNumberAndIndex
↪","params":[1,"0x1","0x0"],"id":1}' http://127.0.0.1:8545 |jq
```

see result in getTransactionByHash

13.22 getTransactionReceipt

Returns the receipt of a transaction by transaction hash.

13.22.1 Parameters

- `groupID`: `unsigned int` - group ID
- `transactionHash`: `string` - hash of a transaction

13.22.2 Returns

- `object`: - transaction information:
  - `blockHash`: `string` - hash of the block where this transaction was in
  - `blockNumber`: `string` - block number where this transaction was in
  - `contractAddress`: `string` - contract address, the contract address created, if the transaction was a contract creation, otherwise "0x0000000000000000000000000000000000000000"
  - `from`: `string` - address of the sender
  - `gasUsed`: `string` - The amount of gas used by this specific transaction
  - `input`: `string` - the data send along with the transaction

  - `logs`: `array` - Array of log objects, which this transaction generated
  - `logsBloom`: `string` - Bloom filter for light clients to quickly retrieve related logs
  - `output`: `string` - the data result along with the transaction
  - `root`: `string` - state root
  - `status`: `string` - status value of the transaction
  - `to`: `string` - address of the receiver. `0x0000000000000000000000000000000000000000` when its a contract creation.
  - `transactionHash`: `string` - hash of the transaction
- `transactionIndex`: `string` - integer of the transaction's index position in the block
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getTransactionReceipt","params":[1,
↪"0x708b5781b62166bd86e543217be6cd954fd815fd192b9a124ee9327580df8f3f"],"id":1}'␣
↪http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": {
        "blockHash":
↪"0x977efec48c248ea4be87016446b40d7785d7b71b7d4e3aa0b103b9cf0f5fe19e",
        "blockNumber": "0xa",
        "contractAddress": "0x0000000000000000000000000000000000000000",
        "from": "0xcdcce60801c0a2e6bb534322c32ae528b9dec8d2",
        "gasUsed": "0x1fb8d",
        "input":
↪"0xb602109a0000000000000000000000000000000000000000000000000000000000000080000000000000000000000
↪",
        "logs": [ ],
        "logsBloom":
↪"0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
        "output":
↪"0x0000000000000000000000000000000000000000000000000000000000000000",
        "root":"0x38723a2e5e8a17aa7950dc008209944e898f69a7bd10a23c839d341e935fd5ca
↪",
        "status": "0x0",
        "to": "0x15538acd403ac1b2ff09083c70d04856b8c0bdfd",
        "transactionHash":
↪"0x708b5781b62166bd86e543217be6cd954fd815fd192b9a124ee9327580df8f3f",
        "transactionIndex": "0x0"
    }
}
```

13.23 getPendingTransactions

Returns the pending transactions list.

13.23.1 Parameters

- `groupID`: `unsigned int` - group ID

13.23.2 Returns

- `object`: - pending transaction information:
  - `from`: `string` - address of the sender
  - `gas`: `string` - gas provided by the sender
  - `gasPrice`: `string` - gas price provided by the sender
  - `hash`: `string` - hash of the transaction
  - `input`: `string` - the data send along with the transaction
  - `nonce`: `string` - the number of transactions made by the sender prior to this one
  - `to`: `string` - address of the receiver, `0x0000000000000000000000000000000000000000` when its a contract creation transaction‘
  - `value`: `string` - value transferred
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getPendingTransactions","params
↪":[1],"id":1}' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": {
        [
            {
                "from": "0x6bc952a2e4db9c0c86a368d83e9df0c6ab481102",
                "gas": "0x9184e729fff",
                "gasPrice": "0x174876e7ff",
                "hash":
↪"0x7536cf1286b5ce6c110cd4fea5c891467884240c9af366d678eb4191e1c31c6f",
                "input":
↪"0x48f85bce000000000000000000000000000000000000000000000000000000000000001bf5bd8a9e7ba8b936ea70
↪",
                "nonce":
↪"0x65f0d06e39dc3c08e32ac10a5070858962bc6c0f5760baca823f2d5582d03f",
                "to": "0xd6f1a71052366dbae2f7ab2d5d5845e77965cf0d",
                "value": "0x0"
            }
        ]
    }
}
```

13.24 getPendingTxSize

Returns the size of pending transactions

13.24.1 Parameters

- `groupID`: `unsigned int` - group ID

13.24.2 Returns

- `string`: - size of pending transactions
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getPendingTxSize","params":[1],"id
↪":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": "0x1"
}
```

13.25 getCode

Returns code by a contract address.

