WeCross Documentation

发布 ***v1.0.0-rc1***

WeCross Community

2020 年 **05** 月 **12** 日

Contents

WeCross 是由微众银行自主研发并完全开源的分布式商业区块链跨链协作平台。该平台能解决业界主流的区块链产品间接口不互通、无法协作的问题，以及区块链系统无法平行扩展、计算能力和存储容量存在瓶颈等问题。WeCross作为未来分布式商业区块链互联的基础架构，秉承公众联盟链多方参与、共享资源、智能协同和价值整合的理念，致力于促进跨行业、机构和地域的跨区块链价值交换和商业合作，实现了高效、通用和安全的区块链跨链协作机制。

CHAPTER 1

平台介绍

1.1 基本介绍

区块链作为构建未来价值互联网的重要基础设施，深度融合分布式存储、点对点通信、分布式架构、共识机制、密码学等前沿技术，正在成为技术创新的前沿阵地。全球主要国家都在加快布局区块链技术，用以推动技术革新和产业变革。经过行业参与者十年砥砺前行，目前区块链在底层技术方案上已趋于完整和成熟，国内外均出现可用于生产环境的区块链解决方案。其所面向的创新应用场景覆盖广泛，已在对账与清结算、跨境支付、供应链金融、司法仲裁、政务服务、物联网、智慧城市等众多领域落地企业级应用。

在广泛的场景应用背后，来自于性能、安全、成本、扩展等方面的技术挑战也愈发严峻。目前不同区块链应用之间互操作性不足，无法有效进行可信数据流通和价值交换，各个区块链俨然成为一座座信任孤岛，很大程度阻碍了区块链应用生态的融合发展。未来，区块链想要跨越到真正的价值互联网，承担传递信任的使命，开启万链互联时代，需要一种通用、高效、安全的区块链跨链协作机制，实现跨场景、跨地域不同区块链应用之间的互联互通，以服务数量更多、地域更广的公众群体。

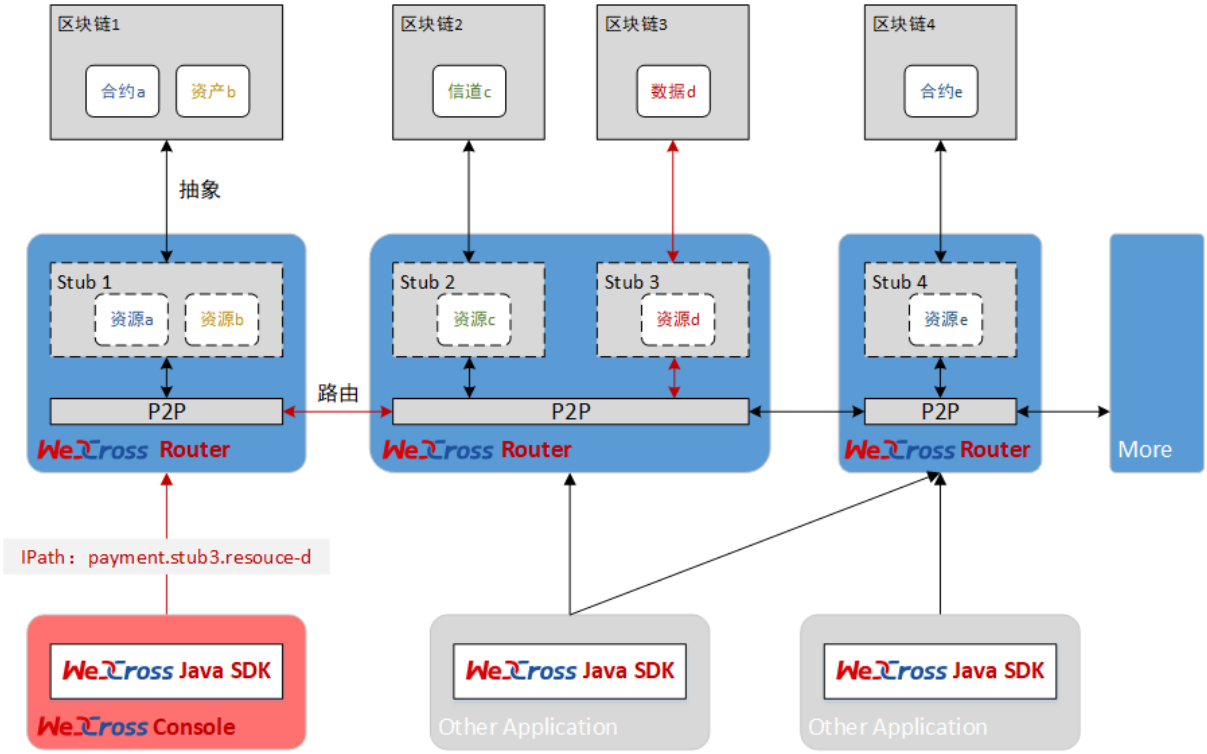
作为一家具有互联网基因的高科技、创新型银行，微众银行自成立之初即高度重视新兴技术的研究和探索，在区块链领域积极开展技术积累和应用实践，不断致力于运用区块链技术提升多机构间的协作效率和降低协作成本，支持国家推进关键技术安全可控战略和推动社会普惠金融发展。微众银行区块链团队基于一揽子自主研发并开源的区块链技术方案，针对不同服务形态、不同区块链平台之间无法进行可信连接与交互的行业痛点，研发区块链跨链协作平台——WeCross，以促进跨行业、机构和地域的跨区块链信任传递和商业合作。

WeCross 着眼应对区块链行业现存挑战，不局限于满足同构区块链平行扩展后的可信数据交换需求，还进一步探索异构区块链之间因底层架构、数据结构、接口协议、安全机制等多维异构性导致无法互联互通问题的有效解决方案。作为未来区块链互联的基础设施，WeCross 秉承多方参与、共享资源、智能协同和价值整合的理念，面向公众完全开源，欢迎广大企业及技术爱好者踊跃参与项目共建。

1.2 关键词

- **跨链路由（WeCross Router）**
  - 与链对接，对链上的资源进行抽象
  - 向外暴露统一的接口
  - 将调用请求路由至对应的区块链

- **控制台（WeCross Console）**
  - 命令行式的交互
  - 查询跨链信息，发送调用请求
- **跨链 SDK（WeCross Java SDK）**
  - WeCross开发工具包，供开发者调用WeCross
  - 集成于各种跨链APP中，提供统一的调用接口
  - 与跨链路由建立连接，调用跨链路由
- **跨链资源（Resource）**
  - 各种区块链上内容的抽象
  - 包括：合约、资产、信道、数据表
- **跨链适配器（Stub）**
  - 跨链路由中对接入的区块链的抽象
  - 跨链路由通过配置Stub与相应的区块链对接
  - FISCO BCOS需配置FISCO BCOS Stub、Fabric需配置Fabric Stub
- **IPath（Interchain Path）**
  - 跨链资源的唯一标识
  - 跨链路由根据IPath将请求路由至相应区块链上
- **跨链分区**
  - 多条链通过跨链路由相连，形成跨链分区
  - 跨链分区有唯一标识，即IPath中的第一项（`payment.stub3.resource-d`的`payment`）



1.3 更多资料

- WeCross白皮书
- WeCross官网

CHAPTER 2

程序版本

2.1 v1.0.0-rc1

功能

- 接入区块链
  - 适配 FISCO BCOS
  - 适配 Fabric
- 统一接口：跨链路由将各种区块链的操作接口进行抽象，向外暴露统一的调用API
- 路由请求：跨链路由可自动将调用请求路由至相应的区块链
- 交易验证：向FISCO BCOS的链发交易时，能验证交易上链后的Merkle证明

架构

- 跨链路由：对接不同区块链的服务，对区块链的调用接口进行统一的抽象，彼此互连，将操作请求路由至相应链
- 跨链SDK：Java语言的API，用统一的接口向不同的链发请求
- 控制台：方便的操作终端，方便进行查询和发送请求

工具

- 跨链分区搭建脚本
- 接入FISCO BCOS和Fabric的配置框架生成脚本

2.2 下载程序

2.2.1 下载WeCross

提供三种方式，根据网络环境选择合适的方式进行下载。

方式1：命令下载

```
bash <(curl -sL https://github.com/WeBankFinTech/WeCross/releases/download/
↪resources/download_wecross.sh)
```

方式2：命令下载（源码编译模式）

```
bash <(curl -sL https://github.com/WeBankFinTech/WeCross/releases/download/
↪resources/download_wecross.sh) -s
```

方式3：手动下载

- 国内资源：点击下载，MD5
- github release（下载最新版本的 `WeCross.tar.gz`）

  手动下载后解压

  ```
  tar -zxvf WeCross.tar.gz
  ```

解压后，目录下包含`WeCross/`文件夹。

2.2.2 下载WeCross控制台

同样提供三种方式，根据网络环境选择合适的方式进行下载。

方式1：命令下载

```
bash <(curl -sL https://github.com/WeBankFinTech/WeCross-Console/releases/download/
↪resources/download_console.sh)
```

方式2：命令下载（源码编译模式）

```
bash <(curl -sL https://github.com/WeBankFinTech/WeCross-Console/releases/download/
↪resources/download_console.sh) -s
```

方式3：手动下载

- 国内资源：点击下载，MD5
- github release（下载最新版本的 `WeCross-Console.tar.gz`）

  手动下载解压

  ```
  tar -zxvf WeCross-Console.tar.gz
  ```

下载后，目录下包含`WeCross-Console/`文件夹。

CHAPTER 3

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。

3.1 环境要求

3.1.1 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	4GB	8GB
核心	4核	8核
带宽	2Mb	10Mb

3.1.2 支持的平台

- Ubuntu 16.04及以上
- CentOS 7.2及以上
- MacOS 10.14及以上

3.1.3 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 -y openssl curl expect tree

# CentOS
sudo yum install -y openssl curl expect tree

# MacOS
brew install openssl curl expect tree
```

3.2 快速部署

本文档指导完成**跨链路由**和**跨链控制台**的部署。

- **跨链路由**：与区块链节点对接，并彼此互连，形成跨链分区，负责跨链请求的转发
- **跨链控制台**：查询和发送交易的操作终端

操作以~/wecross/目录下为例进行

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

3.2.1 部署 WeCross

下载WeCross，用WeCross中的工具生成跨链路由，并启动跨链路由。

下载WeCross

WeCross中包含了生成跨链路由的工具，执行以下命令进行下载（提供三种下载方式，可根据网络环境选择合适的方式进行下载），程序下载至当前目录WeCross/中。

```
bash <(curl -sL https://github.com/WeBankFinTech/WeCross/releases/download/
↪resources/download_wecross.sh)
```

生成跨链路由

用WeCross中的 build_wecross.sh生成一个跨链路由。请确保机器的8250, 25500端口没有被占用。

```
bash ./WeCross/build_wecross.sh -n payment -o routers-payment -l 127.0.0.
↪1:8250:25500
```

注解:

- -n 指定跨链分区标识符(zone id)，跨链分区通过zone id进行区分，可以理解为业务名称。
- -o 指定输出的目录，并在该目录下生成一个跨链路由。
- -l 指定此WeCross跨链路由的ip地址，rpc端口，p2p端口。

命令执行成功，生成routers-payment/目录，目录中包含一个跨链路由127.0.0.1-8250-25500

