WeCross Documentation

Release v0.2.0

WeCross Community

Dec 18, 2019

Contents

WeCross 是

CHAPTER 1

WeCross 介绍

1.1 基本介绍

1.2 关键词

CHAPTER 2

快速入门

本章介绍WeCross所需的软硬件环境配置，以及为用户提供了快速入门WeCross的教程。

2.1 环境要求

2.1.1 硬件

WeCross负责管理多个Stub并与多条链通讯，同时作为Web Server提供RPC调用服务，为了能保证服务的稳定性，尽量使用推荐配置。

配置	最低配置	推荐配置
CPU	1.5GHz	2.4GHz
内存	1GB	8GB
核心	1核	4核
带宽	1Mb	10Mb

2.1.2 支持的平台

- Ubuntu 16.04
- CentOS 7.2+
- MacOS 10.14+

2.1.3 软件依赖

WeCross作为Java项目，需要安装Java环境包括：

- JDK8及以上
- Gradle 5.0及以上

WeCross提供了多种脚本帮助用户快速体验，这些脚本依赖`openssl`, `curl`, `expect`，使用下面的指令安装。

```
# Ubuntu
sudo apt-get install openssl curl expect

# CentOS
sudo yum install openssl curl expect

# MacOS
brew install openssl curl expect
```

2.2 快速部署

如果软硬件已准备就绪，接下来的教程将带领您快速部署WeCross。

- 创建操作目录

```
cd ~ && mkdir -p WeCross && cd WeCross
```

- 下载build_wecross.sh脚本

```
curl -LO https://raw.githubusercontent.com/WeBankFinTech/WeCross/release-0.2/
↪scripts/build_wecross.sh && chmod u+x build_wecross.sh
```

2.2.1 部署WeCross

WeCross提供了测试资源，在不搭任何链的情况下依然能体验WeCross针对跨链资源的相关操作。

构建WeCross

在WeCross目录下执行下面的指令，部署一个WeCross服务。请确保机器的8250, 25500端口没有被占用。

```
bash build_wecross.sh -i payment 127.0.0.1 8250 25500
```

Note:

- 其中-i选项指定服务配置，参数分别代表：[跨链网络标识符]，[rpc_listen_ip]，[rpc_port]，[p2p_port]。
- 详细的使用教程详见[Build WeCross脚本]()。

命令执行成功会输出Build WeCross successfully，并生成目录127.0.0.1-8250-25500。如果执行出错，请查看屏幕打印提示。

该步骤已经帮忙完成了WeCross根配置wecross.toml的自动配置，包括[stubs.path]以及[p2p]，在127.0.0.1-8250-25500/conf目录下查看目录结构如下：

```
cd 127.0.0.1-8250-25500/conf
tree
.
├── log4j2.xml      # 日志配置文件
├── p2p             # P2P证书目录
│   ├── ca.crt
│   ├── node.crt
│   ├── node.key
│   └── node.nodeid
```

(continues on next page)

(continued from previous page)

```
├── stubs          # 存放所有Stub配置的根目录，目录下包含了不同Stub的配置文件示例
│   ├── bcos
│   │   └── stub-sample.toml
│   ├── fabric
│   │   └── stub-sample.toml
│   └── jd
│       └── stub-sample.toml
├── wecross-sample.toml
└── wecross.toml # 根配置
```

启动WeCross

```
# 进入127.0.0.1-8250-25500目录
bash start.sh
```

启动成功会输出类似下面内容的响应。否则请使用netstat -an | grep tcp检查机器的8250, 25500端口是否被占用。

```
Wait for wecross to start ...
Wecross start successfully
```

如果仍然启动失败，则根据提示查看错误日志。

```
cat logs/error.log
```

访问测试资源

WeCross模拟了一个用于测试的合约资源，它不属于任何一条链，但是具备所有的UBI接口。可通过访问测试资源的status接口，确认服务是否完全启动成功。

```
curl http://127.0.0.1:8250/test-network/test-stub/test-resource/status
```

status接口用户查询某个资源的状态，即是否存在于WeCross的资源池中。如果得到如下输出，则代表跨链服务已成功启动。

```
{"version":"0.2","result":0,"message":null,"data":"exists"}
```

使用控制台

WeCross控制台提供了一套访问跨链资源UBI接口的命令，方便用户进行跨链开发和调试。

- 下载编译

```
git clone https://github.com/WeBankFinTech/WeCross-Console.git
cd WeCross-Console
./gradlew assemble
```

- 配置

控制台需要配置所有连接的WeCross的IP和端口信息。

```
cd dist
cp conf/console-sample.xml conf/console.xml
```

Note:

- 若搭建WeCross的IP和端口未使用默认配置，拷贝完配置文件后，需自行更改，详见[控制台配置]()

- 启动

```
bash start.sh
```

启动成功则输出如下信息，通过-h可查看控制台帮助，输入q/quit退出。

```
================================================================================
Welcome to WeCross console(0.2)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
================================================================================
```

- 调用测试资源

可以通过listResources命令查看当前连接的WeCross已配置的资源，可以通过call, sendTransaction等命令实现资源的UBI接口调用。测试资源所有命令的返回值都是**入参**。具体的命令列表与含义详见控制台命令

```
[server1]> listResources
Resources{
    errorCode=0,
    errorMessage='',
    resourceList=[
        WeCrossResource{
            checksum=
↪'0x7644243d71d1b1c154c717075da7bfe2d22bb2a94d7ed7693ab481f6cb11c756',
            type='TEST_RESOURCE',
            distance=0,
            path='test-network.test-stub.test-resource'
        }
    ]
}
[server1]> call test-network.test-stub.test-resource String get
Receipt{
    errorCode=0,
    errorMessage='call test resource success',
    hash='010157f4',
    result=[
        {
            sig=,
            retTypes=[
                String
            ],
            method=get,
            args=[

            ]
        }
    ]
}
[server1]> sendTransaction test-network.test-stub.test-resource String,Int set
↪"Hello World" 12580
Receipt{
    errorCode=0,
    errorMessage='sendTransaction test resource success',
    hash='010157f4',
    result=[
        {
            sig=,
            retTypes=[
```

(continues on next page)

(continued from previous page)

```
                String,
                Int
            ],
            method=set,
            args=[
                Hello World,
                12580
            ]
        }
    ]
}
```

- 停止WeCross

在127.0.0.1-8250-25500目录下提供了启动和停止脚步，运行stop.sh可停止WeCross服务。

```
bash stop.sh
```

2.2.2 体验WeCross+区块链

完成了WeCross的部署，如何让它和一条真实的区块链交互，相信优秀的您一定在跃跃欲试。接下来的教程将以**FISCO BCOS**为例介绍如何体验WeCross+区块链。