13.25.1 Parameters

- `groupID`: `unsigned int` - group ID
- `address`: `string` - contract address

13.25.2 Returns

- `string`: - contract data
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getCode","params":[1,
↪"0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b"],"id":1}' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result":
↪"0x60606040523415600b57fe5b5b60928061001a6000396000f300606060405260000357c0100000000000000000000000
↪"
}
```

13.26 getTotalTransactionCount

Returns current total size of transaction and block number.

13.26.1 Parameters

- `groupID`: `unsigned int` - group ID

13.26.2 Returns

- `object`: - current total size of transaction and block number:
  - `blockNumber`: `string` - block number
  - `failedTxSum`: `string` - the failed total of transaction
  - `txSum`: `string` - the total of transaction
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getTotalTransactionCount","params
↪":[1],"id":1}' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": {
      "blockNumber": "0x1",
      "failedTxSum": "0x0",
      "txSum": "0x1"
    }
}
```

13.27 getSystemConfigByKey

Returns value by a key value

13.27.1 Parameters

- `groupID`: `unsigned int` - group ID
- `key`: `string` - support tx_count_limit and tx_gas_limit

13.27.2 Returns

- `string` - value
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"getSystemConfigByKey","params":[1,
↪"tx_count_limit"],"id":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": "1000"
}
```

13.28 addPeers

Add P2P information to the configuration

13.28.1 Parameters

- `hostPorts`: `array` - the IP and port of nodes

13.28.2 Returns

- `object`: - result of calling
  - `code`: - status，it's meaning can be referenced from [Dynamice group management API status code](#Dynamice\ group management\ API\ status\ code)
  - `message`: - message
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"addPeers","params":[["127.0.0.
↪1:20024","127.0.0.1:20025"]],"id":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "code": "0x0",
    "message": " add peers successfully"
  }
}
```

13.29 erasePeers

Erase P2P information from the configuration

13.29.1 Parameters

- `hostPorts`: `array` - the IP and port of nodes

13.29.2 Returns

- `object`: - result of calling
  - `code`: - status，it's meaning can be referenced from [Dynamice group management API status code](#Dynamice\ group management\ API\ status\ code)
  - `message`: - message
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"erasePeers","params":[["127.0.0.
↪1:20024","127.0.0.1:20025"]],"id":1}' http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "code": "0x0",
```

(continues on next page)

(continued from previous page)

```
    "message": " erase peers successfully"
  }
}
```

13.30 queryPeers

Query the P2P information in the configuration

13.30.1 Parameters

none

13.30.2 Returns

- `array` - the IP and port of the nodes in the configuration
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"queryPeers","params":[],"id":1}'␣
↪http://127.0.0.1:8545 |jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": [
    "127.0.0.1:30301",
    "127.0.0.1:30302",
    "127.0.0.1:30303"
  ]
}
```

13.31 call

Executes a new message call immediately without creating a transaction on the block chain.

13.31.1 Parameters

- `groupID`: `unsigned int` - group ID
- `object`: - the transaction call object:
  - `from`: `string` - address of the sender
  - `to`: `string` - address of the receiver
  - `value`: `string` - (optional) transfer value
  - `data`: `string` - (optional) code parameter. You can read the coding convention in Ethereum Contract ABI

13.31.2 Returns

- `object`: - result object
  - `currentBlockNumber`: `string` - the current block number
  - `output`: `string` - result
  - `status`: `string` - status value of the message(the same as transaction status)
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"call","params":[1,{"from":
↪"0x6bc952a2e4db9c0c86a368d83e9df0c6ab481102","to":
↪"0xd6f1a71052366dbae2f7ab2d5d5845e77965cf0d","value":"0x1","data":"0x3"}],"id":1}
↪' http://127.0.0.1:8545 |jq

// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": {
        "currentBlockNumber": "0x1",
        "output": "0x",
        "status": "0x0"
    }
}
```

13.32 sendRawTransaction

Creates new message call transaction or a contract creation for signed transactions.

13.32.1 Parameters

- `groupID`: `unsigned int` - group ID
- `rlp`: `string` - transaction data of signing

13.32.2 Returns

- `string` - the signed transaction data
- Example

```
// RC1 Request
curl -X POST --data '{"jsonrpc":"2.0","method":"sendRawTransaction","params":[1,
↪"f8ef9f65f0d06e39dc3c08e32ac10a5070858962bc6c0f5760baca823f2d5582d03f85174876e7ff8609184e729fff
↪"],"id":1}' http://127.0.0.1:8545 |jq

// RC1 Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": "0x7536cf1286b5ce6c110cd4fea5c891467884240c9af366d678eb4191e1c31c6f"
}