```
[INFO] All completed. WeCross routers are generated in: routers-payment/
```

生成的跨链路由127.0.0.1-8250-25500目录内容如下

```
# 已屏蔽lib目录，该目录存放所有依赖的jar包
tree routers-payment/127.0.0.1-8250-25500/ -I "lib"
routers-payment/127.0.0.1-8250-25500/
├── apps
│   └── WeCross.jar          # WeCross路由jar包
├── build_wecross.sh
├── conf                     # 配置文件目录
│   ├── application.properties
│   ├── log4j2.xml
│   ├── p2p                  # p2p证书目录
│   │   ├── ca.crt           # 根证书
│   │   ├── node.crt         # 跨链路由证书
│   │   ├── node.key         # 跨链路由私钥
│   │   └── node.nodeid      # 跨链路由nodeid
│   ├── stubs                # stub配置目录，要接入不同的链，在此目录下进行配置
│   └── wecross.toml         # 根配置
├── start.sh                 # 启动脚本
└── stop.sh                  # 停止脚本
```

启动跨链路由

- 启动服务

```
cd ~/wecross/routers-payment/127.0.0.1-8250-25500/
bash start.sh
#停止: bash stop.sh
```

成功

```
WeCross booting up .........
WeCross start successfully
```

如果启动失败，检查8250，25500端口是否被占用

```
netstat -napl | grep 8250
netstat -napl | grep 25500
```

查看失败日志

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

- 检查服务

调用服务的test接口，检查服务是否启动

```
curl http://127.0.0.1:8250/test && echo

# 如果输出如下内容，说明跨链路由服务已完全启动
OK!
```

3.2.2 部署WeCross控制台

WeCross提供了控制台，方便用户进行跨链开发和调试。可通过脚本build_console.sh搭建一个WeCross控制台。

- 下载WeCross控制台

执行如下命令进行下载（提供三种下载方式，可根据网络环境选择合适的方式进行下载），下载后在执行命令的目录下生成WeCross-Console目录。

```
cd ~/wecross/
bash <(curl -sL https://github.com/WeBankFinTech/WeCross-Console/releases/download/
↪resources/download_console.sh)
```

- 配置控制台

```
cd ./WeCross-Console/
cp conf/console-sample.xml conf/console.xml   # 配置控制台连接的跨链路由地址，此处采用默认配
置
```

重要:

- 若搭建WeCross的IP和端口未使用默认配置，需自行更改WeCross-Console/conf/console.xml，详见控制台配置。

- 启动控制台

```
bash start.sh
```

启动成功则输出如下信息，通过help可查看控制台帮助

```
================================================================================
Welcome to WeCross console(v1.0.0-rc1)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
================================================================================
```

- 测试功能

```
# 查看此控制台已连接上的跨链路由
[server1]> currentServer
[server1, 127.0.0.1:8250]

# 退出控制台
[server1]> q
```

更多控制台命令及含义详见控制台命令。

3.3 接入区块链

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**接入FISCO BCOS**为例介绍如何体验WeCross+区块链。

3.3.1 搭链FISCO BCOS链

若已有搭建好的FISCO BCOS链，请忽略本小节。

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

详细步骤如下：

- 脚本建链

```
# 创建操作目录
cd ~ && mkdir -p fisco && cd fisco

# 下载build_chain.sh脚本
curl -LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.2.0/build_
↪chain.sh && chmod u+x build_chain.sh
```

(continues on next page)

(续上页)

```
# 搭建单群组4节点联盟链
# 在fisco目录下执行下面的指令，生成一条单群组4节点的FISCO链。请确保机器的30300~30303，20200~
↪20203，8545~8548端口没有被占用。
# 命令执行成功会输出All completed。如果执行出错，请检查nodes/build.log文件中的错误信息。
bash build_chain.sh -l "127.0.0.1:4" -p 30300,20200,8545
```

- 启动所有节点

```
bash nodes/127.0.0.1/start_all.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30300~30303，20200~20203，8545~8548端口是否被占用。

```
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
```

3.3.2 部署HelloWeCross合约

通过FISCO BCOS控制台部署HelloWeCross合约，控制台的安装和使用详见官方文档配置及使用控制台

HelloWeCross.sol位 于~/wecross/routers-payment/127.0.0.1-8250-25500/conf/stubs-sample/bcos/

控制台安装配置完后启动并部署HelloWeCross.sol，返回的合约地址在之后的WeCross配置中需要用到。详细步骤如下：

- 安装控制台

```
# 获取控制台并回到fisco目录
cd ~/fisco && bash <(curl -s https://raw.githubusercontent.com/FISCO-BCOS/console/
↪master/tools/download_console.sh)

# 拷贝控制台配置文件
# 若节点未采用默认端口，请将文件中的20200替换成节点对应的channle端口。
cp -n console/conf/applicationContext-sample.xml console/conf/applicationContext.
↪xml

# 配置控制台证书
cp nodes/127.0.0.1/sdk/* console/conf/
```

- 拷贝合约文件

将HelloWeCross合约拷贝至控制台目录（用控制台部署）

```
cp ~/wecross/routers-payment/127.0.0.1-8250-25500/conf/stubs-sample/bcos/
↪HelloWeCross.sol console/contracts/solidity/
```

- 启动控制台

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

输出下述信息表明启动成功 否则请检查conf/applicationContext.xml中节点端口配置是否正确

```
================================================================================================
Welcome to FISCO BCOS console(1.0.3)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
 ________  ______   ______    ______    ______         _______    ______    ______
↪   ______
|        \|      \ /      \  /      \  /      \       |       \  /      \  /      \
↪\  /      \
| $$$$$$$$ \$$$$$$|  $$$$$$\|  $$$$$$\|  $$$$$$\      | $$$$$$$\|  $$$$$$\|  $$$$$
↪$$\|  $$$$$$\
| $$__       | $$  | $$___\$$| $$   \$$| $$  | $$      | $$__/ $$| $$   \$$| $$  | $
↪$| $$___\$$
| $$  \      | $$   \$$     \ | $$       | $$  | $$      | $$    $$| $$       | $$  | $
↪$ \$$     \
| $$$$$      | $$   _\$$$$$$$\| $$   __ | $$  | $$      | $$$$$$$\| $$   __ | $$  | $
↪$ _\$$$$$$$\
| $$_____   _| $$_ |  \__| $$| $$__/  \| $$__/ $$      | $$__/ $$| $$__/  \| $$__/ $
↪$|  \__| $$
| $$     \ |   $$ \ \$$    $$ \$$    $$ \$$    $$      | $$    $$ \$$    $$ \$$    $
↪$ \$$    $$
 \$$$$$$$$  \$$$$$$$  \$$$$$$$  \$$$$$$$  \$$$$$$$        \$$$$$$$  \$$$$$$$  \$$$$$
↪$  \$$$$$$$

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

- 部署合约

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

将HelloWeCross的合约地址记录下来，后续步骤中使用：`contract address: 0x04ae9de7bc7397379fad6220ae01529006022d1b`

3.3.3 配置FISCO BCOS stub

完成了FISCO BCOS的搭建以及合约的部署，要完成WeCross和FISCO BCOS的交互，需要配置FISCO BCOS stub，即配置连接信息以及链上的资源。

- 生成stub配置文件

切换至跨链路由的目录，用 create_bcos_stub_config.sh 脚本在`conf`目录下生成FISCO BCOS stub的配置文件框架。

```
cd ~/wecross/routers-payment/127.0.0.1-8250-25500
bash create_bcos_stub_config.sh -n bcos
```

注解:

- -n指定stub的名字（即此链在跨链分区中的名字），默认在跨链路由的conf/stubs目录下生成相关的配置框架。

命令执行成功会输出`[INFO] Create conf/stubs/bcos/stub.toml successfully`；如果执行出错，请查看屏幕打印提示。

生成的目录结构如下：

```
tree conf/stubs/
conf/stubs/
└── bcos
```

(continues on next page)

(续上页)

```
    ├── 0x5c9505d9e2c5c5c21a33187475baf5f512e02cc1.pem          # FISCO BCOS的账户
文件
    └── stub.
↪toml
↪# 链配置文件
```

在conf/stubs/bcos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

将FISCO BCOS节点的证书目录127.0.0.1/sdk下的ca.crt, sdk.key, sdk.crt文件拷贝到conf/stubs/bcos目录下。

```
cp ~/fisco/nodes/127.0.0.1/sdk/* conf/stubs/bcos/
```

- 配置群组

```
vi conf/stubs/bcos/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，则配置如下：

```
[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','127.0.0.1:20201','127.0.0.1:20202','127.0.
↪0.1:20203']
```

- 配置合约资源

在stub.toml文件中配置HelloWeCross合约资源信息，让此跨链路由能够访问此合约。**并将配置中多余无用的举例删除**。

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

```
[[resources]]
    # name cannot be repeated
    name = 'HelloWeCross'
    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。参考此处获取更详尽的配置说明

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

[smCrypto]
    # boolean
    enable = false

[account]
    accountFile = 'classpath:/stubs/bcos/
↪0x0ee5b8ee4af461cac320853aebb7a68d3d4858b4.pem'
    password = ''  # if you choose .p12, then password is required
```

(continues on next page)

(续上页)

```
[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','127.0.0.1:20201','127.0.0.1:20202','127.0.
↪0.1:20203']

# resources is a list
[[resources]]
    # name must be unique
    name = 'HelloWeCross'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x04ae9de7bc7397379fad6220ae01529006022d1b'
```

- 重新启动跨链路由 启动跨链路由加载配置好的跨链资源。

```
# 若WeCross跨链路由未停止，需要先停止
bash stop.sh

# 重新启动
bash start.sh
```

3.3.4 控制台访问跨链资源

- 启动控制台

```
cd ~/wecross/WeCross-Console
bash start.sh
```

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等命令实现合约资源的调用。

具体的命令列表与含义详见控制台命令。

- 查看已有资源

可查看到，跨链分区中有一个跨链资源，其IPath是payment.bcos.HelloWeCross

```
[server1]> listResources
Resources{
    errorCode=0,
    errorMessage='',
    resourceList=[
        WeCrossResource{
            checksum=
↪'0xee1892309d5367c1c3bf5e9d76f312b4134b3209d88048302950bf0dc9395a85',
            type='BCOS_CONTRACT',
            distance=0,
            path='payment.bcos.HelloWeCross'
        }
    ]
}
```

- 调用 HelloWeCross.sol 合约

用IPath调用部署到链上的HelloWeCross合约

```
# payment.bcos.HelloWeCross为跨链资源标识IPath
# String为返回值类型，多个返回值类型用逗号隔开，不能有空格
# getMessage为合约中的方法名
[server1]> call payment.bcos.HelloWeCross String getMessage
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        Hello WeCross
    ]
}

# 方法名后面是参数列表，字符串需要有单引号或者双引号
[server1]> sendTransaction payment.bcos.HelloWeCross Int,String setNumAndMsg 123
↪"Hello World"
Receipt{
    errorCode=0,
    errorMessage='null',
    hash='0x1905f928b980209288280c071ff3574bacc23bbc49538f24325ac07db20133b5',
    result=[
        123,
        Hello World
    ]
}

[server1]> call payment.bcos.HelloWeCross Int,IntArray,String,StringArray getAll
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        123,
        [
            1,
            2,
            3,
            4,
            5
        ],
        Hello World,
        [
            Talk is cheap,
            Show me the code
        ]
    ]
}

# 退出控制台
[server1]> q
```