一键搭链

FISCO BCOS官方提供了一键搭链的教程，详见单群组FISCO BCOS联盟链的搭建

部署HelloWorld合约

FISCO BCOS控制台的安装和使用详见官方文档配置及使用控制台

安装完后启动并部署HelloWorld.sol，得到的合约地址在之后的WeCross配置中需要用到。

```
=====================================================================================================
Welcome to FISCO BCOS console(1.0.6)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
 ________ _______  _______  _______  _______      ________  _______  _______  _______
|        |       \/       \ /       \ /       \     |        \ /       \ /       \ /       \
↪\
| $$$$$$$$\$$$$|  $$$$$$|  $$$$$$|  $$$$$$\    | $$$$$$$|  $$$$$$|  $$$$$$|  $$$$$$
↪$$\
| $$__      | $$ | $$___\$| $$   \$| $$  | $$    | $$__/ $| $$   \$| $$  | $| $$___\
↪$$$
| $$  \     | $$  \$$     \| $$      | $$  | $$    | $$    $| $$      | $$  | $$\$$    \
↪\
| $$$$$     | $$  _\$$$$$$$| $$   __| $$  | $$    | $$$$$$$| $$   __| $$  | $$_\$$$$$
↪$$\
| $$       _| $$_|  \__| $| $$__/  | $$__/ $$    | $$__/ $| $$__/  | $$__/ $|  \__|
↪$$
| $$      |   $$ \\$$    $$\$$    $$\$$    $$    | $$    $$\$$    $$\$$    $$\$$
↪$$
 \$$       \$$$$$$ \$$$$$$  \$$$$$$  \$$$$$$      \$$$$$$$  \$$$$$$  \$$$$$$  \$$$$$
↪$

=====================================================================================================
[group:1]> deploy HelloWorld
contract address: 0x04ae9de7bc7397379fad6220ae01529006022d1b
```

配置FISCO BCOS Stub

完成了FISCO BCOS的部署，需要配置FISCO BCOS Stub，即配置连接信息以及链上的资源。

- 生成配置文件

运行`create_stubs_config.sh`脚本，生成FISCO BCOS Stub的配置文件。

```
bash create_stubs_config.sh -b stubs bcoschain
```

Note:

- 其中-b代表创建FISCO BCOS Stub配置文件，参数分别表示：[配置文件根目录]，[Stub名字]即区块链标识。
- 其中的[配置文件根目录]需要和根配置文件‘wecross.toml‘中的[stubs.path]保存一致。
- 详细的使用教程详见[Build WeCross脚本]()。

命令执行成功会输出`Create stubs/bcoschain/stub.toml successfully`，并生成文件`conf/stubs/bcoschain/stub.toml`。如果执行出错，请查看屏幕打印提示。

在`conf/stubs/bcoschain`目录下的`.pem`文件即FISCO BCOS的账户文件，已自动配置在了`stub.toml`文件中，之后只需要配置证书、群组以及资源信息。

- 配置证书

进入FISCO BCOS节点的证书目录`127.0.0.1/sdk`，将该目录下的`ca.crt, sdk.key, sdk.crt`文件拷贝到`bcoschain`目录下。

- 配置群组

```
vi conf/stubs/bcoschain/stub.toml
```

如果搭FISCO BCOS链采用的都是默认配置，那么将会得到一条单群组四节点的链，群组ID为1，各个节点的channel端口分别为`20200, 20201, 20202, 20203`，则将[channelService.connectionsStr]设置为[‘127.0.0.1:20200’,’127.0.0.1:20201’,’127.0.0.1:20202’,’127.0.0.1:20203’]。

- 配置合约资源

在前面的步骤中，已经通过FISCO BCOS控制台部署了一个`HelloWorld`合约，地址为`0x04ae9de7bc7397379fad6220ae01529006022d1b`

那么可在配置文件中注册一条合约资源信息：

```
[[resources]]
    # name cannot be repeated
    name = 'HelloWorldContract'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x04ae9de7bc7397379fad6220ae01529006022d1b'
```

完成了上述的步骤，那么已经完成了FISCO BCOS Stub的连接配置，并注册了一个合约资源，最终的`stub.toml`文件如下：

```
[common]
    stub = 'bcoschain' # stub must be same with directory name
    type = 'BCOS'

[smCrypto]
    # boolean
    enable = false

[account]
    accountFile = 'classpath:/stubs/bcoschain/
↪0x0ee5b8ee4af461cac320853aebb7a68d3d4858b4.pem'
```

(continues on next page)

(continued from previous page)

```
    password = ''  # if you choose .p12, then password is required

[channelService]
    timeout = 60000  # millisecond
    caCert = 'classpath:/stubs/bcoschain/ca.crt'
    sslCert = 'classpath:/stubs/bcoschain/sdk.crt'
    sslKey = 'classpath:/stubs/bcoschain/sdk.key'
    groupId = 1
    connectionsStr = ['127.0.0.1:20200','127.0.0.1:20201','127.0.0.1:20202','127.0.
↪0.1:20203']

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWorldContract'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x04ae9de7bc7397379fad6220ae01529006022d1b'
```

启动并测试

```
bash start.sh
```

如果启动不成功，那么请检查FISCO BCOS Stub的配置是否正确。

```
curl http://127.0.0.1:8250/payment/bcoschain/HelloWorldContract/status
```

如果得到如下输出，则代表跨链服务已成功启动，并且通过配置文件注册的跨链资源也成功加载了。

```
{"version":"0.2","result":0,"message":null,"data":"exists"}
```

控制台调用

启动WeCross控制台，调用HelloWorld合约

```
[server1]> call payment.bcoschain.HelloWorldContract String get
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        Hello, World!
    ]
}
[server1]> sendTransaction payment.bcoschain.HelloWorldContract Void set "Hello␣
↪WeCross!"
Receipt{
    errorCode=0,
    errorMessage='null',
    hash='0x0f87fc4588d38cce2fbaff2b6d1bbe6c627dd31200f6b95334e69fd367b0b3ec',
    result=[

    ]
}
[server1]> call payment.bcoschain.HelloWorldContract String get
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
```

(continues on next page)

(continued from previous page)

```
    result=[
        Hello WeCross!
    ]
}
```

CHAPTER 3

应用场景

CHAPTER 4

操作手册

4.1 配置文件

WeCross通过配置文件管理Stub以及每个Stub中的跨链资源，启动时首先加载配置文件，根据配置去初始化各个Stub以及相应的资源，如果配置出错，则启动失败。

