WeCross Documentation

发布 ***v1.2.1***

WeCross Community

2022 年 **12** 月 **14** 日

平台介绍

WeCross是由微众银行自主研发并完全开源的区块链跨链协作平台，支持应用与多链互操作、同/异构链间互操作等多维跨链交互。

- Github主页
- Gitee主页
- 深度解析系列文章
- 贡献代码
- 反馈问题
- 微信群
- 公众号

概览

- 系统推荐配置以及软件依赖，请参考 环境要求
- 快速体验WeCross，请参考 BCOS-Fabric Demo
- 网页管理平台：**可视化跨链管理台**，账户注册与登录，跨链资源管理，跨链交易管理等
- 终端控制台：**交互式命令行工具**，部署和调用跨链资源，发起跨链事务等
- Java-SDK：提供Java项目访问跨链跨链、调用跨链资源的接口
- Go-SDK：提供Go项目问跨链跨链、调用跨链资源的接口
- JSON-RPC接口，请参考 JSON-RPC API

关键特性

- 合约跨链: 使用手册
- 跨链事务: 使用手册
- 跨链转账: 使用手册

CHAPTER 1

WeCross

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。

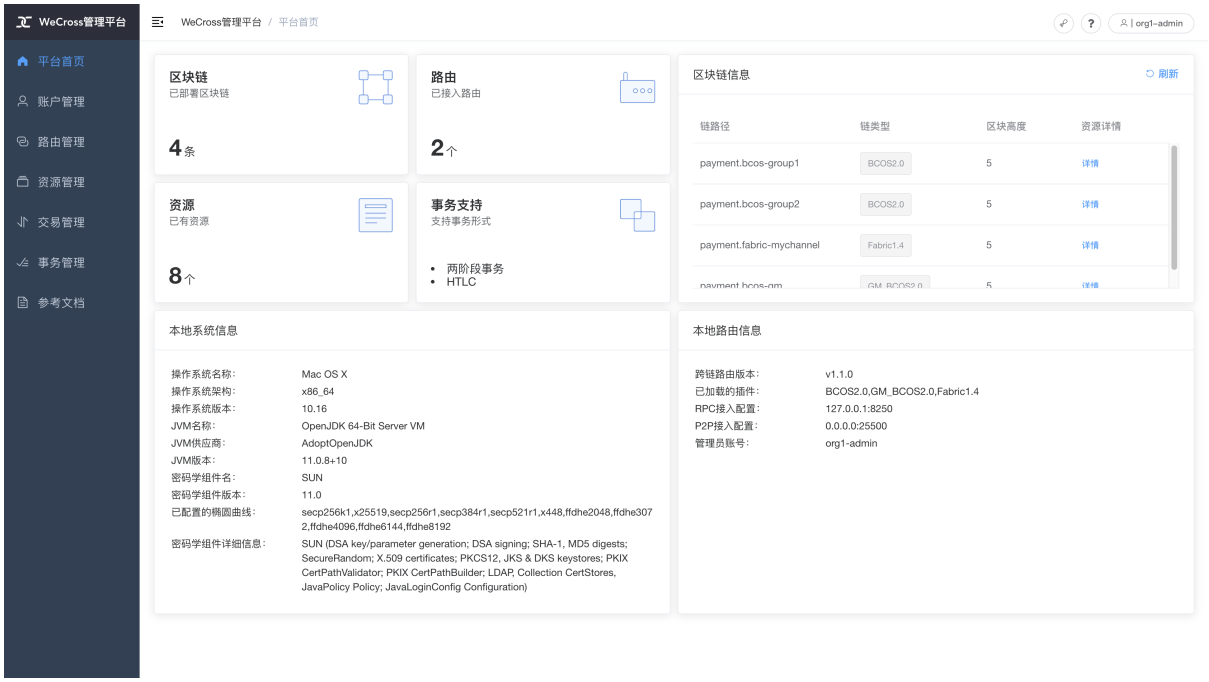
作为一家具有互联网基因的高科技、创新型银行，微众银行自成立之初即高度重视新兴技术的研究和探索，在区块链领域积极开展技术积累和应用实践，不断致力于运用区块链技术提升多机构间的协作效率和降低协作成本，支持国家推进关键技术安全可控战略和推动社会普惠金融发展。微众银行区块链团队基于一揽子自主研发并开源的区块链技术方案，针对不同服务形态、不同区块链平台之间无法进行可信连接与交互的行业痛点，研发区块链跨链协作平台——WeCross，以促进跨行业、机构和地域的跨区块链信任传递和商业合作。

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

- **账户服务（WeCross Account Manager）**
  - 管理跨链账户
  - Router连接所属机构的Account Manager
  - 用户在Router上登录，以跨链账户的身份发交易
- **控制台（WeCross Console）**
  - 命令行式的交互
  - 查询跨链信息，发送调用请求，操作跨链事务
- **网页管理平台**
  - 可视化操作界面
  - 查询跨链信息，发送调用请求，操作跨链事务
- **跨链 SDK（WeCross Java SDK, WeCross Go 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
  - 在代码和文档中将IPath简称为path
- **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 下载程序

2.1.1 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

GitHub下载方式：

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

Gitee下载方式：

```
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪wecross.sh)
```

方式2：命令下载（源码编译模式）

GitHub下载方式：

```
# 默认下载master分支
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_wecross.sh) -s

# 下载特定版本下的
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_wecross.sh) -s -t v1.2.1
```

Gitee下载方式：

```
# 默认下载master分支
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪wecross.sh) -s

# 下载特定版本下的
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪wecross.sh) -s -t v1.2.1
```

(下页继续)

(续上页)

方式3：手动下载

- 国内资源快速下载方式(CDN)：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。

2.1.2 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

GitHub下载方式：

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

Gitee下载方式：

```
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh)
```

方式2：命令下载（源码编译模式）

GitHub下载方式：

```
# 默认下载master分支
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_console.sh) -s

# 下载特定版本下的控制台
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_console.sh) -s -t v1.2.1
```

Gitee下载方式：

```
# 默认下载master分支
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh) -s

# 下载特定版本下的控制台
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh) -s -t v1.2.1
```

方式3：手动下载

- 国内资源快速下载方式(CDN)：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。

2.1.3 下载WeCross Account Manager

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

GitHub下载方式：

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

Gitee下载方式：

```
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪account_manager.sh)
```

方式2：命令下载（源码编译模式）

GitHub下载方式：

```
# 默认下载master分支，
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_account_manager.sh) -s

# 下载特定版本下的控制台
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_account_manager.sh) -s -t v1.2.1
```

Gitee下载方式：

```
# 默认下载master分支，
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪account_manager.sh) -s

# 下载特定版本下的控制台
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪account_manager.sh) -s -t v1.2.1
```

方式3：手动下载

- 国内资源快速下载方式(CDN)：点击下载，MD5
- github release（下载最新版本的 `WeCross-Account-Manager.tar.gz`）

  下载后执行命令

  ```
  tar -zxvf WeCross-Account-Manager.tar.gz # 解压
  cd WeCross-Account-Manager/conf # 进入
  mysql -u your_database_username -p < db_setup.sql # 配置数据库
  ```

下载后，目录下包含`WeCross-Account-Manager/`文件夹。

2.1.4 下载WeCross Demo

同样提供两种方式，根据网络环境选择合适的方式进行下载。

方式1：命令下载

GitHub下载方式：

```
# 下载最新demo
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_demo.sh)
```

(下页继续)

(续上页)

```
# 下载特定版本下的demo
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_demo.sh) -t v1.2.1
```

Gitee下载方式：

```
# 下载最新demo
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_demo.
↪sh)

# 下载特定版本下的demo
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_demo.
↪sh) -t v1.2.1
```

方式2：手动下载

- 国内资源快速下载方式(CDN)：点击下载，MD5
- github release（下载最新release下的`demo.tar.gz`）

  手动下载解压

  ```
  tar -zxvf demo.tar.gz
  mv demo wecross-demo # 命名为wecross-demo目录
  ```

下载后，`wecross-demo/`目录即为WeCross Demo。

2.2 v1.2.1

(2021-12-15)

修复

- 修复log4j的漏洞，将其升级至2.15

兼容性

向前兼容，支持v1.0.x升级至v1.2.x

v1.0.x升级至v1.2.1的方法

下载更新脚本、安装包

```
# 创建新目录，保存升级使用的脚本、安装包
mkdir -p ~/wecross-upgrade/v1.2.1 && cd ~/wecross-upgrade/v1.2.1
```

1. 下载安装包，请根据实际网络情况选择GitHub下载方式和Gitee下载方式：

GitHub下载方式：

```
## 下载 WeCross 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_wecross.sh) -b v1.2.1

## 下载 Console 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_console.sh) -b v1.2.1

## 下载 Account-Manager 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_account_manager.sh) -b v1.2.1
```

Gitee下载方式:

```
## 下载 WeCross 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪wecross.sh) -b v1.2.1

## 下载 Console 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh) -b v1.2.1

## 下载 Account-Manager 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪account_manager.sh) -b v1.2.1
```

1. 获取升级脚本,请根据实际网络情况选择GitHub下载方式和Gitee下载方式:

GitHub下载方式:

```
cd ~/wecross-upgrade/v1.2.1

# Router
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.2.1/
↪upgrade_wecross.sh && chmod u+x upgrade_wecross.sh

# Console
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.2.1/
↪upgrade_console.sh && chmod u+x upgrade_console.sh

# Account-Manager
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.2.1/
↪upgrade_account_manager.sh && chmod u+x upgrade_account_manager.sh
```

Gitee下载方式:

```
cd ~/wecross-upgrade/v1.2.1

# Router
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_wecross.sh &&␣
↪chmod u+x upgrade_wecross.sh

# Console
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_console.sh &&␣
↪chmod u+x upgrade_console.sh

# Account-Manager
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_account_manager.
↪sh && chmod u+x upgrade_account_manager.sh
```

1. 升级操作

```
cd ~/wecross-upgrade/v1.2.1

# Account-Manager
bash upgrade_account_manager.sh -d ~/wecross-demo/WeCross-Account-Manager # 将路径替
换WeCross-Account-Manager的实际部署路径

# Console
bash upgrade_console.sh -d ~/wecross-demo/WeCross-Console # 将路径替换为WeCross-
↪Console的实际部署路径

# WeCross
bash upgrade_wecross.sh -d ~/wecross-demo/routers-payment/127.0.0.1-8250-25500 # 将
路径替换为router的实际部署路径
```

1. 重启服务

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

2.3 v1.2.0

(2021-08-20)

新增

- 接入 Hyperledger Fabric 2+，提供相关演示demo
- 资源访问控制功能，管理员可通过网页管理台给用户授权可访问的资源
- 容器化部署

更改

- 适配外部登录系统，支持通过外部定义的身份操作跨链网络

兼容性

向前兼容，支持v1.0.x升级至v1.2.0

v1.0.x升级至v1.2.0的方法

下载更新脚本、安装包

```
# 创建新目录，保存升级使用的脚本、安装包
mkdir -p ~/wecross-upgrade/v1.2.0 && cd ~/wecross-upgrade/v1.2.0
```

1. 下载安装包，请根据实际网络情况选择GitHub下载方式和Gitee下载方式：

GitHub下载方式：

```
## 下载 WeCross 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_wecross.sh) -b v1.2.0

## 下载 Console 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_console.sh) -b v1.2.0

## 下载 Account-Manager 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_account_manager.sh) -b v1.2.0
```

Gitee下载方式：

```
## 下载 WeCross 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪wecross.sh) -b v1.2.0

## 下载 Console 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh) -b v1.2.0

## 下载 Account-Manager 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪account_manager.sh) -b v1.2.0
```

1. 获取升级脚本,请根据实际网络情况选择GitHub下载方式和Gitee下载方式：

GitHub下载方式：

```
cd ~/wecross-upgrade/v1.2.0

# Router
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.2.0/
↪upgrade_wecross.sh && chmod u+x upgrade_wecross.sh

# Console
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.2.0/
↪upgrade_console.sh && chmod u+x upgrade_console.sh

# Account-Manager
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.2.0/
↪upgrade_account_manager.sh && chmod u+x upgrade_account_manager.sh
```

Gitee下载方式:

```
cd ~/wecross-upgrade/v1.2.0

# Router
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_wecross.sh &&␣
↪chmod u+x upgrade_wecross.sh

# Console
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_console.sh &&␣
↪chmod u+x upgrade_console.sh

# Account-Manager
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_account_manager.
↪sh && chmod u+x upgrade_account_manager.sh
```

1. 升级操作

```
cd ~/wecross-upgrade/v1.2.0

# Account-Manager
bash upgrade_account_manager.sh -d ~/wecross-demo/WeCross-Account-Manager # 将路径替
换WeCross-Account-Manager的实际部署路径

# Console
bash upgrade_console.sh -d ~/wecross-demo/WeCross-Console # 将路径替换为WeCross-
↪Console的实际部署路径

# WeCross
bash upgrade_wecross.sh -d ~/wecross-demo/routers-payment/127.0.0.1-8250-25500 # 将
路径替换为router的实际部署路径
```

1. 重启服务

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

2.4 v1.1.1

(2021-04-02)

更改

- 支持RPC端口的URL地址前缀可配，以兼容更多的部署环境
- 各组件启动时打印版本号，使操作更清晰

- Demo添加启停脚本，支持启动、停止已经部署好的Demo
- 可视化管理台用户体验优化
- 优化区块头验证代码结构

兼容性

向前兼容，支持v1.0.x升级至v1.1.x

v1.0.x升级至v1.1.1的方法

下载更新脚本、安装包

```
# 创建新目录，保存升级使用的脚本、安装包
mkdir -p ~/wecross-upgrade/v1.1.1 && cd ~/wecross-upgrade/v1.1.1
```

1. 下载安装包，请根据实际网络情况选择GitHub下载方式和Gitee下载方式：

GitHub下载方式：

```
## 下载 WeCross 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_wecross.sh) -b v1.1.1

## 下载 Console 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_console.sh) -b v1.1.1

## 下载 Account-Manager 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_account_manager.sh) -b v1.1.1
```

Gitee下载方式：

```
## 下载 WeCross 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪wecross.sh) -b v1.1.1

## 下载 Console 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh) -b v1.1.1

## 下载 Account-Manager 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪account_manager.sh) -b v1.1.1
```

1. 获取升级脚本,请根据实际网络情况选择GitHub下载方式和Gitee下载方式:

GitHub下载方式：

```
cd ~/wecross-upgrade/v1.1.1

# Router
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.1.1/
↪upgrade_wecross.sh && chmod u+x upgrade_wecross.sh

# Console
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.1.1/
↪upgrade_console.sh && chmod u+x upgrade_console.sh

# Account-Manager
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.1.1/
↪upgrade_account_manager.sh && chmod u+x upgrade_account_manager.sh
```

Gitee下载方式：

```
cd ~/wecross-upgrade/v1.1.1

# Router
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_wecross.sh &&␣
↪chmod u+x upgrade_wecross.sh

# Console
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_console.sh &&␣
↪chmod u+x upgrade_console.sh

# Account-Manager
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_account_manager.
↪sh && chmod u+x upgrade_account_manager.sh
```

1. 升级操作

```
cd ~/wecross-upgrade/v1.1.1

# Account-Manager
bash upgrade_account_manager.sh -d ~/wecross-demo/WeCross-Account-Manager # 将路径替
换WeCross-Account-Manager的实际部署路径

# Console
bash upgrade_console.sh -d ~/wecross-demo/WeCross-Console # 将路径替换为WeCross-
↪Console的实际部署路径

# WeCross
bash upgrade_wecross.sh -d ~/wecross-demo/routers-payment/127.0.0.1-8250-25500 # 将
路径替换为router的实际部署路径
```

1. 重启服务

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

2.5 v1.1.0

(2020-02-02)

功能

- 适配FISCO BCOS 2.1-2.5版本
- SSL连接证书格式修改为RSA
- BCOS Stub支持国密SSL连接
- 支持FISCO BCOS和Hyperledger Fabric区块头校验

新增

- 添加changePassword接口
- 添加修改密码功能
- 添加页面帮助指引

更改

- demo脚本优化，支持非交互式运行
- 升级依赖版本，详情参考build.gradle修改内容
- 修改login register接口，添加token认证流程，简化使用方式

- 修改login逻辑，参数加密以及token验证逻辑移植到java-sdk中

删除

- 删除status命令

兼容性

向前兼容，支持v1.0.x升级至v1.1.0

v1.0.x升级至v1.1.0的方法

下载更新脚本、安装包

```
# 创建新目录，保存升级使用的脚本、安装包
mkdir -p ~/wecross-upgrade/v1.1.0 && cd ~/wecross-upgrade/v1.1.0
```

1. 下载安装包，请根据实际网络情况选择GitHub下载方式和Gitee下载方式：

GitHub下载方式：

```
## 下载 WeCross 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_wecross.sh) -b v1.1.0

## 下载 Console 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_console.sh) -b v1.1.0

## 下载 Account-Manager 新版本安装包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_account_manager.sh) -b v1.1.0
```

Gitee下载方式：

```
## 下载 WeCross 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪wecross.sh) -b v1.1.0

## 下载 Console 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh) -b v1.1.0

## 下载 Account-Manager 新版本安装包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪account_manager.sh) -b v1.1.0
```

1. 获取升级脚本,请根据实际网络情况选择GitHub下载方式和Gitee下载方式:

GitHub下载方式：

```
cd ~/wecross-upgrade/v1.1.0

# Router
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.1.0/
↪upgrade_wecross.sh && chmod u+x upgrade_wecross.sh

# Console
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.1.0/
↪upgrade_console.sh && chmod u+x upgrade_console.sh

# Account-Manager
curl -LO# https://github.com/WeBankBlockchain/WeCross/releases/download/v1.1.0/
↪upgrade_account_manager.sh && chmod u+x upgrade_account_manager.sh
```

Gitee下载方式：

```
cd ~/wecross-upgrade/v1.1.0

# Router
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_wecross.sh &&␣
↪chmod u+x upgrade_wecross.sh

# Console
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_console.sh &&␣
↪chmod u+x upgrade_console.sh

# Account-Manager
curl -LO# https://gitee.com/WeBank/WeCross/raw/dev/scripts/upgrade_account_manager.
↪sh && chmod u+x upgrade_account_manager.sh
```

1. 升级操作

```
cd ~/wecross-upgrade/v1.1.0

# Account-Manager
bash upgrade_account_manager.sh -d ~/wecross-demo/WeCross-Account-Manager # 将路径替
换WeCross-Account-Manager的实际部署路径

# Console
bash upgrade_console.sh -d ~/wecross-demo/WeCross-Console # 将路径替换为WeCross-
↪Console的实际部署路径

# WeCross
bash upgrade_wecross.sh -d ~/wecross-demo/routers-payment/127.0.0.1-8250-25500 # 将
路径替换为router的实际部署路径
```

1. 重启服务

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

2.6 v1.0.1

(2020-01-15)

新增

- 新增deploy_system_contract.sh脚本，替换Java命令部署Proxy、Hub

更改

- 启动脚本添加参数，修复新版本JDK无法使用的问题

兼容性

向前兼容

2.7 v1.0.0

(2020-12-17)

功能

- 合约跨链调用：支持由合约发起跨链调用
- 跨链账户管理：支持跨链账户管理，统一跨链身份

- 网页管理平台：可视化的跨链管理组件

新增

- 桥接合约：合约跨链调用统一入口，管理跨链调用请求
- 账户管理：新增UniversalAccount管理链账户，新增注册与登录接口，透传账户相关请求至账户服务
- 网页管理平台：支持网页管理平台静态资源的打包和加载
- RPC接口：新增listTransactions、getTransaction、listXATransactions、getXATransaction等接口
- 更多Demo：新增跨 FISCO BCOS 群组、FISCO BCOS 国密链、Hyperledger Fabric 链 Demo

更改

- 资源调用：参数链账户名替换为跨链账户用户名
- 账户配置：无需在跨链路由配置链账户，账户统一由账户服务管理
- 默认账户：跨链路由新增默认账户，负责合约跨链和HTLC的调度
- 事务优化：开启事务若部分链失败则提交已成功的链，优化各个步骤的错误反馈
- HTLC优化：删除默认账户配置，HTLC合约初始化无需指定对手方合约地址

兼容性

WeCross v1.0.0是发布的第一个正式版本，与已经发布的RC版本不保持兼容性。

2.8 v1.0.0-rc4

(2020-08-18)

功能

- 两阶段事务框架：基于框架进行开发，实现多条异构链间的原子操作
- 跨链资源动态管理：通过API动态部署、更新跨链资源，无需编辑配置文件

新增

- 两阶段事务框架：框架、示例、逻辑和API等
- 代理合约：支持在运行时通过API对跨链资源进行部署和更新
- 更多的Demo
  - 两阶段 Demo
  - 跨 FISCO BCOS 群组 Demo
  - 跨 FISCO BCOS 国密与非国密链 Demo

更改

- 区块头同步逻辑更新：去除区块头落盘、router重启拉取最新区块头
- HTLC更新：使用WeCross-Console替代ledger-tool来初始化资产

兼容性

WeCross v1.0.0-rc4为RC版本，与其他版本不保持兼容性，推荐使用最新的正式发布版本。

2.9 v1.0.0-rc3

(2020-06-16)

新增

- Driver新增异步API定义：asyncCall、asyncSendTransaction，采用异步的方式调用插件接口

更改

- P2P通信：Router间的通信更新为异步的方式
- RPC接口：将性能较差的spring boot tomcat替换成netty的http server
- HTLC：适配Driver的异步API，采用异步的方式进行调用

兼容性

WeCross v1.0.0-rc3为RC版本，与其他版本不保持兼容性，推荐使用最新的正式发布版本。

2.10 v1.0.0-rc2

(2020-05-12)

新增

- 账户管理：用户账户统一由Router管理
- HTLC事务：支持同/异构链之间基于HTLC合约完成跨链转账
- Stub插件化：FISCO BCOS Stub和Fabric Stub通过jar包方式引入
- 安全通讯：WeCross SDK和Router之间采用TLS协议通讯
- 跨链Demo: 支持快速搭建WeCross Demo，体验简单跨链调用

更改

- 跨链接口：跨链调用需要指定账户名
- 跨链合约：跨链合约的参数类型和返回值类型限定为字符串数组
- 配置文件：主配置新增TLS以及HTLC配置项，Stub配置移除账户配置项
- 使用脚本：部署脚本、配置脚本以及证书生成脚本适配新的配置项

兼容性

WeCross v1.0.0-rc2为RC版本，与其他版本不保持兼容性，推荐使用最新的正式发布版本。

2.11 v1.0.0-rc1

(2019-12-30)

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

兼容性

WeCross v1.0.0-rc1为RC版本，与其他版本不保持兼容性，推荐使用最新的正式发布版本。

CHAPTER 3

环境要求

3.1 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

3.2 支持的平台

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

3.3 软件依赖

WeCross作为Java项目，需要安装Java环境包括：

- JDK1.8.0_251以上，可参考链接

目前已经覆盖测试的JDK版本：OracleJDK 1.8.0_251，OracleJDK 1.8.0_271，OracleJDK 14，OracleJDK 15，OpenJDK 1.8.0_282，OpenJDK 14，OpenJDK 15

- Gradle 5.0及以上
- MySQL 5.6及以上
  - MySQL官方安装文档
  - 安装教程

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 fontconfig

# CentOS
sudo yum install -y openssl curl expect tree

# macOS
brew install openssl curl expect tree md5sha1sum
```

运行WeCross Demo时，需安装

- Docker 17.06.2-ce 及以上
  - Docker官方安装文档
  - Docker安装教程

CHAPTER 4

快速体验

我们提供跨链Demo帮助用户快速体验并理解WeCross的原理，如果想搭建WeCross并**接入已有区块链**，请参考手动组网。

下载 Demo

执行命令下载Demo，若下载较慢，可选择更多下载方式。

```
cd ~
# 下载WeCross demo合集，生成wecross-demo目录，目录下包含各种类型的demo
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_demo.sh)