// RC2 Request
curl -X POST --data '{"jsonrpc":"2.0","method":"sendRawTransaction","params":[1,
↪"f8d3a003922ee720bb7445e3a914d8ab8f507d1a647296d563100e49548d83fd98865c8411e1a3008411e1a3008201
↪"],"id":1}' http://127.0.0.1:8545 |jq
```

(continues on next page)

(continued from previous page)

```
// RC2 Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": "0x0accad4228274b0d78939f48149767883a6e99c95941baa950156e926f1c96ba"
}

// FISCO BCOS supports OSCCA algorithm, request of OSCCA is exemplified here
// RC1 Request
curl -X POST --data '{"jsonrpc":"2.0","method":"sendRawTransaction","params":[1,
↪"f8ef9f65f0d06e39dc3c08e32ac10a5070858962bc6c0f5760baca823f2d5582d03f85174876e7ff8609184e729fff
↪"],"id":1}' http://127.0.0.1:8545 |jq
// RC2 Request
curl -X POST --data '{"jsonrpc":"2.0","method":"sendRawTransaction","params":[1,
↪"f90114a003eebc46c9c0e3b84799097c5a6ccd6657a9295c11270407707366d0750fcd598411e1a30084b2d05e0082
↪"],"id":1}' http://127.0.0.1:8545 |jq
```

13.33 sendRawTransactionAndGetProof

To execute a signed transaction, after the transaction is chained, push the transaction receipt, transaction Merkle certificate, and transaction receipt Merkle certificate. For the Merkle certificate, please refer to [here] (./design/merkle_proof.md).

Note:

- `supported_version < 2.2.0`: Call the `sendRawTransactionAndGetProof` interface, only push the transaction receipt after the transaction is chained
- `supported_version >= 2.2.0`: Call the `sendRawTransactionAndGetProof` interface to push the transaction receipt, transaction Merkle certificate, and transaction receipt Merkle certificate after the transaction is chained

---

13.33.1 Parameters

- `groupID`: `unsigned int` - group ID
- `rlp`: `string` - transaction data of signing

13.33.2 Returns

- `string` - Transaction hash
- Example: Same as `sendRawTransaction`, refer to here

13.34 getTransactionByHashWithProof

Returns the information about the transaction and its proof by a transaction hash. Please note that this function is supported since 2.2.0.

13.34.1 Parameters

- `groupID`: `unsigned int` - group ID
- `transactionHash`: `string` - transaction hash

13.34.2 Returns

- `object`: - A transaction object:
  - `blockHash`: `string` - hash of the block where this transaction was in
  - `blockNumber`: `string` - block number where this transaction was in
  - `from`: `string` - address of the sender
  - `gas`: `string` - gas provided by the sender
  - `gasPrice`: `string` - gas price provided by the sender
  - `hash`: `string` - hash of the transaction
  - `input`: `string` - the data send along with the transaction
  - `nonce`: `string` - the number of transactions made by the sender prior to this one
  - `to`: `string` - address of the receiver, `0x0000000000000000000000000000000000000000` when its a contract creation transaction‘
  - `transactionIndex`: `string` - integer of the transaction’s index position in the block
  - `value`: `string` - value transferred
- `array` - proof of transaction:
  - `left`: `array` - the hash list of left
  - `right`: `array` - the hash list of right
- Examples

```
curl -X POST --data '{"jsonrpc":"2.0","method":"getTransactionByHashWithProof",
↪"params":[1,"0xd2c12e211315ef09dbad53407bc820d062780232841534954f9c23ab11d8ab4c
↪"],"id":1}' http://127.0.0.1:8585 |jq
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "transaction": {
      "blockHash":
↪"0xcd31b05e466bce99460b1ed70d6069fdfbb15e6eef84e9b9e4534358edb3899a",
      "blockNumber": "0x5",
      "from": "0x148947262ec5e21739fe3a931c29e8b84ee34a0f",
      "gas": "0x1c9c380",
      "gasPrice": "0x1c9c380",
      "hash": "0xd2c12e211315ef09dbad53407bc820d062780232841534954f9c23ab11d8ab4c",
      "input":
↪"0x8a42ebe900000000000000000000000000000000000000000000000000000000000000400000000000000000000000
↪",
      "nonce": "0x208f6fd78d48aad370df51c6fdf866f8ab022de765c2959841ff2e81bfd9af9",
      "to": "0xd6c8a04b8826b0a37c6d4aa0eaa8644d8e35b79f",
      "transactionIndex": "0x32",
      "value": "0x0"
    },
    "txProof": [
      {
        "left": [
```

(continues on next page)

(continued from previous page)