Note:

- WeCross在开发阶段尝试了XML、YAML等不同的配置了文件格式，在综合易用性、灵活性、功能性以及需求契合性等多方面因素的考量后，最终采用了Toml的配置格式。
- Toml是一种语义化配置文件格式，可以无二义性地转换为一个哈希表，支持多层级配置，无缩进和空格要求，配置容错率高。

4.1.1 配置结构

WeCross的配置分为根配置和子配置两级，根配置负责P2P、Tomcat Server等和WeCross服务相关的必要信息；子配置主要有两个重要的配置项，分别是Stub对应区块链客户端的建连配置以及区块链上的资源列表信息。如果子配置缺省，WeCross仍能启动成功，只是不能提供任何跨链服务。

WeCross根配置文件名为`wecross.toml`，子配置文件名为`stub.toml`，配置的目录结构如下：

```
.
├── log4j2.xml    // 日志配置文件，无需更改
├── stubs
│   ├── bcos
│   │   └── stub.toml
│   └── fabric
│       └── stub.toml
└── wecross.toml
```

4.1.2 根配置

WeCross的根配置示例文件`wecross-sample.toml`编译后位于`WeCross/dist/conf`目录，使用前需拷贝成指定文件名`wecross.toml`。

```
cd dist
cp conf/wecross-sample.toml conf/wecross.toml
```

配置示例如下：

```
[common]
    network = 'payment'
    visible = true

[stubs]
    path = 'classpath:stubs'

[server]
    address = '127.0.0.1'
    port = 8250

[p2p]
    listenIP = '0.0.0.0'
    listenPort = 25500
    caCert = 'classpath:p2p/ca.crt'
    sslCert = 'classpath:p2p/node.crt'
    sslKey = 'classpath:p2p/node.key'
    peers = ['127.0.0.1:25501','127.0.0.1:25502']

[test]
    enableTestResource = false
```

根配置有五个配置项，分别是[common]、[stubs]、[server]、[p2p]以及[test]，各个配置项含义如下：

- [common] 通用配置
  - network：字符串；跨链网络标识符；通常一种跨链业务/应用为一个跨链网络
  - visible：布尔；可见性；标明当前跨链网络下的资源是否对其他跨链网络可见
- [stubs] Stub配置
  - path：字符串；Stub配置的根目录；WeCross从该目录下去加载各个Stub的配置
- [server] Tomcat Server配置
  - address：字符串；本机IP地址；WeCross通过Spring Boot内置的Tomcat启动Web服务
  - port：整型；WeCross服务端口；需要未被占用
- [p2p] 组网配置
  - listenIP：字符串；监听地址；一般为'0.0.0.0'
  - listenPort ： 整型；监听端口；WeCross节点之间的消息端口
  - caCert ： 字符串；根证书路径；拥有相同根证书的WeCross节点才能互相通讯
  - sslCert ： 字符串；节点证书路径；WeCross节点的证书
  - slKey ： 字符串；节点私钥路径；WeCross节点的私钥
  - peers：字符串数组；peer列表；需要互相连接的WeCross节点列表
- [test] 测试配置
  - enableTestResource：布尔；测试资源开关；如果开启，那么即使没有配置Stub的资源信息，也可以根据测试资源体验WeCross的部分功能。

注：

1. WeCross启动时会把conf目录指定为classpath，若配置项的路径中开头为classpath:，则以conf为相对目录。

2. [p2p]配置项中的证书和私钥可以通过build_cert.sh脚本生成，-h可查看帮助信息。使用示例如下：

```
# 生成根证书ca.crt
sh build_cert.sh -c

# 生成节点证书和私钥node.crt和node.key
# 必须先生成根证书ca.crt，才能生成节点证书和私钥
# 该命令还会生成node.nodeid，主要用于P2P出错时的程序调试，可以忽略
sh build_cert.sh -n

# 批量生成节点证书和私钥
# -C后面为数量
sh build_cert.sh -n -C 10
```

1. 若通过build_wecross.sh脚本生成的项目，那么已自动帮忙配置好了wecross.toml，包括P2P的配置，其中Stub的根目录默认为stubs。

4.1.3 子配置

子配置即每个Stub的配置，是WeCross跨链业务的核心，配置了Stub和区块链交互所需的信息，以及注册了各个链需要参与跨链的资源。WeCross启动后会在wecross.toml中所指定的Stub的根目录下去遍历所有的一级目录，目录名即为Stub的名字，不同的目录代表不同的链，然后尝试读取每个目录下的stub.toml‘文件。

目前WeCross支持的Stub类型包括：FISCO BCOS、Fabric和JDChain

FISCO BCOS

配置示例如下：

```
[common]
    stub = 'bcos' # stub must be same with directory name
    type = 'BCOS'

[smCrypto]
    # boolean
    enable = false

[account]
    accountFile = 'classpath:/stubs/bcos/
↪0xa1ca07c7ff567183c889e1ad5f4dcd37716831ca.pem'
    password = ''  # if you choose .p12, then password is required

[channelService]
    timeout = 60000  # millisecond
    caCert = 'classpath:/stubs/bcos/ca.crt'
    sslCert = 'classpath:/stubs/bcos/sdk.crt'
    sslKey = 'classpath:/stubs/bcos/sdk.key'
    groupId = 1
    connectionsStr = ['127.0.0.1:20200']

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWorldContract'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x8827cca7f0f38b861b62dae6d711efe92a1e3602'
[[resources]]
    name = 'FirstTomlContract'
```

(continues on next page)

(continued from previous page)

```
    type = 'BCOS_CONTRACT'
    contractAddress = '0x584ecb848dd84499639fbe2581bfb8a8774b485c'
```

配置方法详见FISCO BCOS Stub配置

Fabric

配置示例如下：

```
[common]
    stub = 'fabric' # stub must be same with directory name
    type = 'FABRIC'

# fabricServices is a list
[fabricServices]
     channelName = 'mychannel'
     orgName = 'Org1'
     mspId = 'Org1MSP'
     orgUserName = 'Admin'
     orgUserKeyFile = 'classpath:/stubs/fabric/
↪5895923570c12e5a0ba4ff9a908ed10574b475797b1fa838a4a465d6121b8ddf_sk'
     orgUserCertFile = 'classpath:/stubs/fabric/Admin@org1.example.com-cert.pem'
     ordererTlsCaFile = 'classpath:/stubs/fabric/tlsca.example.com-cert.pem'
     ordererAddress = 'grpcs://127.0.0.1:7050'

[peers]
    [peers.a]
        peerTlsCaFile = 'classpath:/stubs/fabric/tlsca.org1.example.com-cert.pem'
        peerAddress = 'grpcs://127.0.0.1:7051'
    [peers.b]
         peerTlsCaFile = 'classpath:/stubs/fabric/tlsca.org2.example.com-cert.pem'
         peerAddress = 'grpcs://127.0.0.1:9051'

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWorldContract'
    type = 'FABRIC_CONTRACT'
    chainCodeName = 'mycc'
    chainLanguage = "go"
    peers=['a','b']
[[resources]]
    name = 'FirstTomlContract'
    type = 'FABRIC_CONTRACT'
    chainLanguage = "go"
    chainCodeName = 'cscc'
    peers=['a','b']
```