# 若出现长时间下载Demo包失败，请尝试以下命令重新下载:
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_demo.
↪sh)
```

注解:

- macOS用户若出现“无法打开”，“无法验证开发者”的情况，可参考 FAQ问题3 的方式解决

Demo 场景

1. Demo：混合场景（Fabric、FISCO BCOS 国密非国密、多群组）
2. Demo：跨平台 FISCO BCOS & Fabric
   - 跨链资源操作
   - 跨链转账（HTLC）
   - 跨链存证（2PC）
3. Demo：跨平台 FISCO BCOS & Fabric2
4. Demo：跨多群组
5. Demo：跨国密、非国密

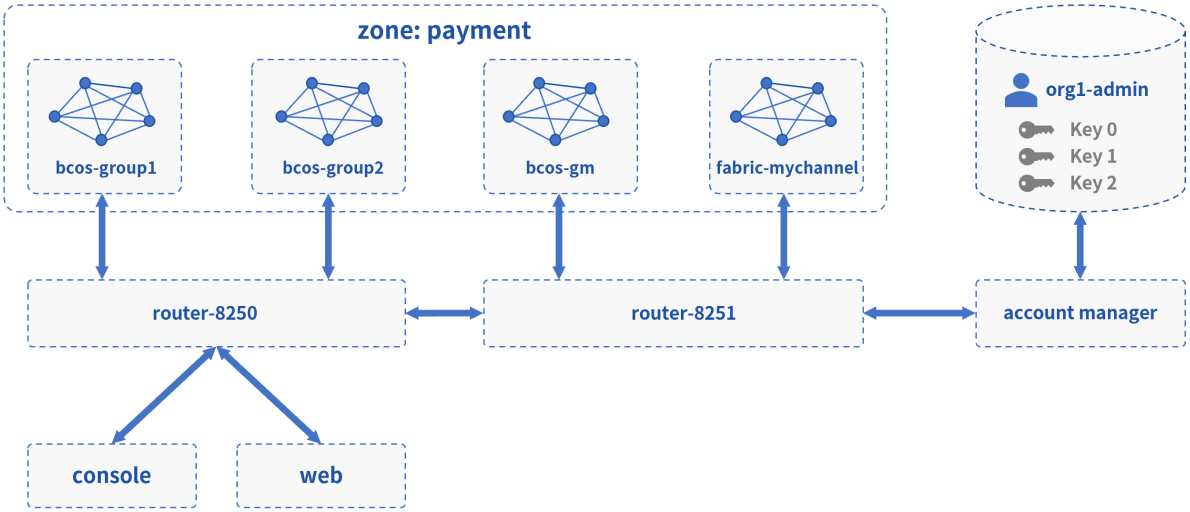
实际情况下，WeCross的场景不仅限于两条链，用户可配置接入多条各种类型的区块链。

4.1 混合场景

此Demo搭建了一个WeCross跨链网络，连接以下四条链（群组）：

- Hyperledger Fabric 链：fabric-mychannel
- FISCO BCOS 国密链：bcos-gm
- FISCO BCOS 非国密链群组1：bcos-group1
- FISCO BCOS 非国密链群组2：bcos-group2

搭建后，用户可使用网页管理平台和WeCross控制台，对不同的链上资源进行操作。



4.1.1 网络部署

在已下载的demo目录下进行操作

```
cd ~/wecross-demo

#清理旧demo环境
bash clear.sh

# 运行部署脚本，第一次运行需耗时10-30分钟左右
bash build_cross_all.sh # 若出错，可用 bash clear.sh 清理后重试。bash build.sh -h 可查看更多用法
```

重要:

- 若出现“command not found”， 则说明缺少依赖，请参考 环境要求 安装相关依赖
- macOS用户若出现“无法打开”，“无法验证开发者”的情况，可参考 FAQ问题3 的方式解决
- 输入数据库IP时，若”127.0.0.1”无法成功，请尝试输入”localhost”
- 若出现其它问题，请参考常见问题说明 FAQ

部署成功后会输出Demo的网络架构，FISCO BCOS和Fabric通过各自的WeCross Router相连。（输入Y，回车，进入WeCross控制台）

```
[INFO] Success! WeCross demo network is running. Framework:

       FISCO BCOS          FISCO BCOS          FISCO BCOS            Fabric
```

(下页继续)

(续上页)

```
        Group 1             Group 2              Guomi           first-network
   (HelloWorldGroup1)  (HelloWorldGroup2)     (HelloWorld)         (sacc.go)
            \                 /                    \                /
             \               /                      \              /
              \             /                        \            /
               WeCross  Router  <-------------------->  WeCross  Router <----------> WeCross Account Manager
           (127.0.0.1-8250-25500)                  (127.0.0.1-8251-25501)      (127.0.0.1:8340)
              /             \
             /               \
            /                 \
     WeCross WebApp      WeCross Console

Start WeCross Console? [Y/n]
```

4.1.2 控制台操作

登录跨链账户

进入控制台，首先登录跨链账户。（Demo中已配置好一个账户：org1-admin，密码：123456）

```
[WeCross]> login org1-admin 123456
Result: success
=============================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

查看账户

用`listAccount`命令查看此跨链账户下，向不同类型的链发送交易的链账户。

```
[WeCross.org1-admin]> listAccount
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
chainAccounts: [
        BCOS2.0 Account:
        keyID    : 0
        type     : BCOS2.0
        address  : 0x445554f52beb4e85ccb2edbab3a6f4e125eb61b7
        isDefault: true
        ----------
        GM_BCOS2.0 Account:
        keyID    : 4
        type     : GM_BCOS2.0
        address  : 0x4fef3c35a60490d1f01b2e749c17ea83c7c6611e
        isDefault: true
        ----------
        Fabric1.4 Account:
        keyID    : 1
        type     : Fabric1.4
        MembershipID : Org1MSP
        isDefault: true
        ----------
        Fabric1.4 Account:
        keyID    : 3
```

(下页继续)

(续上页)

```
        type     : Fabric1.4
        MembershipID : Org1MSP
        isDefault: false
        ----------
        Fabric1.4 Account:
        keyID    : 2
        type     : Fabric1.4
        MembershipID : Org2MSP
        isDefault: false
        ----------
]
```

查看资源

用`listResources`命令查看WeCross跨连网络中的所有资源。可看到有多个跨链资源：

- `payment.bcos-group1.HelloWorldGroup1`
  - bcos-group1链上的HelloWorld.sol合约
- `payment.bcos-group2.HelloWorldGroup2`
  - bcos-group2链上的HelloWorld.sol合约
- `payment.bcos-gm.HelloWorld`
  - 对应于国密FISCO BCOS链上的HelloWorld.sol合约
- `payment.fabric-mychannel.sacc`
  - 对应于Fabric链上的sacc.go合约
- `payment.xxxx.WeCrossHub`
  - 每条链默认安装的Hub合约，用于接收链上合约发起的跨链调用，可参考《合约跨链》

```
path: payment.bcos-group2.HelloWorldGroup2, type: BCOS2.0, distance: 0
path: payment.bcos-group2.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.bcos-group1.HelloWorldGroup1, type: BCOS2.0, distance: 0
path: payment.bcos-group1.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.fabric-mychannel.WeCrossHub, type: Fabric1.4, distance: 1
path: payment.bcos-gm.WeCrossHub, type: GM_BCOS2.0, distance: 1
path: payment.bcos-gm.HelloWorld, type: GM_BCOS2.0, distance: 1
path: payment.fabric-mychannel.sacc, type: Fabric1.4, distance: 1
total: 8
```

操作资源

本demo涉及资源较多，可在其他demo体验。请直接访问网页管理平台，更直观的操作跨链资源。

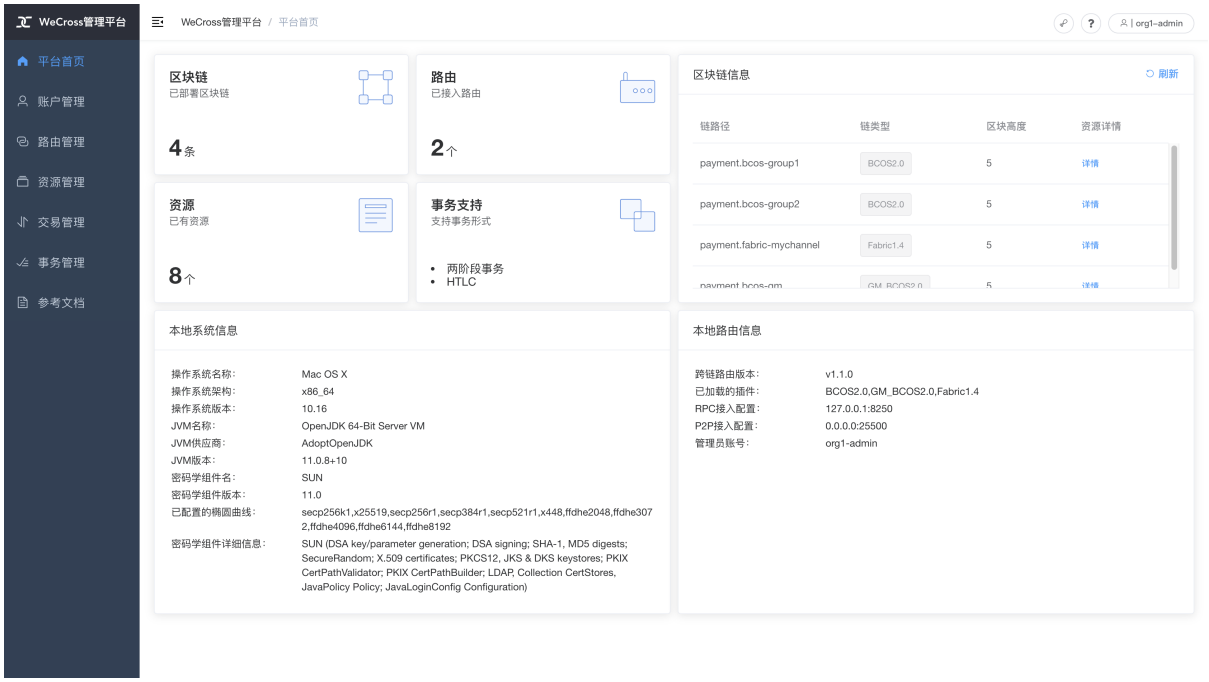
```
# 退出当前控制台
[WeCross.org1-admin]> quit
```

4.1.3 访问网页管理平台

浏览器访问`router-8250`的网页管理平台

```
http://localhost:8250/s/index.html#/login
```

用demo已配置账户进行登录：`org1-admin`，密码：`123456`



管理台中包含如下内容，点击链接进入相关操作指导。

- 登录/注册
- 平台首页
- 账户管理
- 路由管理
- 资源管理
- 交易管理
- 事务管理

注解:

- 若需要远程访问，请修改router的主配置（如：~/demo/routers-payment/127.0.0.1-8250-25500/conf/wecross.toml），将 `[rpc]` 标签下的 `address` 修改为所需ip（如：0.0.0.0）。保存后，重启router即可。

4.1.4 清理 Demo

为了不影响其它章节的体验，可将搭建的Demo清理掉。

```
cd ~/wecross-demo/
bash clear.sh

# 可使用脚本drop_account_database.sh删除MySQL中demo使用的数据库
bash drop_account_database.sh

# Tips: 可选用配置MySQL参数，进行无交互式部署，详情请参考下述脚本输出
bash drop_account_database.sh -h

Drop wecross-account-manager database named wecross_account_manager.
Usage:
    -d                                [Optional] Use default db configuration: -H␣
↪127.0.0.1 -P 3306 -u root -p 123456
```

(下页继续)

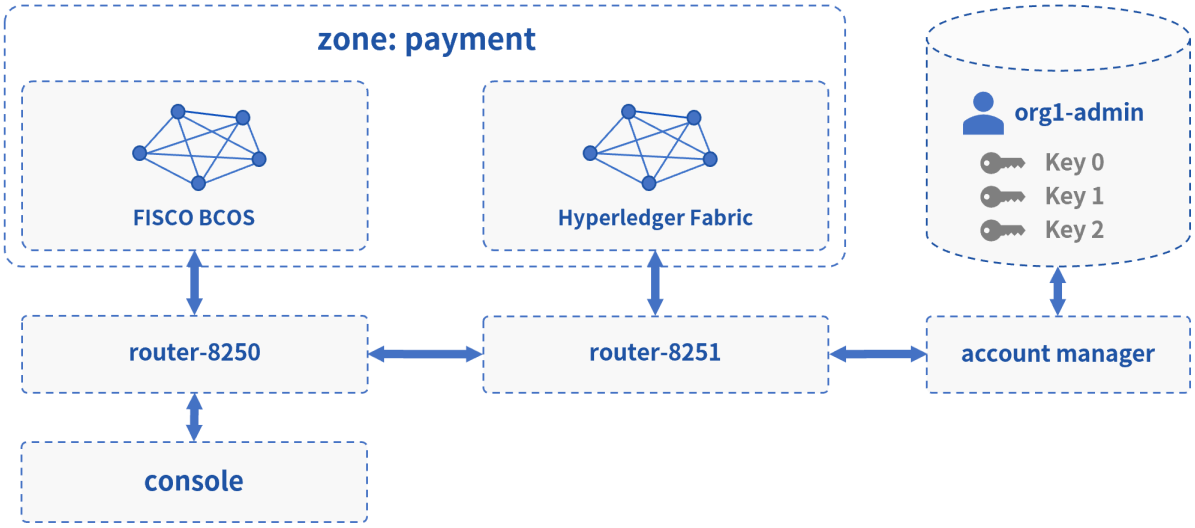
(续上页)

```
    -H                                  [Optional] DB ip
    -P                                  [Optional] DB port
    -u                                  [Optional] DB username
    -p                                  [Optional] DB password
    -h  call for help
e.g
    bash drop_account_database.sh -H 127.0.0.1 -P 3306 -u root -p 123456
    bash drop_account_database.sh
```

至此，恭喜你，快速体验完成！可进一步操作，体验其他Demo。

4.2 跨平台 FISCO BCOS & Fabric

此Demo搭建了一个WeCross跨链网络，连接FISCO BCOS和Hyperledger Fabric区块链。用户可通过WeCross控制台，对不同的链上资源进行操作。



4.2.1 网络部署

在已下载的demo目录下进行操作

```
cd ~/wecross-demo

#清理旧demo环境
bash clear.sh

# 运行部署脚本，输入数据库账号密码，第一次运行需耗时10-30分钟左右
bash build.sh # 若出错，可用 bash clear.sh 清理后重试。bash build.sh -h 可查看更多用法
```

重要:

- 若出现“command not found”，则说明缺少依赖，请参考 环境要求 安装相关依赖
- macOS用户若出现“无法打开”，“无法验证开发者”的情况，可参考 FAQ问题3 的方式解决
- 输入数据库IP时，若”127.0.0.1”无法成功，请尝试输入”localhost”
- 若出现其它问题，请参考常见问题说明 FAQ

部署成功后会输出Demo的网络架构，FISCO BCOS和Fabric通过各自的WeCross Router相连。（输入Y，回车，进入WeCross控制台）

```
[INFO] Success! WeCross demo network is running. Framework:

            FISCO BCOS                      Fabric
           (4node pbft)                (first-network)
          (HelloWorld.sol)                (sacc.go)
                 |                            |
                 |                            |
                 |                            |
          WeCross Router <----------> WeCross Router <----------> WeCross Account␣
↪Manager
      (127.0.0.1-8250-25500)      (127.0.0.1-8251-25501)             (127.0.0.
↪1:8340)
          /            \
         /              \
        /                \
 WeCross WebApp     WeCross Console

Start WeCross Console? [Y/n]
```

4.2.2 操作跨链资源

登录跨链账户

进入控制台，首先登录跨链账户。（Demo中已配置好一个账户：org1-admin，密码：123456）

```
[WeCross]> login org1-admin 123456
Result: success
=============================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

查看账户

用`listAccount`命令查看此跨链账户下，向不同类型的链发送交易的链账户。

```
[WeCross.org1-admin]> listAccount
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
chainAccounts: [
        BCOS2.0 Account:
        keyID    : 0
        type     : BCOS2.0
        address  : 0x07234bd37393a97268f91ee2b7c76cb3ec3601b0
        isDefault: true
        ----------
        Fabric1.4 Account:
        keyID    : 1
        type     : Fabric1.4
        MembershipID : Org1MSP
        isDefault: true
        ----------
        Fabric1.4 Account:
        keyID    : 2
```

(下页继续)

(续上页)

```
        type     : Fabric1.4
        MembershipID : Org2MSP
        isDefault: false
        ----------
]
```

查看资源

用listResources命令查看WeCross跨连网络中的所有资源。可看到已经部署了多个资源:

- payment.bcos.HelloWorld
  - 对应于FISCO BCOS链上的HelloWorld.sol合约
- payment.fabric.sacc
  - 对应于Fabric链上的sacc.go合约
- payment.xxxx.WeCrossHub
  - 每条链默认安装的Hub合约，用于接收链上合约发起的跨链调用，可参考《合约跨链》

```
[WeCross.org1-admin]> listResources
path: payment.bcos.HelloWorld, type: BCOS2.0, distance: 0
path: payment.fabric.WeCrossHub, type: Fabric1.4, distance: 1
path: payment.bcos.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.fabric.sacc, type: Fabric1.4, distance: 1
total: 4
```

操作资源：payment.bcos.HelloWorld

- 读资源
  - 命令：call path 接口名 [参数列表]
  - 示例：call payment.bcos.HelloWorld get

```
# 调用HelloWorld合约中的get接口
[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Hello, World!]
```

- 写资源
  - 命令：sendTransaction path 接口名 [参数列表]
  - 示例：sendTransaction payment.bcos.HelloWeCross set Tom

```
# 调用HelloWeCross合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.bcos.HelloWorld set Tom
Txhash  : 0x7043064899fa48b6c3138f545ecf0f8d6f823d45e0783406bc2afe489061c77c
BlockNum: 6
Result  : []     // 将Tom给set进去

[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Tom]    // 再次get，Tom已set
```

操作资源：payment.fabric.sacc

跨链资源是对各个不同链上资源的统一和抽象，因此操作的命令是保持一致的。

- 读资源

```
# 调用mycc合约中的query接口
[WeCross.org1-admin]> call payment.fabric.sacc get a
Result: [10] // 初次get，a的值为10
```

- 写资源

```
# 调用sacc合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.fabric.sacc set a 666
Txhash  : 85dd6c2c76b959cd5d6d5c60d1f4bc509df4c84a48ef7066f6c66bd59b67c6be
BlockNum: 8
Result  : [666]

[WeCross.org1-admin]> call payment.fabric.sacc get a
Result: [666] // 再次get，a的值变成666

# 退出WeCross控制台
[WeCross.org1-admin]> quit # 若想再次启动控制台，cd至WeCross-Console，执行start.sh即可
```

WeCross Console是基于WeCross Java SDK开发的跨链应用。搭建好跨链网络后，可基于WeCross Java SDK开发更多的跨链应用，通过统一的接口对各种链上的资源进行操作。

4.2.3 跨链转账

WeCross支持多种事务机制。此跨链转账的Demo是哈希时间锁合约(htlc)的举例。WeCross基于其htlc框架实现了**异构链之间资产的原子互换**，如下图所示：

场景描述如下

- BCOS 链
  - 转账金额：700
  - 发送者：org2-admin（bcos链账户：`0x4305196480b029bbecb071b4b68e95dfef36a7b7`）
  - 接收者：org1-admin（bcos链账户：`0x2b5ad5c4795c026514f8317c7a215e218dccd6cf`）
- Fabric 链
  - 转账金额：500
  - 发送者：org1-admin（Fabric链账户：`Admin@org1.example.com`）
  - 接收者：org2-admin（Fabric链账户：`User1@org2.example.com`）
- 哈希锁：`bea2dfec011d830a86d0fbeeb383e622b576bb2c15287b1a86aacdba0a387e11`
  - 解锁秘钥：`9dda9a5e175a919ee98ff0198927b0a765ef96cf917144b589bb8e510e04843c`
- 原子互换：FISCO BCOS链与Fabric链的转账同时发生

部署哈希时间锁合约

可通过脚本htlc_config.sh完成相关部署，并体验跨链转账。

```
# 请确保demo已搭建完毕，并在demo根目录执行，耗时5分钟左右
bash htlc_config.sh
```

跨链转账涉及两条链、两个用户（org1-admin，org2-admin），两条链上的资产转出者各自通过WeCross控制台创建一个转账提案，之后router会自动完成跨链转账。

创建转账提案

跨链转账需在跨链的两端都提交转账提案，提交后，router自动实现跨链转账。

- BCOS链资产转出者（org2-admin）提交提案
  - FISCO BCOS 链
    - 转账金额：700
    - 发送者：org2-admin（bcos链账户：0x4305196480b029bbecb071b4b68e95dfef36a7b7）
    - 接收者：org1-admin（bcos链账户：0x2b5ad5c4795c026514f8317c7a215e218dccd6cf）
  - Fabric 链
    - 转账金额：500
    - 发送者：org1-admin（Fabric链账户：Admin@org1.example.com）
    - 接收者：org2-admin（Fabric链账户：User1@org2.example.com）
  - 哈希锁：bea2dfec011d830a86d0fbeeb383e622b576bb2c15287b1a86aacdba0a387e11
    - 是否是发起方：是
    - 解锁秘钥：9dda9a5e175a919ee98ff0198927b0a765ef96cf917144b589bb8e510e04843c

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

# 转出者登录
[WeCross]> login org2-admin 123456

# 查看发送方余额
[WeCross.org2-admin]> call payment.bcos.htlc balanceOf␣
↪0x4305196480b029bbecb071b4b68e95dfef36a7b7
Result: [1000000000]

# 查看接收方余额
[WeCross.org2-admin]> call payment.bcos.htlc balanceOf␣
↪0x2b5ad5c4795c026514f8317c7a215e218dccd6cf
Result: [0]

# 创建转账提案
[WeCross.org2-admin]> newHTLCProposal payment.bcos.htlc␣
↪bea2dfec011d830a86d0fbeeb383e622b576bb2c15287b1a86aacdba0a387e11␣
↪9dda9a5e175a919ee98ff0198927b0a765ef96cf917144b589bb8e510e04843c true␣
↪0x4305196480b029bbecb071b4b68e95dfef36a7b7␣
↪0x2b5ad5c4795c026514f8317c7a215e218dccd6cf 700 2000010000 Admin@org1.example.com␣
↪User1@org2.example.com 500 2000000000
# 输出
Txhash: 0xc521d7dca1094d53773dc61faec1a95273d8ef2b681c5c50c94cde55263015c7
BlockNum: 11
Result: create a htlc proposal successfully
# 退出当前控制台
[WeCross.org2-admin]> quit
```

- Fabric链资产转出者（org1-admin）提交提案

- FISCO BCOS 链
  - 转账金额：700
  - 发送者：org2-admin（bcos链账户：0x4305196480b029bbecb071b4b68e95dfef36a7b7）
  - 接收者：org1-admin（bcos链账户：0x2b5ad5c4795c026514f8317c7a215e218dccd6cf）
- Fabric 链
  - 转账金额：500
  - 发送者：org1-admin（Fabric链账户：Admin@org1.example.com）
  - 接收者：org2-admin（Fabric链账户：User1@org2.example.com）
- 哈希锁：bea2dfec011d830a86d0fbeeb383e622b576bb2c15287b1a86aacdba0a387e11
  - 是否是发起方：否
  - 解锁秘钥：空（非发起方无秘钥）