```
            "30f0abfcf4ca152815548620e33d21fd0feaa7c78867791c751e57cb5aa38248c2",
            "31a864156ca9841da8176738bb981d5da9102d9703746039b3e5407fa987e5183e"
          ],
          "right": [
            "33d8078d7e71df3544f8845a9db35aa35b2638e8468a321423152e64b9004367b4",
            "34343a4bce325ec8f6cf48517588830cd15f69b60a05598b78b03c3656d1fbf2f5",
            "35ac231554047ce77c0b31cd1c469f1f39ebe23404fa8ff6cc7819ad83e2c029e7",
            "361f6c588e650323e03afe6460dd89a9c061583e0d62c117ba64729d2c9d79317c",
            "377606f79f3e08b1ba3759eceada7fde3584f01822467855aa6356652f2499c738",
            "386722fe270659232c5572ba54ce23b474c85d8b709e7c08e85230afb1c155fe18",
            "39a9441d668e5e09a5619c365577c8c31365f44a984bde04300d4dbba190330c0b",
            "3a78a8c288120cbe612c24a33cce2731dd3a8fe6927d9ee25cb2350dba08a541f5",
            "3bd9b67256e201b5736f6081f39f83bcb917261144384570bdbb8766586c3bb417",
            "3c3158e5a82a1ac1ed41c4fd78d5be06bf79327f60b094895b886e7aae57cff375",
            "3de9a4d98c5ae658ffe764fbfa81edfdd4774e01b35ccb42beacb67064a5457863",
            "3e525e60c0f7eb935125f1156a692eb455ab4038c6b16390ce30937b0d1b314298",
            "3f1600afe67dec2d21582b8c7b76a15e569371d736d7bfc7a96c0327d280b91dfc"
          ]
        },
        {
          "left": [
            "3577673b86ad4d594d86941d731f17d1515f4669483aed091d49f279d677cb19",
            "75603bfea5b44df4c41fbb99268364641896334f006af3a3f67edaa4b26477ca",
            "1339d43c526f0f34d8a0f4fb3bb47b716fdfde8d35697be5992e0888e4d794c9"
          ],
          "right": [
            "63c8e574fb2ef52e995427a8acaa72c27073dd8e37736add8dbf36be4f609ecb",
            "e65353d911d6cc8ead3fad53ab24cab69a1e31df8397517b124f578ba908558d"
          ]
        },
        {
          "left": [],
          "right": []
        }
      ]
    }
}
```


13.35 getTransactionReceiptByHashWithProof

Returns the information about the receipt and its proof by a transaction hash. Please note that this function is supported since 2.2.0.

13.35.1 Parameters

- `groupID`: `unsigned int` - group ID
- `transactionHash`: `string` - transaction hash

13.35.2 Returns

- `array` - proof of receipt:
  - `left`: `array` - the hash list of left
  - `right`: `array` - the hash list of right
- `object`: - transaction information:

  - `blockHash`: `string` - hash of the block where this transaction was in
  - `blockNumber`: `string` - block number where this transaction was in
  - `contractAddress`: `string` - contract address, the contract address created, if the transaction was a contract creation, otherwise "0x0000000000000000000000000000000000000000"
  - `from`: `string` - address of the sender
  - `gasUsed`: `string` - The amount of gas used by this specific transaction
  - `input`: `string` - the data send along with the transaction
  - `logs`: `array` - Array of log objects, which this transaction generated
  - `logsBloom`: `string` - Bloom filter for light clients to quickly retrieve related logs
  - `output`: `string` - the data result along with the transaction
  - `status`: `string` - status value of the transaction
  - `to`: `string` - address of the receiver. `0x0000000000000000000000000000000000000000` when its a contract creation.
  - `transactionHash`: `string` - hash of the transaction
  - `transactionIndex`: `string` - integer of the transaction's index position in the block

- Examples

```
curl -X POST --data '{"jsonrpc":"2.0","method":
↪"getTransactionReceiptByHashWithProof","params":[1,
↪"0xd2c12e211315ef09dbad53407bc820d062780232841534954f9c23ab11d8ab4c"],"id":1}'␣
↪http://127.0.0.1:8585 |jq