停止跨链路由

```
cd ~/wecross/routers-payment/127.0.0.1-8250-25500
bash stop.sh
```

3.4 跨链组网

如果已完成接入区块链的体验，接下来将教您如何搭建一个WeCross跨链组网，实现不同的跨链路由之间的相互调用。请求发送至任意的跨链路由，都能够访问到正确的跨链资源。

重要:

- WeCross跨链路由之间采用TLS协议实现安全通讯，只有持有相同ca证书的跨链路由才能建立连接。

3.4.1 构建多个WeCross跨链路由

```
cd ~/wecross
vi ipfile
```

本举例中将构造两个跨链路由，构建一个ipfile文件，将需要构建的两个跨链路由信息内容保存到文件中（按行区分：ip地址:rpc端口:p2p端口）：

```
127.0.0.1:8251:25501
127.0.0.1:8252:25502
```

生成跨链路由

```
# -f 表示以文件为输入
bash ./WeCross/build_wecross.sh -n bill -o routers-bill -f ipfile

# 成功输出如下信息
[INFO] Create routers/127.0.0.1-8251-25501 successfully
[INFO] Create routers/127.0.0.1-8252-25502 successfully
[INFO] All completed. WeCross routers are generated in: routers-bill/
```

在routers-bill目录下生成了两个跨链路由

```
tree routers-bill/ -L 1
routers-bill/
├── 127.0.0.1-8251-25501
├── 127.0.0.1-8252-25502
└── cert
```

3.4.2 跨链路由接入区块链

两个跨链路由配置同一条FISCO BCOS链的不同合约资源。

配置127.0.0.1-8251-25501

让此跨链路由操作HelloWorld合约。

- 部署合约

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

# 部署HelloWorld合约，请将合约地址记录下来后续步骤使用
[group:1]> deploy HelloWorld
contract address: 0xe51eb006c96345f8f0d431f100f0bf619f6145d4

# 部署HelloWeCross合约，请将合约地址记录下来后续步骤使用
[group:1]> deploy HelloWeCross
contract address: 0x5854394d40e60b203e3807d6218b36d4bc0f3437

# 退出控制台
[group:1]> q
```

- 配置stub.toml

```
cd ~/wecross/routers-bill/127.0.0.1-8251-25501
bash create_bcos_stub_config.sh -n bcos1
vi conf/stubs/bcos1/stub.toml
# 配置通过哪些节点接入此链
connectionsStr = ['127.0.0.1:20200','127.0.0.1:20201','127.0.0.1:20202','127.0.0.
↪1:20203']

# 配置资源: HelloWorld合约（同时删除其他资源示例）
[[resources]]
    # name must be unique
    name = 'HelloWorld'
    type = 'BCOS_CONTRACT'
    contractAddress = '0xe51eb006c96345f8f0d431f100f0bf619f6145d4'
```

- 拷贝证书

```
cp ~/fisco/nodes/127.0.0.1/sdk/* ~/wecross/routers-bill/127.0.0.1-8251-25501/conf/
↪stubs/bcos1
```

配置127.0.0.1-8252-25502

让此跨链路由操作HelloWeCross合约

- 配置stub.toml

```
cd ~/wecross/routers-bill/127.0.0.1-8252-25502
bash create_bcos_stub_config.sh -n bcos2
vi conf/stubs/bcos2/stub.toml
# 配置通过哪些节点接入此链
connectionsStr = ['127.0.0.1:20200','127.0.0.1:20201','127.0.0.1:20202','127.0.0.
↪1:20203']

# 配置资源: HelloWeCross合约（同时删除其他资源示例）
[[resources]]
    # name must be unique
    name = 'HelloWeCross'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x5854394d40e60b203e3807d6218b36d4bc0f3437'
```

- 拷贝证书

```
cp ~/fisco/nodes/127.0.0.1/sdk/* ~/wecross/routers-bill/127.0.0.1-8252-25502/conf/
↪stubs/bcos2
```

启动跨链路由

```
cd ~/wecross/routers-bill/127.0.0.1-8251-25501
bash start.sh

cd ~/wecross/routers-bill/127.0.0.1-8252-25502
bash start.sh
```

3.4.3 调用跨链资源

用控制台调用跨链资源，请求发到任意的跨链路由上，都可访问到跨链分区中的任意资源。

配置跨链控制台

在控制台配置文件中增加新的跨链路由的信息。

```
cd ~/wecross/WeCross-Console
vi conf/console.xml
# 配置map信息
<map>
    <entry key="server1" value="127.0.0.1:8251"/>
    <entry key="server2" value="127.0.0.1:8252"/>
</map>
```

启动跨链控制台

```
cd ~/wecross/WeCross-Console
bash start.sh
```

测试请求路由功能

跨链路由能够进行请求转发，将调用请求正确地路由至相应的跨链资源。无论请求发至哪个跨链路由，都可正确路由至相应跨链路由配置的链上资源。

```
# 查看配置的所有跨链路由
[server1]> listServers
{server1=127.0.0.1:8251, server2=127.0.0.1:8252}

# 查看server1的资源列表，可以看到server2所连接的跨链路由的资源（bill.bcos2.HelloWeCross）也在列表中
[server1]> listResources
Resources{
    errorCode=0,
    errorMessage='',
    resourceList=[
        WeCrossResource{
            checksum=
↪'0x7027331008fe69a87ec703a006461a14a652f5380071c8868cc08d3c7247d608',
            type='REMOTE_RESOURCE',
            distance=1,
            path='bill.bcos2.HelloWeCross'
        },
        WeCrossResource{
            checksum=
↪'0x3d58f2c92c9894abbcff5192c78f9ea823c39b089c44f976bb53e63d9285b0b0',
            type='BCOS_CONTRACT',
            distance=0,
            path='bill.bcos1.HelloWorld'
        }
    ]
}

# 在server1中调用server2连接的资源
[server1]> call bill.bcos2.HelloWeCross Int getNumber
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        2019
    ]
```

(continues on next page)

(续上页)

```
}

# 切换server2
[server1]> switch server2

# server2调用server1的资源
[server2]> call bill.bcos1.HelloWorld String get
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        Hello,
        World!
    ]
}

# 退出控制台
[server2]> q
```

停止跨链路由

```
cd ~/wecross/routers-bill/127.0.0.1-8251-25501
bash stop.sh
cd ~/wecross/routers-bill/127.0.0.1-8252-25502
bash stop.sh
```

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。

注解:

- 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的配置分为跨链服务配置和链配置两级。

- 跨链服务配置：P2P、RPC等和WeCross服务相关的必要信息。
- 链配置：与区块链建连信息和链上资源信息。

如果链配置缺省，WeCross仍能启动成功，只是不能提供任何跨链服务。

WeCross跨链服务配置文件名为`wecross.toml`，链配置文件名为`stub.toml`，配置的目录结构如下：

```
# 这是conf目录下标准的配置结构，表示配置了两条链，分别叫bcos和fabric
.
├── log4j2.xml    // 日志配置文件，无需更改
├── stubs
│   ├── bcos
│   │   └── stub.toml
│   └── fabric
│       └── stub.toml
└── wecross.toml
```

4.1.2 跨链服务配置

配置示例文件`wecross-sample.toml`位于`conf`目录，使用前需拷贝成指定文件名`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
```

跨链服务配置有五个配置项，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；跨链分区标识符；通常一种跨链业务/应用为一个跨链分区
  - visible：布尔；可见性；标明当前跨链分区下的资源是否对其他跨链分区可见
- [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] RPC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create_cert.sh脚本生成。
3. 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 链配置

链配置即每个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Stubs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和Fabric。

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'
    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'
    type = 'FABRIC'

# fabricServices is a list
[fabricServices]
    channelName = 'mychannel'
    orgName = 'Org1'
```

(continues on next page)

(续上页)

```
    mspId = 'Org1MSP'
    orgUserName = 'Admin'
    orgUserKeyFile = 'classpath:/stub/fabric/orgUserKeyFile'
    orgUserCertFile = 'classpath:/stub/fabric/orgUserCertFile'
    ordererTlsCaFile = 'classpath:/stub/fabric/ordererTlsCaFile'
    ordererAddress = 'grpcs://127.0.0.1:7050'

[peers]
    [peers.org1]
        peerTlsCaFile = 'classpath:/stub/fabric/peerOrg1CertFile'
        peerAddress = 'grpcs://127.0.0.1:7051'
    [peers.org2]
         peerTlsCaFile = 'classpath:/stub/fabric/peerOrg2CertFile'
         peerAddress = 'grpcs://127.0.0.1:9051'

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWeCross'
    type = 'FABRIC_CONTRACT'
    chainCodeName = 'mycc'
    chainLanguage = "go"
    peers=['org1','org2']
[[resources]]
    name = 'HelloWorld'
    type = 'FABRIC_CONTRACT'
    chainCodeName = 'mygg'
    chainLanguage = "go"
    peers=['org1','org2']
```

配置方法详见Fabric Stub配置

4.2 控制台

控制台是WeCross重要的交互式客户端工具，它通过WeCross-Java-SDK与WeCross 跨链代理建立连接，实现对跨链资源的读写访问请求。控制台拥有丰富的命令，包括获取跨链资源列表，查询资源状态，以及所有的JSON-RPC接口命令。

4.2.1 控制台命令

控制台命令可分为两类，普通命令和交互式命令。

普通命令

普通命令由两部分组成，即指令和指令相关的参数:

- **指令**: 指令是执行的操作命令，包括获取跨链资源列表，查询资源状态指令等，其中部分指令调用JSON-RPC接口，因此与JSON-RPC接口同名。 **使用提示： 指令可以使用tab键补全，并且支持按上下键显示历史输入指令。**
- **指令相关的参数**: 指令调用接口需要的参数，指令与参数以及参数与参数之间均用空格分隔。与JSON-RPC接口同名命令的输入参数和获取信息字段的详细解释参考JSON-RPC API。

交互式命令

WeCross控制台为了方便用户使用，还提供了交互式的使用方式，比如将跨链资源标识赋值给变量，初始化一个类，并用`.command`的方式访问方法。 详见：交互式命令

4.2.2 常用命令链接

普通命令

- call(不发交易): *call*
- sendTransaction(发交易)): *sendTransaction*

重要:

- 其中 call 和 sendTransaction 需要传入返回值类型，多个返回值类型使用逗号分隔，不能有空格。
- 目前支持的类型包括：Int(整型)，IntArray(整型数组)，String(字符串)，StringArray(字符串数组)。
- 如果返回值为空，控制台需要传入关键字：Void。
- 参数列表传入字面量，因此字符串需要用单引号或双引号括起来。

交互式命令

- 初始化资源实例: *WeCross.getResource*
- 访问资源UBI接口: *[resource].[command]*

4.2.3 快捷键

- `Ctrl+A`：光标移动到行首
- `Ctrl+E`：光标移动到行尾
- `Ctrl+R`：搜索输入的历史命令
- ↑： 向前浏览历史命令
- ↓： 向后浏览历史命令
- `table`： 自动补全，支持命令、变量名、资源名以及其它固定参数的补全

4.2.4 控制台响应

当发起一个控制台命令时，控制台会获取命令执行的结果，并且在终端展示执行结果，执行结果分为2类：

- **正确结果:** 命令返回正确的执行结果，以字符串或是json的形式返回。
- **错误结果:** 命令返回错误的执行结果，以字符串或是json的形式返回。
- **状态码:** 控制台的命令调用JSON-RPC接口时，状态码参考这里。

4.2.5 控制台配置与运行

重要: 前置条件：部署WeCross请参考 快速部署。

获取控制台

可通过脚本download_console.sh获取控制台。