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

# 转出者登录
[WeCross]> login org1-admin 123456

# 查看接收方余额
[WeCross.org1-admin]> call payment.fabric.htlc balanceOf User1@org2.example.com
Result: [0]

# 创建转账提案
[WeCross.org1-admin]> newHTLCProposal payment.fabric.htlc
↪bea2dfec011d830a86d0fbeeb383e622b576bb2c15287b1a86aacdba0a387e11 null false
↪0x4305196480b029bbecb071b4b68e95dfef36a7b7
↪0x2b5ad5c4795c026514f8317c7a215e218dccd6cf 700 2000010000 Admin@org1.example.com
↪User1@org2.example.com 500 2000000000
# 输出
Txhash: bbbe3c65a54f883151075129d4666a5ba40896bf75a3be09cfb96ef0354cc55f
BlockNum: 10
Result: create a htlc proposal successfully
# 退出当前控制台
[WeCross]> quit
```

跨链资产转移

当两个资产转出者都创建完提案后，router开始执行调度，并完成跨链转账。一次跨链转账存在5-25s的交易时延，主要取决于两条链以及机器的性能。

查询转账结果

在通过WeCross控制台查询资产是否到账。

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

# 登录
[WeCross]> login org1-admin 123456

# 查看FISCO BCOS接收方余额，收到700
[WeCross.org1-admin]> call payment.bcos.htlc balanceOf
↪0x2b5ad5c4795c026514f8317c7a215e218dccd6cf
Result: [700]

# 查看Fabric接收方余额，收到500
[WeCross.org1-admin]> call payment.fabric.htlc balanceOf User1@org2.example.com
```

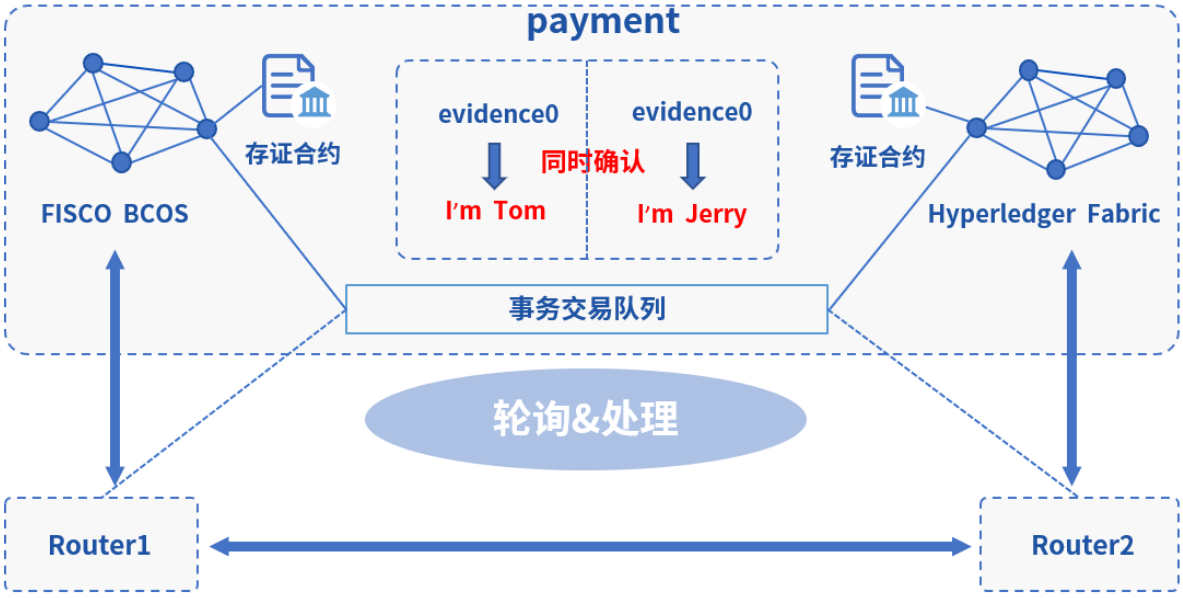
(下页继续)

(续上页)

```
Result: [500]

# 退出当前控制台
[WeCross.org1-admin]> quit
```

4.2.4 跨链存证

WeCross支持多种事务机制。此跨链转账的demo是两阶段事务机制（2PC）的举例。WeCross基于其2PC框架实现了**异构链之间证据的同时确认**，如下图所示：



部署跨链存证demo的合约

用一键脚本，在FISCO BCOS和Fabric上分别部署存证跨链的两个存证demo合约

```
cd ~/wecross-demo/
bash xa_config_evidence.sh
```

部署成功，输入Y进入控制台

```
[INFO] SUCCESS: 2PC evidence example has been deployed to FISCO BCOS and Fabric:

      FISCO BCOS                    Fabric
(payment.bcos.evidence)    (payment.fabric.evidence)
           |                           |
           |                           |
    WeCross Router <----------> WeCross Router
(127.0.0.1-8250-25500)       (127.0.0.1-8251-25501)
           |
           |
    WeCross Console

Start WeCross Console to try? [Y/n]
```

进入控制台，首先登录跨链账户。（Demo中已配置好一个账户：org1-admin，密码：123456）

```
[WeCross]> login org1-admin  123456
Result: success
================================================================================================
```

(下页继续)

(续上页)

```
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

发起跨链事务（start）

发起一个跨链事务，指定事务涉及的跨链资源，此处FISCO BCOS和Fabric上各一个资源

- payment.bcos.evidence
- payment.fabric.evidence

```
# 发起事务
[WeCross.org1-admin]> startTransaction payment.bcos.evidence payment.fabric.
↪evidence
Result: success!

# 查看当前存证evidence0的内容
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence0
Result: [] # 未存入，空

[WeCross.org1-admin]> call payment.fabric.evidence queryEvidence evidence0
Result: [] # 未存入，空
```

发送事务交易（exec）

事务开始后，通过execTransaction发送事务交易至向此事务涉及的资源，交易会被缓存入事务交易队列，在下一步commit时让所有交易同时被确认。

```
# 在FISCO BCOS链上进行存证，证据名：evidence0，内容：I'm Tom
[WeCross.org1-admin]> execTransaction payment.bcos.evidence newEvidence evidence0
↪"I'm Tom"
Txhash  : 0xa5bdd60240622438be1a99b8cca70b6a71aa3cfa444eefac405b22febf6ce4c9
BlockNum: 8
Result  : [true]

# 在Fabric链上进行存证，证据名：evidence0，内容：I'm Jerry
[WeCross.org1-admin]> execTransaction payment.fabric.evidence newEvidence␣
↪evidence0 "I'm Jerry"
Txhash  : c01809f3cf9154d09fb1a057c743a07dbb0ab2f0c544a9f9d644eb55fb384604
BlockNum: 10
Result  : [Success]

# 可发送更多的操作
```

确认跨链事务（commit）

在此事务下缓存了一些列的事务交易后，通过commitTransaction让所有事务交易同时被确认，结束此事务。

```
# 确认事务，事务结束
[WeCross.org1-admin]> commitTransaction payment.bcos.evidence payment.fabric.
↪evidence
Result: success!

# 查看当前存证内容，两条链都已完成存证
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence0
Result: [I'm Tom]

[WeCross.org1-admin]> call payment.fabric.evidence queryEvidence evidence0
Result: [I'm Jerry]
```

回滚跨链事务（rollback）

在commit前，若不想让事务发生，则用rollbackTransaction回滚此事务，所有缓存的事务交易被丢弃，链上数据回退到事务发起前状态，事务结束。（每个事务要么commit，要么rollback）

```
# 查看当前存证evidence1的内容
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence1
Result: [] # 未存入，空

[WeCross.org1-admin]> call payment.fabric.evidence queryEvidence evidence1
Result: [] # 未存入，空

# 发起另一个事务，事务号101
[WeCross.org1-admin]> startTransaction payment.bcos.evidence payment.fabric.
↪evidence
Result: success!

# 向FISCO BCOS链发送事务交易，设置evidence1，内容为I'm TomGG
[WeCross.org1-admin]> execTransaction payment.bcos.evidence newEvidence evidence1
↪"I'm TomGG"
Txhash  : 0x1f81a63a3986702a6c1b532c1d94074f2771a825c3bef1b9dffc73321b45cbe2
BlockNum: 11
Result  : [true]

# 向Fabric链发送事务交易，设置evidence1，内容为I'm JerryMM
[WeCross.org1-admin]> execTransaction payment.fabric.evidence newEvidence␣
↪evidence1 "I'm JerryMM"
Txhash  : fd18cbecaa3f2528e865ece6ea243c1f18c37e306ee6a59c56407476cce6cc37
BlockNum: 13
Result  : [Success]

# 查看当前事务状态下的数据
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence1
Result: [I'm TomGG]

# 查看当前事务状态下的数据
[WeCross.org1-admin]> call payment.fabric.evidence queryEvidence evidence1
Result: [I'm JerryMM]

# 尝试发送普通交易修改此事务下的资源，由于此资源处在事务状态中，被锁定，不可修改
[WeCross.org1-admin]> sendTransaction payment.bcos.evidence newEvidence evidence1
↪"I'm TomDD"
Error: code(2031), message(evidence is locked by unfinished xa transaction:␣
↪b04a651171644bf591f2047c2a0a79eb)

# 查看当前事务状态下的数据，未变化，普通交易修改失败，符合预期
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence1
Result: [I'm TomGG]

# 回滚操作！假设此事务不符合预期，需回滚至事务开始前状态，执行如下命令进行回滚，事务结束
[WeCross.org1-admin]> rollbackTransaction payment.bcos.evidence payment.fabric.
↪evidence
Result: success!

# 再次查看当前存证evidence1的内容
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence1
Result: [] # 已回滚至开始状态

[WeCross.org1-admin]> call payment.fabric.evidence queryEvidence evidence1
Result: [] # 已回滚至开始状态

# 退出当前控制台
[WeCross.org1-admin]> quit
```

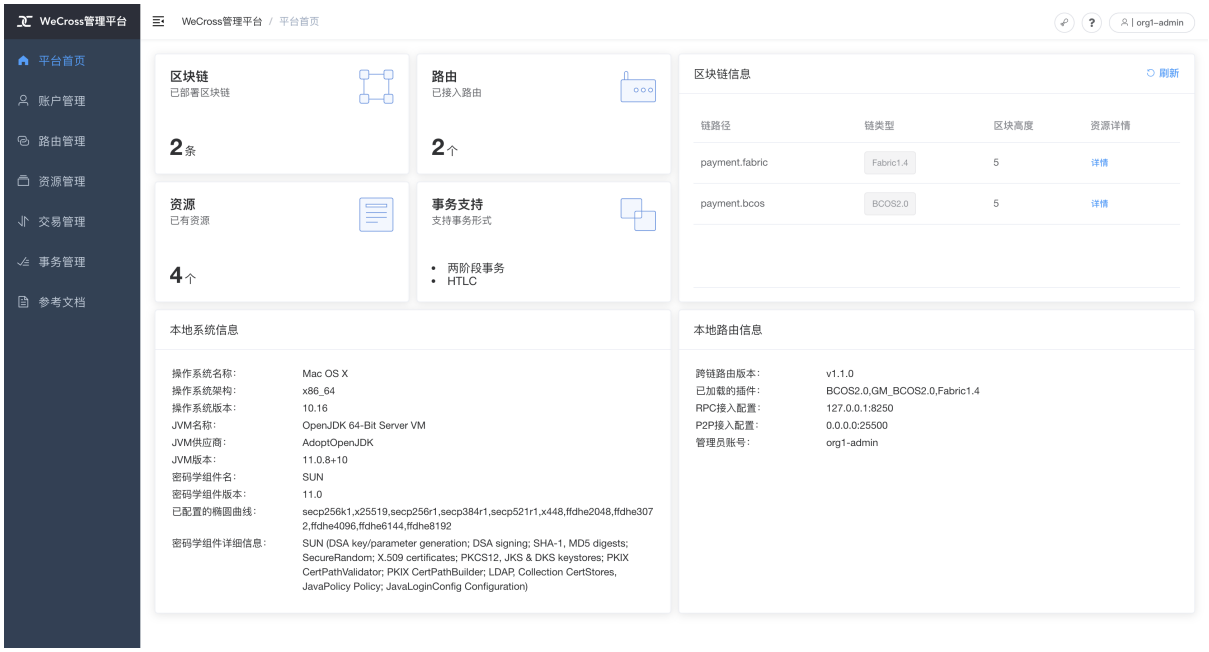
此demo基于2PC框架实现，用户可根据业务需要基于框架开发自己的跨链应用，实现链间的原子操作。

4.2.5 访问网页管理平台

浏览器访问router-8250的网页管理平台

```
http://localhost:8250/s/index.html#/login
```

用demo已配置账户进行登录：org1-admin，密码：123456



管理台中包含如下内容，点击链接进入相关操作指导。

- 登录/注册
- 平台首页
- 账户管理
- 路由管理
- 资源管理
- 交易管理
- 事务管理

注解:

- 若需要远程访问，请修改router的主配置（如：~/demo/routers-payment/127.0.0.1-8250-25500/conf/wecross.toml），将 [rpc] 标签下的 address 修改为所需ip（如：0.0.0.0）。保存后，重启router即可。

4.2.6 清理 Demo

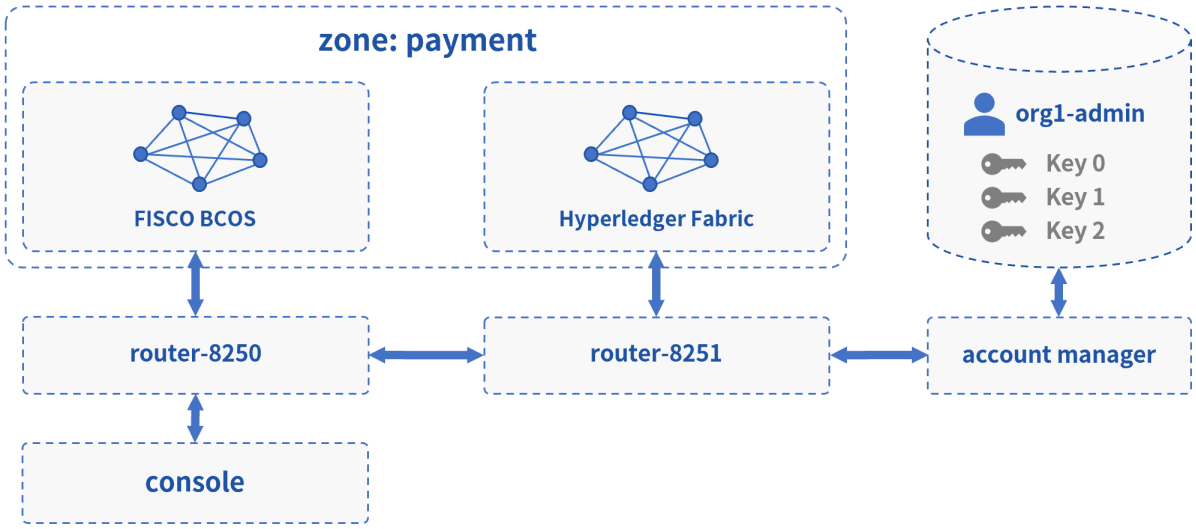
为了不影响其它章节的体验，可将搭建的Demo清理掉。

```
cd ~/wecross-demo/
bash clear.sh

# 可使用脚本drop_account_database.sh删除MySQL中demo使用的数据库
bash drop_account_database.sh

# Tips: 可选用配置MySQL参数，进行无交互式部署，详情请参考下述脚本输出
bash drop_account_database.sh -h

Drop wecross-account-manager database named wecross_account_manager.
Usage:
    -d                                [Optional] Use default db configuration: -H␣
↪127.0.0.1 -P 3306 -u root -p 123456
    -H                                [Optional] DB ip
    -P                                [Optional] DB port
    -u                                [Optional] DB username
    -p                                [Optional] DB password
    -h  call for help
e.g
    bash drop_account_database.sh -H 127.0.0.1 -P 3306 -u root -p 123456
    bash drop_account_database.sh
```

至此，恭喜你，快速体验完成！可进入手动组网章节深入了解更多细节。

4.3 跨平台 FISCO BCOS & Fabric2

此Demo搭建了一个WeCross跨链网络，连接FISCO BCOS和Hyperledger Fabric2区块链。用户可通过WeCross控制台，对不同的链上资源进行操作。



4.3.1 网络部署

在已下载的demo目录下进行操作

```
cd ~/wecross-demo

#清理旧demo环境
bash clear.sh
```

(下页继续)

(续上页)

```
# 运行部署脚本，输入数据库账号密码，第一次运行需耗时10-30分钟左右
bash build_cross_fabric2.sh # 若出错，可用 bash clear.sh 清理后重试。bash build_cross_
↪fabric2.sh -h 可查看更多用法
```

重要:

- 若出现“command not found”，则说明缺少依赖，请参考 环境要求 安装相关依赖
- macOS用户若出现“无法打开”，“无法验证开发者”的情况，可参考 FAQ问题3 的方式解决
- 输入数据库IP时，若”127.0.0.1”无法成功，请尝试输入”localhost”
- 若出现其它问题，请参考常见问题说明 FAQ

部署成功后会输出Demo的网络架构，FISCO BCOS和Fabric2通过各自的WeCross Router相连。（输入Y，回车，进入WeCross控制台）

```
[INFO] Success! WeCross demo network is running. Framework:

            FISCO BCOS                  Fabric2
           (4node pbft)              (test-network)
          (HelloWorld.sol)              (sacc.go)
                 |                          |
                 |                          |
                 |                          |
          WeCross Router <----------> WeCross Router <----------> WeCross Account␣
↪Manager
      (127.0.0.1-8250-25500)      (127.0.0.1-8251-25501)             (127.0.0.
↪1:8340)
          /            \
         /              \
        /                \
 WeCross WebApp     WeCross Console

Start WeCross Console? [Y/n]
```

4.3.2 操作跨链资源

登录跨链账户

进入控制台，首先登录跨链账户。（Demo中已配置好一个账户：org1-admin，密码：123456）

```
[WeCross]> login org1-admin 123456
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

查看账户

用listAccount命令查看此跨链账户下，向不同类型的链发送交易的链账户。

```
[WeCross.org1-admin]> listAccount
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

(下页继续)

(续上页)

```
chainAccounts: [
        BCOS2.0 Account:
        keyID    : 0
        type     : BCOS2.0
        address  : 0x347271c6b24cb9ee6c7c18fd258f83a03bd6d3df
        isDefault: true
        ----------
        Fabric2.0 Account:
        keyID    : 2
        type     : Fabric2.0
        MembershipID : Org2MSP
        isDefault: true
        ----------
        Fabric2.0 Account:
        keyID    : 1
        type     : Fabric2.0
        MembershipID : Org1MSP
        isDefault: false
        ----------
]
```

查看资源

用`listResources`命令查看WeCross跨连网络中的所有资源。可看到已经部署了多个资源:

- `payment.bcos.HelloWorld`
  - 对应于FISCO BCOS链上的HelloWorld.sol合约
- `payment.fabric2.sacc`
  - 对应于Fabric链上的sacc.go合约
- `payment.xxxx.WeCrossHub`
  - 每条链默认安装的Hub合约，用于接收链上合约发起的跨链调用，可参考《合约跨链》

```
[WeCross.org1-admin]> listResources
path: payment.bcos.HelloWorld, type: BCOS2.0, distance: 0
path: payment.bcos.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.fabric2.WeCrossHub, type: Fabric2.0, distance: 1
path: payment.fabric2.sacc, type: Fabric2.0, distance: 1
total: 4
```

操作资源：payment.bcos.HelloWorld

- 读资源
  - 命令：`call path 接口名 [参数列表]`
  - 示例：`call payment.bcos.HelloWorld get`

```
# 调用HelloWorld合约中的get接口
[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Hello, World!]
```

- 写资源
  - 命令：`sendTransaction path 接口名 [参数列表]`
  - 示例：`sendTransaction payment.bcos.HelloWeCross set Tom`

```
# 调用HelloWeCross合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.bcos.HelloWorld set Tom
Txhash  : 0x7043064899fa48b6c3138f545ecf0f8d6f823d45e0783406bc2afe489061c77c
```

(下页继续)

(续上页)

```
BlockNum: 6
Result  : []      // 将Tom给set进去

[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Tom]     // 再次get，Tom已set
```

操作资源：payment.fabric2.sacc

跨链资源是对各个不同链上资源的统一和抽象，因此操作的命令是保持一致的。

- 写资源

```
# 调用sacc合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.fabric2.sacc set a 666
Txhash  : 85dd6c2c76b959cd5d6d5c60d1f4bc509df4c84a48ef7066f6c66bd59b67c6be
BlockNum: 14
Result  : [666]
```

- 读资源

```
[WeCross.org1-admin]> call payment.fabric2.sacc get a
Result: [666] // get，a的值已是666

# 退出WeCross控制台
[WeCross.org1-admin]> quit # 若想再次启动控制台，cd至WeCross-Console，执行start.sh即可
```

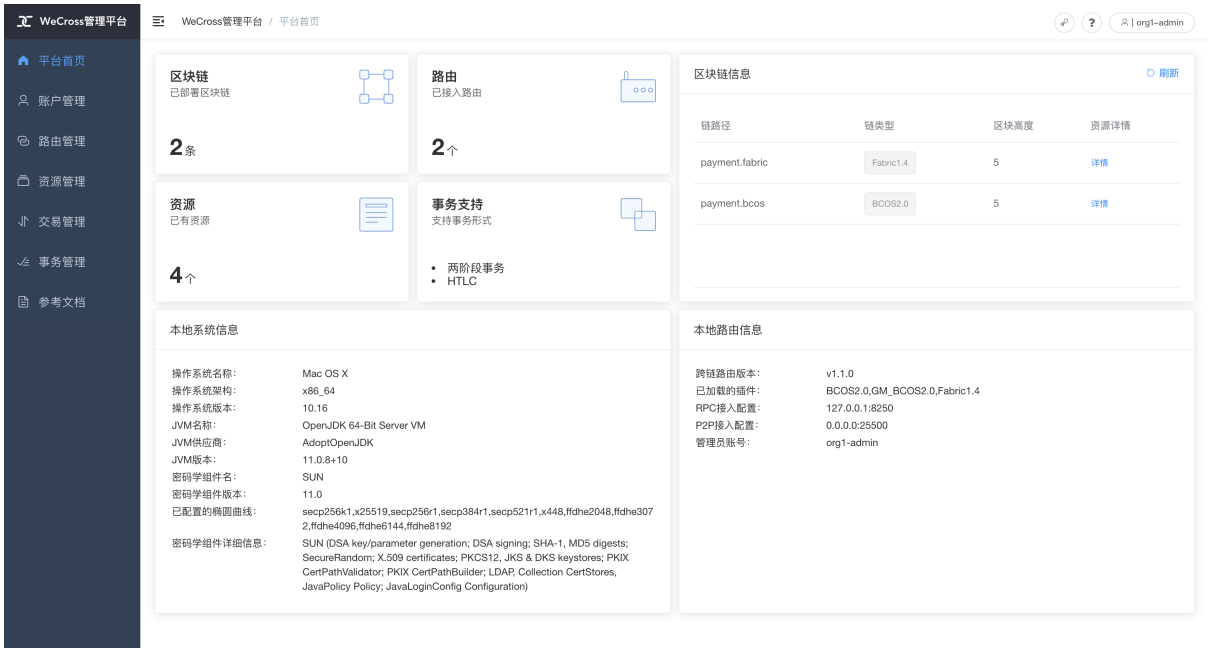
WeCross Console是基于WeCross Java SDK开发的跨链应用。搭建好跨链网络后，可基于WeCross Java SDK开发更多的跨链应用，通过统一的接口对各种链上的资源进行操作。

4.3.3 访问网页管理平台

浏览器访问router-8250的网页管理平台