{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "receiptProof": [
      {
        "left": [
          "3088b5c8f9d92a3411a911f35ff0119a02e8f8f04852cf2fdfaa659843eac6a3ad",
          "31170ac8fd555dc50e59050841da0d96e4c4bc7e6266e1c6865c08c3b2391801dd"
        ],
        "right": [
          "33c572c8f961e0c56689d641fcf274916857819769a74e6424c58659bf530e90e3",
          "341233933ea3d357b4fdd6b3d1ed732dcff15cfd54e527c93c15a4e0238585ed11",
          "351e7ba09965cce1cfb820aced1d37204b06d96a21c5c2cf36850ffc62cf1fc84c",
          "361f65633d9ae843d4d3679b255fd448546a7b531c0056e8161ea0adbf1af12c0f",
          "37744f6e0d320314536b230d28b2fd6ac90b0111fb1e3bf4a750689abc282d8589",
          "386e60d9daa0be9825019fcf3d08cdaf51a90dc62a22a6e11371f94a8e516679cc",
          "391ef2f2cee81f3561a9900d5333af18f59aa3cd14e70241b5e86305ba697bf5f2",
          "3ac9999d4f36d76c95c61761879eb9ec60b964a489527f5af844398ffaa8617f0d",
          "3b0039ce903e275170640f3a464ce2e1adc2a7caee41267c195469365074032401",
          "3ca53017502028a0cb5bbf6c47c4779f365138da6910ffcfebf9591b45b89abd48",
          "3de04fc8766a344bb73d3fe6360c61d036e2eeedfd9ecdb86a0498d7849ed591f0",
          "3e2fc73ee22c4986111423dd20e8db317a313c9df29fa5aa3090f27097ecc4e1a9",
          "3fa7d31ad5c6e7bba3f99f9efc03ed8dd97cb1504003c34ad6bde5a662481f00a0"
        ]
      },
      {
        "left": [
          "cd46118c0e99be585ffcf50423630348dbc486e54e9d9293a6a8754020a68a92",
          "3be78209b3e3c83af3668ec3192b5bf232531323ef66b66de80a11f386270132",
          "bd3a11d74a3fd79b1e1ea17e45b76eda4d25f6a5ec7fc5f067ea0d086b1ce70f"
        ],
```

(continues on next page)

(continued from previous page)


```
        "right": [
          "6a6cefef8b48e455287a8c8694b06f4f7cb7950017ab048d6e6bdd8029f9f8c9",
          "0a27c5ee02e618d919d228e6a754dc201d299c91c9e4420a48783bb6fcd09be5"
        ]
      },
      {
        "left": [],
        "right": []
      }
    ],
    "transactionReceipt": {
      "blockHash":
↪"0xcd31b05e466bce99460b1ed70d6069fdfbb15e6eef84e9b9e4534358edb3899a",
      "blockNumber": "0x5",
      "contractAddress": "0x0000000000000000000000000000000000000000",
      "from": "0x148947262ec5e21739fe3a931c29e8b84ee34a0f",
      "gasUsed": "0x21dc1b",
      "input":
↪"0x8a42ebe900000000000000000000000000000000000000000000000000000000000000400000000000000000000000
↪",
      "logs": [],
      "logsBloom":
↪"0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
↪",
      "output": "0x",
      "root": "0xc3b4185963c78a4ca8eb90240e5cd95371d7217a9ce2bfa1149d53f79c73afbb",
      "status": "0x0",
      "to": "0xd6c8a04b8826b0a37c6d4aa0eaa8644d8e35b79f",
      "transactionHash":
↪"0xd2c12e211315ef09dbad53407bc820d062780232841534954f9c23ab11d8ab4c",
      "transactionIndex": "0x32"
    }
  }
}
```


13.36 generateGroup

Generate new group with group ID and parameters of genesis block. Please note that this function is supported since 2.2.0

13.36.1 Parameters

- `groupID`: `unsigned int` - ID of the group
- `params`: `object` - parameters of genesis block
  - `timestamp`: `unsigned int` - timestamp of genesis block
  - `sealers`: `array` - node ID of sealers, sealers should have valid P2P connection to each other
  - `enable_free_storage`: `bool` - optional，whether to enable "free storage" mode, after activation, the gas usage of storage instructions will be decreased

13.36.2 Returns

- `object`: - result of calling

  - `code`: - status, it's meaning can be referenced from [Dynamice group management API status code](#Dynamice\ group management\ API\ status\ code)
  - `message`: - message

- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"generateGroup","params":[2, {
↪"timestamp":"1585214879000","sealers":[
↪"70f18c055d366615e86df99f91b6d3f16f07d66293b203b73498442c0366d2c8ff7a21bb56923d9d81b1c291625188
↪",
↪"dde37f534885f08db914566efeb03183d59363a4be972bbcdde25c37f0b350e1980a7de4fdc4aaf956b931aab00b73
↪",
↪"d41672b29b3b1bfe6cad563d0f0b2a2735865b27918307b85085f892043a63f681ac8799243e920f7bb144b111d854
↪",
↪"7ba2717f81f38e7371ccdbe173751f051b86819f709e940957664dbde028698fd31ba3042f7dd9accd73741ba42afc
↪"],"enable_free_storage":true}],"id":1}' http://127.0.0.1:8545 | jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "code": "0x0",
    "message": "Group 2 generated successfully"
  }
}
```