```
cd ~ && mkdir -p wecross && cd wecross
# 获取控制台
bash <(curl -sL https://github.com/WeBankFinTech/WeCross-Console/releases/download/
↪resources/download_console.sh)
```

执行成功后，会生成WeCross-Console目录，结构如下：

```
├── apps
│   └── wecross-console.jar   # 控制台jar包
├── conf
│   ├── console-sample.xml    # 配置示例文件
│   └── log4j2.xml            # 日志配置文件
├── download_console.sh       # 获取控制台脚本
├── lib                       # 相关依赖的jar包目录
├── logs                      # 日志文件
└── start.sh                  # 启动脚本
```

配置控制台

配置前需要将console-sample.xml拷贝成console.xml，再配置console.xml文件。

控制台唯一需要配置的是所连接的WeCross跨链代理的服务地址，包括IP和端口号。

```
<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="weCrossServers" class="com.webank.wecross.console.common.
↪WeCrossServers">
        <property name="servers">
            <map>
                <!-- 配置需要连接的WeCross跨链代理服务列表 -->
                <entry key="server1" value="127.0.0.1:8250"/>
                <entry key="server2" value="127.0.0.1:8251"/>
            </map>
        </property>
        <!-- 启动控制台默认连接的WeCross跨链代理 -->
        <property name="defaultServer" value="server1"/>
    </bean>

</beans>
```

注：配置中的key只能是字母和数字的组合，不能出现其他字符。

启动控制台

在WeCross服务已经开启的情况下，启动控制台：

```
cd ~/wecross/WeCross-Console
bash start.sh
# 输出下述信息表明启动成功
================================================================================================
Welcome to WeCross console(1.0.0-rc1)!
```

(continues on next page)

(续上页)

```
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.

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

4.2.6 普通命令

以下所有跨链资源相关命令的执行结果以实际配置为准，此处只是示例。

help

输入help或者h，查看控制台所有的命令。

```
[server1]> help
--------------------------------------------------------------------------------
↪----------
quit                                Quit console.
currentServer                       Show currently connected WeCross server.
listServers                         List all configured WeCross servers.
switch                              Switch to a specific WeCross server.
listLocalResources                  List local resources configured by WeCross␣
↪server.
listResources                       List all resources including remote resources.
status                              Check if the resource exists.
getData                             Get data from contract.
setData                             Set data for contract.
call                                Call constant method of smart contract.
callInt                             Call constant method of smart contract with int␣
↪returned.
callIntArray                        Call constant method of smart contract with int␣
↪array returned.
callString                          Call constant method of smart contract with␣
↪string returned.
callStringArray                     Call constant method of smart contract with␣
↪string array returned.
sendTransaction                     Call non-constant method of smart contract.
sendTransactionInt                  Call non-constant method of smart contract with␣
↪int returned.
sendTransactionIntArray             Call non-constant method of smart contract with␣
↪int array returned.
sendTransactionString               Call non-constant method of smart contract with␣
↪string returned.
sendTransactionStringArray          Call non-constant method of smart contract with␣
↪string array returned.
WeCross.getResource                 Init resource by path, and assign it to a␣
↪custom variable.
[resource].[command]                Equal to command: command [path].

--------------------------------------------------------------------------------
↪----------
```

注：

- help显示每条命令的含义是：命令 命令功能描述
- 查看具体命令的使用介绍说明，输入命令 -h或–help查看。例如：

```
[server1]> currentServer -h
--------------------------------------------------------------------------------
↪----------
```

(continues on next page)

(续上页)

```
Show currently connected WeCross server.
Usage: currentServer
--------------------------------------------------------------------------------
↪----------
```

currentServer

显示当前连接的WeCross跨链代理。

```
[server1]> currentServer
[server1, 127.0.0.1:8250]
```

listServers

显示所有已配置的WeCross跨链代理。

```
[server1]> listServers
{server1=127.0.0.1:8250, server2=127.0.0.1:8251}
```

switch

根据key切换连接的WeCross跨链代理。

```
[server1]> switch server2
[server2]>
```

注：需要切换的WeCross跨链代理，请确保已在dist/conf目录下的console.xml进行了配置，并且节点ip和端口正确，服务正常运行。

listLocalResources

查看WeCross跨链代理本地配置的跨链资源。

```
[server1]> listLocalResources
Resources{
    errorCode=0,
    errorMessage='',
    resourceList=[
        WeCrossResource{
            checksum=
↪'0x320a7e701e655cb67a3d1a5283d6083cbe228b080503d4b791fa060bf7f7d926',
            type='BCOS_CONTRACT',
            distance=0,
            path='payment.bcos.HelloWeCross'
        },
    ]
}
```

listResources

查看WeCross跨链代理本地配置的跨链资源和所有的远程资源。

```
[server1]> listResources
Resources{
    errorCode=0,
    errorMessage='',
    resourceList=[
        WeCrossResource{
            checksum=
↪'0xdcb8e609e025c8e091d18fe18b8d66d34836bd3051a08ce615118d27e4a29ebe',
            type='REMOTE_RESOURCE',
            distance=1,
            path='payment.bcos.HelloWorld'
        },
        WeCrossResource{
            checksum=
↪'0xdcb8e609e025c8e091d18fe18b8d66d34836bd3051a08ce615118d27e4a29ebe',
            type='REMOTE_RESOURCE',
            distance=1,
            path='payment.bcos1.HelloWorld'
        }
    ]
}
```

status

查看跨链资源的状态，即是否存在于连接的WeCross跨链代理中。

参数:

- IPath：跨链资源标识。

```
[server1]> status payment.bcos.HelloWeCross
Result ==> exists
```

getData

预留接口，根据key查询value，目前没有链支持。

参数:

- IPath：跨链资源标识。
- key：字符串。

```
[server1]> getData payment.bcos.HelloWeCross "number"
StatusAndValue{
    errorCode=101,
    errorMessage='Not supported by BCOS_CONTRACT',
    value='null'
}
```

setData

预留接口，根据key更新value，目前没有链支持。

参数:

- IPath：跨链资源标识。
- key：字符串。
- value：字符串。

```
[server1]> setData payment.bcos.HelloWeCross "message" "haha"
StatusAndValue{
    errorCode=101,
    errorMessage='Not supported by BCOS_CONTRACT',
    value='null'
}
```

call

调用智能合约的方法，不涉及状态的更改，不发交易。

参数:

- IPath：跨链资源标识。
- retTypes：返回值类型列表。
- method：合约方法名。
- args：参数列表。

```
[server1]> call payment.bcos.HelloWeCross Int,String getNumAndMsg
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        2019,
        Hello WeCross
    ]
}
```

callInt

`call`的衍生命令，返回值为整型。

参数:

- IPath：跨链资源标识。
- method：合约方法名。
- args：参数列表。

```
[server1]> callInt payment.bcos.HelloWeCross getNumber
CallResult{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=2019
}
```

callIntArray

`call`的衍生命令，返回值为整型数组。

参数:

- IPath：跨链资源标识。
- method：合约方法名。

- args：参数列表。

```
[server1]> callIntArray payment.bcos.HelloWeCross getNumbers
CallResult{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        1,
        2,
        3
    ]
}
```

callString

call的衍生命令，返回值为字符串。

参数：

- IPath：跨链资源标识。
- method：合约方法名。
- args：参数列表。

```
[server1]> call payment.bcos.HelloWeCross String getMessage
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        Hello WeCross
    ]
}
```

callStringArray

call的衍生命令，返回值为字符串数组。

参数：

- IPath：跨链资源标识。
- method：合约方法名。
- args：参数列表。

```
[server1]> callStringArray payment.bcos.HelloWeCross getMessages
CallResult{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        Bei Bei,
        Jing Jing,
        Huan Huan,
        Ying Ying
    ]
}
```

sendTransaction

调用智能合约的方法，会更改链上状态，需要发交易。

参数：

- IPath：跨链资源标识。
- retTypes：返回值类型列表。
- method：合约方法名。
- args：参数列表。

```
[server1]> sendTransaction payment.bcos.HelloWeCross Int,String setNumAndMsg 2020
↪"Hello World"
Receipt{
    errorCode=0,
    errorMessage='null',
    hash='0xa1c4ff31e21e97bbaf06ee228f7e84753f3da51ac6b33ffd326100d9a2a40307',
    result=[
        2020,
        Hello World
    ]
}
```

sendTransactionInt

`sendTransactionInt`的衍生命令，返回值为整型。

参数：

- IPath：跨链资源标识。
- method：合约方法名。
- args：参数列表。

```
[server1]> sendTransactionInt payment.bcos.HelloWeCross setNumber 2019
CallResult{
    errorCode=0,
    errorMessage='null',
    hash='0x9847777b33130b0c4f0cbbc8491890bad437e0e5f6e87366b1f8dcd2b0761535',
    result=2019
}
```

sendTransactionIntArray

`sendTransactionIntArray`的衍生命令，返回值为整型数组。

参数：

- IPath：跨链资源标识。
- method：合约方法名。
- args：参数列表。

```
[server1]> sendTransactionIntArray payment.bcos.HelloWeCross getNumbers
CallResult{
    errorCode=0,
    errorMessage='null',
    hash='0x7ee8b8b9c553c0be0b8c3d649eb0c0525604bb3ab6de4d50b002edc891faec2a',
    result=[
```

(continues on next page)

(续上页)

```
        1,
        2,
        3
    ]
}
```

sendTransactionString

sendTransactionString的衍生命令，返回值为字符串。

参数：

- IPath：跨链资源标识。
- method：合约方法名。
- args：参数列表。

```
[server1]> sendTransactionString payment.bcos.HelloWeCross setMessage "Xi Xi"
CallResult{
    errorCode=0,
    errorMessage='null',
    hash='0x7ff89e4c4ec4d5af600e4548a49452d2bb2a21ae881a346b1623375ece12cbb6',
    result=Xi Xi
}
```

sendTransactionStringArray

sendTransactionStringArray的衍生命令，返回值为字符串数组。

参数：

- IPath：跨链资源标识。
- method：合约方法名。
- args：参数列表。

```
[server1]> sendTransactionStringArray payment.bcos.HelloWeCross getMessages
CallResult{
    errorCode=0,
    errorMessage='null',
    hash='0x22416a793263bc52bde28f1eb8850678824d74bf09a3322a923ca0351b271304',
    result=[
        Bei Bei,
        Jing Jing,
        Huan Huan,
        Ying Ying
    ]
}
```