```
http://localhost:8250/s/index.html#/login
```

用demo已配置账户进行登录：org1-admin，密码：123456



管理台中包含如下内容，点击链接进入相关操作指导。

- 登录/注册
- 平台首页
- 账户管理
- 路由管理
- 资源管理
- 交易管理
- 事务管理

注解:

- 若需要远程访问，请修改router的主配置（如：~/demo/routers-payment/127.0.0.1-8250-25500/conf/wecross.toml），将 `[rpc]` 标签下的 `address` 修改为所需ip（如：0.0.0.0）。保存后，重启router即可。

4.3.4 清理 Demo

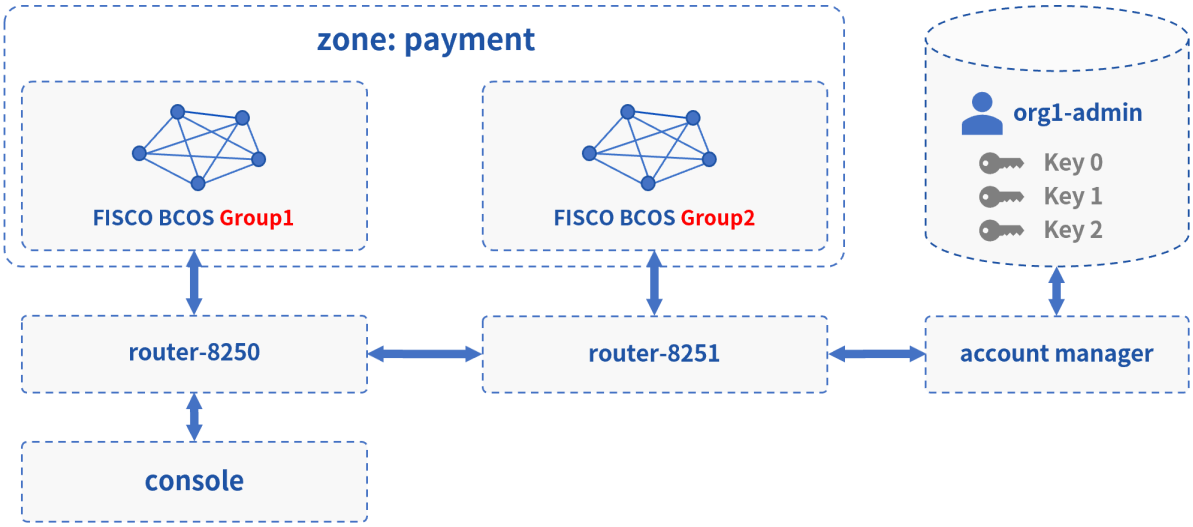
为了不影响其它章节的体验，可将搭建的Demo清理掉。

```
cd ~/wecross-demo/
bash clear.sh

# 可使用脚本drop_account_database.sh删除MySQL中demo使用的数据库
bash drop_account_database.sh

# Tips: 可选用配置MySQL参数，进行无交互式部署，详情请参考下述脚本输出
bash drop_account_database.sh -h

Drop wecross-account-manager database named wecross_account_manager.
Usage:
    -d                                [Optional] Use default db configuration: -H␣
↪127.0.0.1 -P 3306 -u root -p 123456
    -H                                [Optional] DB ip
    -P                                [Optional] DB port
    -u                                [Optional] DB username
    -p                                [Optional] DB password
    -h  call for help
e.g
    bash drop_account_database.sh -H 127.0.0.1 -P 3306 -u root -p 123456
    bash drop_account_database.sh
```

至此，恭喜你，快速体验完成！可进入手动组网章节深入了解更多细节。

4.4 跨多群组

此Demo搭建了一个WeCross跨链网络，连接FISCO BCOS中的两个群组。用户可通过WeCross控制台，对不同群组的资源进行操作。实际情况下，WeCross接入链的场景不受限制，用户可配置接入多条各种类型的区块链。



4.4.1 网络部署

在已下载的demo目录下进行操作

```
cd ~/wecross-demo

#清理旧demo环境
bash clear.sh

# 运行部署脚本，输入数据库账号密码，第一次运行需耗时10-30分钟左右
bash build_cross_groups.sh # 若出错，可用 bash clear.sh 清理后重试。bash build.sh -h 可查看更多用法
```

重要:

- 若出现“command not found”， 则说明缺少依赖，请参考 环境要求 安装相关依赖
- macOS用户若出现“无法打开”，“无法验证开发者”的情况，可参考 FAQ问题3 的方式解决
- 输入数据库IP时，若”127.0.0.1”无法成功，请尝试输入”localhost”
- 若出现其它问题，请参考常见问题说明 FAQ

部署成功后会输出Demo的网络架构，两个群组通过各自的WeCross Router相连。（输入Y，回车，进入WeCross控制台）

```
[INFO] Success! WeCross demo network is running. Framework:

                          FISCO BCOS
              Group 1                     Group 2
         (HelloWorldGroup1)          (HelloWorldGroup2)
                 |                           |
                 |                           |
                 |                           |
          WeCross Router <----------> WeCross Router <----------> WeCross Account␣
↪Manager
      (127.0.0.1-8250-25500)       (127.0.0.1-8251-25501)              (127.0.0.
↪1:8340)
           /             \
          /               \
         /                 \
```

(下页继续)

(续上页)

```
 WeCross WebApp     WeCross Console

Start console? [Y/n]
```

4.4.2 操作跨链资源

登录跨链账户

进入控制台，首先登录跨链账户。（Demo中已配置好一个账户：org1-admin，密码：123456）

```
[WeCross]> login org1-admin 123456
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

查看账户

用`listAccount`命令查看此跨链账户下，向不同类型的链发送交易的链账户。

```
[WeCross.org1-admin]> listAccount
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
chainAccounts: [
        BCOS2.0 Account:
        keyID    : 0
        type     : BCOS2.0
        address  : 0x64a0494c16c4bd6d7e3bb0cf0b9bd6949b65500d
        isDefault: true
        ----------
]
```

查看资源

用`listResources`命令查看WeCross跨连网络中的所有资源。可看到有多个资源：

- `payment.group1.HelloWorldGroup1`
  - 对应于Group1上的HelloWorld.sol合约
- `payment.group2.HelloWorldGroup2`
  - 对应于Group2上的HelloWorld.sol合约
- `payment.xxxx.WeCrossHub`
  - 每条链默认安装的Hub合约，用于接收链上合约发起的跨链调用，可参考《合约跨链》

```
[WeCross.org1-admin]> listResources
path: payment.group2.WeCrossHub, type: BCOS2.0, distance: 1
path: payment.group1.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.group2.HelloWorldGroup2, type: BCOS2.0, distance: 1
path: payment.group1.HelloWorldGroup1, type: BCOS2.0, distance: 0
total: 4
```

操作资源：payment.group1.HelloWorldGroup1

- 读资源
  - 命令：`call path 接口名 [参数列表]`

  - 示例：call payment.group1.HelloWorldGroup1 get

```
# 调用Group1 HelloWorld合约中的get接口
[WeCross.org1-admin]> call payment.group1.HelloWorldGroup1 get
Result: [Hello, World!] // 初次get，值为Hello World!
```

- 写资源
  - 命令：sendTransaction path 接口名 [参数列表]
  - 示例：sendTransaction payment.group1.HelloWorldGroup1 set Tom

```
# 调用Group1 HelloWeCross合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.group1.HelloWorldGroup1 set Tom
Txhash  : 0xd510203ce12b0ca8c9bceca50dfd4702f64efc2e147aa334a6cfff7988ada686
BlockNum: 6
Result  : []      // 将Tom给set进去

[WeCross.org1-admin]> call payment.group1.HelloWorldGroup1 get
Result: [Tom]     // 再次get，Tom已set
```

操作资源：payment.group2.HelloWorldGroup2

跨链资源是对各个不同链上资源的统一和抽象，因此操作的命令是保持一致的。

- 读资源

```
# 调用Group2 HelloWorld合约中的get接口
[WeCross.org1-admin]> call payment.group2.HelloWorldGroup2 get
Result: [Hello, World!] // 初次get，值为Hello World!
```

- 写资源

```
# 调用Group2 HelloWeCross合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.group2.HelloWorldGroup2 set Jerry
Txhash  : 0xf5a52604750dd4a8b10df69f238815378db9444fef365e102d1a5a43603f18d0
BlockNum: 6
Result  : []      // 将Jerry给set进去

[WeCross.org1-admin]> call payment.group2.HelloWorldGroup2 get
Result: [Jerry]     // 再次get，Jerry已set

# 检查Group1资源，不会因为Group2的资源被修改而改变
[WeCross.org1-admin]> call payment.group1.HelloWorldGroup1 get
Result: [Tom]

# 退出WeCross控制台
[WeCross.org1-admin]> quit # 若想再次启动控制台，cd至WeCross-Console，执行start.sh即可
```

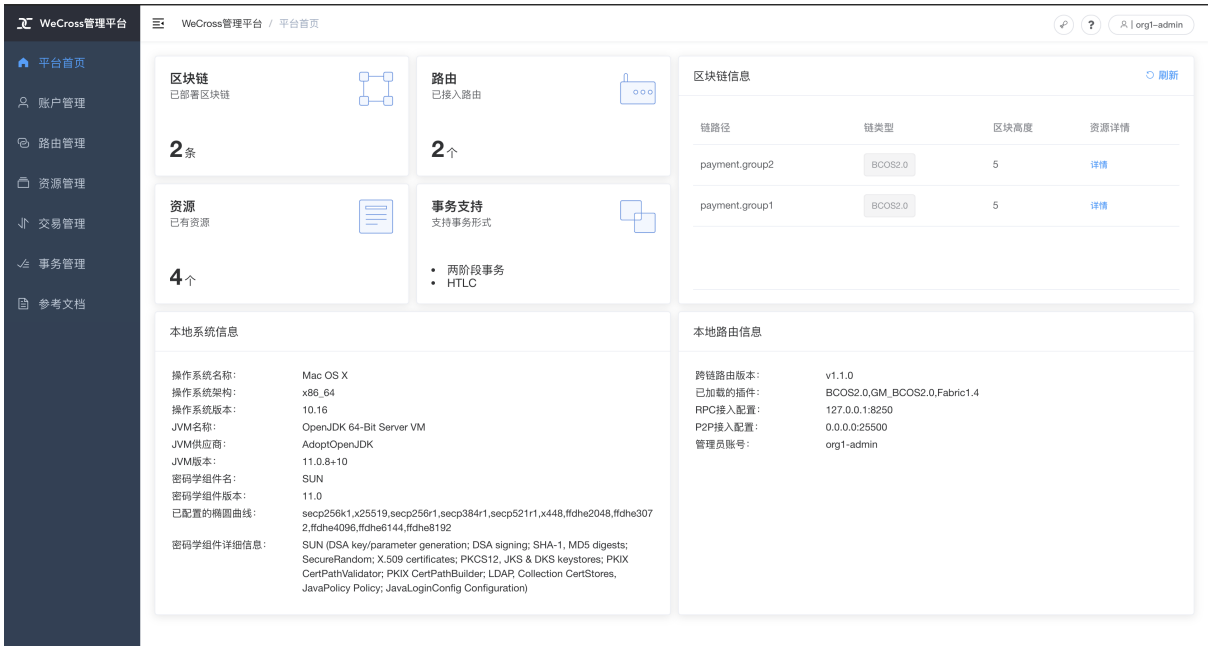
WeCross Console是基于WeCross Java SDK开发的跨链应用。搭建好跨链网络后，可基于WeCross Java SDK开发更多的跨链应用，通过统一的接口对各种链上的资源进行操作。

4.4.3 访问网页管理平台

浏览器访问router-8250的网页管理平台

```
http://localhost:8250/s/index.html#/login
```

用demo已配置账户进行登录：org1-admin，密码：123456



管理台中包含如下内容，点击链接进入相关操作指导。

- 登录/注册
- 平台首页
- 账户管理
- 路由管理
- 资源管理
- 交易管理
- 事务管理

注解:

- 若需要远程访问，请修改router的主配置（如：~/demo/routers-payment/127.0.0.1-8250-25500/conf/wecross.toml），将 `[rpc]` 标签下的 `address` 修改为所需ip（如：0.0.0.0）。保存后，重启router即可。

4.4.4 清理 Demo

为了不影响其它章节的体验，可将搭建的Demo清理掉。

```
cd ~/wecross-demo/
bash clear.sh

# 可使用脚本drop_account_database.sh删除MySQL中demo使用的数据库
bash drop_account_database.sh

# Tips: 可选用配置MySQL参数，进行无交互式部署，详情请参考下述脚本输出
bash drop_account_database.sh -h

Drop wecross-account-manager database named wecross_account_manager.
Usage:
    -d                                [Optional] Use default db configuration: -H␣
↪127.0.0.1 -P 3306 -u root -p 123456
```

(下页继续)

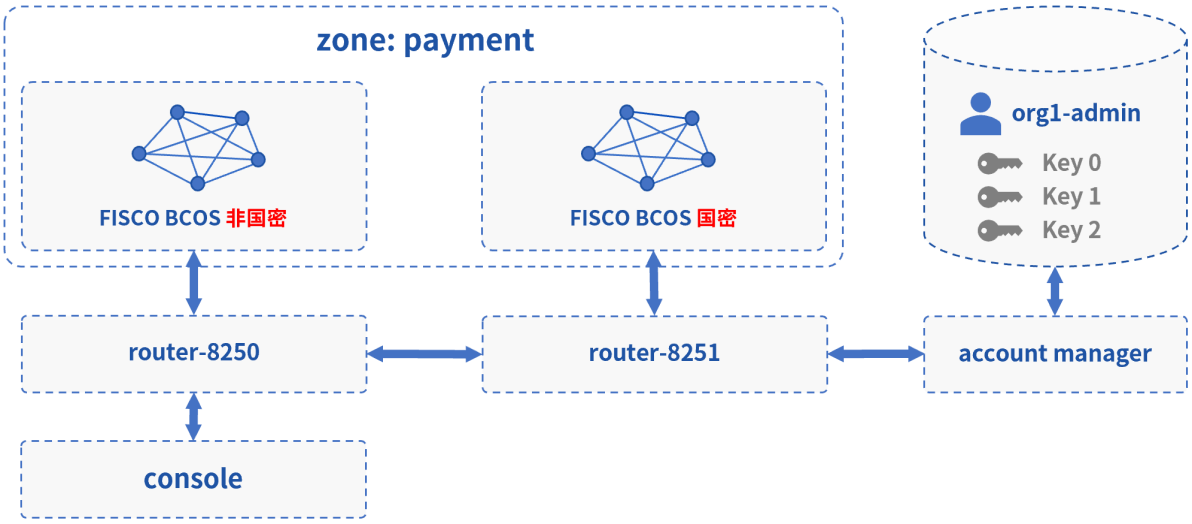
(续上页)

```
    -H                                  [Optional] DB ip
    -P                                  [Optional] DB port
    -u                                  [Optional] DB username
    -p                                  [Optional] DB password
    -h  call for help
e.g
    bash drop_account_database.sh -H 127.0.0.1 -P 3306 -u root -p 123456
    bash drop_account_database.sh
```

至此，恭喜你，快速体验完成！可进入手动组网章节深入了解更多细节。

4.5 跨国密、非国密

此Demo搭建了一个WeCross跨链网络，连接FISCO BCOS的国密区块链和非国密区块链。用户可通过WeCross控制台，对不同链的链上资源进行操作。实际情况下，WeCross接入链的场景不受限制，用户可配置接入多条各种类型的区块链。



4.5.1 网络部署

在已下载的demo目录下进行操作

```
cd ~/wecross-demo

#清理旧demo环境
bash clear.sh

# 运行部署脚本，输入数据库账号密码，第一次运行需耗时10-30分钟左右
bash build_cross_gm.sh # 若出错，可用 bash clear.sh 清理后重试。bash build.sh -h 可查看更多用法
```

重要:

- 若出现“command not found”， 则说明缺少依赖，请参考 环境要求 安装相关依赖
- macOS用户若出现“无法打开”，“无法验证开发者”的情况，可参考 FAQ问题3 的方式解决
- 输入数据库IP时，若”127.0.0.1”无法成功，请尝试输入”localhost”
- 若出现其它问题，请参考常见问题说明 FAQ

部署成功后会输出Demo的网络架构，FISCO BCOS的国密和非国密链通过Cross Router相连。（输入Y，回车，进入WeCross控制台）

```
[INFO] Success! WeCross demo network is running. Framework:

                          FISCO BCOS
              Normal                      Guomi
            (HelloWorld)               (HelloWorld)
                 |                          |
                 |                          |
                 |                          |
          WeCross Router <----------> WeCross Router <----------> WeCross Account␣
↪Manager
      (127.0.0.1-8250-25500)      (127.0.0.1-8251-25501)             (127.0.0.
↪1:8340)
          /            \
         /              \
        /                \
 WeCross WebApp     WeCross Console

Start console? [Y/n]
```

4.5.2 操作跨链资源

登录跨链账户

进入控制台，首先登录跨链账户。（Demo中已配置好一个账户：org1-admin，密码：123456）

```
[WeCross]> login org1-admin 123456
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

查看账户

用`listAccount`命令查看此跨链账户下，向不同类型的链发送交易的链账户。

```
[WeCross.org1-admin]> listAccount
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
chainAccounts: [
        BCOS2.0 Account:
        keyID    : 0
        type     : BCOS2.0
        address  : 0x8120dde72685b1ac07d0c20d0069696e42ace1db
        isDefault: true
        ----------
        GM_BCOS2.0 Account:
        keyID    : 1
        type     : GM_BCOS2.0
        address  : 0x0e5c45e9431578fd8ac114781c6d61a2ad4946e5
        isDefault: true
        ----------
]
```

查看资源

用listResources命令查看WeCross跨连网络中的所有资源。可看到有多个资源：

- payment.bcos.HelloWorld
  - 对应于**非国密FISCO BCOS链**上的HelloWorld.sol合约
- payment.bcos_gm.HelloWorld
  - 对应于**国密FISCO BCOS链**上的HelloWorld.sol合约
- payment.xxxx.WeCrossHub
  - 每条链默认安装的Hub合约，用于接收链上合约发起的跨链调用，可参考《合约跨链》

```
[WeCross.org1-admin]> listResources
path: payment.bcos.HelloWorld, type: BCOS2.0, distance: 0
path: payment.bcos_gm.HelloWorld, type: GM_BCOS2.0, distance: 1
path: payment.bcos.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.bcos_gm.WeCrossHub, type: GM_BCOS2.0, distance: 1
total: 4
```

操作资源：payment.bcos.HelloWorld

- 读资源
  - 命令：call path 接口名 [参数列表]
  - 示例：call payment.bcos.HelloWorld get

```
# 调用非国密链上HelloWorld合约中的get接口
[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Hello, World!] // 初次get，值为Hello World!
```

- 写资源
  - 命令：sendTransaction path 接口名 [参数列表]
  - 示例：sendTransaction payment.bcos.HelloWorld set Tom

```
# 调用非国密链上HelloWeCross合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.bcos.HelloWorld set Tom
Txhash  : 0x4b2d6a5f2365318b6574a02fba2df1f4cdba8e581513c8588033f7b793afc061
BlockNum: 6
Result  : []      // 将Tom给set进去

[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Tom]     // 再次get，Tom已set
```

操作资源：payment.bcos_gm.HelloWorld

跨链资源是对各个不同链上资源的统一和抽象，因此操作的命令是保持一致的。

- 读资源

```
# 调用国密链上HelloWorld合约中的get接口
[WeCross.org1-admin]> call payment.bcos_gm.HelloWorld get
Result: [Hello, World!] // 初次get，值为Hello World!
```

- 写资源

```
# 调用国密链上HelloWeCross合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.bcos_gm.HelloWorld set Jerry
Txhash  : 0x95d54faaca7499b5cb19e7af0dafc4965676c3b136809e5957c50d8ca07ed408
BlockNum: 6
Result  : []      // 将Jerry给set进去
```

(下页继续)

(续上页)

```
[WeCross.org1-admin]> call payment.bcos_gm.HelloWorld get
Result: [Jerry]    // 再次get，Jerry已set

# 检查非国密链上的资源，不会因为国密链上的资源被修改而改变
[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Tom]

# 退出WeCross控制台
[WeCross.org1-admin]> quit  # 若想再次启动控制台，cd至WeCross-Console，执行start.sh即可
```

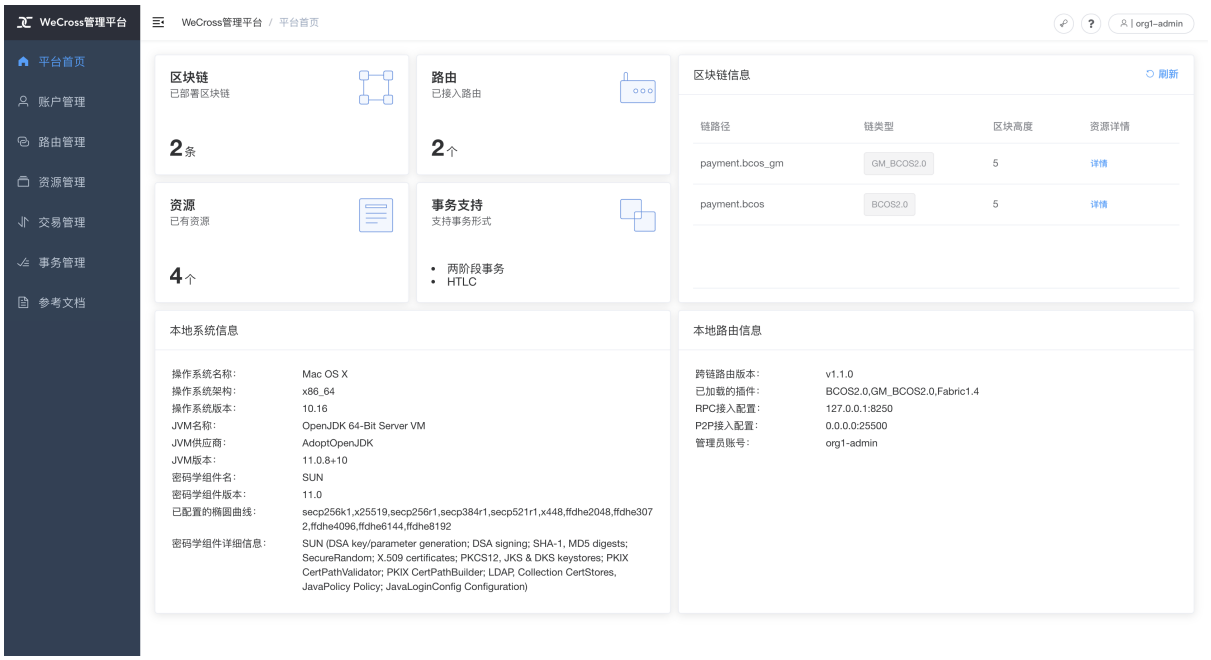
WeCross Console是基于WeCross Java SDK开发的跨链应用。搭建好跨链网络后，可基于WeCross Java SDK开发更多的跨链应用，通过统一的接口对各种链上的资源进行操作。

4.5.3 访问网页管理平台

浏览器访问router-8250的网页管理平台

```
http://localhost:8250/s/index.html#/login
```

用demo已配置账户进行登录：org1-admin，密码：123456



管理台中包含如下内容，点击链接进入相关操作指导。

- 登录/注册
- 平台首页
- 账户管理
- 路由管理
- 资源管理
- 交易管理
- 事务管理

注解:

- 若需要远程访问，请修改router的主配置（如：~/demo/routers-payment/127.0.0.1-8250-25500/conf/wecross.toml），将 `[rpc]` 标签下的 `address` 修改为所需ip（如：0.0.0.0）。保存后，重启router即可。

4.5.4 清理 Demo

为了不影响其它章节的体验，可将搭建的Demo清理掉。

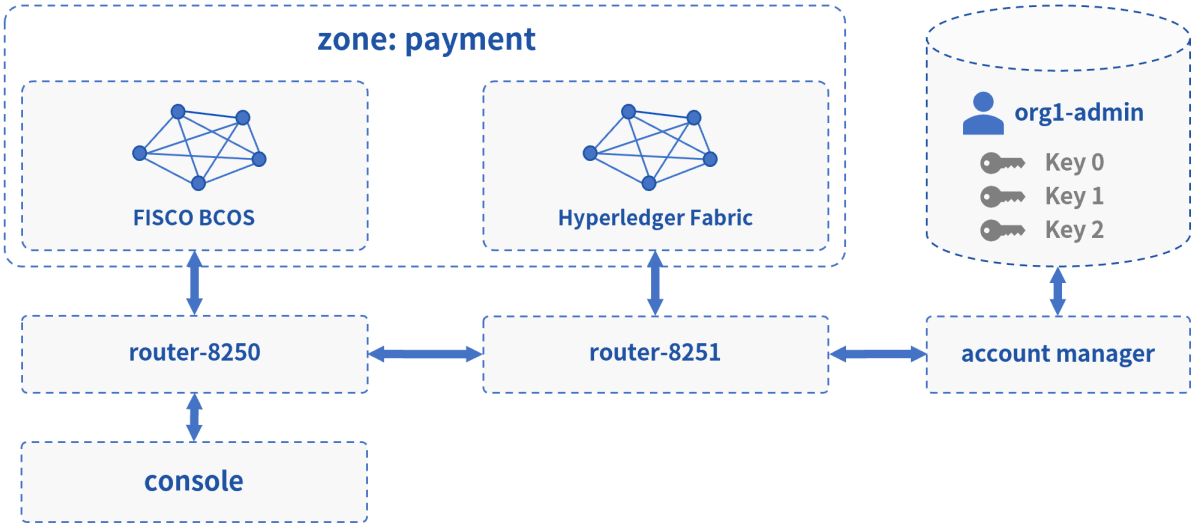
```
cd ~/wecross-demo/
bash clear.sh