13.37 startGroup

Starts group. Please note that this function is supported since 2.2.0

13.37.1 Parameters

- `groupID`: `unsigned int` - ID of the group

13.37.2 Returns

- `object`: - result of calling
  - `code`: - status, it's meaning can be referenced from [Dynamice group management API status code](#Dynamice\ group management\ API\ status\ code)
  - `message`: - message

- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"startGroup","params":[2],"id":1}'␣
↪http://127.0.0.1:8545 | jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "code": "0x0",
    "message": "Group 2 started successfully"
```

(continues on next page)

(continued from previous page)

```
  }
}
```

13.38 stopGroup

Stops group. Please note that this function is supported since 2.2.0

13.38.1 Parameters

- `groupID`: `unsigned int` - ID of the group

13.38.2 Returns

- `object`: - result of calling
  - `code`: - status, it's meaning can be referenced from [Dynamice group management API status code](#Dynamice\ group management\ API\ status\ code)
  - `message`: - message
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"stopGroup","params":[2],"id":1}'␣
↪http://127.0.0.1:8545 | jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "code": "0x0",
    "message": "Group 2 stopped successfully"
  }
}
```

13.39 removeGroup

Deletes group, but the data of group will be reserved for future recover. Please note that this function is supported since 2.2.0

13.39.1 Parameters

- `groupID`: `unsigned int` - ID of the group

13.39.2 Returns

- `object`: - result of calling
  - `code`: - status, it's meaning can be referenced from [Dynamice group management API status code](#Dynamice\ group management\ API\ status\ code)
  - `message`: - message

- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"removeGroup","params":[2],"id":1}' http://127.0.0.1:8545 | jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "code": "0x0",
    "message": "Group 2 deleted successfully"
  }
}
```

13.40 recoverGroup

Recovers group. Please note that this function is supported since 2.2.0

13.40.1 Parameters

- `groupID`: `unsigned int` - ID of the group

13.40.2 Returns

- `object`: - result of calling
  - `code`: - status，it's meaning can be referenced from [Dynamice group management API status code](#Dynamice\ group management\ API\ status\ code)
  - `message`: - message
- Example

```
// Request
curl -Ss -X POST --data '{"jsonrpc":"2.0","method":"recoverGroup","params":[2],"id":1}' http://127.0.0.1:8545 | jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "code": "0x0",
    "message": "Group 2 recovered successfully"
  }
}
```

13.41 queryGroupStatus

Queries status of group. Please note that this function is supported since 2.2.0

13.41.1 Parameters

- `groupID`: `unsigned int` - ID of the group

13.41.2 Returns

- `object`: - result of calling
  - `code`: - status，it's meaning can be referenced from [Dynamice group management API status code](#Dynamice\ group management\ API\ status\ code)
  - `message`: - message
  - `status`: - status of the group:
    - `INEXISTENT`: the group is inexistent
    - `STOPPING`: the group is stopping
    - `RUNNING`: the group is running
    - `STOPPED`: the group is stopped
    - `DELETED`: the group is deleted
- Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"queryGroupStatus","params":[2],"id
↪":1}' http://127.0.0.1:8545 | jq

// Result
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "code": "0x0",
    "message": "",
    "status": "STOPPED"
  }
}
```

13.42 getNodeInfo

Get the NodeID, Topic and other information of the requested node

13.42.1 Parameters

- 无

13.42.2 Returns

- NodeInfo: Get node information, including the agency where the node is located, the node name, the NodeID, and the topic subscribed by the node
- Example:

```
curl -X POST --data '{"jsonrpc":"2.0","method":"getNodeInfo","params":[],"id":1}'␣
↪http://127.0.0.1:8545 |jq
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "Agency": "agency",
    "IPAndPort": "0.0.0.0:30300",
    "Node": "node1",
    "NodeID":
↪"2263a586cba1c1b8419f0dd9e623c7db79b2ba9127367aa854b1493e218efaa2ef58b08de1b36defdeb1d407449014e
↪",
    "Topic": []
  }
}
```

13.43 getBatchReceiptsByBlockNumberAndRange

According to the block number and transaction range, batch return the transaction receipt information of the specified group.

13.43.1 Parameters

- `groupID`: Group ID
- `blockNumber`: The height of the block where the requested receipt information is located
- `from`: The start index of the receipt to be obtained
- `count`: The number of receipts that need to be obtained in batches. When set to -1, return all receipt information in the block
- `compressFlag`: Compression flag. When set to false, the batch transaction receipt information is returned in plain text; when set to true, the batch transaction receipt information is compressed in zlib format, and the compressed receipt information is returned in Base64 encoded format.