配置方法详见Fabric Stub配置

JDChain

配置示例如下：

```
[common]
    stub = 'jd' # stub must be same with directory name
    type = 'JDCHAIN'

# jdServices is a list
```

(continues on next page)

(continued from previous page)

```
[[jdServices]]
     privateKey = '0000000000000000'
     publicKey = '111111111111111'
     password = '222222222222222'
     connectionsStr = '127.0.0.1:18081'
[[jdServices]]
     privateKey = '0000000000000000'
     publicKey = '111111111111111'
     password = '222222222222222'
     connectionsStr = '127.0.0.1:18082'

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWorldContract'
    type = 'JDCHAIN_CONTRACT'
    contractAddress = '0x38735ad749aebd9d6e9c7350ae00c28c8903dc7a'
[[resources]]
    name = 'FirstTomlContract'
    type = 'JDCHAIN_CONTRACT'
    contractAddress = '0x38735ad749aebd9d6e9c7350ae00c28c8903dc7a'
```

配置方法详见JDChain Stub配置

4.2 控制台

4.3 WeCross SDK

4.4 JSON-RPC API

4.5 脚本介绍

为了方便用户使用，WeCross提供了丰富的脚本，所有脚本编译后都位于`dist`目录，本章节将对这些脚步做详细介绍。

4.5.1 启动脚本

启动脚本`start.sh`用于启动WeCross服务，启动过程中的完整信息记录在start.out中。

```
bash start.sh
```

成功输出：

```
Wecross start successfully
```

失败输出：

```
WeCross start failed
See logs/error.log for details
```

4.5.2 停止脚本

停止脚本`stop.sh`用于停止WeCross服务。

```
bash  stop.sh
```

4.5.3 构建WeCross脚本

构建WeCross脚本`build_wecross.sh`用于快速部署WeCross，该脚本默认从GitHub下载`master`分支代码进行相关环境的搭建。

`build_wecross.sh`主要完成的工作包括：下载和编译源码，生成P2P证书，帮助用户完成根配置文件`wecross.toml`的配置。

可通过-h查看帮助信息:

```
bash build_wecross.sh -h

Usage:
    -i [Network ID] [IP] [Port] [Port]      Init wecross project by wecross network id, ip, rpc_port and p2p_port, e.g: payment 127.0.0.1 8250 25500
    -f [Network ID] [File]                  Init wecross project by wecross network id and ip&ports file. file should be splited by line "ip rpc_port p2p_port" e.g: 127.0.0.1 8250 25500
    -h                                      Call for help
e.g
    bash build_wecross.sh -i payment 127.0.0.1 8250 25500
```

- **`i`选项:** 用于指定跨链网络标识，RPC监听的IP和端口，以及用于P2P通讯的端口。
- **`f`选项**
  - 参数包括跨链网络标识和文件名。
  - 用于根据配置文件生成多个WeCross项目，最后输出`tar.gz`格式的压缩文件。
  - 文件按行分割，每一行表示一个WeCross项目，格式为`[IP] [Port] [Port]`，每行内的项使用空格分割，**不可有空行**。

下面是一个配置文件的例子，每个配置项以空格分隔。

```
192.168.0.1 8250 25500
192.168.0.1 8251 25501
192.168.0.2 8252 25502
192.168.0.3 8253 25503
192.168.0.4 8254 25504
```

如果执行成功，则输出如下信息，生成的五个WeCross项目互为Peer。

```
Create 192.168.0.1-8250-25500.tar.gz  successfully
Create 192.168.0.1-8251-25501.tar.gz  successfully
Create 192.168.0.2-8252-25502.tar.gz  successfully
Create 192.168.0.3-8253-25503.tar.gz  successfully
Create 192.168.0.4-8254-25504.tar.gz  successfully
Build Wecross successfully
```

4.5.4 创建Stubs配置脚本

创建Stubs配置脚本`create_stubs_config.sh`用于快速创建各类Stub的配置文件。

目前支持的类型包括`FISCO BCOS`, `Fabric`, `JDChain`

`build_wecross.sh`主要完成的工作包括：创建目录，根据Stub类型拷贝配置示例，然后完成stub.toml的部分配置。

可通过-h查看帮助信息:

```
Usage:
    -a [Root Dir] [[Stub type] [Stub name]]   Generate stub configuration by list␣
↪of types and names
                                              Supported types: BCOS, FABRIC, JD
    -b [Root Dir] [Stub name]                 Generate FISCO BCOS stub␣
↪configuration
    -f [Root Dir] [Stub name]                 Generate FABRIC stub configuration
    -j [Root Dir] [Stub name]                 Generate JDChain stub configuration
    -h                                        Call for help
e.g
    bash create_stubs_config.sh -a stubs BCOS bcoschain FABRIC fabricchain
```

- **a选项:** 批量生成Stub配置文件。输入Stub根目录，Stub类型和跨链标识的列表。
- **b选项:** 生成FISCO BCOS Stub配置文件。输入Stub根目录，以及跨链标识。已帮忙生成了FISCO BCOS的账户文件。
- **f选项:** 生成Fabric Stub配置文件。输入Stub根目录，以及跨链标识。
- **j选项:** 生成JDChain Stub配置文件。输入Stub根目录，以及跨链标识。

例如:

```
bash create_stubs_config.sh -a stubs BCOS bcoschain FABRIC fabricchain JD jdchain
```

在stubs目录下查看目录结构:

```
tree
.
├── bcos
│   └── stub-sample.toml
├── bcoschain
│   ├── 0x0ee5b8ee4af461cac320853aebb7a68d3d4858b4.pem
│   └── stub.toml
├── fabric
│   └── stub-sample.toml
├── fabricchain
│   └── stub.toml
├── jd
│   └── stub-sample.toml
└── jdchain
    └── stub.toml
```