# 可使用脚本drop_account_database.sh删除MySQL中demo使用的数据库
bash drop_account_database.sh

# Tips: 可选用配置MySQl参数，进行无交互式部署，详情请参考下述脚本输出
bash drop_account_database.sh -h

Drop wecross-account-manager database named wecross_account_manager.
Usage:
    -d                                [Optional] Use default db configuration: -H␣
↪127.0.0.1 -P 3306 -u root -p 123456
    -H                                [Optional] DB ip
    -P                                [Optional] DB port
    -u                                [Optional] DB username
    -p                                [Optional] DB password
    -h  call for help
e.g
    bash drop_account_database.sh -H 127.0.0.1 -P 3306 -u root -p 123456
    bash drop_account_database.sh
```

至此，恭喜你，快速体验完成！可进入手动组网章节深入了解更多细节。

CHAPTER 5

手动组网

你可以基于已有（或新部署）的区块链环境，搭建一个与Demo相似的跨链网络。



操作步骤分为以下4项：

- **基础组件部署**：部署WeCross基础组件，包括跨链路由、账户服务、控制台、网页管理台
- **区块链接入与账户配置**：将区块链接入WeCross跨链网络，并配置跨链账户
- **资源部署与操作**：基于搭建好的WeCross环境部署和操作跨链资源
- **区块头验证配置**：配置区块头验证参数，完善跨链交易的验证逻辑

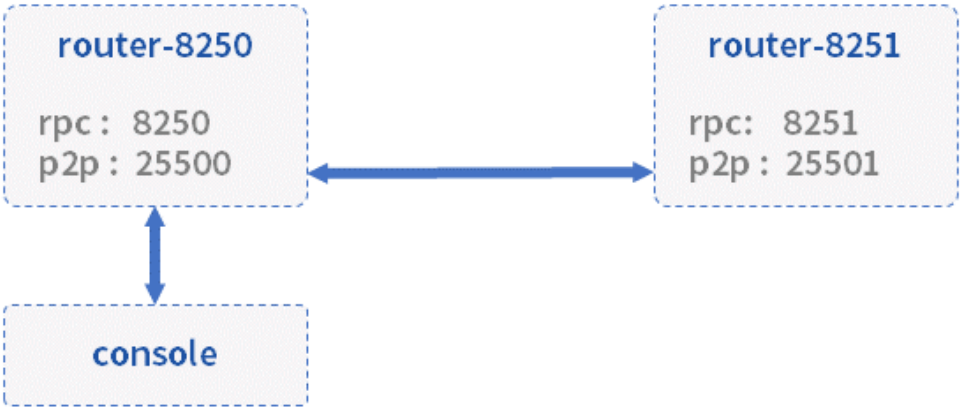
手动组网教程以`~/wecross-networks/`目录下为例进行：

```
# 若已搭建WeCross Demo，请先关闭所有服务

# 创建手动组网的操作目录
mkdir -p ~/wecross-networks && cd ~/wecross-networks
```

5.1 基础组件部署

本章节指导完成以下组件的部署，完成WeCross基础网络的搭建。

- **跨链路由（router）**：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
- **账户服务（account manager）**：为跨链系统提供账户管理
- **跨链控制台（console）**：查询和发送交易的操作终端



进入操作目录：

```
cd ~/wecross-networks
```

5.1.1 下载WeCross

下载WeCross，用WeCross中的工具生成跨链路由，并启动跨链路由。

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~/wecross-networks/WeCross/中。

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

# 若出现长时间下载WeCross包失败，请尝试以下命令重新下载:
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪wecross.sh)
```

5.1.2 部署跨链路由

本例将构建两个跨链路由。首先创建一个ipfile配置文件，将需要构建的两个跨链路由信息（ip:rpc_port:p2p_port）按行分隔，保存到文件中。

注：请确保机器的8250，8251, 25500，25501端口没有被占用。

```
cd ~/wecross-networks
vim ipfile

# 在文件中键入以下内容
127.0.0.1:8250:25500
127.0.0.1:8251:25501
```

生成好ipfile文件后，使用脚本build_wecross.sh生成两个跨链路由。

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

# 成功输出如下信息
[INFO] Create routers-payment/127.0.0.1-8250-25500 successfully
[INFO] Create routers-payment/127.0.0.1-8251-25501 successfully
[INFO] All completed. WeCross routers are generated in: routers-payment/
```

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。
- -f 指定需要生成的WeCross跨链路由的列表，包括ip地址，rpc端口，p2p端口，生成后的router已完成互联配置。

在routers-payment目录下生成了两个跨链路由。

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

生成的跨链路由目录内容如下，以127.0.0.1-8250-25500为例。

```
# 已屏蔽lib和pages目录，该目录存放所有依赖的jar包
tree routers-payment/127.0.0.1-8250-25500/
routers-payment/127.0.0.1-8250-25500/
├── add_chain.sh       # 区块链配置文件创建脚本
├── apps
│   └── WeCross.jar    # WeCross路由jar包
├── build_wecross.sh
├── conf               # 配置文件目录
│   ├── accounts       # 账户配置目录
│   ├── application.properties
│   ├── chains         # 区块链配置目录，要接入不同的链，在此目录下进行配置
│   ├── log4j2.xml
│   ├── ca.crt         # 根证书
│   ├── ssl.crt        # 跨链路由证书
│   ├── ssl.key        # 跨链路由私钥
│   ├── node.nodeid    # 跨链路由nodeid
│   └── wecross.toml   # WeCross Router主配置文件
├── create_cert.sh     # 证书生成脚本
├── download_wecross.sh
├── pages              # 网页管理平台页面文件
├── plugin             # 插件目录，接入相应类型链的插件
│   ├── bcos-stub-gm.jar
│   ├── bcos-stub.jar
│   └── fabric-stub.jar
├── start.sh           # 启动脚本
└── stop.sh            # 停止脚本
```

5.1.3 部署账户服务

- 下载

执行过程中需输入相应数据库的配置。

```
cd ~/wecross-networks
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_account_manager.sh)

# 若出现长时间下载WeCross-Account-Manager包失败，请尝试以下命令重新下载：
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪account_manager.sh)
```

- 拷贝证书

```
cd ~/wecross-networks/WeCross-Account-Manager/
cp ~/wecross-networks/routers-payment/cert/sdk/* conf/
```

- 生成私钥

```
bash create_rsa_keypair.sh -d conf/
```

- 配置

```
cp conf/application-sample.toml conf/application.toml
vim conf/application.toml
```

需配置内容包括：

admin：配置admin账户，此处可默认，router中的admin账户需与此处对应，用于登录账户服务

db：配置自己的数据库账号密码

```
[service]
    address = '0.0.0.0'
    port = 8340
    sslKey = 'classpath:ssl.key'
    sslCert = 'classpath:ssl.crt'
    caCert = 'classpath:ca.crt'
    sslOn = true

[admin]
    # admin账户配置，第一次启动时写入db，之后作为启动校验字段
    name = 'org1-admin' # admin账户名
    password = '123456' # 密码

[auth]
    # for issuring token
    name = 'org1'
    expires = 18000 # 5 h
    noActiveExpires = 600 # 10 min

[db]
    # for connect database
    url = 'jdbc:mysql://localhost:3306/wecross_account_manager'
    username = 'root' # 配置数据库账户
    password = '123456' # 配置数据库密码，不支接受空密码
[ext]
    # for image auth code, allow image auth token empty
    allowImageAuthCodeEmpty = true
```

- 启动

```
bash start.sh
```

5.1.4 启动跨链路由

```
# 启动 router-8250
cd ~/wecross-networks/routers-payment/127.0.0.1-8250-25500/
bash start.sh

# 启动 router-8251
cd ~/wecross-networks/routers-payment/127.0.0.1-8251-25501/
bash start.sh
```

启动成功，输出如下：

```
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
netstat -napl | grep 8251
netstat -napl | grep 25501
```

5.1.5 部署控制台

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搭建控制台。

- 下载

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），下载完成后在当前目录下生成WeCross-Console目录。

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

# 若出现长时间下载WeCross-Console包失败，请尝试以下命令重新下载:
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh)
```

- 配置

```
cd ~/wecross-networks/WeCross-Console

# 拷贝连接router所需的TLS证书，从生成的routers-payment/cert/sdk目录下拷贝
cp ~/wecross-networks/routers-payment/cert/sdk/* conf/

# 拷贝配置文件，并配置跨链路由RPC服务地址以及端口。此处采用默认配置，默认连接至本地8250端口。
cp conf/application-sample.toml conf/application.toml
```

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/application.toml，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help可查看控制台帮助。

```
================================================================================
Welcome to WeCross console(v1.1.0)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.
================================================================================
```

- 测试功能

```
# 正常进入，可先退出控制台，等待后续配置
[WeCross]> quit
```

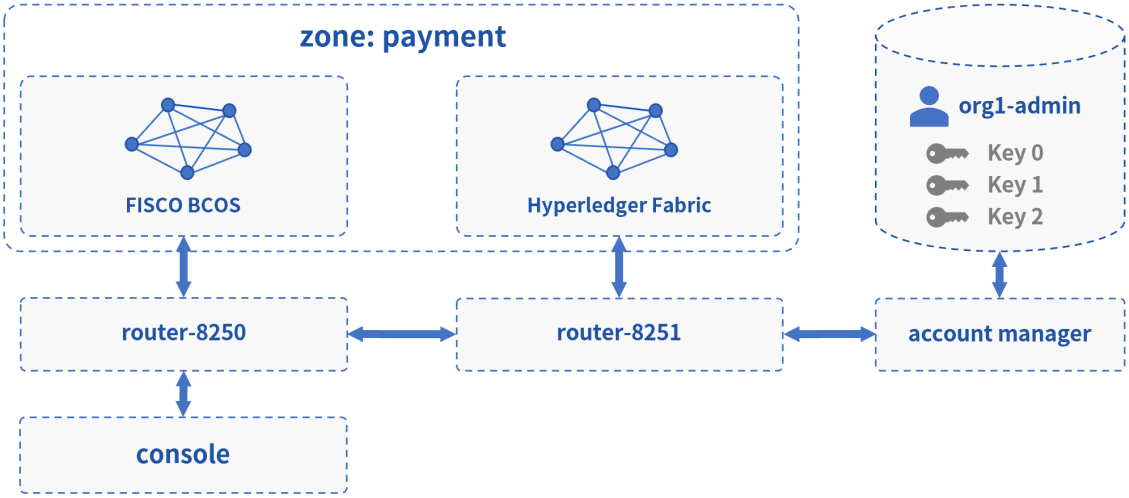
更多控制台命令及含义详见控制台命令。

5.2 区块链接入与账户配置

完成WeCross基础组件的部署后，接下来就是接入区块链，配置跨链账户，构建一个真实的跨链网络！

5.2.1 接入区块链

- **接入FISCO BCOS**：接入至跨链路由router-8250，详细教程
- **接入Hyperledger Fabric**：接入至跨链路由router-8251，详细教程



完成区块链接入后，重启跨链路由加载已配置的跨链资源。

```
# 重启 router-8250
cd ~/wecross-networks/routers-payment/127.0.0.1-8250-25500/
bash stop.sh && bash start.sh

# 重启 router-8251
cd ~/wecross-networks/routers-payment/127.0.0.1-8251-25501/
bash stop.sh && bash start.sh
```

检查日志，可看到刷出已加载的跨链资源，ctrl + c 退出。

```
tail -f logs/info.log |grep "active resources"

2020-12-05 21:07:30.925 [Thread-3] INFO  WeCrossHost() - Current active resources:␣
↪payment.bcos.WeCrossProxy(local), payment.bcos.WeCrossHub(local)
2020-12-05 21:07:40.940 [Thread-3] INFO  WeCrossHost() - Current active resources:␣
↪payment.bcos.WeCrossProxy(local), payment.bcos.WeCrossHub(local)
2020-12-05 21:07:50.956 [Thread-3] INFO  WeCrossHost() - Current active resources:␣
↪payment.bcos.WeCrossProxy(local), payment.bcos.WeCrossHub(local)
```

(下页继续)

(续上页)

5.2.2 配置跨链账户

区块链接入后，必须配置相应的账户才能完成资源的调用。

WeCross将各种类型的链账户进行了汇总，统一用跨链账户进行管理。在WeCross中，用户以跨链账户身份进行登陆，再操作各种资源。要往特定类型的链上发交易，只需要在跨链账户中添加相应类型的链账户即可。

添加BCOS链账户

- 生成公私钥

在控制台的目录中生成FISCO BCOS的账户公私钥，为添加链账户做准备。

```
cd ~/wecross-networks/WeCross-Console/conf/accounts/
bash get_account.sh # 生成accounts目录
```

生成成功，输出账户地址（address），请记录，用于后续添加链账户。

```
[INFO] Account Address: 0x129f336960ae6632ac3de903619720dde916d922
```

- 启动控制台

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

- 登录

用默认的跨链账户登录：org1-admin，密码：123456（默认账户在WeCross-Account-Manager/conf/application.toml配置）。

```
[WeCross]> login org1-admin 123456
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

- 添加链账户

为当前登录的跨链账户添加用于发交易的链账户，示例如下（每次生成的账户地址有所不同，需按具体情况填入）。

```
# 参数: addChainAccount BCOS2.0 私钥位置 公钥位置 账户地址(address) 是否设置为发交易的默认链账户
[WeCross.org1-admin]> addChainAccount BCOS2.0 conf/accounts/accounts/
↪0x4e89af80184147fcddc391c64ad673512236af67.public.pem conf/accounts/accounts/
↪0x4e89af80184147fcddc391c64ad673512236af67.pem␣
↪0x4e89af80184147fcddc391c64ad673512236af67 true
```

添加成功，退出控制台。

```
[WeCross.org1-admin]> quit
```

添加Fabric链账户

注意： 在添加Fabric链账户之前，请确保已搭建Fabric链，请参考搭建Fabric区块链

- 拷贝公私钥

将fabric链的公私钥拷贝至控制台目录，为添加链账户做准备。

```
cp -r ~/wecross-networks/fabric/certs/accounts/* ~/wecross-networks/WeCross-
↪Console/conf/accounts/
```

- 启动控制台

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

- 登录

用默认的跨链账户登录：org1-admin，密码：123456（默认账户在WeCross-Account-Manager/conf/application.toml配置）。

```
[WeCross]> login org1-admin 123456
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

- 添加链账户

为当前登录的跨链账户添加用于发交易的链账户。

```
# 参数: addChainAccount Fabric1.4 私钥位置 公钥位置 MSPID 是否设置为发交易的默认链账户

# 添加 fabric_admin_org1
[WeCross.org1-admin]> addChainAccount Fabric1.4 conf/accounts/fabric_admin_org1/
↪account.crt conf/accounts/fabric_admin_org1/account.key Org1MSP true

# 添加 fabric_admin_org2
[WeCross.org1-admin]> addChainAccount Fabric1.4 conf/accounts/fabric_admin_org2/
↪account.crt conf/accounts/fabric_admin_org2/account.key Org2MSP true
```

查看链账户

查看当前登录的跨链账户下的所有链账户，isDefault为true表示发交易的默认账户。

```
[WeCross.org1-admin]> listAccount
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
chainAccounts: [
        BCOS2.0 Account:
        keyID    : 0
        type     : BCOS2.0
        address  : 0x4e89af80184147fcddc391c64ad673512236af67
        isDefault: true
        ----------
        Fabric1.4 Account:
        keyID    : 2
        type     : Fabric1.4
```

(下页继续)

(续上页)

```
        MembershipID : Org2MSP
        isDefault: true
        ----------
        Fabric1.4 Account:
        keyID     : 1
        type      : Fabric1.4
        MembershipID : Org1MSP
        isDefault: false
        ----------
]
```

操作成功，退出控制台。

```
[WeCross.org1-admin]> quit
```

5.3 资源部署与操作

整个跨链网络环境搭建完毕后，就可以基于WeCross控制台部署和操作跨链资源了。

5.3.1 部署FISCO BCOS跨链资源

WeCross支持通过控制台向BCOS链上部署合约，部署步骤如下：

- 准备合约代码（以HelloWorld为例）

控制台的合约存放目录：`conf/contracts/solidity/`，目录下已有HelloWorld合约文件，若需部署其它合约，可将合约拷贝至相同位置。

```
tree conf/contracts/solidity/
conf/contracts/solidity/
└── HelloWorld.sol
```

- 启动控制台

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

- 部署合约

用`bcosDeploy`命令进行部署。

参数：ipath，代码目录，合约名，设置一个版本号

```
# 登录
[WeCross]> login org1-admin 123456

# 部署
[WeCross.org1-admin]> bcosDeploy payment.bcos.HelloWorld contracts/solidity/
↪HelloWorld.sol HelloWorld 1.0
Result: 0x953c8f97f9ea5930e6ca8d5eabbd9dfdcb142e6c
```

用`listResources`可查看此资源（`payment.bcos.HelloWorld`）已部署

```
[WeCross.org1-admin]> listResources
path: payment.bcos.HelloWorld, type: BCOS2.0, distance: 0
path: payment.bcos.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.fabric.WeCrossHub, type: Fabric1.4, distance: 1
total: 3
```

5.3.2 部署Hyperledger Fabric跨链资源

WeCross支持通过控制台向Hyperledger Fabric链上部署chaincode，部署步骤如下：

- 准备chaincode代码（以sacc为例）

控制台的chaincode存放目录：`conf/contracts/chaincode/`，其中sacc代码放入目录：`conf/contracts/chaincode/sacc`（目录名sacc为chaincode的名字），sacc目录中放置chaincode代码：sacc.go （代码名任意）。

```
tree conf/contracts/chaincode/sacc
conf/contracts/chaincode/sacc
├── policy.yaml
└── sacc.go
```

- 部署chaincode

为不同的Org分别安装（install）相同的chaincode。

```
# 在登录态下，查看默认链账户，可看到Fabric1.4的默认账户是Org2MSP的
[WeCross.org1-admin]> listAccount
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
chainAccounts: [
        BCOS2.0 Account:
        keyID    : 0
        type     : BCOS2.0
        address  : 0x4e89af80184147fcddc391c64ad673512236af67
        isDefault: true
        ----------
        Fabric1.4 Account:
        keyID    : 2
        type     : Fabric1.4
        MembershipID : Org2MSP
        isDefault: true
        ----------
        Fabric1.4 Account:
        keyID    : 1
        type     : Fabric1.4
        MembershipID : Org1MSP
        isDefault: false
        ----------
]

# 在向Org1进行install前，设置Fabric1.4的默认账户为Org1MSP，参数：setDefaultAccount␣
↪Fabric1.4 keyID
[WeCross.org1-admin]> setDefaultAccount Fabric1.4 1

# 给Org1安装sacc，参数：path Org 链码位置 版本号 链码语言
[WeCross.org1-admin]> fabricInstall payment.fabric.sacc Org1 contracts/chaincode/
↪sacc 1.0 GO_LANG
path: classpath:contracts/chaincode/sacc
Result: Success

# 在向Org2进行install前，设置Fabric1.4的默认账户为Org2MSP，参数：setDefaultAccount␣
↪Fabric1.4 keyID
[WeCross.org1-admin]> setDefaultAccount Fabric1.4 2

# 给Org2安装sacc，参数：path Org 链码位置 版本号 链码语言
[WeCross.org1-admin]> fabricInstall payment.fabric.sacc Org2 contracts/chaincode/
↪sacc 1.0 GO_LANG
```

(下页继续)

(续上页)

```
path: classpath:contracts/chaincode/sacc
Result: Success
```

实例化（instantiate）指定chaincode。

参数：ipath，对应的几个Org，chaincode代码工程目录，指定的版本，chaincode语言，背书策略（此处用默认），初始化参数

```
# fabricInstantiate 时默认Org1MSP或Org2MSP的链账户都可，此处用的Org2MSP
[WeCross.org1-admin]> fabricInstantiate payment.fabric.sacc ["Org1","Org2"]␣
↪contracts/chaincode/sacc 1.0 GO_LANG default ["a","10"]

Result: Instantiating... Please wait and use 'listResources' to check. See router
↪'s log for more information.
```

instantiate请求后，需等待1分钟左右。用`listResources`查看是否成功。若instantiate成功，可查询到资源`payment.fabric.sacc`。

```
[WeCross.org1-admin]> listResources
path: payment.bcos.HelloWorld, type: BCOS2.0, distance: 0
path: payment.fabric.WeCrossHub, type: Fabric1.4, distance: 1
path: payment.bcos.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.fabric.sacc, type: Fabric1.4, distance: 1
total: 4

[WeCross.org1-admin]> quit # 退出控制台
```

5.3.3 操作跨链资源

查看跨链资源

- 登录

用默认的跨链账户登录：org1-admin，密码：123456。（默认账户在WeCross-Account-Manager/conf/application.toml配置）

```
[WeCross]> login org1-admin 123456
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

- 获取资源列表

用`listResources`命令查看WeCross跨链网络中的所有资源。可看到有多个资源：

`payment.bcos.HelloWorld`：对应于FISCO BCOS链上的HelloWorld.sol合约。

`payment.fabric.sacc`：对应于Fabric链上的sacc.go合约。

`payment.xxxx.WeCrossHub`：每条链默认安装的Hub合约，用于接收链上合约发起的跨链调用。可参考《合约跨链》

```
[WeCross.org1-admin]> listResources
path: payment.bcos.HelloWorld, type: BCOS2.0, distance: 0
path: payment.fabric.WeCrossHub, type: Fabric1.4, distance: 1
path: payment.bcos.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.fabric.sacc, type: Fabric1.4, distance: 1
total: 4
```

操作payment.bcos.HelloWorld

- 读资源
  - 命令：call path 接口名 [参数列表]
  - 示例：call payment.bcos.HelloWorld get

```
# 调用HelloWorld合约中的get接口
[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Hello, World!]
```

- 写资源
  - 命令：sendTransaction path 接口名 [参数列表]
  - 示例：sendTransaction payment.bcos.HelloWeCross set Tom

```
# 调用HelloWeCross合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.bcos.HelloWorld set Tom
Txhash  : 0xd9cefb8c3ba28084583ba340e1d73a37574e1661926c3116729b1ec029f59828
BlockNum: 6
Result  : []      // 将Tom给set进去

[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Tom]     // 再次get，Tom已set
```

操作payment.fabric.sacc

跨链资源是对各个不同链上资源的统一和抽象，因此操作的命令是保持一致的。

- 读资源