4.2.7 交互式命令

WeCross.getResource

WeCross控制台提供了一个资源类，通过方法getResource来初始化一个跨链资源实例，并且赋值给一个变量。这样调用同一个跨链资源的不同UBI接口时，不再需要每次都输入跨链资源标识。

```
# myResource 是自定义的变量名
[server1]> myResource = WeCross.getResource payment.bcos.HelloWeCross
Result ==> {"path":"payment.bcos.HelloWeCross","weCrossRPC":{"weCrossService":{
↪"server":"127.0.0.1:8250"}}}

# 还可以将跨链资源标识赋值给变量，通过变量名来初始化一个跨链资源实例
[server1]> path = payment.bcos.HelloWorldContract
Result ==> "payment.bcos.HelloWorldContract"

[server1]> myResource = WeCross.getResource path
Result ==> {"path":"payment.bcos.HelloWorldContract","weCrossRPC":{"weCrossService
↪":{"server":"127.0.0.1:8250"}}}
```

[resource].[command]

当初始化一个跨链资源实例后，就可以通过.command的方式，调用跨链资源的UBI接口。

```
# 输入变量名，通过table键可以看到能够访问的所有命令
[server1]> myResource.
myResource.call                          myResource.callStringArray
myResource.status                        myResource.sendTransaction
myResource.callInt                       myResource.sendTransactionInt
myResource.getData                       myResource.sendTransactionString
myResource.setData                       myResource.sendTransactionIntArray
myResource.callString                    myResource.sendTransactionStringArray
myResource.callIntArray

# 查看状态
[server1]> myResource.status
payment.bcoschain.HelloWorldContract : exists

# getData
[server1]> myResource.getData "name"
StatusAndValue{
    errorCode=101,
    errorMessage='Not supported by BCOS_CONTRACT',
    value='null'
}

# setData
[server1]> myResource.setData "name" "dou dou"
Status{
    errorCode=101,
    errorMessage='Not supported by BCOS_CONTRACT'
}

# call
[server1]> myResource.call Int,IntArray,String,StringArray getAll
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        2019,
        [
            1,
            2,
            3
        ],
        Xi Xi,
```

(continues on next page)

(续上页)

```
        [
            Bei Bei,
            Jing Jing,
            Huan Huan,
            Ying Ying
        ]
    ]
}

# sendTransaction
[server1]> myResource.sendTransaction Int,String setNumAndMsg 100 "Ha Ha"
Receipt{
    errorCode=0,
    errorMessage='null',
    hash='0x8a9dac589c269c837262c30b5a81dcccbdd7d823cb20621a5ce7f1e9174e4dac',
    result=[
        100,
        Ha Ha
    ]
}

[server1]> myResource.call Int,String getNumAndMsg
Receipt{
    errorCode=0,
    errorMessage='success',
    hash='null',
    result=[
        100,
        Ha Ha
    ]
}
```

4.3 脚本介绍

为了方便用户使用，WeCross提供了丰富的脚本，脚本位于WeCross跨链路由的根目录下，本章节将对这些脚步做详细介绍。

4.3.1 启动脚本

start.sh

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
```

4.3.2 停止脚本

stop.sh

停止脚本stop.sh用于停止WeCross服务。

```
bash  stop.sh
```

4.3.3 构建WeCross脚本

build_wecross.sh

生成WeCross跨链路由网络

```
Usage:
    -n  <zone id>                   [Required]   set zone ID
    -l  <ip:rpc-port:p2p-port>      [Optional]   "ip:rpc-port:p2p-port" e.g:"127.0.
↪0.1:8250:25500"
    -f  <ip list file>              [Optional]   split by line, every line should␣
↪be "ip:rpc-port:p2p-port". eg "127.0.0.1:8250:25500"
    -c  <ca dir>                    [Optional]   dir of existing ca
    -o  <output dir>                [Optional]   default ./${router_output}/
    -z  <generate tar packet>       [Optional]   default no
    -T  <enable test mode>          [Optional]   default no. Enable test resource.
    -h  call for help
e.g
    bash $0 -n payment -l 127.0.0.1:8250:25500
    bash $0 -n payment -f ipfile
```

- **-n**：指定跨链分区标识
- **-l**：可选，指定生成一个跨链路由，与-f二选一，单行，如：192.168.0.1:8250:25500
- **-f**：可选，指定生成多个跨链路由，与-l二选一，多行，**不可有空行**，例如：

```
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
```

- **-o**：可选，指定跨链路由生成目录，默认wecross/
- **-z**：可选，若设置，则生成跨链路由的压缩包，方便拷贝至其它机器
- **-T**：可选，若设置，生成的跨链路由开启测试资源
- **-h**：可选，打印Usage

4.3.4 BCOS stub配置脚本

脚本create_bcos_stub_config.sh用于快速创建FISCO BCOS stub的配置文件。

可通过-h查看帮助信息：

```
Usage:
    -n  <stub name>         [Required]    specify the name of stub
    -r  <root dir>          [Optional]    specify the stubs root dir, default is␣
↪stubs
    -c  <conf path>         [Optional]    specify the path of conf dir, default is␣
↪conf
    -p  <password>          [Optional]    password for p12 a type of FISCO BCOS␣
↪account, default is null and use pem
    -h  call for help
e.g
    bash create_bcos_stub_config.sh -n bcos
```

(continues on next page)

(续上页)

```
    bash create_bcos_stub_config.sh -n bcos -r stubs
    bash create_bcos_stub_config.sh -n bcos -r stubs -c conf
    bash create_bcos_stub_config.sh -n bcos -r stubs -c conf -p 123456
```

- **-n**：指定区块链跨链标识，即stub的名字；同时也会在**-r**指定的目录下生成与stub相同名字的目录来保存配置文件。
- **-r**：可选，指定跨链路由的stub配置根目录，默认stubs/，需要和根配置文件wecross.toml中的[stubs.path]保存一致。
- **-c**：可选，指定跨链路由的配置根目录，默认为与启动脚本同级的conf/目录，所有的配置文件都保存在该路径下。
- **-p**：可选，表示使用p12格式的账户文件，并指定口令。默认是pem格式，无需口令。

例如：

```
bash create_bcos_stub_config.sh -n bcos -r stubs -c conf -p 123456
```

执行后，在conf/stubs/目录下生成名字为bcos的stub目录，之后按照接入FISCO BCOS的指引进行配置

```
tree conf/
conf/
├── stubs
│   └── bcos
│       ├── 0x7dfbec42690e83004687eae5d0e3738750c7c153.pem
│       ├── 0xaa8f54cce575b4cc92f48b6138d6393b2b7d3e86.p12
│       ├── ca.crt
│       ├── node.crt
│       ├── node.key
│       ├── sdk.crt
│       ├── sdk.key
│       └── stub.toml
```

4.3.5 Fabric stub配置脚本

脚本create_fabric_stub_config.sh用于快速创建Fabric stub的配置文件。

可通过-h查看帮助信息：

```
Usage:
    -n  <stub name>         [Required]    specify the name of stub
    -r  <root dir>          [Optional]    specify the stubs root dir, default is␣
↪stubs
    -c  <conf path>         [Optional]    specify the path of conf dir, default is␣
↪conf
    -h  call for help
e.g
    bash create_fabric_stub_config.sh -n fabric
    bash create_fabric_stub_config.sh -n fabric -r stubs
    bash create_fabric_stub_config.sh -n fabric -r stubs -c conf
```

- **-n**：指定区块链跨链标识，即stub的名字；同时也会生成相同名字的目录来保存配置文件。
- **-r**：指定配置文件根目录，需要和配置文件wecross.toml中的[stubs.path]保存一致。
- **-c**：指定加载配置文件时的classpath路径，所有的配置文件都保存在该路径下，默认为与启动脚本同级的conf目录。
- **-h**：可选，打印Usage

例如：

```
bash create_fabric_stub_config -r stubs -o fabric -d conf
```

执行结果的目录结构如下:

```
.
├── conf
│   └── stubs
│       └── fabric
│           └── stub.toml
```

4.3.6 创建P2P证书脚本

create_cert.sh

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

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

4.4 跨链SDK

WeCross向外部暴露了所有的UBI接口，开发者可以通过SDK实现这些接口的快速调用。

4.4.1 环境要求

重要:

- java版本

  要求 JDK8或以上

- WeCross服务部署

参考 WeCross快速入门

4.4.2 Java应用引入SDK

通过gradle或maven引入SDK到java应用

gradle:

```
compile ('com.webank:wecross-java-sdk:1.0.0-rc1')
```

maven:

```
<dependency>
    <groupId>com.webank</groupId>
    <artifactId>wecross-java-sdk</artifactId>
    <version>1.0.0-rc1</version>
</dependency>
```

4.4.3 使用方法

调用SDK的JSON-RPC API

示例代码如下：

```
    // 使用IP和端口初始化WeCrossService
    WeCrossService weCrossService = new WeCrossRPCService("127.0.0.1:8250");

    // 初始化WeCrossRPC
    WeCrossRPC weCrossRPC = WeCrossRPC.init(weCrossService);

    // 调用RPC接口，send表示同步调用。
    Response response = weCrossRPC.status("payment.bcoschain.HelloWorldContract").
↪send();
```

4.5 JSON-RPC API

下列接口的示例中采用curl命令。curl是一个利用url语法在命令行下运行的数据传输工具，通过curl命令发送http请求，可以访问WeCross的JSON RPC接口。curl命令的url地址需要设置为WeCross跨链代理的RPC监听IP和端口。

可使用jq工具对结果进行格式化显示。RPC状态码参考*RPC*状态码。

4.5.1 API列表

- RemoteCall status(String path);
- RemoteCall list(Boolean ignoreRemote);
- RemoteCall getData(String path, String key);
- RemoteCall setData(String path, String key, String value);
- RemoteCall call(String path, String method, Object... args);
- RemoteCall call(String path, String retTypes[], String method, Object... args);
- RemoteCall callInt(String path, String method, Object... args);

- RemoteCall callIntArray(String path, String method, Object... args);
- RemoteCall callString(String path, String method, Object... args);
- RemoteCall callStringArray(String path, String method, Object... args);
- RemoteCall sendTransaction(String path, String method, Object... args);
- RemoteCall sendTransaction(String path, String retTypes[], String method, Object... args);
- RemoteCall sendTransactionInt(String path, String method, Object... args);
- RemoteCall sendTransactionIntArray(String path, String method, Object... args);
- RemoteCall sendTransactionString(String path, String method, Object... args);
- RemoteCall sendTransactionStringArray(String path, String method, Object... args);

重要:

- 所有UBI接口的RPC API都是针对跨链资源，因此需要将跨链资源标识转换成url再进行调用。
- *call* 和 *sendTransaction* 需要传入返回值类型列表，类型为字符串数组。
- 目前支持的类型包括：Int(整型)，IntArray(整型数组)，String(字符串)，StringArray(字符串数组)。

4.5.2 API解析

status

查看跨链资源状态

参数

- `path`: `String` - 跨链资源标识

返回值

- `Response` - 相应包
  - `version`: `String` - 版本号
  - `result`: `int` - 状态码
  - `message`: `String` - 错误消息
  - `data`: `object` - 返回的数据

java示例