13.43.2 Returns

When `compressFlag` is true, return the batch transaction receipt information in plain text; when `compressFlag` is false, return the batch receipt information compressed by zlib and Base64 encoding. The specific fields returned are as follows:

- `blockHash`: The hash of the block where the receipt is located
- `blockNumber`:The height of the block where the receipt is located
- `receiptRoot`: The receiptRoot of the block where the receipt is located
- `receiptsCount`: Number of receipts contained in the block

The fields included in each receipt message are as follows:

- contractAddress: Contract address corresponding to the receipt
- from: External account information corresponding to the transaction receipt
- gasUsed: gas information
- logs: Event log information included in the receipt
- output: Transaction execution result

- status: Receipt status
- to: Target account address
- transactionHash: The transaction hash that generated the receipt
- transactionIndex: Index of the transaction that generated the receipt in the block

Example:

```
# Get the plaintext information of all receipts of the block with a block height␣
↪of 1
curl -X POST --data '{"jsonrpc":"2.0","method":
↪"getBatchReceiptsByBlockNumberAndRange","params":[1,"0x1","0","-1",false],"id":1}
↪' http://127.0.0.1:8545 |jq
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "blockInfo": {
      "blockHash":
↪"0xcef82a3c1e7770aa4e388af5c70e97ae177a3617c5020ae052be4095dfdd39a2",
      "blockNumber": "0x1",
      "receiptRoot":
↪"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
      "receiptsCount": "0x1"
    },
    "transactionReceipts": [
      {
        "contractAddress": "0x0000000000000000000000000000000000000000",
        "from": "0xb8e3901e6f5f842499fd537a7ac7151e546863ad",
        "gasUsed": "0x5798",
        "logs": [],
        "output":
↪"0x08c379a0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000
↪",
        "status": "0x1a",
        "to": "0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744",
        "transactionHash":
↪"0xc6ec15fd1e4d696e66d7fbef6064bda3ed012bcb7744d09903ee289df65f7d53",
        "transactionIndex": "0x0"
      }
    ]
  }
}

# Get all receipt information after compression encoding with a block height of 1
curl -X POST --data '{"jsonrpc":"2.0","method":
↪"getBatchReceiptsByBlockNumberAndRange","params":[1,"0x1","0","-1",true],"id":1}
↪' http://127.0.0.1:8545 |jq
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": "eJylUrFu3DAM3fsZmm+gRImUbiu6NEuHAJ2KDLJEpkETK7B9QIDD/
↪XsZX4AgQIcCEWCLj+YjafKd3fQ42p+bWYc7nq/ge11/
↪u6ODlyaaQ8XmhZmh1iiYc9XUGKRwFc9ckTy3BAGqQAqTRCipa+9YanCHa8Ifp6dJlj2lN98iTR6et9sxtt1HpULUSsAoURu
↪Yw5tu3CHf8dXZtzPapbV97X2RddwL857FquoynnTNlwQJeSJPmGGIp2hNyZWvfJy8pUias3Tj3df25St9piUs21+O4f23n7u
↪TUPNhwJ2q5dM2cpFArCpqmoByj+7D/d1GTNJ/
↪UNBU7FRKizjqJElCcekXp4MPUJrYUHUyBKBJeK1BStu1+THszd3m5Ls5d7i5f/gLIg98l"
}
```

13.44 getBatchReceiptsByBlockHashAndRange

According to the block hash and transaction range, batch return the transaction receipt information of the specified group.

13.44.1 Parameters

- `groupID`: Group ID
- `blockHash`: The hash of the block where the requested receipt information is located
- `from`: The start index of the receipt to be obtained
- `count`: The number of receipts that need to be obtained in batches. When set to -1, return all receipt information in the block
- `compressFlag`: Compression flag. When set to false, the batch transaction receipt information is returned in plain text; when set to true, the batch transaction receipt information is compressed in zlib format, and the compressed receipt information is returned in Base64 encoded format.