4.5.5 创建P2P证书脚本

创建P2P证书脚本create_cert.sh用于创建P2P证书文件。WeCross Router之间通讯需要证书用于认证，只有具有相同ca.crt根证书的WeCross Router直接才能建立连接。

可通过-h查看帮助信息:

```
Usage:
    -c                                  [Optional] generate ca certificate
    -C <number>                         [Optional] the number of node certificate␣
↪generated, work with '-n' opt, default: 1
    -D <dir>                            [Optional] the ca certificate directory,␣
↪work with '-n', default: './'
    -d <dir>                            [Required] generated target_directory
    -n                                  [Optional] generate node certificate
    -t                                  [Optional] cert.cnf path, default: cert.cnf
    -h                                  [Optional] Help
e.g
    bash create_cert.sh -c -d ./ca
```

(continues on next page)

(continued from previous page)

```
bash create_cert.sh -n -D ./ca -d ./ca/node
bash create_cert.sh -n -D ./ca -d ./ca/node -C 10
```

- **c选项:** 生成ca证书，只有生成了ca证书，才能生成节点证书。
- **n选项:** 生成节点证书。
- **C选项:** 配合-n，指定生成节点证书的数量。
- **D选项:** 配合-n，指定ca证书路径。
- **d选项:** 指定输出目录。
- **t选项:** 指定`cert.cnf`的路径

4.5.6 创建FISCO BCOS账户脚本

脚本`create_bcos_account.sh`用于生成FISCO BCOS的账户。

FISCO BCOS账户文件有两种类型`.pem`, `.p12`，其中`.p12`需要输入口令。

脚本`create_bcos_account.sh`会同时生成两种类型的账户，用户可根据需求选择使用不同的账户。

CHAPTER 5

跨链接入

5.1 接入FISCO BCOS

5.1.1 FISCO BCOS环境搭建

FISCO BCOS环境搭建参考部署文档

5.1.2 FISCO BCOS stub配置

WeCross配置好之后，默认的conf目录结构如下：

```
├── log4j2.xml
├── p2p
│   ├── ca.crt
│   ├── node.crt
│   ├── node.key
│   └── node.nodeid
├── stubs
│   ├── bcos
│   │   └── stub-sample.toml
│   ├── fabric
│   │   └── stub-sample.toml
│   └── jd
│       └── stub-sample.toml
├── wecross-sample.toml
└── wecross.toml
```

假定当前目录在conf，执行如下操作：

```
    cd bcos;
    cp stub-sample.toml  stub.toml
```

查看stub.toml，可以看到文件内容如下：

```
[common]
    stub = 'bcos'
```

(continues on next page)

(continued from previous page)

```
    type = 'BCOS'

[guomi]
    # boolean
    enable = false

[account]
    accountFile = 'classpath:/stubs/bcos/
↪0xa1ca07c7ff567183c889e1ad5f4dcd37716831ca.pem'

[channelService]
    timeout = 60000  # millisecond
    caCert = 'classpath:/stubs/bcos/ca.crt'
    sslCert = 'classpath:/stubs/bcos/sdk.crt'
    sslKey = 'classpath:/stubs/bcos/sdk.key'
    groupId = 1
    connectionsStr = ['127.0.0.1:20200']

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWorldContract'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x8827cca7f0f38b861b62dae6d711efe92a1e3602'
[[resources]]
    name = 'FirstTomlContract'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x8827cca7f0f38b861b62dae6d711efe92a1e3602'
```

[account]:发送交易的账户信息。 accountFile:发送交易的账户信息,账户产生请参考账户创建

[channelService]:连接的FISCO BCOS的节点信息配置。 timeout:连接超时时间，单位毫秒。 caCert:链证书，证书和私钥相关的文件是从FISCO BCOS链中拷贝。 sslCert:SDK证书，证书和私钥相关的文件是从FISCO BCOS链中拷贝。 sslKey:SDK私钥，证书和私钥相关的文件是从FISCO BCOS链中拷贝。 groupId:groupId。 connectionsStr:连接节点的地址，多个地址使用,分隔。

[[resources]]: 配置资源相关信息，包括资源名称，类型，合约地址等。

name:资源名称，需要唯一。 type:类型，默认都是BCOS_CONTRACT。 contractAddress:合约地址。

5.2 接入Fabric

5.2.1 Fabric环境搭建

5.2.2 前置准备工作

docker安装

centos环境下docker安装

docker安装需要满足内核版本不低于3.10。

卸载旧版本

```
sudo yum remove docker docker-common docker-selinux docker-engine
```

安装依赖

```
sudo yum install -y yum-utils device-mapper-persistent-data lvm2
```

设置yum源

```
sudo  yum-config-manager --add-repo https://download.docker.com/linux/centos/
↪docker-ce.repo
```

安装docker

```
sudo yum install docker-ce
```

启动docker

```
service docker start
```

ubuntu环境下docker安装

卸载旧版本

```
sudo apt-get remove docker docker-engine docker-ce docker.io
```

更新apt包索引

```
sudo apt-get update
```

安装HTTPS依赖库

```
sudo apt-get install -y apt-transport-https ca-certificates curl software-
↪properties-common
```

添加Docker官方的GPG密钥

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

设置stable存储库

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
↪$(lsb_release -cs) stable"
```

更新apt包索引

```
sudo apt-get update
```

安装最新版本的Docker-ce

```
sudo apt-get install -y docker-ce
```

启动docker

```
service docker start
```

docker-compose安装

安装docker-compose

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.18.0/docker-
↪compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

查看docker-compose版本

```
    docker-compose --version
    docker-compose version 1.18.0, build 8dd22a9
```

go安装

go版本需要1.11以上。

安装go

```
    wget https://dl.google.com/go/go1.11.5.linux-amd64.tar.gz
    sudo tar zxvf go1.11.5.linux-amd64.tar.gz -C /usr/local
```

环境变量配置

创建文件夹和软链

```
cd ~
mkdir /data/go
ln -s /data/go go
```

修改环境变量

```
sudo vim /etc/profile
```

添加如下内容:

```
export PATH=$PATH:/usr/local/go/bin
export GOROOT=/usr/local/go
export GOPATH=/home/app/go/
export PATH=$PATH:$GOROOT/bin
```

修改完成后,执行如下操作:

```
    source /etc/profile
```

确认go环境安装成功

新增helloworld.go文件，内容如下：

```
    package main
    import "fmt"
    func main() {
    fmt.Println("hello world.")
    }
```

运行helloworld.go文件

```
    go run helloworld.go
```

如果安装和配置成功,将输出:

```
    hello world.
```

Fabric链搭建

目录准备

```
    cd ~
    mkdir go/src/github.com/hyperledger -p
    cd go/src/github.com/hyperledger
```

源码下载

```
    git clone -b release-1.4 https://github.com/hyperledger/fabric.git
    git clone -b release-1.4 https://github.com/hyperledger/fabric-samples.git
```

源码编译

```
    cd ~/go/src/github.com/hyperledger/fabric
    make release
```

环境变量修改

sudo vi /etc/profile 新增如下一行

```
    export PATH=$PATH:$GOPATH/src/github.com/hyperledger/fabric/release/linux-
↪amd64/bin
```

修改完成后执行

```
    source /etc/profile
```

节点启动

```
    cd ~/go/src/github.com/hyperledger/fabric-samples/first-network
    ./byfn.sh up
```

验证

执行如下命令，进入容器

```
    sudo docker exec -it cli bash
```

进入操作界面，执行如下命令:

```
    peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
    peer chaincode invoke -o orderer.example.com:7050 --tls true --cafile /opt/
↪gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/
↪example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.
↪pem -C mychannel -n mycc --peerAddresses peer0.org1.example.com:7051 --
↪tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
↪peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt --
↪peerAddresses peer0.org2.example.com:9051 --tlsRootCertFiles /opt/gopath/src/
↪github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.example.com/
↪peers/peer0.org2.example.com/tls/ca.crt -c '{"Args":["invoke","b","a","1"]}'
    peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
```

5.2.3 Fabric stub配置

WeCross配置好之后，默认的conf目录结构如下:

```
├── log4j2.xml
├── p2p
│   ├── ca.crt
│   ├── node.crt
│   ├── node.key
│   └── node.nodeid
├── stubs
│   ├── bcos
```

(continues on next page)

(continued from previous page)

```
│   │   └── stub-sample.toml
│   ├── fabric
│   │   └── stub-sample.toml
│   └── jd
│       └── stub-sample.toml
├── wecross-sample.toml
└── wecross.toml
```

假定当前目录在conf，执行如下操作:

```
    cd fabric;
    cp stub-sample.toml  stub.toml
```

查看stub.toml，可以看到文件内容如下:

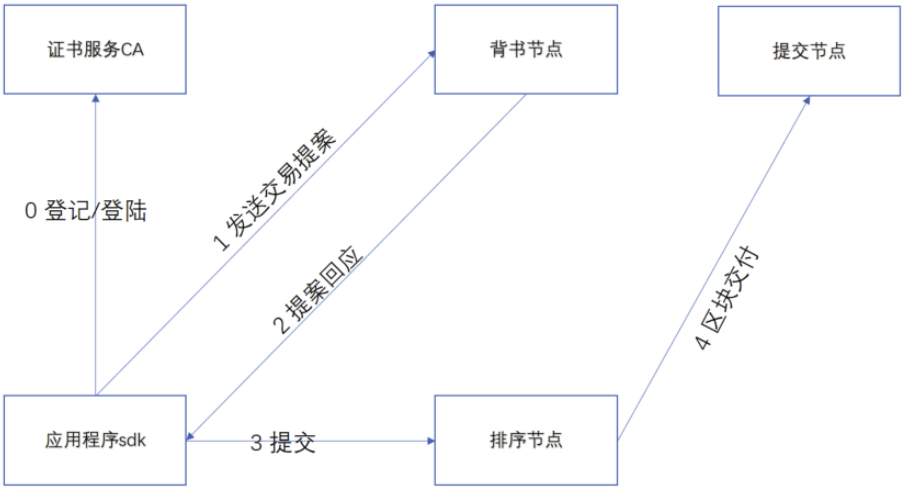
```
[common]
    stub = 'fabric'
    type = 'FABRIC'

# fabricServices is a list
[fabricServices]
     channelName = 'mychannel'
     orgName = 'Org1'
     mspId = 'Org1MSP'
     orgUserName = 'Admin'
     orgUserKeyFile = 'classpath:/fabric/
↪5895923570c12e5a0ba4ff9a908ed10574b475797b1fa838a4a465d6121b8ddf_sk'
     orgUserCertFile = 'classpath:/fabric/Admin@org1.example.com-cert.pem'
     ordererTlsCaFile = 'classpath:/fabric/tlsca.example.com-cert.pem'
     ordererAddress = 'grpcs://127.0.0.1:7050'

[peers]
    [peers.a]
        peerTlsCaFile = 'classpath:/fabric/tlsca.org1.example.com-cert.pem'
        peerAddress = 'grpcs://127.0.0.1:7051'
    [peers.b]
         peerTlsCaFile = 'classpath:/fabric/tlsca.org2.example.com-cert.pem'
         peerAddress = 'grpcs://127.0.0.1:9051'