```
    // 初始化RPC
    WeCrossService weCrossService = new WeCrossRPCService("127.0.0.1:8250");
    WeCrossRPC weCrossRPC = WeCrossRPC.init(weCrossService);

    // 调用RPC接口，目前只支持同步调用
    Response response = weCrossRPC.status("payment.bcos.HelloWeCross").send();
```

注 - 之后的java示例，会省去初始化WeCrossRPC的步骤。

curl示例

```
# Request
curl --data '{"version":"1", "path":"payment.bcos.HelloWeCross", "method":"status"}
↪' http://127.0.0.1:8250/payment/bcos/HelloWeCross/status | jq

# Result
{
  "version": "1",
  "result": 0,
  "message": null,
  "data": "exists"
}
```

list

获取资源列表

参数

- ignoreRemote: Boolean - true: 忽略远程资源，即不显示WeCross跨链代理连接的peers的资源。

返回值

- ResourcesResponse - 相应包
  - version: String - 版本号
  - result: int - 状态码
  - message: String - 错误消息
  - data: Resources - 跨链资源列表

java示例

```
    ResourcesResponse response = weCrossRPC.list(true).send();
```

curl示例

```
# Request
curl --data '{"version":"1", "path":"", "method":"list", "data": {"ignoreRemote":␣
↪true}}' -H "Content-Type:application/json"  http://127.0.0.1:8250/list | jq

# Result
{
  "version": "1",
  "result": 0,
  "message": null,
  "data": {
    "errorCode": 0,
    "errorMessage": "",
    "resources": [
```

(continues on next page)

(续上页)

```
      {
        "checksum":
↪"0xa88063c594ede65ee3c4089371a1e28482bd21d05ec1e15821c5ec7366bb0456",
        "type": "BCOS_CONTRACT",
        "distance": 0,
        "path": "payment.bcos.HelloWeCross"
      }
    ]
  }
}
```

getData

根据key获取value，目前只有JDChain支持该接口。

参数

- path: String - 跨链资源标识
- key: String - 数据的键

返回值

- GetDataResponse - 相应包
  - version: String - 版本号
  - result: int - 状态码
  - message: String - 错误消息
  - data: GetDataResponse.StatusAndValue - 状态和数据

java示例

```
    GetDataResponse response = weCrossRPC.getData("payment.bcos.HelloWeCross", "get
↪").send();
```

curl示例

```
# Request
curl --data '{"version":"1", "path":"payment.bcos.HelloWeCross", "method":"getData
↪", "data": {"key": "name"}}' -H "Content-Type:application/json"  http://127.0.0.
↪1:8250/payment/bcos/HelloWeCross/getData | jq

# Result
{
  "version": "1",
  "result": 0,
  "message": null,
  "data": {
    "errorCode": 101,
    "errorMessage": "Not supported by BCOS_CONTRACT",
    "value": null
```

(continues on next page)

(续上页)

```
  }
}
```

setData

根据key更新value，预留接口，目前没有链支持。

参数

- path: String - 跨链资源标识
- key: String - 数据的键
- value: String - 数据的值

返回值

- SetDataResponse - 相应包
  - version: String - 版本号
  - result: int - 状态码
  - message: String - 错误消息
  - data: SetDataResponse.Status - 返回的数据

java示例

```
    GetDataResponse response = weCrossRPC.setData("payment.bcos.HelloWeCross", "set
↪", "Hello World").send();
```

curl示例

```
# Request
curl --data '{"version":"1", "path":"payment.bcos.HelloWeCross", "method":"setData
↪", "data": {"key": "name", "value":"dou dou"}}' -H "Content-Type:application/json
↪"  http://127.0.0.1:8250/payment/bcos/HelloWeCross/setData | jq

# Result
{
  "version": "1",
  "result": 0,
  "message": null,
  "data": {
    "errorCode": 101,
    "errorMessage": "Not supported by BCOS_CONTRACT"
  }
}
```

call

调用智能合约，不更改链状态，不发交易，无返回值。

参数

- path: String - 跨链资源标识
- method: String - 调用的方法
- args : object[] - 可变参数列表

返回值

- TransactionResponse - 相应包
  - version: String - 版本号
  - result: int - 状态码
  - message: String - 错误消息
  - data: CallContractResult - 调用结果

java示例

```
    TransactionResponse transactionResponseVoid =
        weCrossRPC
            .call(
                "payment.bcos.HelloWeCross",
                    "getNumber")
            .send();
```

curl示例

```
# Request
curl --data '{"version":"1", "path":"payment.bcos.HelloWeCross", "method":"call",
↪"data": {"method": "getNumber", "args":[]}}' -H "Content-Type:application/json"
↪http://127.0.0.1:8250/payment/bcos/HelloWeCross/call | jq

# Result
{
  "version": "1",
  "result": 0,
  "message": null,
  "data": {
    "errorCode": 0,
    "errorMessage": "success",
    "hash": null,
    "extraHashes": null,
    "result": [],
    "type": "NORMAL",
    "encryptType": "NORMAL",
    "blockHeader": null,
    "proofs": null
  }
}
```

call

调用智能合约，不更改链状态，不发交易，有返回值。

参数

- path: String - 跨链资源标识
- retTypes: String[] - 返回值类型列表
- method: String - 调用的方法
- args : object[] - 可变参数列表

返回值

- TransactionResponse - 相应包
  - version: String - 版本号
  - result: int - 状态码
  - message: String - 错误消息
  - data: CallContractResult - 调用结果

java示例

```
    TransactionResponse transactionResponseVoid =
        weCrossRPC
            .call(
                "payment.bcos.HelloWeCross",
                new String[] {"String"}
                    "getMessage")
            .send();
```

curl示例

```
# Request
curl --data '{"version":"1", "path":"payment.bcos.HelloWeCross", "method":"call",
↪"data": {"retTypes":["String"], "method": "getMessage", "args":[]}}' -H "Content-
↪Type:application/json"  http://127.0.0.1:8250/payment/bcos/HelloWeCross/call | jq

# Result
{
  "version": "1",
  "result": 0,
  "message": null,
  "data": {
    "errorCode": 0,
    "errorMessage": "success",
    "hash": null,
    "extraHashes": null,
    "result": [
      "Ha Ha"
    ],
    "type": "NORMAL",
    "encryptType": "NORMAL",
    "blockHeader": null,
    "proofs": null
  }
}
```

衍生接口

- callInt: 调用返回值为Int类型的call
- callIntArray: 调用返回值为IntArray类型的call
- callString: 调用返回值为String类型的call
- callStringArray: 调用返回值为StringArray类型的call

sendTransaction

调用智能合约，更改链状态，发交易，无返回值。

参数

- `path`: `String` - 跨链资源标识
- `method`: `String` - 调用的方法
- `args` : `object[]` - 可变参数列表

返回值

- `TransactionResponse` - 相应包
  - `version`: `String` - 版本号
  - `result`: `int` - 状态码
  - `message`: `String` - 错误消息
  - `data`: `CallContractResult` - 调用结果

java示例

```
    TransactionResponse transactionResponseVoid =
        weCrossRPC
            .sendTransaction(
                "payment.bcos.HelloWeCross",
                    "setNumber"
                    123)
            .send();
```

curl示例

```
# Request
curl --data '{"version":"1", "path":"payment.bcos.HelloWeCross", "method":
↪"sendTransaction", "data": {"method": "setNumber", "args":[123]}}' -H "Content-
↪Type:application/json"  http://127.0.0.1:8250/payment/bcos/HelloWeCross/
↪sendTransaction | jq

# Result
{
  "version": "1",
  "result": 0,
  "message": null,
```

(continues on next page)

(续上页)


```
  "data": {
    "errorCode": 0,
    "errorMessage": null,
    "hash": "0x5fe42cf98516c70b0b490ee8957088e341faa1382d4071cd6bbad7b704fd1870",
    "extraHashes": [
      "0x4c17d7a9ef8c244fcc864251143bcbc773632d5356086f08c84e95cb80e0f894"
    ],
    "result": [],
    "type": "NORMAL",
    "encryptType": "NORMAL",
    "blockHeader": {
      "blockNumber": 40,
      "hash": "0x2e592998b752369087c9286048e0638d606ebf20cb9964350fa9926d5c23d30d",
      "roots": [
        "0x13874fc5554d177b8739595b8ea6a97443805697238279e170d7219533fb1542",
        "0xc9a1779a4ceaf2134653f2be3468e6081cfabcd9044d602d9e9229512dc5a1b2",
        "0x767cc8d0cd30b74c42698b224cd65a46db251241aa6c74ab028f72f5f40c0afb"
      ]
    },
    "proofs": [
      {
        "root": "0x767cc8d0cd30b74c42698b224cd65a46db251241aa6c74ab028f72f5f40c0afb
↪",
        "path": [
          {
            "left": [],
            "right": []
          }
        ],
        "leaf": {
          "index": "0x0",
          "leaf":
↪"0x5fe42cf98516c70b0b490ee8957088e341faa1382d4071cd6bbad7b704fd1870",
          "proof":
↪"0x805fe42cf98516c70b0b490ee8957088e341faa1382d4071cd6bbad7b704fd1870"
        }
      },
      {
        "root": "0xc9a1779a4ceaf2134653f2be3468e6081cfabcd9044d602d9e9229512dc5a1b2
↪",
        "path": [
          {
            "left": [],
            "right": []
          }
        ],
        "leaf": {
          "index": "0x0",
          "leaf":
↪"0x4c17d7a9ef8c244fcc864251143bcbc773632d5356086f08c84e95cb80e0f894",
          "proof":
↪"0x804c17d7a9ef8c244fcc864251143bcbc773632d5356086f08c84e95cb80e0f894"
        }
      }
    ]
  }
}
```

sendTransaction

调用智能合约，更改链状态，发交易，有返回值。

参数

- path: String - 跨链资源标识
- retTypes: String[] - 返回值类型列表
- method: String - 调用的方法
- args : object[] - 可变参数列表

返回值

- TransactionResponse - 相应包
  - version: String - 版本号
  - result: int - 状态码
  - message: String - 错误消息
  - data: CallContractResult - 调用结果

java示例

```
    TransactionResponse transactionResponseVoid =
        weCrossRPC
            .sendTransaction(
                "payment.bcos.HelloWeCross",
                new String[] {"Int", "String"},
                "setNumAndMsg",
                234,
                "Hello WeCross")
            .send();
```

curl示例

```
# Request
curl --data '{"version":"1", "path":"payment.bcos.HelloWeCross", "method":
↪"sendTransaction", "data": {"retTypes":["Int", "String"], "method": "setNumAndMsg
↪", "args":[234,"Hello WeCross"]}}' -H "Content-Type:application/json"  http://
↪127.0.0.1:8250/payment/bcos/HelloWeCross/sendTransaction | jq