13.44.2 Returns

When `compressFlag` is true, return the batch transaction receipt information in plain text; when `compressFlag` is false, return the batch receipt information compressed by zlib and Base64 encoding. The specific fields returned are as follows:

- `blockHash`: The hash of the block where the receipt is located
- `blockNumber`:The height of the block where the receipt is located
- `receiptRoot`: The receiptRoot of the block where the receipt is located
- `receiptsCount`: Number of receipts contained in the block

The fields included in each receipt message are as follows:

- contractAddress: Contract address corresponding to the receipt
- from: External account information corresponding to the transaction receipt
- gasUsed: gas information
- logs: Event log information included in the receipt
- output: Transaction execution result
- status: Receipt status
- to: Target account address
- transactionHash: The transaction hash that generated the receipt
- transactionIndex: Index of the transaction that generated the receipt in the block

Example:

```
# 获取区块哈希为0xcef82a3c1e7770aa4e388af5c70e97ae177a3617c5020ae052be4095dfdd39a2的区
块所有回执明文信息
 curl -X POST --data '{"jsonrpc":"2.0","method":
↪"getBatchReceiptsByBlockHashAndRange","params":[1,
↪"0xcef82a3c1e7770aa4e388af5c70e97ae177a3617c5020ae052be4095dfdd39a2","0","1",
↪false],"id":1}' http://127.0.0.1:8545 |jq
{
  "id": 1,
  "jsonrpc": "2.0",
```

(continues on next page)

(continued from previous page)

```
  "result": {
    "blockInfo": {
      "blockHash":
↪"0xcef82a3c1e7770aa4e388af5c70e97ae177a3617c5020ae052be4095dfdd39a2",
      "blockNumber": "0x1",
      "receiptRoot":
↪"0x69a04fa6073e4fc0947bac7ee6990e788d1e2c5ec0fe6c2436d0892e7f3c09d2",
      "receiptsCount": "0x1"
    },
    "transactionReceipts": [
      {
        "contractAddress": "0x0000000000000000000000000000000000000000",
        "from": "0xb8e3901e6f5f842499fd537a7ac7151e546863ad",
        "gasUsed": "0x5798",
        "logs": [],
        "output":
↪"0x08c379a0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000
↪",
        "status": "0x1a",
        "to": "0x8c17cf316c1063ab6c89df875e96c9f0f5b2f744",
        "transactionHash":
↪"0xc6ec15fd1e4d696e66d7fbef6064bda3ed012bcb7744d09903ee289df65f7d53",
        "transactionIndex": "0x0"
      }
    ]
  }
}

# 获取区块哈希为0xcef82a3c1e7770aa4e388af5c70e97ae177a3617c5020ae052be4095dfdd39a2的压
缩编码后的所有回执信息
curl -X POST --data '{"jsonrpc":"2.0","method":"getBatchReceiptsByBlockHashAndRange
↪","params":[1,"0xcef82a3c1e7770aa4e388af5c70e97ae177a3617c5020ae052be4095dfdd39a2
↪","0","1",true],"id":1}' http://127.0.0.1:8545 |jq
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": "eJylUrFu3DAM3fsZmm+gRImUbiu6NEuHAJ2KDLJEpkETK7B9QIDD/
↪XsZX4AgQIcCEWCLj+YjafKd3fQ42p+bWYc7nq/ge11/
↪u6ODlyaaQ8XmhZmh1iiYc9XUGKRwFc9ckTy3BAGqQAqTRCipa+9YanCHa8Ifp6dJlj2lN98iTR6et9sxtt1HpULUSsAoURuU
↪Yw5tu3CHf8dXZtzPapbV97X2RddwL857FquoynnTNlwQJeSJPmGGIp2hNyZWvfJy8pUias3Tj3df25St9piUs21+O4f23n7u
↪TUPNhwJ2q5dM2cpFArCpqmoByj+7D/d1GTNJ/
↪UNBU7FRKizjqJElCcekXp4MPUJrYUHUyBKBJeK1BStu1+THszd3m5Ls5d7i5f/gLIg98l"
}
```

CHAPTER

FOURTEEN

《深入浅出FISCO BCOS》

CHAPTER

FIFTEEN

COMMUNITY

FISCO BCOS, officially launched in December 2017, is the first China-developed open source consortium blockchain platform. It was collaboratively built by the FISCO open source working group, which was formed by Beyondsoft, Huawei, Shenzhen Securities Communications, DCITS, Forms Syntron, Tencent, WeBank, YIBI Technology, Yuexiu Financial Holdings (Fintech) and more.

15.1 FISCO BCOS resources

- Github homepage
- Technical documents
- Inlightful articles
- Code contribution
- Feedbacks
- Application cases
- 2021FISCO BCOS Industrial Applications Whitepaper

15.2 Join FISCO BCOS community

docs/../imag

成为贡献者

希望以后你可以拿这个项目给自己加分：
“FISCO BCOS是我一手搞起来的！”

★ Star	于你是收藏，于我是鼓励
New issue	反馈bug | 问题交流
New PR	文档修改 | bug修复 | 提交新功能特性