```
# 调用mycc合约中的query接口
[WeCross.org1-admin]> call payment.fabric.sacc get a
Result: [10] // 初次get，a的值为10
```

- 写资源

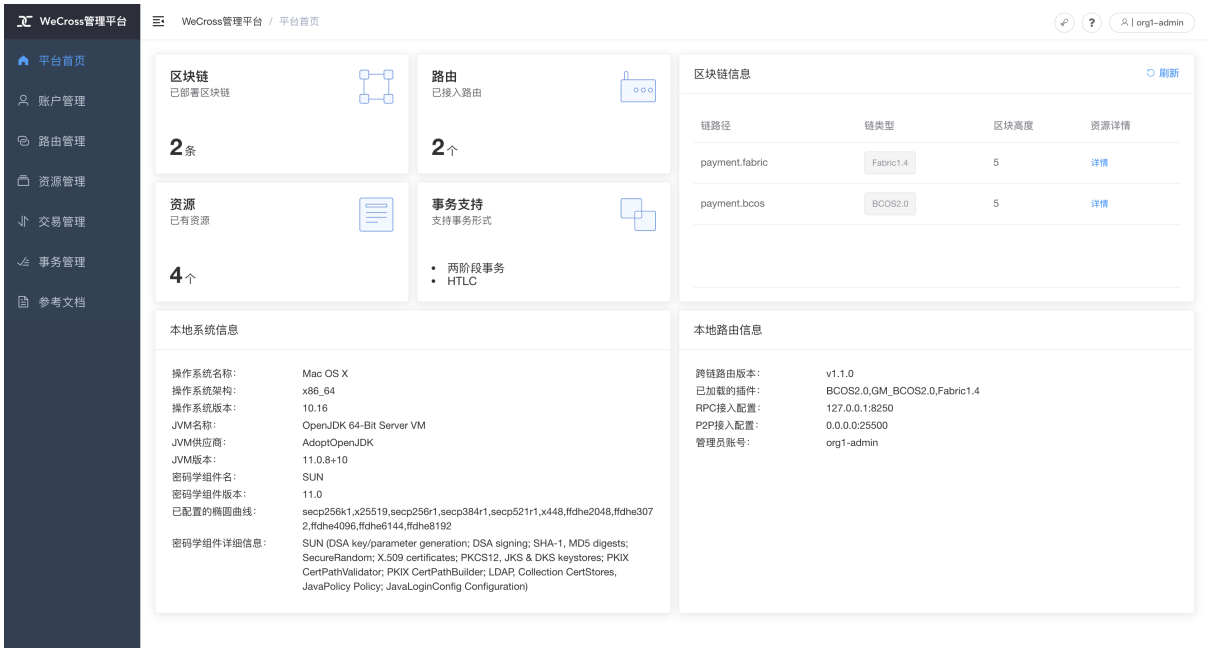
```
# 调用sacc合约中的set接口
[WeCross.org1-admin]> sendTransaction payment.fabric.sacc set a 666
Txhash  : aa3a7cd62d4b4c56b486f11fae2d903b7f07c2a3fa315ee2b44d5f5c43f5a8dc
BlockNum: 8
Result  : [666]

[WeCross.org1-admin]> call payment.fabric.sacc get a
Result: [666] // 再次get，a的值变成666

# 退出WeCross控制台
[WeCross.org1-admin]> quit # 若想再次启动控制台，cd至WeCross-Console，执行start.sh即可
```

5.3.4 访问网页管理平台

除了控制台，还可以通过浏览器访问WeCross网页管理台，实现跨链管理和资源调用。

```
http://localhost:8250/s/index.html#/login
```

用demo已配置账户进行登录：org1-admin，密码：123456



管理台中包含如下内容，点击链接进入相关操作指导。

- 登录/注册
- 平台首页
- 账户管理
- 路由管理
- 资源管理
- 交易管理
- 事务管理

注解:

- 若需要远程访问，请在router的conf/wecross.toml中，修改[rpc]标签下的address为所需ip（如：0.0.0.0）。保存后，重启router即可。

5.4 区块头验证配置

可在WeCross跨链路由端设定其他远端路由接入的区块链配置信息，遵循“谁配置，谁验证”的原则。

用户在连接BCOS和Fabric链的Router端可配置Fabric链各个Org Peer的CA证书和Orderer 的CA证书，以及BCOS链各个节点的公钥，用于验证对应链发来的区块。当在配置中未做该链的配置，则默认为不验证这个链的区块。

若要配置验证逻辑，需要在跨链路由的`conf`目录下新增配置文件`verifier.toml`（和`wecross.toml`在相同目录）。

这里以跨平台 FISCO BCOS & Fabric部署的网络作为举例，配置如下所示（请以实际情况配置）：

```
[verifiers]
    [verifiers.payment.bcos]
        chainType = 'BCOS2.0'
        # 填写所有共识节点的公钥
        pubKey = [
```

(下页继续)

(续上页)

```
            'b949f25fa39a6b3797ece30a2a9e025...',
            '...'
        ]
    [verifiers.payment.fabric]
        chainType = 'Fabric1.4'
        # 填写所有机构CA的证书路径
        [verifiers.payment.fabric.endorserCA]
            Org1MSP = 'classpath:verifiers/org1CA/ca.org1.example.com-cert.pem'
            Org2MSP = 'classpath:verifiers/org2CA/ca.org2.example.com-cert.pem'
        [verifiers.payment.fabric.ordererCA]
            OrdererMSP = 'classpath:verifiers/ordererCA/ca.example.com-cert.pem'
```

配置详情请查看 区块头验证配置。

CHAPTER 6

跨链接入

6.1 接入FISCO BCOS 2.0

WeCross-BCOS2-Stub作为WeCross的插件，让跨链路由具备接入FISCO-BCOS 2.0的能力。本章节将介绍如何基于BCOS插件接入FISCO BCOS链，内容包括：

- 搭建区块链
- 安装插件
- 配置插件
- 部署系统合约

WeCross-BCOS2-Stub 源码访问链接： GitHub访问链接， Gitee访问链接

重要:

- FISCO-BCOS版本需要 >= v2.1.0
- 若还未完成WeCross搭建，请参考 部署指南
- 以下教程的目录结构基于 部署指南 搭建的WeCross环境作介绍

6.1.1 1. 搭建区块链

如果已存在FISCO BCOS链，不需搭建新链，可跳过本节内容。

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
mkdir -p ~/wecross-networks/bcos && cd ~/wecross-networks/bcos

# 下载build_chain.sh脚本
curl -LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v2.7.1/build_
↪chain.sh && chmod u+x build_chain.sh
```

(下页继续)

(续上页)

```
# 若因为网络原因出现长时间下载失败，请尝试以下命令：
curl -LO https://gitee.com/FISCO-BCOS/FISCO-BCOS/raw/v2.7.1/tools/build_chain.sh &&
↪ chmod u+x build_chain.sh

# 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
```

6.1.2 2. 安装插件

基于部署指南搭建的WeCross，已完成插件的安装，位于跨链路由的plugin目录，可跳过本节内容。

```
plugin/
|-- bcos2-stub-gm-xxxx.jar     # 国密插件
|-- bcos2-stub-xxxx.jar        # 非国密插件
└-- fabric1-stub-xxxx.jar
```

用户如有特殊需求，可以自行编译，替换plugin目录下的插件。

2.1 下载编译

```
git clone https://github.com/WeBankBlockchain/WeCross-BCOS2-Stub.git

# 若因网络原因出现长时间下载失败，请尝试以下命令：
git clone https://gitee.com/WeBank/WeCross-BCOS2-Stub.git

cd WeCross-BCOS2-Stub
bash gradlew assemble # 在 dist/apps/ 下生成 bcos2-stub-XXXXX.jar 和 bcos2-stub-gm-
↪xxxx.jar
```

WeCross-BCOS2-Stub编译生成两个插件：

- 国密插件
- 非国密插件

```
dist/apps
├── bcos2-stub-gm-xxxx.jar     # 国密插件
└── bcos2-stub-xxxx.jar        # 非国密插件
```

2.2 拷贝安装

在跨链路由的主目录下创建plugin目录，然后将插件拷贝到该目录下完成安装。

```
cp dist/apps/* ~/wecross-networks/routers-payment/127.0.0.1-8250-25500/plugin/
```

注：若跨链路由中配置了两个相同的插件，插件冲突，会导致跨链路由启动失败。

6.1.3 3. 配置插件

3.1 生成配置框架

进入跨链路由的主目录，用add_chain.sh脚本在conf目录下生成BCOS链的配置框架。

```
cd ~/wecross-networks/routers-payment/127.0.0.1-8250-25500

 # -t 链类型，-n 指定链名字，可根据-h查看使用说明
bash add_chain.sh -t BCOS2.0 -n bcos
```

执行成功，输出如下。如果执行出错，请查看屏幕打印提示。

```
Chain "bcos" config framework has been generated to "conf/chains/bcos"
```

生成的目录结构如下：

```
tree conf/chains/bcos/
conf/chains/bcos/
├── WeCrossHub
│   └── WeCrossHub.sol          # 桥接合约
├── WeCrossProxy                # 代理合约
│   └── WeCrossProxy.sol
├── admin                       # stub内部内置账户，部署代理合约和桥接合约的默认账户
│   ├── xxxxx_secp256k1.key
│   └── account.toml
└── stub.toml                   # 插件配置文件
```

3.2 完成配置

拷贝证书

为了能够连接BCOS链的节点，需要拷贝节点SDK的证书，证书位于节点的nodes/127.0.0.1/sdk/目录。

```
# 证书目录以实际情况为准
cp -r xxxxxx/nodes/127.0.0.1/sdk/*   ~/wecross-networks/routers-payment/127.0.0.1-
↪8250-25500/conf/chains/bcos/
```

编辑配置文件

插件配置文件stub.toml配置项包括：

- 配置资源信息
- 配置SDK连接信息，与链进行交互

根据实际情况编辑[chain]、[channelService]的各个配置项。

```
[common]                    # 通用配置
    name = 'bcos'           # stub配置名称，即 [stubName] = bcos
    type = 'BCOS2.0'        # stub类型，`GM_BCOS2.0`或者`BCOS2.0`，`GM_BCOS2.0`国密类
型，`BCOS2.0`非国密类型
```

(下页继续)

(续上页)

```
[chain]                  # FISCO-BCOS 链配置
    groupId = 1          # 连接FISCO-BCOS群组id，默认为1
    chainId = 1          # 连接FISCO-BCOS链id，默认为1

[channelService]         # FISCO-BCOS 配置，以下文件在BCOS链的nodes/127.0.0.1/sdk目录下拷
贝
    caCert = 'ca.crt'    # 根证书
    sslCert = 'sdk.crt' # SDK证书
    sslKey = 'sdk.key'   # SDK私钥
    gmConnect = false    # 国密连接开关，若为true则使用BCOS节点SDK的国密证书进行连接，反之则使
用非国密证书连接
    gmCaCert = 'gm/gmca.crt'          # 国密CA证书
    gmSslCert = 'gm/gmsdk.crt'        # 国密SDK证书
    gmSslKey = 'gm/gmsdk.key'         # 国密SDK密钥
    gmEnSslCert = 'gm/gmensdk.crt'    # 国密加密证书
    gmEnSslKey = 'gm/gmensdk.key'     # 国密加密密钥
    timeout = 5000                    # SDK请求超时时间
    connectionsStr = ['127.0.0.1:20200']     # 连接列表

# [[resources]] 资源列表，配置链已有的资源
# [[resources]]
    #name = 'htlc'            # 资源名称
    #type = 'BCOS_CONTRACT'   # 资源类型，BCOS_CONTRACT
    #contractAddress = '0x7540601cce8b0802980f9ebf7aeee22bb4d73c22'   # 合约地址
```

6.1.4 4. 部署系统合约

每个Stub需要部署两个系统合约，分别是代理合约和桥接合约，代理合约负责管理事务以及业务合约的调用，桥接合约用于记录合约跨链请求。在跨链路由主目录执行以下命令：

4.1 非国密链

```
# 部署代理合约
bash deploy_system_contract.sh -t BCOS2.0 -c chains/bcos -P

# 部署桥接合约
bash deploy_system_contract.sh -t BCOS2.0 -c chains/bcos -H

# 若后续有更新系统合约的需求，首先更新conf/chains/bcos下的系统合约代码，在上述命令添加-u参数，执
行并重启跨链路由
```

部署成功，则输出如下内容。若失败可查看提示信息和错误日志。

```
SUCCESS: WeCrossProxy:xxxxxxxx has been deployed! chain: chains/bcos
SUCCESS: WeCrossHub:xxxxxxxx has been deployed! chain: chains/bcos
```

4.2 国密链

```
# 部署代理合约
bash deploy_system_contract.sh -t GM_BCOS2.0 -c chains/bcos -P

# 部署桥接合约
bash deploy_system_contract.sh -t GM_BCOS2.0 -c chains/bcos -H

# 若后续有更新系统合约的需求，首先更新conf/chains/bcos下的系统合约代码，在上述命令添加-u参数，执
行并重启跨链路由
```

(下页继续)

(续上页)

部署成功，则输出如下内容。若失败可查看提示信息和错误日志。

```
SUCCESS: WeCrossProxy:xxxxxxxx has been deployed! chain: chains/bcos
SUCCESS: WeCrossHub:xxxxxxxx has been deployed! chain: chains/bcos
```

6.2 接入Hyperledger Fabric 1.4

WeCross-Fabric1-Stub作为WeCross的插件，让跨链路由具备接入Hyperledger Fabric 1.4的能力。本章节将介绍如何基于Fabric插件接入Hyperledger Fabric链，内容包括：

- 搭建区块链
- 安装插件
- 配置内置账户
- 配置插件
- 部署系统合约

重要:

- 若还未完成WeCross搭建，请参考 部署指南
- 以下教程的目录结构基于 部署指南 搭建的WeCross环境作介绍

WeCross-Fabric1-Stub源码访问地址：

- GitHub访问地址
- Gitee访问地址

6.2.1 1. 搭建区块链

如果已存在Fabric链，不需搭建新链，可跳过本节内容。

为方便Fabric链的搭建，WeCross Demo包中提供了Fabric链搭建脚本。若下载较慢，可选择更多下载方式。

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

# 下载Demo包
bash <(curl -sL https://github.com/WeBankBlockchain/wecross/releases/download/
↪resources/download_demo.sh)

# 若出现长时间下载Demo包失败，请尝试以下命令重新下载:
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_demo.
↪sh)

# 拷贝其中的Fabric demo链环境
cp ./wecross-demo/fabric/* ./

# 搭链，若出错，执行 bash clear.sh 后重新 bash build.sh
bash build.sh
```

搭建成功，查看Fabric链各个容器运行状态。

```
docker ps
```

可看到各个容器的状态：

```
CONTAINER ID        IMAGE                                  COMMAND
↪CREATED             STATUS              PORTS                     NAMES
11c7358b5f59        hyperledger/fabric-tools:latest     "/bin/bash"          2␣
↪minutes ago        Up 2 minutes                                  cli
63bb98e16c20        hyperledger/fabric-peer:latest      "peer node start"    2␣
↪minutes ago        Up 2 minutes        0.0.0.0:10051->10051/tcp  peer1.org2.
↪example.com
823f2c4034b7        hyperledger/fabric-peer:latest      "peer node start"    2␣
↪minutes ago        Up 2 minutes        0.0.0.0:8051->8051/tcp    peer1.org1.
↪example.com
1468956a60c6        hyperledger/fabric-peer:latest      "peer node start"    2␣
↪minutes ago        Up 2 minutes        0.0.0.0:9051->9051/tcp    peer0.org2.
↪example.com
1b3f50ed07ad        hyperledger/fabric-orderer:latest   "orderer"            2␣
↪minutes ago        Up 2 minutes        0.0.0.0:7050->7050/tcp    orderer.example.
↪com
18747185608f        hyperledger/fabric-peer:latest      "peer node start"    2␣
↪minutes ago        Up 2 minutes        0.0.0.0:7051->7051/tcp    peer0.org1.
↪example.com
```

6.2.2 2. 安装插件

基于部署指南搭建的WeCross，已完成插件的安装，位于跨链路由的plugin目录，可跳过本节内容。

```
plugin/
|-- bcos2-stub-gm-xxxx.jar
|-- bcos2-stub-xxxx.jar
└-- fabric1-stub-xxxx.jar      # Fabric插件
```

用户如有特殊需求，可以自行编译，替换plugin目录下的插件。

2.1 下载编译

```
git clone https://github.com/WeBankBlockchain/WeCross-Fabric1-Stub.git

# 若因网络原因出现长时间拉取代码失败，请尝试以下命令：
git clone https://gitee.com/WeBank/WeCross-Fabric1-Stub.git

cd WeCross-Fabric1-Stub
bash gradlew assemble # 在 dist/apps/ 下生成 fabric1-stub-XXXXX.jar
```

2.2 拷贝安装

在跨链路由的主目录下创建plugin目录，然后将插件拷贝到该目录下完成安装。

```
cp dist/apps/* ~/wecross-networks/routers-payment/127.0.0.1-8251-25501/plugin/
```

注：若跨链路由中配置了两个相同的插件，插件冲突，会导致跨链路由启动失败。

6.2.3 3. 配置内置账户

在跨链路由中需配置用于与Fabric链进行交互的内置账户。内置账户需配置多个：

- admin账户：必配，一个admin账户，用于接入此Fabric链
- 机构admin账户：选配，每个Fabric的Org配置一个admin账户，用于在每个Org上部署系统合约

3.1 生成账户配置框架

```
# 切换至对应跨链路由的主目录
cd ~/wecross-networks/routers-payment/127.0.0.1-8251-25501/

# 用脚本生成Fabric账户配置：账户类型（Fabric1.4），账户名（fabric_admin）
# 接入Fabric链，需要配置一个admin账户
bash add_account.sh -t Fabric1.4 -n fabric_admin


# 为Fabric链的每个Org都配置一个admin账户，此处有两个org（Org1和Org2），分别配两个账户

# 配Org1的admin
bash add_account.sh -t Fabric1.4 -n fabric_admin_org1

# 配Org2的admin
bash add_account.sh -t Fabric1.4 -n fabric_admin_org2
```

3.2 完成配置

修改账户配置

生成的账户配置，默认的mspid为Org1MSP，需将Org2的账户的mspid配置为Org2MSP的。此处只需修改fabric_admin_org2账户的配置。

```
vim conf/accounts/fabric_admin_org2/account.toml

# 修改mspid，将 'Org1MSP' 更改为 'Org2MSP'
```

拷贝证书文件

Fabric链的证书位于crypto-config目录，请参考以下命令并**根据实际情况**完成相关证书的拷贝。

```
cd ~/wecross-networks/routers-payment/127.0.0.1-8251-25501

# 配置fabric_admin
# 拷贝私钥
cp xxxxxx/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.
↪example.com/msp/keystore/*_sk   conf/accounts/fabric_admin/account.key
# 拷贝证书
cp xxxxxx/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.
↪example.com/msp/signcerts/Admin@org1.example.com-cert.pem   conf/accounts/fabric_
↪admin/account.crt

# 配置fabric_admin_org1
# 拷贝私钥
cp xxxxxx/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.
↪example.com/msp/keystore/*_sk   conf/accounts/fabric_admin_org1/account.key
# 拷贝证书
cp xxxxxx/crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.
↪example.com/msp/signcerts/Admin@org1.example.com-cert.pem   conf/accounts/fabric_
↪admin_org1/account.crt

# 配置fabric_admin_org2
# 拷贝私钥
cp xxxxxx/crypto-config/peerOrganizations/org2.example.com/users/Admin@org2.
↪example.com/msp/keystore/*_sk   conf/accounts/fabric_admin_org2/account.key
```

(下页继续)

(续上页)

```
# 拷贝证书
cp xxxxxx/crypto-config/peerOrganizations/org2.example.com/users/Admin@org2.
↪example.com/msp/signcerts/Admin@org2.example.com-cert.pem   conf/accounts/fabric_
↪admin_org2/account.crt
```

完成证书拷贝后，账户目录结构如下：

```
tree conf/accounts/
conf/accounts/
├── fabric_admin
│   ├── account.crt
│   ├── account.key
│   └── account.toml
├── fabric_admin_org1
│   ├── account.crt
│   ├── account.key
│   └── account.toml
└── fabric_admin_org2
    ├── account.crt
    ├── account.key
    └── account.toml
```

6.2.4 4. 配置插件

4.1 生成插件配置框架

进入跨链路由的主目录，用add_chain.sh脚本在conf目录下生成Fabric链的配置框架。

```
cd ~/wecross-networks/routers-payment/127.0.0.1-8251-25501

 # -t 链类型，-n 指定链名字，可根据-h查看使用说明
bash add_chain.sh -t Fabric1.4 -n fabric
```

执行成功，输出如下。如果执行出错，请查看屏幕打印提示。

```
SUCCESS: Chain "fabric" config framework has been generated to "conf/chains/fabric"
```

生成的目录结构如下：

```
tree conf/chains/fabric/
conf/chains/fabric/
├── chaincode
│   ├── WeCrossHub
│   │   └── hub.go      # 桥接合约
│   └── WeCrossProxy
│       └── proxy.go    # 代理合约
└── stub.toml           # 插件配置文件
```

4.2 完成配置

拷贝证书

相关证书同样位于crypto-config目录，请参考以下命令并**根据实际情况**完成相关证书的拷贝。

```
# 假设当前位于跨链路由的主目录
# 拷贝orderer证书
cp xxxxxx/crypto-config/ordererOrganizations/example.com/orderers/orderer.example.
↪com/msp/tlscacerts/tlsca.example.com-cert.pem   conf/chains/fabric/orderer-tlsca.
↪crt
```

(下页继续)

(续上页)

```
# 拷贝org1证书
cp xxxxxx/crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.
↪example.com/tls/ca.crt    conf/chains/fabric/org1-tlsca.crt

# 拷贝org2证书
cp xxxxxx/crypto-config/peerOrganizations/org2.example.com/peers/peer0.org2.
↪example.com/tls/ca.crt    conf/chains/fabric/org2-tlsca.crt
```

编辑配置文件

插件配置文件stub.toml配置项包括：

- 配置资源信息
- 配置SDK连接信息，与链进行交互

根据实际情况编辑[fabricServices]、[orgs]的各个配置项。

```
[common]
    name = 'fabric'                                    # 指定的连接的链的名字，与该配置文件所
在的目录名一致，对应path中的{zone}/{chain}/{resource}的chain
    type = 'Fabric1.4'                             # 插件的类型

[fabricServices]
    channelName = 'mychannel'
    orgUserName = 'fabric_admin' # 指定一个机构的admin账户，用于与orderer通信
    ordererTlsCaFile = 'orderer-tlsca.crt' # orderer证书名字，指向与此配置文件相同目录下的
证书
    ordererAddress = 'grpcs://localhost:7050' # orderer的url

[orgs] # 机构节点列表
    [orgs.Org1] # 机构1：Org1
        tlsCaFile = 'org1-tlsca.crt' # Org1的证书
        adminName = 'fabric_admin_org1' # Org1的admin账户，在下一步骤中配置
        endorsers = ['grpcs://localhost:7051'] # endorser的ip:port列表，可配置多个

    [orgs.Org2] # 机构2：Org2
        tlsCaFile = 'org2-tlsca.crt' # Org2的证书
        adminName = 'fabric_admin_org2' # Org2的admin账户，在下一步骤中配置
        endorsers = ['grpcs://localhost:9051'] # endorser的ip:port列表，可配置多个
```

6.2.5 5. 部署系统合约

每个Stub需要部署两个系统合约，分别是代理合约和桥接合约，代理合约负责管理事务以及业务合约的调用，桥接合约用于记录合约跨链请求。在跨链路由主目录执行以下命令：