# Result
{
  "version": "1",
  "result": 0,
  "message": null,
  "data": {
    "errorCode": 0,
    "errorMessage": null,
    "hash": "0xc0a842c9edf35484ad7d3eb1bdb12deb7f20bd356ed9eb3c063ec9fe0de4401e",
    "extraHashes": [
      "0x58a663c23ebac75168d8e9859f0db803c926f0a452cbd537beffe0b3426f4394"
    ],
```

(continues on next page)

(续上页)


```
    "result": [
      234,
      "Hello WeCross"
    ],
    "type": "NORMAL",
    "encryptType": "NORMAL",
    "blockHeader": {
      "blockNumber": 42,
      "hash": "0x43956a4330ee652bbd2234894960f6c78234775a25ca66741cfcc7ddf22f9cfb",
      "roots": [
        "0xfb146b917eacedf386e40396d3b5c526576308008ebbd2f42f0cef89615ac961",
        "0x99cbcd0d1aa8742e23148db18436a120ddfecf0635dc1289129c3e0b260e323f",
        "0x3e58b5157ecd7e5024aaa66c925ca8685dd5b8444a534c0f9387d36051067a00"
      ]
    },
    "proofs": [
      {
        "root": "0x3e58b5157ecd7e5024aaa66c925ca8685dd5b8444a534c0f9387d36051067a00
↪",
        "path": [
          {
            "left": [],
            "right": []
          }
        ],
        "leaf": {
          "index": "0x0",
          "leaf":
↪"0xc0a842c9edf35484ad7d3eb1bdb12deb7f20bd356ed9eb3c063ec9fe0de4401e",
          "proof":
↪"0x80c0a842c9edf35484ad7d3eb1bdb12deb7f20bd356ed9eb3c063ec9fe0de4401e"
        }
      },
      {
        "root": "0xfb146b917eacedf386e40396d3b5c526576308008ebbd2f42f0cef89615ac961
↪",
        "path": [
          {
            "left": [],
            "right": []
          }
        ],
        "leaf": {
          "index": "0x0",
          "leaf":
↪"0x58a663c23ebac75168d8e9859f0db803c926f0a452cbd537beffe0b3426f4394",
          "proof":
↪"0x8058a663c23ebac75168d8e9859f0db803c926f0a452cbd537beffe0b3426f4394"
        }
      }
    ]
  }
}
```


衍生接口

- sendTransactionInt: 调用返回值为Int类型的sendTransaction
- sendTransactionIntArray: 调用返回值为IntArray类型的sendTransaction
- sendTransactionString: 调用返回值为String类型的sendTransaction

- sendTransactionStringArray: 调用返回值为StringArray类型的sendTransaction

4.5.3 RPC状态码

当一个RPC调用遇到错误时，返回的响应对象必须包含error错误结果字段，该字段有下列成员参数：

- result: 使用数值表示该异常的错误类型，必须为整数。
- message: 对该错误的简单描述字符串。

标准状态码及其对应的含义如下：

CHAPTER 5

跨链接入

5.1 接入FISCO BCOS

接入FISCO BCOS需要满足底层版本和兼容性版本均不低于v2.2.0。

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

假定当前目录在conf，执行如下操作：

```
    mkdir -p stubs/bcos;
    cp stubs-sample/bcos/stub-sample.toml  stubs/bcos/stub.toml
```

查看stub.toml，可以看到文件内容如下：

```
[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 must be unique
    name = 'HelloWeCross'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x8827cca7f0f38b861b62dae6d711efe92a1e3602'
[[resources]]
    name = 'HelloWorld'
    type = 'BCOS_CONTRACT'
    contractAddress = '0x584ecb848dd84499639fbe2581bfb8a8774b485c'
```

`[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

docker安装需要满足内核版本不低于3.10。

centos环境下docker安装

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方式1：

```
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
```

若安装完成后输入docker-compose --version命令报如下错误，是由于网络不稳定导致下载失败，可尝试方式2进行安装。

```
Cannot open self /usr/local/bin/docker-compose or archive /usr/local/bin/
docker-compose.pkg
```

安装-docker-compose方式2：

```
sudo pip install docker-compose==1.18.0
```

查看docker-compose版本

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

go安装

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 ~
sudo 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=$PATH:/usr/local/go/bin
export GOROOT=/usr/local/go
export GOPATH=/data/go/
export PATH=$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.
```

请确保这一步可以正常输出，如果不能正常输出，请检查go的版本以及环境变量配置。

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

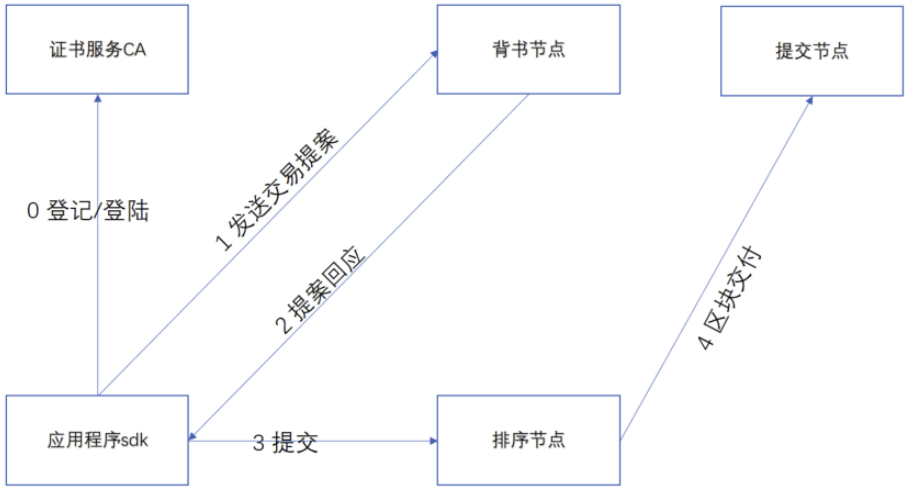
```
sudo docker exec -it cli bash
```

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

Fabric证书介绍



Fabric交易执行流程如下图:

从图中可以看出，和交易sdk直接交互的CA节点，背书节点和排序节点。 所以需要配置的证书包括： 1 CA证书，包括用户私钥和证书文件。 2 背书节点证书，背书节点有多少个就需要拷贝多少个背书节点证书。 3 排序节点证书。

Fabric证书路径说明

访问Fabric链依赖的证书需要从链上拷贝，以上面搭建的链为例，说明相关证书文件路径。

首先进入容器

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

CA证书路径

CA证书是分用户的，如果用户为Admin,则用户私钥和证书文件的路径分别为

```
    #私钥文件
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.
↪example.com/users/Admin@org1.example.com/msp/keystore/*_sk

    #证书文件
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.
↪example.com/users/Admin@org1.example.com/msp/signcerts/Admin@org1.example.com-
↪cert.pem
```

如果用户是User1,则用户私钥和证书文件的路径分别为:

```
    #私钥文件
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.
↪example.com/users/User1@org1.example.com/msp/keystore/*_sk

    #证书文件
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.
↪example.com/users/User1@org1.example.com/msp/signcerts/User1@org1.example.com-
↪cert.pem
```

背书节点证书

按照上面步骤部署出来的链码有两个背书节点，路径分别为:

```
#peer0.org1证书
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.
↪example.com/peers/peer0.org1.example.com/tls/ca.crt

#peer0.org2证书
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org2.
↪example.com/peers/peer0.org2.example.com/tls/ca.crt
```

排序节点证书

按照上面步骤部署出来的链码有一个排序节点，路径为:

```
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/
↪example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.
↪pem
```

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

假定当前目录在conf，执行如下操作:

```
mkdir -p stubs/fabric;
cp stubs-sample/fabric/stub-sample.toml  stubs/fabric/stub.toml
```

执行上述命令之后，目录结构变成如下：

```
├── log4j2.xml
├── p2p
│   ├── ca.crt
│   ├── node.crt
│   ├── node.key
│   └── node.nodeid
├── stubs
│   ├── bcos
│   │   └── stub-sample.toml
│   ├── fabric
│   │   └── stub-sample.toml
│   │   └── stub.toml
│
├── wecross-sample.toml
└── wecross.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:/stubs/fabric/orgUserKeyFile'
    orgUserCertFile = 'classpath:/stubs/fabric/orgUserCertFile'
    ordererTlsCaFile = 'classpath:/stubs/fabric/ordererTlsCaFile'
    ordererAddress = 'grpcs://127.0.0.1:7050'

[peers]
    [peers.org1]
        peerTlsCaFile = 'classpath:/stubs/fabric/peerOrg1CertFile'
        peerAddress = 'grpcs://127.0.0.1:7051'
    [peers.org2]
         peerTlsCaFile = 'classpath:/stubs/fabric/peerOrg2CertFile'
         peerAddress = 'grpcs://127.0.0.1:9051'

# resources is a list
[[resources]]
    # name cannot be repeated
    name = 'HelloWeCross'
    type = 'FABRIC_CONTRACT'
    chainCodeName = 'mycc'
    chainLanguage = "go"
    peers=['org1','org2']
[[resources]]
    name = 'HelloWorld'
    type = 'FABRIC_CONTRACT'
    chainCodeName = 'mygg'
    chainLanguage = "go"
    peers=['org1','org2']
```

需要配置的项包括：

orgUserName:用户名称，按照上面搭出来的链可选为Admin或者User1。

orgUserKeyFile:用户私钥文件，需要从链上拷贝，文件路径请参考ca证书路径。请拷贝文件,修改文件名为orgUserKeyFile并将文件拷贝到conf/stubs/fabric目录。

orgUserCertFile:用户证书文件，需要从链上拷贝。文件路径请参考ca证书路径。请拷贝文件,修改文件名为orgUserCertFile并将文件拷贝到conf/stubs/fabric目录。

ordererTlsCaFile:排序节点证书文件，需要从链上拷贝。文件路径请参考排序节点证书路径。请拷贝文件,修改文件名为ordererTlsCaFile并将文件拷贝到conf/stubs/fabric目录。

ordererAddress:排序节点地址，将默认的127.0.0.1改成真实ip。

peerTlsCaFile:背书节点证书文件，需要从链上拷贝。文件路径请参考背书节点证书路径。请拷贝对应的两个文件,分别修改文件名为peerOrg1CertFile和peerOrg2CertFile，并将文件拷贝到conf/stubs/fabric目录。

peerAddress:背书节点地址，将默认的127.0.0.1改成真实ip。

Fabric环境搭建常见问题定位

1. 启动docker提示:"dial unix /var/run/docker.sock: connect: permission denied" 解决方案：将当前用户加入到docker用户组

```
sudo gpasswd -a ${USER} docker
```

2 节点启动或者停止过程出现类似错误:

```
ERROR: for peer0.org2.example.com  container␣
↪4cd74d7c81ed915ebee257e1b9d73a0b53dd92447a44f7654aa36563adabbd06: driver
↪"overlay2" failed to remove root filesystem: unlinkat /var/lib/docker/overlay2/
↪14bc15bfac499738c5e4f12083b2e9907f5a304ff234d68d3ba95eef839f4a31/merged: device␣
↪or resource busy
```