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWorldContract'
    type = 'FABRIC_CONTRACT'
    chainCodeName = 'mycc'
    chainLanguage = "go"
    peers=['a','b']
```

再讲述配置文件配置之前，需要讲述下fabric的交易运行过程：



fabric交易流程

[fabricServices]:配置的是fabric的用户证书以及连接的order节点信息。channelName:channel名称，每个Channel之间是相互隔离。orgName:机构名称，使用默认即可。mspId:使用默认即可。orgUserName:机构用户名称，使用默认即可。也可以配置成User1，配置不一样会影响orgUserKeyFile和orgUserCertFile配置。orgUserKeyFile:用户私钥文件，如果orgUserName为Admin则此文件的源路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/keystore/*_sk，如果为User1，则此文件的源路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/keystore/*_sk ordererTlsCaFile:连接的order节点的证书。上述搭链生成的证书路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem。orgUserCertFile:用户私钥文件，如果orgUserName为Admin则此文件的源路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/signcerts/Admin@org1.example.com-cert.pem，如果为User1，则此文件的源路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/signcerts/User1@org1.example.com-cert.pem ordererTlsCaFile:连接的order节点的证书。上述搭链生成的证书路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem。ordererAddress:连接的order节点的地址，默认将ordererAddress中127.0.0.1替换为实际ip地址的即可。

[peers]:用于配置背书节点集合信息，包括证书文件和地址。默认将peerAddress中127.0.0.1替换为实际ip地址的即可，上述搭链的证书路径分别为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network//crypto-config/peerOrganizations/org1.example.com/tlsca/tlsca.org1.example.com-cert.pem和$GOPATH/src/github.com/hyperledger/fabric-samples/first-network//crypto-config/peerOrganizations/org2.example.com/tlsca/tlsca.org2.example.com-cert.pem。

[[resources]]:用于配置合约信息 [fabricServices]配置的是fabric的用户证书以及连接的order节点信息。channelName：channel名称，每个Channel之间是相互隔离。orgName：机构名称，使用默认即可。mspId：使用默认即可。orgUserName：机构用户名称，使用默认即可。也可以配置成User1，配置不一样会影响orgUserKeyFile和orgUserCertFile配置。orgUserKeyFile：用户私钥文件，如果orgUserName为Admin则此文件的源路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.

com/users/Admin@org1.example.com/msp/keystore/*_sk，如果为User1，则此文件的源路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/keystore/*_sk ordererTlsCaFile：连接的order节点的证书。上述搭链生成的证书路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem。 orgUserCertFile：用户私钥文件，如果orgUserName为Admin则此文件的源路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/signcerts/Admin@org1.example.com-cert.pem，如果为User1，则此文件的源路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/signcerts/User1@org1.example.com-cert.pem ordererTlsCaFile：连接的order节点的证书。上述搭链生成的证书路径为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network/crypto-config/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem。 ordererAddress：连接的order节点的地址，默认将ordererAddress中127.0.0.1替换为实际ip地址的即可。

[peers]:用于配置背书节点集合信息，包括证书文件和地址。默认将peerAddress中127.0.0.1替换为实际ip地址的即可，上述搭链的证书路径分别为$GOPATH/src/github.com/hyperledger/fabric-samples/first-network//crypto-config/peerOrganizations/org1.example.com/tlsca/tlsca.org1.example.com-cert.pem和$GOPATH/src/github.com/hyperledger/fabric-samples/first-network//crypto-config/peerOrganizations/org2.example.com/tlsca/tlsca.org2.example.com-cert.pem。

[[resources]]:用于配置合约信息

name:资源名称，需要唯一。

type:类型，默认都是FABRIC_CONTRACT。

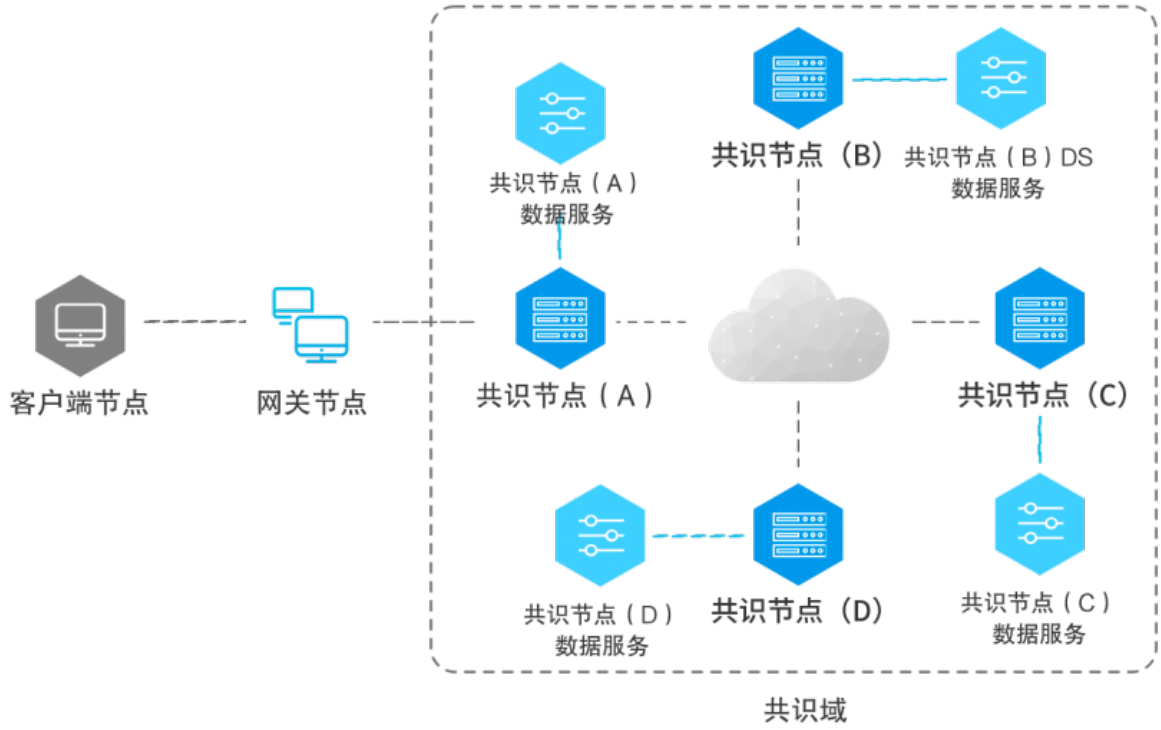
chainCodeName:chainCode名称，由链码初始化时指定。

chainLanguage:合约代码的开发语言，当前包括go,node,java；分别代表go，nodejs和java语言。

peers: 背书节点列表，必须是[peers]的子集。

5.3 接入JDChain

5.3.1 JDChain逻辑架构图



JDChain架构图

5.3.2 JDChain部署流程

JDChain部署流程

为简化部署条件、方便开发者学习，我们的示例使用一台服务器进行部署演示，因此我们将4个共识节点的端口进行如下约定： 管理工具的端口定义分别为：8000/8001/8002/8003 peer节点的启动端口定义为：7080/7081/7082/7083 请确保上述端口没有被占用。

附件下载和解压

```
mkdir ~/jdchain
cd  ~/jdchain
wget http://storage.jd.com/jd.block.chain/jdchain-peer-1.1.0.RELEASE.zip
 mkdir ~/jdchain/peer0/
```

(continues on next page)

(continued from previous page)

```
mkdir ~/jdchain/peer1/
mkdir ~/jdchain/peer2/
mkdir ~/jdchain/peer3/
unzip jdchain-peer-1.1.0.RELEASE.zip -d ~/jdchain/peer0/
unzip jdchain-peer-1.1.0.RELEASE.zip -d ~/jdchain/peer1/
unzip jdchain-peer-1.1.0.RELEASE.zip -d ~/jdchain/peer2/
unzip jdchain-peer-1.1.0.RELEASE.zip -d ~/jdchain/peer3/
```

修改manager和peer监听端口

修改manager监听端口

分别针对~/jdchain/peer1/ ~/jdchain/peer2/ ~/jdchain/peer3/目录下manager-startup.sh文件修改监听端口，分别修改为8001,8002,8003

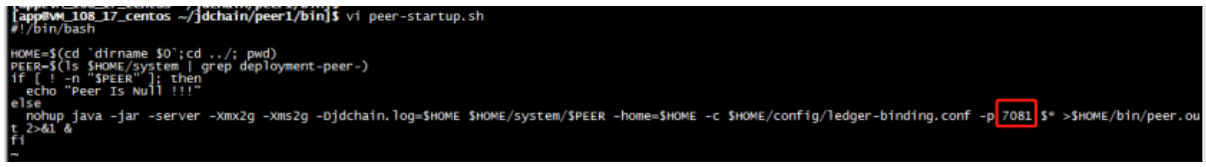
```
cd ~/jdchain/peer1/bin/
vi manager-startup.sh
```

manager.png

修改peer监听端口

分别针对~/jdchain/peer1/ ~/jdchain/peer2/ ~/jdchain/peer3/目录下peer-startup.sh文件修改监听端口,分别改为7081，7082，7083