```
# 部署代理合约
bash deploy_system_contract.sh -t Fabric1.4 -c chains/fabric -P

# 部署桥接合约
bash deploy_system_contract.sh -t Fabric1.4 -c chains/fabric -H

# 若后续有更新系统合约的需求，首先更新conf/chains/fabric下的系统合约代码，在上述命令添加-u参数，
执行并重启跨链路由
```

部署成功，则输出如下内容。若失败可查看提示信息和错误日志。

```
SUCCESS: WeCrossProxy has been deployed to chains/fabric
SUCCESS: WeCrossHub has been deployed to chains/fabric
```

6.3 接入Hyperledger Fabric 2

WeCross-Fabric2-Stub作为WeCross的插件，让跨链路由具备接入Hyperledger Fabric 2的能力。本章节将介绍如何基于Fabric插件接入Hyperledger Fabric链，内容包括：

- 搭建区块链
- 安装插件
- 配置内置账户
- 配置插件
- 部署系统合约

重要:

- 若还未完成WeCross搭建，请参考 部署指南
- 以下教程的目录结构基于 部署指南 搭建的WeCross环境作介绍

WeCross-Fabric2-Stub源码访问地址：

- GitHub访问地址
- Gitee访问地址

6.3.1 1. 搭建区块链

如果已存在Fabric链，不需搭建新链，可跳过本节内容。

为方便Fabric链的搭建，WeCross Demo包中提供了Fabric链搭建脚本。若下载较慢，可选择更多下载方式。

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

# 下载Demo包
bash <(curl -sL https://github.com/WeBankBlockchain/wecross/releases/download/
↪resources/download_demo.sh)

# 若出现长时间下载Demo包失败，请尝试以下命令重新下载:
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_demo.
↪sh)

# 拷贝其中的Fabric demo链环境
cp ./wecross-demo/fabric2/* ./

# 搭链，若出错，执行 bash clear.sh 后重新 bash build.sh
bash build.sh
```

搭建成功，查看Fabric链各个容器运行状态。

```
docker ps
```

可看到各个容器的状态：

```
CONTAINER ID        IMAGE
↪
↪                       COMMAND                  CREATED             STATUS
↪          PORTS                                         NAMES
22355c5e1cb5        hyperledger/fabric-tools:2.3.0
↪
↪                       "/bin/bash"              2 minutes ago       Up 2
↪minutes                                                  cli
```

(下页继续)

(续上页)

```
43749216ed56        hyperledger/fabric-peer:2.3.0
↪
↪                       "peer node start"        2 minutes ago        Up 2␣
↪minutes        7051/tcp, 0.0.0.0:9051->9051/tcp                  peer0.org2.
↪example.com
5d8e86ea87d2        hyperledger/fabric-orderer:2.3.0
↪
↪                       "orderer"                2 minutes ago        Up 2␣
↪minutes        0.0.0.0:7050->7050/tcp, 0.0.0.0:7053->7053/tcp   orderer.example.
↪com
a510273be2fb        hyperledger/fabric-peer:2.3.0
↪
↪                       "peer node start"        2 minutes ago        Up 2␣
↪minutes        0.0.0.0:7051->7051/tcp                            peer0.org1.
↪example.com
```

6.3.2 2. 安装插件

基于部署指南搭建的WeCross，已完成插件的安装，位于跨链路由的plugin目录，可跳过本节内容。

```
plugin/
|-- bcos2-stub-gm-xxxx.jar
|-- bcos2-stub-xxxx.jar
|-- fabric1-stub-xxxx.jar
└-- fabric2-stub-xxxx.jar      # Fabric2插件
```

用户如有特殊需求，可以自行编译，替换plugin目录下的插件。

2.1 下载编译

```
git clone https://github.com/WeBankBlockchain/WeCross-Fabric2-Stub.git

# 若因网络原因出现长时间拉取代码失败，请尝试以下命令：
git clone https://gitee.com/WeBank/WeCross-Fabric2-Stub.git

cd WeCross-Fabric2-Stub
bash gradlew assemble # 在 dist/apps/ 下生成 fabric2-stub-XXXXX.jar
```

2.2 拷贝安装

在跨链路由的主目录下创建plugin目录，然后将插件拷贝到该目录下完成安装。

```
cp dist/apps/* ~/wecross-networks/routers-payment/127.0.0.1-8251-25501/plugin/
```

注：若跨链路由中配置了两个相同的插件，插件冲突，会导致跨链路由启动失败。

6.3.3 3. 配置内置账户

在跨链路由中需配置用于与Fabric链进行交互的内置账户。内置账户需配置多个：

- admin账户：必配，一个admin账户，用于接入此Fabric链
- 机构admin账户：选配，每个Fabric的Org配置一个admin账户，用于在每个Org上部署系统合约

3.1 生成账户配置框架

```
# 切换至对应跨链路由的主目录
cd ~/wecross-networks/routers-payment/127.0.0.1-8251-25501/

# 用脚本生成Fabric账户配置：账户类型（Fabric1.4），账户名（fabric_admin）
# 接入Fabric链，需要配置一个admin账户
bash add_account.sh -t Fabric2.0 -n fabric2_admin


# 为Fabric链的每个Org都配置一个admin账户，此处有两个org（Org1和Org2），分别配两个账户

# 配Org1的admin
bash add_account.sh -t Fabric2.0 -n fabric2_admin_org1

# 配Org2的admin
bash add_account.sh -t Fabric2.0 -n fabric2_admin_org2
```

3.2 完成配置

修改账户配置

生成的账户配置，默认的mspid为Org1MSP，需将Org2的账户的mspid配置为Org2MSP的。此处只需修改fabric2_admin_org2账户的配置。

```
vim conf/accounts/fabric2_admin_org2/account.toml

# 修改mspid，将 'Org1MSP' 更改为 'Org2MSP'
```

拷贝证书文件

Fabric链的证书位于organizations目录，请参考以下命令并**根据实际情况**完成相关证书的拷贝。

```
cd ~/wecross-networks/routers-payment/127.0.0.1-8251-25501

# 配置fabric_admin
# 拷贝私钥
cp xxxxxx/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/
↪keystore/*_sk   conf/accounts/fabric2_admin/account.key
# 拷贝证书
cp xxxxxx/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/
↪signcerts/Admin@org1.example.com-cert.pem   conf/accounts/fabric2_admin/account.
↪crt

# 配置fabric_admin_org1
# 拷贝私钥
cp xxxxxx/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/
↪keystore/*_sk   conf/accounts/fabric2_admin_org1/account.key
# 拷贝证书
cp xxxxxx/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp/
↪signcerts/Admin@org1.example.com-cert.pem   conf/accounts/fabric2_admin_org1/
↪account.crt

# 配置fabric_admin_org2
# 拷贝私钥
cp xxxxxx/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp/
↪keystore/*_sk   conf/accounts/fabric2_admin_org2/account.key
# 拷贝证书
cp xxxxxx/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp/
↪signcerts/Admin@org2.example.com-cert.pem   conf/accounts/fabric2_admin_org2/
↪account.crt
```

完成证书拷贝后，账户目录结构如下：

```
tree conf/accounts/
conf/accounts/
├── fabric2_admin
│   ├── account.crt
│   ├── account.key
│   └── account.toml
├── fabric2_admin_org1
│   ├── account.crt
│   ├── account.key
│   └── account.toml
└── fabric2_admin_org2
    ├── account.crt
    ├── account.key
    └── account.toml
```

6.3.4 4. 配置插件

4.1 生成插件配置框架

进入跨链路由的主目录，用add_chain.sh脚本在conf目录下生成Fabric链的配置框架。

```
cd ~/wecross-networks/routers-payment/127.0.0.1-8251-25501

 # -t 链类型，-n 指定链名字，可根据-h查看使用说明
bash add_chain.sh -t Fabric2.0 -n fabric2
```

执行成功，输出如下。如果执行出错，请查看屏幕打印提示。

```
SUCCESS: Chain "fabric2" config framework has been generated to "conf/chains/
↪fabric2"
```

生成的目录结构如下：

```
tree conf/chains/fabric2/ -L 2
conf/chains/fabric2/
├── chaincode
│   ├── WeCrossHub    # 桥接合约代码目录
│   └── WeCrossProxy  # 代理合约代码目录
└── stub.toml                    # 插件配置文件
```

4.2 完成配置

拷贝证书

相关证书同样位于organizations目录，请参考以下命令并**根据实际情况**完成相关证书的拷贝。

```
# 假设当前位于跨链路由的主目录
# 拷贝orderer证书
cp xxxxxx/ordererOrganizations/example.com/orderers/orderer.example.com/msp/
↪tlscacerts/tlsca.example.com-cert.pem   conf/chains/fabric2/orderer-tlsca.crt

# 拷贝org1证书
cp xxxxxx/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.
↪crt   conf/chains/fabric2/org1-tlsca.crt

# 拷贝org2证书
cp xxxxxx/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.
↪crt   conf/chains/fabric2/org2-tlsca.crt
```

编辑配置文件

插件配置文件stub.toml配置项包括：

- 配置资源信息
- 配置SDK连接信息，与链进行交互

根据实际情况编辑[fabricServices]、[orgs]的各个配置项。

```
[common]
    name = 'fabric2'                                    # 指定的连接的链的名字，与该配置文件所在的目录名一致，对应path中的{zone}/{chain}/{resource}的chain
    type = 'Fabric2.0'                                  # 插件的类型

[fabricServices]
    channelName = 'mychannel'
    orgUserName = 'fabric2_admin' # 指定一个机构的admin账户，用于与orderer通信
    ordererTlsCaFile = 'orderer-tlsca.crt' # orderer证书名字，指向与此配置文件相同目录下的证书
    ordererAddress = 'grpcs://localhost:7050' # orderer的url

[orgs] # 机构节点列表
    [orgs.Org1] # 机构1：Org1
         tlsCaFile = 'org1-tlsca.crt' # Org1的证书
         adminName = 'fabric2_admin_org1' # Org1的admin账户，在下一步骤中配置
         endorsers = ['grpcs://localhost:7051'] # endorser的ip:port列表，可配置多个

    [orgs.Org2] # 机构2：Org2
         tlsCaFile = 'org2-tlsca.crt' # Org2的证书
         adminName = 'fabric2_admin_org2' # Org2的admin账户，在下一步骤中配置
         endorsers = ['grpcs://localhost:9051'] # endorser的ip:port列表，可配置多个
```

6.3.5 5. 部署系统合约

每个Stub需要部署两个系统合约，分别是代理合约和桥接合约，代理合约负责管理事务以及业务合约的调用，桥接合约用于记录合约跨链请求。在跨链路由主目录执行以下命令：

```
# 部署代理合约
bash deploy_system_contract.sh -t Fabric2.0 -c chains/fabric2 -P

# 部署桥接合约
bash deploy_system_contract.sh -t Fabric2.0 -c chains/fabric2 -H

# 若后续有更新系统合约的需求，首先更新conf/chains/fabric下的系统合约代码，在上述命令添加-u参数，执行并重启跨链路由
```

部署成功，则输出如下内容。若失败可查看提示信息和错误日志。

```
SUCCESS: WeCrossProxy has been deployed to chains/fabric2
SUCCESS: WeCrossHub has been deployed to chains/fabric2
```

重启跨链路由

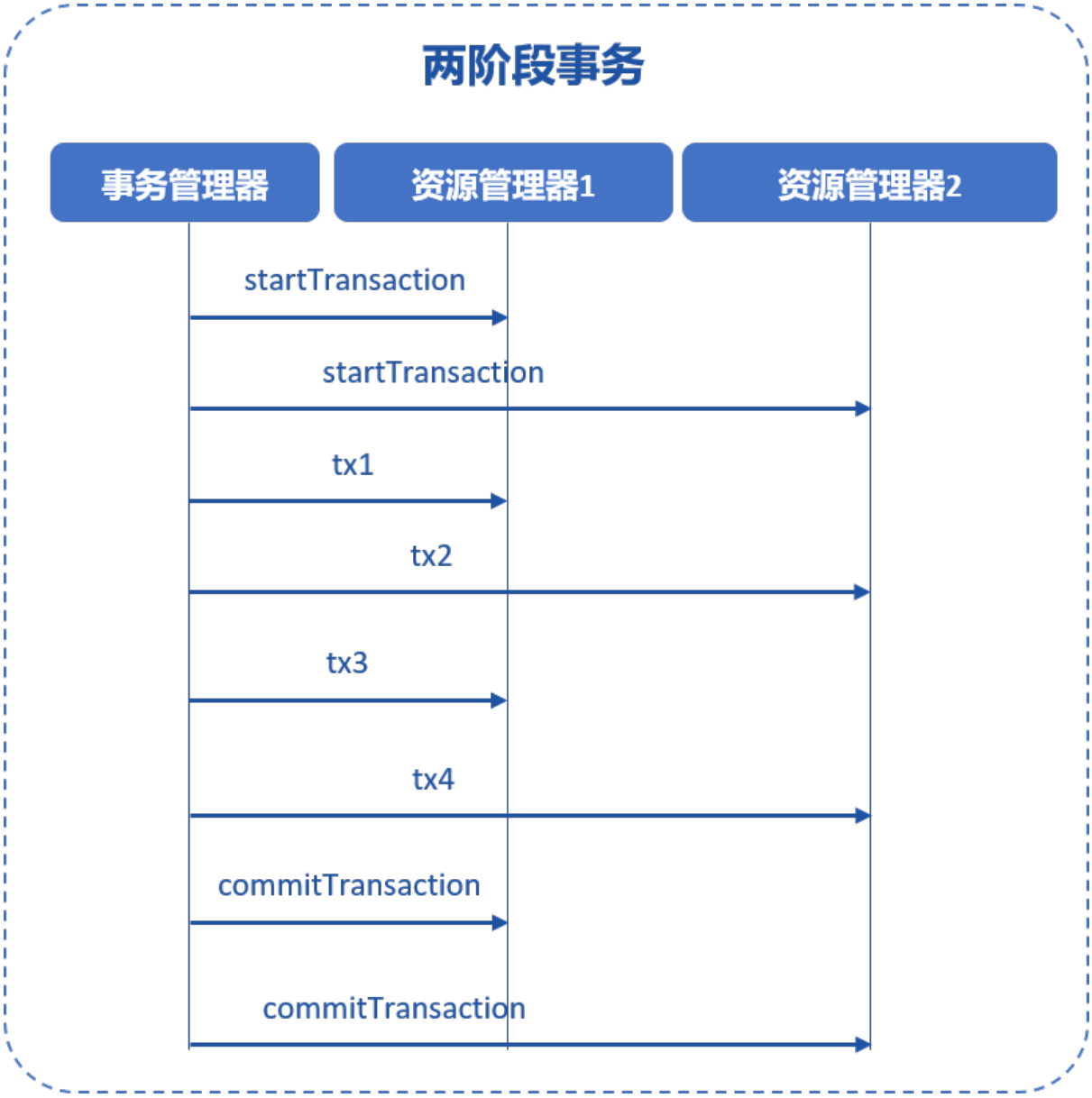
```
bash stop.sh
bash start.sh
```

CHAPTER 7

跨链事务

7.1 两阶段事务

两阶段事务是分布式数据库和分布式系统中常见的事务模型，两阶段事务的架构中包含事务管理器和资源管理器等多个角色。



在WeCross事务模型中，代理合约负责管理事务状态以及对资源的读写控制，事务管理器则由跨链路由实现。

用户通过使用WeCross SDK，可以让多个跨链路由和多个链上的资源参与事务，以保障对各个资源操作的原子性。

7.1.1 准备工作

无论何种智能合约或链码，参与基本的两阶段事务流程时不需要提前做修改。但如果要支持两阶段事务的**回滚操作**，参与事务的资源接口必须实现对应的**逆向接口**。

正向接口示例

以Solidity智能合约为例，原合约包含一个交易接口，如转账：

```
function transfer(string memory from, string memory to, int balance) public {
// balance check...
    balances[from] -= balance;
    balances[to] += balance;
}
```

注： 案例省略了余额和透支检查的逻辑，只列出了正向交易接口的关键逻辑。

逆向接口示例

为了支持回滚，需实现一个逆向接口。逆向接口的参数与原交易接口相同，函数名增加_revert的后缀，表明这是一个用于两阶段事务回滚的接口。当两阶段事务需要回滚时，WeCross会自动执行Solidity合约的逆向接口。

参考实现如下：

```
function transfer_revert(string memory from, string memory to, int balance) public
↪{
  // balance check...
  balance[from] += balance;
  balance[to] -= balance;
}
```

可以看到，在相同输入参数的情况下逆向接口的行为和执行结果与正向接口相反，以恢复资源执行事务之前的状态。

7.1.2 原理说明

当开始事务时，参与事务的资源会被代理合约记录并锁定。之后的调用请求必须提供相应的事务ID才能被正确执行。代理合约还会保存事务中每一个执行步骤的资源、调用方法和参数列表。

WeCross回滚事务时，会按照正向接口执行的顺序，以相反的顺序调用逆向接口。举例，事务过程中执行了一系列正向交易接口如下：

```
transfer1('from1', 'to1', 100)
transfer2('from2', 'to2', 200)
transfer3('from3', 'to3', 300)
```

回滚时，WeCross会按以下顺序执行逆向交易接口：

```
transfer3_revert('from3', 'to3', 300)
transfer2_revert('from2', 'to2', 200)
transfer1_revert('from1', 'to1', 100)
```

7.1.3 使用

Java项目可通过调用Java-SDK的事务相关接口实现事务操作，其它语言的项目可通过JSON-RPC接口完成事务调用。

WeCross控制台也提供了事务相关的命令，操作示例如下：

开始两阶段事务

使用startTransaction命令，开始两阶段事务

```
startTransaction [path_1] ... [path_n]
```

参数解析：

- path_1 ... path_n: 参与事务的资源路径列表，路径列表中的资源会被本次事务锁定，锁定后仅限本事务相关的交易才能对这些资源发起读写操作，非本次事务的所有读写操作都会被拒绝

例子：

开始一个事务，资源为zone.chainA.res1、zone.chainB.res2

```
startTransaction zone.chainA.res1 zone.chainB.res2
```

发起事务交易

使用execTransaction命令，发起事务交易，execTransaction命令与sendTransaction命令类似。

任何资源一旦参与了事务，就无法用sendTransaction来向该资源发送交易，必须使用execTransaction

```
execTransaction [path] [method] [args]
```

参数解析：

- path：资源路径
- method：接口名，同sendTransaction
- args：参数，同sendTransaction

举例，通过控制台，调用transfer接口：

```
#调用事务资源zone.chainA.res1的transfer接口
execTransaction zone.chainA.res1 transfer 'fromUserName' 'toUserName' 100

#调用事务资源zone.chainB.res2的transfer接口
execTransaction zone.chainB.res2 transfer 'fromUserName' 'toUserName' 200
```

注： 单个事务的步骤不宜过多，否则在获取事务详情或者回滚事务时容易出现gas不足导致执行失败。

提交事务

使用commitTransaction命令，提交事务，确认事务执行过程中所有的变动。

```
# 控制台记录了该事务的上下文，因此提交的时候不需要传入参数
commitTransaction
```

通过控制台，执行一个完整的事务步骤

```
#开始事务
startTransaction zone.chainA.res1 zone.chainB.res2

#调用事务资源zone.chainA.res1的transfer接口
execTransaction zone.chainA.res1 transfer 'fromUserName' 'toUserName' 100

#调用事务资源zone.chainB.res2的transfer接口
execTransaction zone.chainB.res2 transfer 'fromUserName' 'toUserName' 200

#提交事务
commitTransaction
```

回滚事务

当事务的某个步骤执行失败，需要撤销本次事务的所有变更时，使用rollbackTransaction命令

```
# 控制台记录了该事务的上下文，因此回滚的时候不需要传入参数
rollbackTransaction
```

举例，通过控制台，执行一个完整的事务步骤:

```
startTransaction zone.chainA.res1 zone.chainB.res2 #开始事务

# 调用事务资源zone.chainA.res1的transfer接口
execTransaction zone.chainA.res1 transfer 'fromUserName' 'toUserName' 100

# 调用事务资源zone.chainB.res2的transfer接口，假设失败，则事务需要回滚
execTransaction zone.chainB.res2 transfer 'fromUserName' 'toUserName' 200

# 回滚事务
rollbackTransaction
```

7.2 哈希时间锁合约

哈希时间锁合约（hash time lock contract，htlc）能够实现两条异构链之间资产的原子交换，即跨链转账。

如果你想**快速体验**跨链转账可以参考体验WeCross。

WeCross提供了Solidity版本和Golang版本的htlc基类合约，基于htlc基类合约可以轻松开发适用于不同资产类型的htlc应用合约。

Solidity版本源码:

- GitHub访问链接
- Gitee访问链接

Golang版本源码:

- GitHub访问链接

- Gitee访问链接

本章节以FISCO BCOS和Hyperledger Fabric的**示例资产合约**为例，演示如何实现两条异构链的资产互换。

7.2.1 准备工作

要完成资产互换，需要在各自链上部署资产合约以及哈希时间锁合约，然后通过WeCross控制台创建跨链转账提案，跨链路由会根据提案信息自动完成跨链转账。

部署WeCross和控制台

- 以组网方式搭建两个跨链路由
- 搭建WeCross控制台，连接跨链路由

BCOS链前期准备

部署合约

假设已存在一个跨链账户`org2-admin`，且分别添加了一个BCOS2.0链账户以及Fabric1.4链账户。

进入BCOS链的WeCross控制台。

```
bash start.sh

# 登录
[WeCross]> login org2-admin 123456

# 发行资产，资产名为htlc，最小单位1，发行数量100000000
[WeCross.org2-admin]> bcosDeploy payment.bcos.ledger contracts/solidity/
↪LedgerSample.sol LedgerSample 1.0 token htlc 1 100000000

# 资产合约地址需要记录下来
Result: 0xf4fdcdfe0184644f09a1cfa16a945cc71a5d44ff

# 部署htlc合约
[WeCross.org2-admin]> bcosDeploy payment.bcos.htlc contracts/solidity/
↪LedgerSampleHTLC.sol LedgerSampleHTLC 1.0

# htlc合约地址需要记录下来
Result: 0x22a83719f748da09845d91fe1a2f44437f0ad13b
```

资产授权

要完成跨链转账，资产拥有者需要将资产的转移权授权给哈希时间锁合约。

进入BCOS链的WeCross控制台。

```
bash start.sh

# 登录
[WeCross]> login org2-admin 123456

# approve [被授权者地址]（此处为自己的哈希时间锁合约地址），[授权金额]
[WeCross.org2-admin]> sendTransaction payment.bcos.ledger approve␣
↪0x22a83719f748da09845d91fe1a2f44437f0ad13b 1000000

Txhash  : 0x718e948b0ab55697c61675253acfd580104e539c85e0fcb23c0686457ea429d4
BlockNum: 46
Result  : [true]
```

哈希时间锁合约初始化

需要将资产合约的地址和对手方的哈希时间锁合约地址保存到自己的哈希时间锁合约。

进入BCOS链的WeCross控制台。

```
bash start.sh

# 登录
[WeCross]> login org2-admin 123456

# init [己方资产合约地址]
[WeCross.org2-admin]> sendTransaction payment.bcos.htlc init␣
↪0xf4fdcdfe0184644f09a1cfa16a945cc71a5d44ff

Txhash  : 0x7df25ce20e7db6f6bba836bf54c258bb5386873e14b57e74a2371ec367b31779
BlockNum: 51
Result  : [success]

# 查看资产发行者owner的地址
[WeCross.org2-admin]> call payment.bcos.htlc queryAddress

# 该地址需要记录下来，发起转账提案时需要
Result: [0x55f934bcbe1e9aef8337f5551142a442fdde781c]

# 查看owner余额，检查是否初始化成功
[WeCross.org2-admin]> call payment.bcos.htlc balanceOf␣
↪0x55f934bcbe1e9aef8337f5551142a442fdde781c

Result: [100000000]
```

Fabric链前期准备

假设已存在一个跨链账户org1-admin，且分别添加了一个BCOS2.0链账户以及两个不同机构的Fabric1.4链账户。

进入Fabric链的WeCross控制台。

部署合约

```
bash start.sh

# 登录
[WeCross]> login org1-admin 123456

# 切换默认账户
[WeCross.org1-admin]> setDefaultAccount Fabric1.4 2

# 在机构2安装资产合约链码
[WeCross.org1-admin]> fabricInstall payment.fabric.ledger Org2 contracts/chaincode/
↪ledger 1.0 GO_LANG

path: classpath:contracts/chaincode/ledger
Result: Success

# 在机构2安装哈希时间锁合约链码
[WeCross.org1-admin]> fabricInstall payment.fabric.htlc Org2 contracts/chaincode/
↪htlc 1.0 GO_LANG

path: classpath:contracts/chaincode/htlc
Result: Success
```