解决方案:获得所有和docker相关的进程，找到正在使用的设备号对应的进程，kill掉进程。

```
grep docker /proc/*/mountinfo | grep␣
↪14bc15bfac499738c5e4f12083b2e9907f5a304ff234d68d3ba95eef839f4a31 | awk -F ':' '
↪{print $1}' | awk -F'/' '{print $3}'
```

CHAPTER 6

应用场景

6.1 数字资产跨链

区块链天然具有金融属性，有望为金融业带来更多创新。支付清算方面，在基于区块链技术的架构下，市场多个参与者维护的多个账本或区块链融合连通并实时交互，短短几分钟内就能完成现在两三天才能完成的支付、对账、清算任务，降低了跨行跨境交易的复杂性和成本；同时，区块链技术能够确保交易记录透明安全，方便监管部门追踪链上交易，快速定位高风险交易流向。数字票据和供应链金融方面，区块链技术可以有效解决中小企业融资难问题。目前的供应链金融很难惠及产业链上游的中小企业，因为他们跟核心企业往往没有直接贸易往来，金融机构难以评估其信用资质。基于区块链技术，可以建立一种联盟多链网络，涵盖核心企业、上下游供应商、金融机构等，核心企业发放应收账款凭证给其供应商，票据数字化上链后可在供应商之间跨链流转，每一级供应商可凭数字票据实现对应额度的融资。

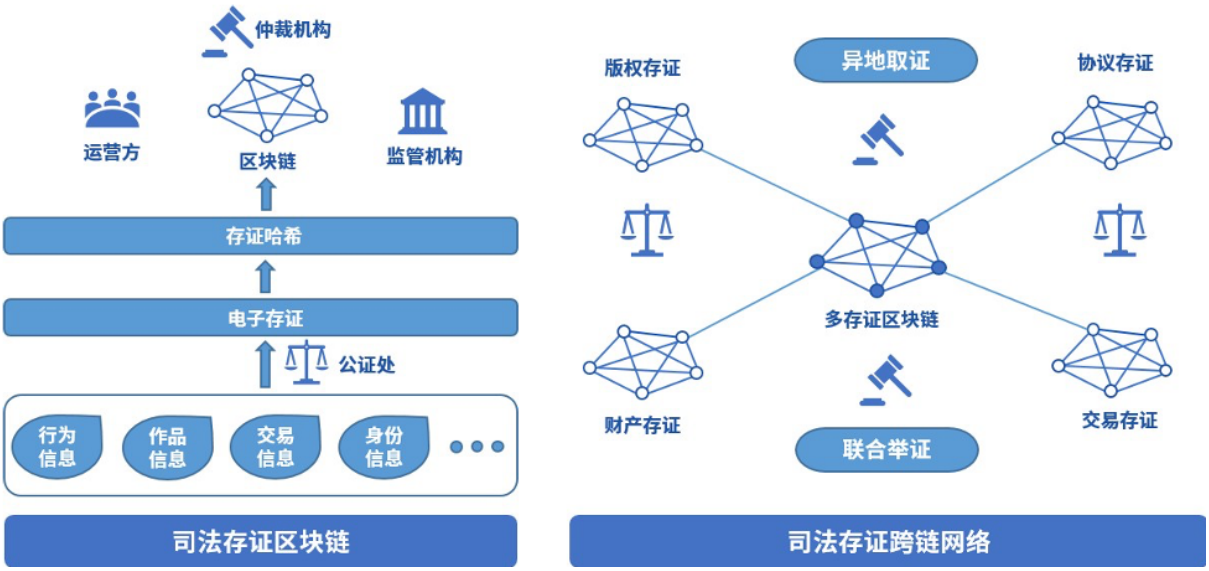
伴随着区块链在金融领域落地应用的飞速增长，多元化的数字资产场景和区块链应用带来了区块链资产相互隔离的问题，不同数字资产业务彼此搭建的区块链上的数字资产无法安全可信地实现互通，区块链上存在的数字资产价值越来越大，跨链的需求愈发迫切。

WeCross支持以多种网络拓扑模型搭建数字资产的跨链分区。在交易逻辑上，两阶段事务模型和HTLC事务模型将实现数字资产的去中心、去信任和不可篡改的转移。在安全防护上，加密和准入机制将保障数字资产转移的安全与可信。通过以上技术优势，WeCross将助力过去纸质形态的资产凭证全面数字化，让资产和信用层层深入传递到产业链末端，促进数字经济的发展。

6.2 司法跨域仲裁

随着数字经济高速发展，司法证据正逐步进入电子化时代。2017年9月，微众银行区块链团队与第三方存证公司合作，推出区块链司法存证与仲裁平台，开创将仲裁、法院等机构作为链上节点的先河，并于2018年2月，联合仲裁机构基于该平台出具业内首份裁决书，标志着区块链应用在司法领域的真正落地并完成价值验证；2018年6月，杭州互联网法院开始探求区块链在司法场景中的运用，进一步确立了区块链存证电子证据的合法性；2018年9月，北京互联网法院推出电子证据平台“天平链”，加速推动在网络空间治理的法治化进程。由于区块链司法应用能够极大缩减仲裁流程，仲裁机构得以快速完成证据核实，快速解决纠纷。

随着区块链应用在司法存证领域的普及，不同司法存证链之间连通的需求愈发强烈。但区块链的信任模型使得不同的司法存证链上的证据无法互通互信，当司法仲裁需要异地取证或是联合举证时，需要引入一个中心化的可信机构来进行协调，影响了区块链的实用价值。



WeCross跨链技术可以将各家存证链的证据统一抽象成证据资源，在不同的司法存证链之间可信地传输证据。WeCross可以搭建一个拥有多类型存证的存证链网络，在面向重大问题和重大纠纷时，去中心化地帮助各个链交互完备、可信和强有力的证据材料，帮助仲裁机构完成裁决。

6.3 个体数据跨域授权

随着WeIdentity、Hyperledger Indy等遵循DID协议的区块链身份认证系统出现，多个国家和地区开展了多中心化身份认证的实践与落地，多中心化身份认证目前市场需求巨大，加之政策鼓励支持，行业方兴未艾，处于高速发展的黄金时期。2019年2月27日，微众银行区块链团队与澳门政府设立的澳门科学技术发展基金签署合作协议，在智慧城市、民生服务、政务管理、人才培训等方面开展合作。双方合作的首个项目基于“WeIdentity”的实体身份标识及可信数据交换解决方案展开，这是区块链在粤港澳大湾区应用落地的重要一步。

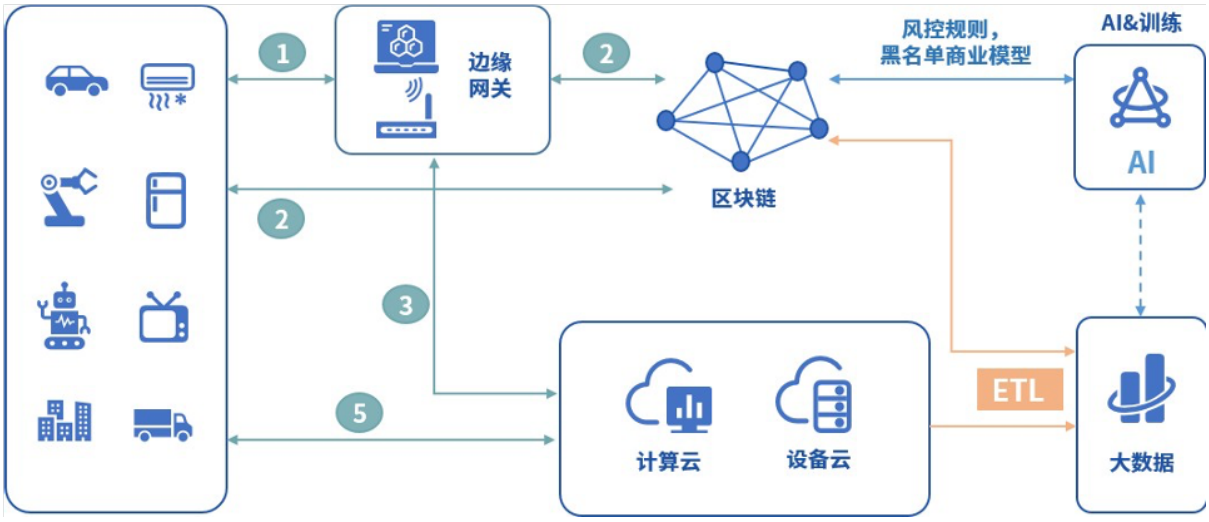
身份认证正向跨地域的方向发展，不同地域、业务和基于不同区块链平台的身份认证产品之间尚不能互认的现状造成信息的鸿沟，导致身份和资质等数据仍然局限在小范围的地域和业务内，无法互通。

WeCross 可以将多个不同架构、行业和地域的多中心化身份认证平台联结起来，帮助多中心化身份认证更好地解决数据孤岛、数据滥用和数据黑产的问题，在推进数据资源开放共享与信息流通，促进跨行业、跨领域、跨地域大数据应用，形成良性互动的产业发展格局上，发挥更大的作用。

6.4 物联网跨平台联动

随着智能穿戴、智能家居、无人机及以人脸识别等人工智能设备的普及，智能设备的类别越来越多，人机交互的频次也越来越高，物联网数据的类型和结构呈现多样化和复杂化的趋势。在5G时代，实现万物互联之后，数据和场景的复杂度更是呈几何倍数增长。区块链技术为物联网设备提供信任机制，保证所有权、交易等记录的可信性、可靠性及透明性，同时还可为用户隐私提供保障机制，从而有效解决物联网发展面临的大数据管理、信任、安全和隐私等问题，推进物联网向更加灵活化、智能化的形态演进。

目前物联网行业的区块链项目，有的旨在解决物联网碎片化严重、物联网产品没有标准化等痛点，有的则探索区块链在智能城市、基础设施、智能电网、供应链以及运输等领域的应用。然而，它们都面临着相同的困境。物联网设备硬件模块的选择和组合非常多样，对区块链平台的支持能力不尽相同，一旦硬件部署完成后难以更新，单一的区块链平台在连通多样化的物联网设备时必然会遇到瓶颈，无法全面满足所有物联网设备在多样化场景中的需求。



WeCross跨链技术支持物联网设备跨链平行扩展，可用于构建高效、安全的分布式物联网网络，以及部署海量设备网络中运行的数据密集型应用；WeCross跨链技术可以安全可信地融合连通多个物联网设备的区块链，在功能和安全上满足多样的场景需求。

CHAPTER 7

FAQ

CHAPTER 8

社区

8.1 贡献代码

8.1.1 接入更多链

WeCross，目前适配了

- FISCO BCOS
- Fabric

WeCross一直在迭代，想接入什么链？一起来写代码吧

- 联系社区（issue） -> 手把手指导 -> 开发 -> 贡献PR -> 合入 -> WeCross！

8.1.2 更多

- 点Star！！！
- 提交代码(Pull Requests)。
- 提问和提交BUG