```
cd ~/jdchain/peer1/bin/
vi peer-startup.sh
```



peer.png

启动管理端

四个节点对应端口的管理工具，例如：http://192.168.0.1:8000（请自行替换对应ip和端口）

通过如下命令启动管理端

```
cd  ~/jdchain/peer0/bin
sh manager-startup.sh

cd  ~/jdchain/peer1/bin
sh manager-startup.sh

cd  ~/jdchain/peer2/bin
sh manager-startup.sh

cd  ~/jdchain/peer3/bin
sh manager-startup.sh
```

启动完成后，可以看到服务器新增了8000~8003的端口监听。同时可以使用 tail -f jump.out查看日志。

```
[app@VM_108_17_centos ~/jdchain/peer0/bin]$ netstat -pan|grep -i listen|grep 800
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 0.0.0.0:8000            0.0.0.0:*               LISTEN      13962/java
tcp        0      0 0.0.0.0:8001            0.0.0.0:*               LISTEN      14667/java
tcp        0      0 0.0.0.0:8002            0.0.0.0:*               LISTEN      14942/java
tcp        0      0 0.0.0.0:8003            0.0.0.0:*               LISTEN      15236/java
[app@VM_108_17_centos ~/jdchain/peer0/bin]$
```

manager_port.png

创建公私钥

分别登录四个节点对应端口的管理工具，例如：http://192.168.0.1:8000（请自行替换对应ip和端口）

点击左侧菜单：公私钥管理→生成公私钥。在弹出的界面中填写相关信息：

generate_ppk.png

账本初始化

创建数据账本时，四个节点需要指定其中一个作为协调方，发起账本的创建。另外三个作为参与方，接受协调方的邀请码，共同创建数据账本。

点击左侧菜单：账本→初始化账本。

四个节点都配置完成后，请确保四个peer节点的配置均无误的前提下，在尽可能短的时间内同时点击界面下方的“保存配置信息”按钮。

协调方配置信息参考如下：

京

东链协调方.png

参与配置信息参考如下：

京东链参与方.png

四个参与方都点击“保存配置信息”后，配置信息会在四个参与方之间共享。界面如下：

docs/stubs/../images/jdchainInit.png

参与方.png

然后点击界面下方的“开始”按钮，则启动四个参与方的初始化操作，初始化进度会在下方展示。最终展示界面如下：

docs/stubs/../images/jdchainSave.png

启动前.png

然后点击菜单：账本→查看账本，刚生成的账本会在内容区展示。点击其中的“启动节点”按钮，即可启动peer节点（见下图）。

docs/stubs/../images/jdchainLoad.png

启动后.png

启动无误后，状态展示为：已启动→已加载。

安装和启动Gateway节点

附件下载和解压

```
cd  ~/jdchain
wget http://storage.jd.com/jd.block.chain/jdchain-gateway-1.1.0.RELEASE.zip
mkdir ~/jdchain/gateway/
unzip jdchain-gateway-1.1.0.RELEASE.zip -d ~/jdchain/gateway/
```

配置文件修改

gateway的配置需要配置3个东西，公钥，私钥，以及加密后的密码。公钥，私钥可以通过管理端查看。加密后的口令查看命令如下：

```
cat ~/jdchain/peer0/config/keys/*.pwd
```

gateway.png

配置完成之后，启动gateway。

```
cd  ~/jdchain
cd /home/app/jdchain/gateway/bin
sh startup.sh
```

通过web页面访问区块链浏览器，格式为：http://192.168.0.1:18081，（请自行替换对应ip和端口）。界

面如下： 

浏览器.png

5.3.3 JDChain stub配置

WeCross配置好之后，默认的conf目录结构如下：

```
├── log4j2.xml
├── p2p
│   ├── ca.crt
│   ├── node.crt
│   ├── node.key
│   └── node.nodeid
├── stubs
│   ├── bcos
│   │   └── stub-sample.toml
│   ├── fabric
│   │   └── stub-sample.toml
│   └── jd
│       └── stub-sample.toml
├── wecross-sample.toml
└── wecross.toml
```

假定当前目录在conf，执行如下操作：

```
    cd jd;
    cp stub-sample.toml  stub.toml
```

查看stub.toml，可以看到文件内容如下：

```
    [common]
    stub = 'jd'
    type = 'JDCHAIN'

# jdServices is a list
[[jdServices]]
     privateKey = '0000000000000000'
     publicKey = '111111111111111'
     password = '222222222222222'
     connectionsStr = '127.0.0.1:18081'
[[jdServices]]
```

(continues on next page)

(continued from previous page)

```
    privateKey = '0000000000000000'
    publicKey = '111111111111111'
    password = '222222222222222'
    connectionsStr = '127.0.0.1:18082'

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWorldContract'
    type = 'JDCHAIN_CONTRACT'
    contractAddress = '0x38735ad749aebd9d6e9c7350ae00c28c8903dc7a'
[[resources]]
    name = 'FirstTomlContract'
    type = 'JDCHAIN_CONTRACT'
    contractAddress = '0x38735ad749aebd9d6e9c7350ae00c28c8903dc7a'
```

`[[jdServices]]`：配置的是WeCross连接的JDChain的gateway配置信息，包括gateway连接的JDChain的公私钥以及密码。`privateKey`：配置JDChain节点的私钥，搭链过程中的生成的私钥。`publicKey`：配置JDChain节点的公钥，搭链过程中的生成的公钥。`password`：密码，搭链过程输入的密码。`connectionsStr`：gateway地址。

`[[resources]]`：配置资源相关信息，包括资源名称，类型，合约地址等。

`name`:资源名称，需要唯一。`type`:类型，默认都是`JDCHAIN_CONTRACT`。`contractAddress`:合约地址。

CHAPTER 6

系统设计

CHAPTER 7

版本描述

7.1 1.0

CHAPTER 8

FAQ

CHAPTER 9

社区

9.1 QR