(下页继续)

(续上页)

```
# 切换默认账户
[WeCross.org1-admin]> setDefaultAccount Fabric1.4 1

# 在机构1安装资产合约链码
[WeCross.org1-admin]> fabricInstall payment.fabric.ledger Org1 contracts/chaincode/
↪ledger 1.0 GO_LANG

path: classpath:contracts/chaincode/ledger
Result: Success

# 在机构1安装哈希时间锁合约链码
[WeCross.org1-admin]> fabricInstall payment.fabric.htlc Org1 contracts/chaincode/
↪htlc 1.0 GO_LANG

path: classpath:contracts/chaincode/htlc
Result: Success

# 实例化链码，发行资产100000000
[WeCross.org1-admin]> fabricInstantiate payment.fabric.ledger ["Org1","Org2"]␣
↪contracts/chaincode/ledger 1.0 GO_LANG default ["token","htlc","100000000"]

Result: Query success. Please wait and use 'listResources' to check.

# 用listResources确认payment.fabric.ledger已实例化完成（约1分钟）
[WeCross.org1-admin]> listResources

path: payment.fabric.ledger, type: Fabric1.4, distance: 0

# 实例化哈希时间锁合约，需要写入[己方资产合约名，channel]
[WeCross.org1-admin]> fabricInstantiate payment.fabric.htlc ["Org1","Org2"]␣
↪contracts/chaincode/htlc 1.0 GO_LANG default ["ledger","mychannel"]

Result: Query success. Please wait and use 'listResources' to check.

# 用listResources确认payment.fabric.htlc已实例化完成（约1分钟）
[WeCross.org1-admin]> listResources

path: payment.fabric.htlc, type: Fabric1.4, distance: 0
```

资产授权

fabric的示例资产合约通过创建一个托管账户实现资产的授权。

进入Fabric链的WeCross控制台。

```
bash start.sh

# 登录
[WeCross]> login org1-admin 123456

# 创建一个托管账户完成授权
[WeCross.org1-admin]> sendTransaction payment.fabric.ledger createEscrowAccount␣
↪1000000

Txhash  : d6a6241cbaac9cad768465213f3462f54c62e32f168c26bcce23d060315f0751
BlockNum: 1097
Result  : [HTLCoin-Admin@org1.example.com-EscrowAccount]

# 查看资产发行者owner的地址
[WeCross.org1-admin]> call payment.fabric.htlc queryAddress

Result: [Admin@org1.example.com]
```

(下页继续)

(续上页)

```
# 查看owner授权后的余额
[WeCross.org1-admin]> call payment.fabric.htlc balanceOf Admin@org1.example.com

# 通过授权的方式已转移1000000到托管账户
Result: [99000000]
```

7.2.2 哈希时间锁合约配置

配置FISCO BCOS端跨链路由 在主配置文件wecross.toml中配置htlc任务。

```
[[htlc]]
     #直连链的哈希时间锁资源路径
    selfPath = 'payment.bcos.htlc'

    #对手方的哈希时间锁资源路径
    counterpartyPath = 'payment.fabric.htlc'
```

重要:

- htlc资源路径请结合实际情况配置。

配置Fabric端跨链路由

在主配置文件wecross.toml中配置htlc任务。

```
[[htlc]]
     #直连链的的哈希时间锁资源路径
    selfPath = 'payment.fabric.htlc'

    #对手方的哈希时间锁资源路径
    counterpartyPath = 'payment.bcos.htlc'
```

重启两个跨链路由

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

7.2.3 发起跨链转账

假设跨链转账发起方是FISCO BCOS的用户，发起方选择一个secret，计算 $$ hash = sha256(secret) $$ 得到hash，然后和Fabric的用户即参与方协商好各自转账的金额、账户以及时间戳。

协商内容

- FISCO BCOS的两个账户：
  - 资产转出者：org2-admin(BCOS链发行资产的地址：0x55f934bcbe1e9aef8337f5551142a442fdde781c)
  - 资产接收者：org1-admin(BCOS链账户：0x2b5ad5c4795c026514f8317c7a215e218dccd6cf)
- FISCO BCOS转账金额：700
- Fabric的两个账户：
  - 资产转出者：org1-admin(Fabric链发行资产的地址：Admin@org1.example.com)
  - 资产接收者：org2-admin(Fabric链账户：User1@org2.example.com)
- Fabric转账金额500

- 哈希: 2afe76d15f32c37e9f219ffd0a8fcefc76d850fa9b0e0e603cbd64a2f4aa670d
- 哈希原像: e7d5c31dcc6acae547bb0d84c2af05413994c92599bff89c4abd72866f6ac5c6（协商时只有发起方知道）
- 时间戳t0：发起方的超时时间，单位s
- 时间戳t1： 参与方的超时时间，单位s

注解:

- 哈希原像和哈希可通过控制台命令 genSecretAndHash 生成。
- 两个时间戳可通过控制台命令 genTimelock 生成。
- 时间戳需要满足条件：t0 - 300 > t1 > now + 600

创建跨链转账提案

两条链的**资产转出者**通过WeCross控制台创建跨链转账提案，将协商的转账信息写入各自的区块链。

- 命令介绍
  - 命令：`newHTLCProposal`
  - 参数：`path,hash，secret，role，sender0，receiver0，amount0，timelock0，sender1，rece`
- 注意事项
  - 其中下标为0的参数是发起方信息。
  - 发起方`secret`传入哈希原像，`role`传入true；参与方`secret`传入null，`role`传入false。
- 启动控制台
- 发起方创建转账提案

该步骤由发起方即BCOS链的资产转出者完成。

进入BCOS链的WeCross控制台。

```
bash start.sh

# 登录
[WeCross]> login org2-admin 123456

[WeCross.org2-admin]> newHTLCProposal payment.bcos.htlc␣
↪edafd70a27887b361174ba5b831777c761eb34ef23ee7343106c0b545ec1052f␣
↪049db09dd9cf6fcf69486512c1498a1f6ea11d33b271aaad1893cd590c16542a true␣
↪0x55f934bcbe1e9aef8337f5551142a442fdde781c␣
↪0x2b5ad5c4795c026514f8317c7a215e218dccd6cf 700 2000010000 Admin@org1.example.com␣
↪User1@org2.example.com 500 2000000000
```

- 参与方创建转账提案

该步骤由参与方即Fabric链的资产转出者完成。

进入Fabric链的WeCross控制台。

```
bash start.sh

# 登录
[WeCross]> login org1-admin 123456

[WeCross.org1-admin]> newHTLCProposal payment.fabric.htlc␣
↪edafd70a27887b361174ba5b831777c761eb34ef23ee7343106c0b545ec1052f null false␣
↪0x55f934bcbe1e9aef8337f5551142a442fdde781c␣
↪0x2b5ad5c4795c026514f8317c7a215e218dccd6cf 700 2000010000 Admin@org1.example.com␣
↪User1@org2.example.com 500 2000000000
```

(下页继续)

(续上页)

结果确认

哈希时间锁合约提供了查询余额的接口，可通过WeCross控制台调用，查看两方的接收者是否到账。

- FISCO BCOS到账确认

```
[WeCross.org2-admin]> call payment.bcos.htlc balanceOf␣
↪0x2b5ad5c4795c026514f8317c7a215e218dccd6cf

Result: [700]
```

- Fabric到账确认

```
[WeCross.org1-admin]> call payment.fabric.htlc balanceOf User1@org2.example.com

Result: [500]
```

注：跨链转账存在交易时延，取决于两条链以及机器的性能，一般需要5~25s完成转账。Fabric链资产转出者Admin@org1.example.com在授权的时候已经把钱转移到了托管账户，因此htlc扣的是托管账户，他的余额不会发生变化。

CHAPTER 8

网页管理平台

WeCross网页管理平台是继WeCross控制台以来，更加便捷、友好的可视化客户端工具，通过发送HTTP请求与WeCross JSON-RPC接口进行跨链路由连接与通信，实现对跨链资源的读写访问等请求。WeCross网页管理平台拥有多维度的操作，包括账户管理、资源管理、路由管理、交易管理以及事务管理等多个管理模块。

可参考WeCross系列文章：五分钟可视化演绎跨链互操作 对网页管理平台的整体流程有大致了解。

本文分为部署手册和使用手册两部分，部署手册介绍WeCross网页管理平台的部署启动方式，使用手册介绍WeCross网页管理平台的实际操作和使用方式。

8.1 1. 部署手册

本手册主要介绍WeCross网页管理平台在使用和编译的环境要求、两种启动方式以及远程访问跨链路由的方式。

8.1.1 1.1 环境要求

- 为了您有最好的操作体验，请尽量使用Chrome浏览器，或使用支持ECMAScript 2015语法的浏览器，可参考链接：caniuse。若使用过时的浏览器，有可能会出现部分功能不可使用的问题 *:(*
- 若您需要进行手动源码编译网页管理平台，请确保在您的编译环境中`node.js`和npm版本支持ECMAScript 2015语法，可参考链接：node.green。推荐版本为`node.js >= 8.10, npm >= 6.4.1`。

node.js及npm版本查看方式：

```
# node version
node -v
v8.16.0
# npm version
npm -v
6.4.1
```

8.1.2 1.2 启动方式

网页管理平台的启动方式分为两种方式：快速体验、手动编译。

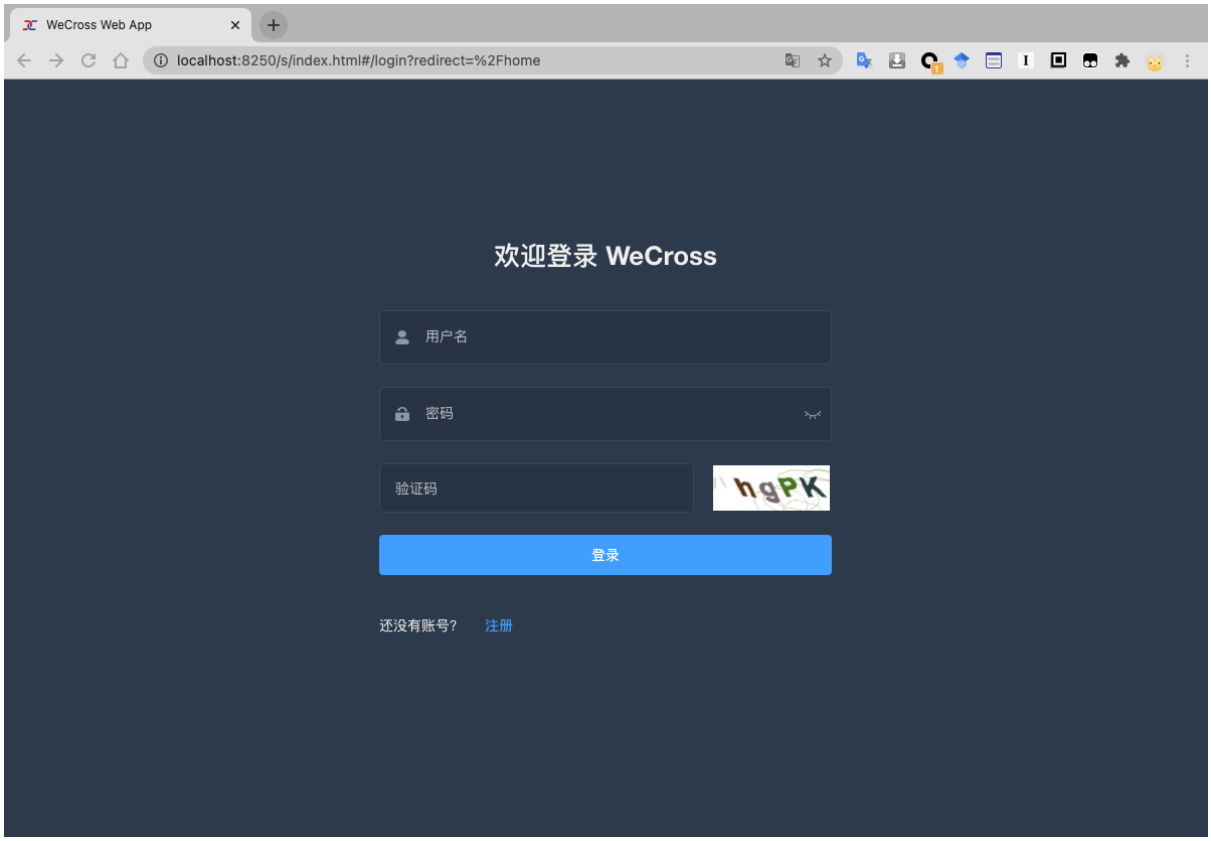
1.2.1 快速体验

若您是从WeCross 快速体验搭建的跨链Demo，那么在Demo中已经默认完成对网页管理平台的下载部署工作，您可以直接从浏览器输入以下地址即可访问：

```
http://localhost:8250/s/index.html#/login
```

其中localhost为跨链路由所配置的IP地址，8250为所访问的跨链路由的端口号，在这里的URL为默认配置，可根据您本地的配置进行修改。

输入后可以看到如下界面：



在快速体验Demo中，已配置默认账号org1-admin，可输入密码123456进行登录体验。

1.2.2 手动编译

若您目前已经有跨链路由正在运行，请确认跨链路由的版本为1.0.0及以上，即可进行手动源码编译网页管理平台，进行跨链路由的访问。主要步骤如下：

1.从远程仓库拉取代码到本地；

```
git clone https://github.com/WeBankBlockchain/WeCross-WebApp.git

# 若因为网络原因出现长时间拉取代码失败，请尝试以下命令：
git clone https://gitee.com/WeBank/WeCross-WebApp.git
```

2.npm安装相关依赖，请注意编译环境要求；

```
cd WeCross-WebApp
npm install
```

3.安装完依赖以后，进行npm编译源代码，编译好的静态文件在dist文件夹中；

```
npm run build:prod
cd dist
```

4.在跨链路由所在文件中，创建pages文件夹，**注意**： 这里以routers-payment/127.0.0.1-8250-25500作为网页管理平台访问的跨链路由所在文件夹，实际操作请以实际情况为准；

```
cd ./routers-payment/127.0.0.1-8250-25500
mkdir -p pages
```

5.将dist文件夹中编译好的静态文件全部拷贝至刚创建的pages文件夹中；

```
cp -r ./WeCross-WebApp/dist/* ./routers-payment/127.0.0.1-8250-25500/pages/
```

6.在浏览器中输入[跨链路由IP]:[跨链路由端口]/s/index.html访问WeCross网页管理平台，请注意浏览器版本要求。

8.1.3 1.3 远程访问

若需要网页管理平台进行远程访问跨链路由，请将跨链路由所在目录的 conf/wecross.toml 文件，修改[rpc]标签下的address字段为所需IP（如：0.0.0.0），再通过以下命令重启当前的路由模块，在浏览器输入远程IP进行访问。

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

8.1.4 1.4 开启HTTPS访问

注意： 请确认跨链路由的配置 wecross.toml中已开启SSL配置：

```
[rpc]
    sslSwitch = 1 # 默认为2
```

修改之后需要重新启动跨链路由。

在浏览器中导入证书，开启HTTPS：

- 根证书位于所连接的跨链路由目录下的conf/ca.crt
- 将ca.crt证书导入到浏览器

这里以Chrome浏览器为例，设置 ==> 隐私设置和安全性 ==> 高级 ==> 管理证书

8.1.5 1.5 可视化管理台变更请求URL前缀

为了运维方便，所有请求都收敛到同一个URL前缀，支持修改URL前缀。

注意： 该功能只支持v1.1.1版本以上的WeCross跨链路由，且只支持源码编译WeCross-WebApp。

修改方式如下，本节以`/wecross/`为例（手动编译方法请参考本文1.2.2小节）：

```
# 手动修改.env.production文件
# vim打开
vim .env.production
```

修改`.env.production`文件，修改规则请参考`dotenv`规则dotenv rules或dotenv rules gitee

```
ENV = 'production'
# base api
# 原配置:
# VUE_APP_BASE_API = '/'
# 修改为
VUE_APP_BASE_API = '/wecross/'
```

重新编译：

```
# 修改后的配置会在编译时注入配置
npm run build:prod
# 在dist生成静态文件，替换router跨链路由的pages中的静态文件，这里以~/wecross-demo/routers-
↪payment/127.0.0.1-8250-25500/为例
cp -r ./dist/* /wecross-demo/routers-payment/127.0.0.1-8250-25500/pages
```

再次访问（这里必须保证router跨链路由也配置了URL前缀，可参考链接router主配置中`urlPrefix`字段）：

在浏览器输入 http://127.0.0.1:8250/wecross/s/index.html 访问。

若需要进行控制台访问修改URL前置的跨链路由router，那么也需要配置控制台的配置，可参考控制台配置中的`urlPrefix`字段。

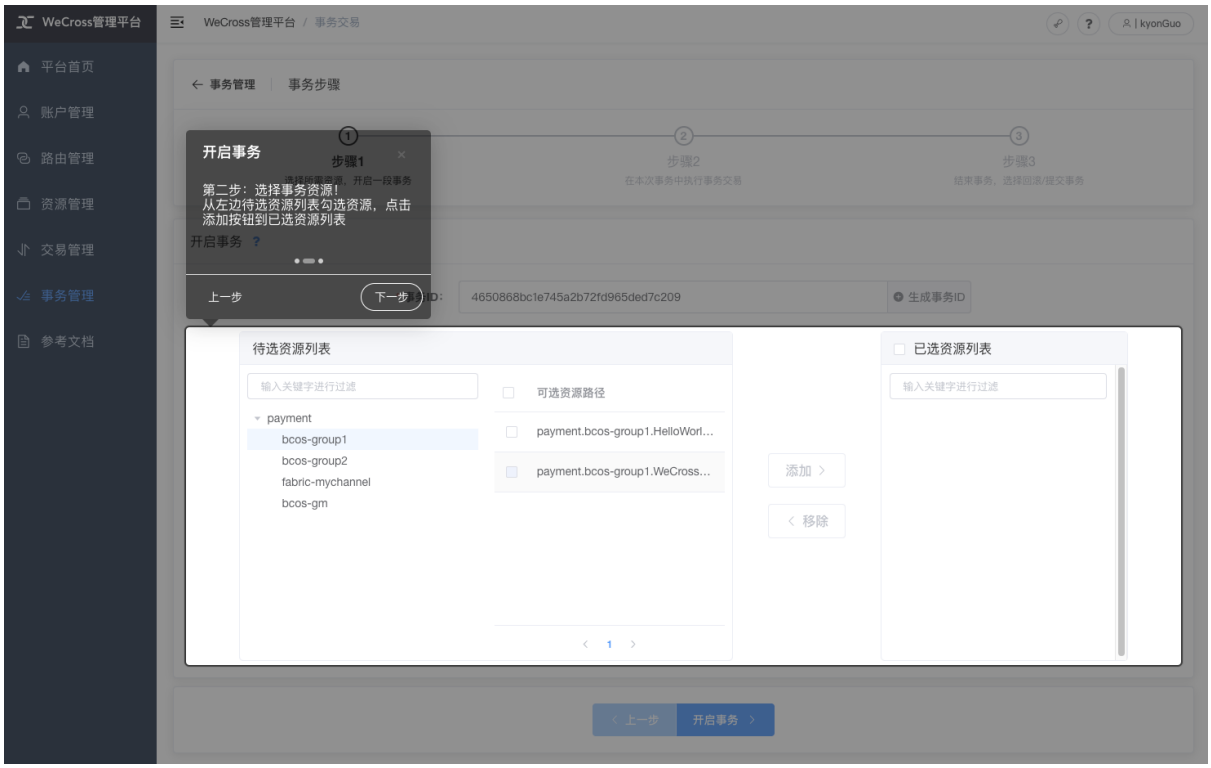
8.2 2. 使用手册

本手册主要介绍WeCross网页管理平台的功能模块，以及各个功能的使用方法。

8.2.1 2.1 新手用户指引

对于初次登录的新手用户，将会弹框引导用户进行操作。

此外，可点击页面中 **"?"** 按钮获取更多帮助信息。



8.2.2 2.2 功能模块

WeCross网页管理平台共包含以下功能模块：

- 登录与注册模块：用户可进行Universal Account（UA）的注册和登录，UA的详细介绍可参考：链接；
- 平台首页展示模块：可根据首页展示的信息大致了解已部署WeCross跨链网络的整体情况；
- 账户管理模块：用户可通过该模块查看当前UA账户信息，并进行链账户的管理；
- 路由管理模块：查看当前跨链路由的Peer路由信息；
- 资源管理模块：可对跨链资源进行查看、部署、调用等操作；
- 交易管理模块：查看所有跨链资源的每一笔交易详情，可进行跨链资源调用；
- 事务管理模块：查看所有跨链事务详情，可在页面开启/执行/结束事务；
- 其他模块：查看参考文档、快速提Issue、修改账号密码、账号登出网页等。

8.2.3 2.3 登录与注册模块

在注册模块，用户可自行申请注册一个UA账号，界面如下图所示，用户填入用户名和密码，并填入正确的验证码即可。若出现验证码不能显示的问题，请尝试刷新页面，或点击验证码手动刷新。

值得注意的是，用户名长度3～18个字符，支持数字、大小写字母、下划线_、连接符-；密码长度6~18个字符，支持数字、大小写字母、特殊字符~!@#$%^&*()，至少包含一个数字和字母。

在已有UA账户的前提下，可进行登录操作，界面如下图所示，用户填入用户名和密码，并填入正确的验证码即可。

8.2.4 2.4 平台首页展示模块

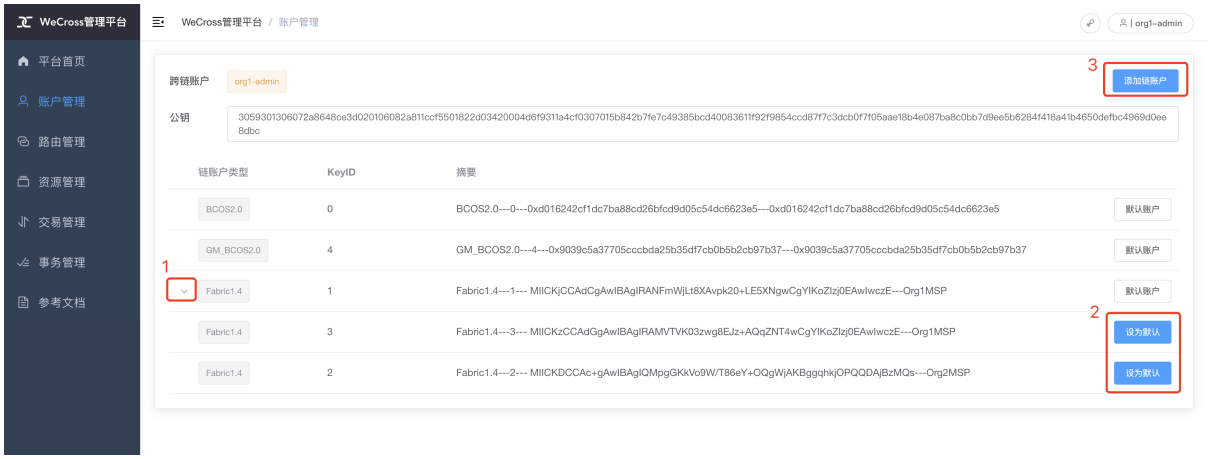
如下图所示，在登录成功后会进入平台首页模块，展示出已部署的WeCross跨链网络的整体信息情况。总共可分为三个部分（在图中已圈出）：

- 在第1块，展示了WeCross网页管理平台所配置的区块链条数、路由个数、总资源个数以及事务功能支持；
- 在第2块，展示了每一条链的简要信息，包括路径、类型和区块高度；
- 在第3块，展示了跨链路由的本地信息，包括运行的环境信息和本地的跨链路由系统信息。

8.2.5 2.5 账户管理模块

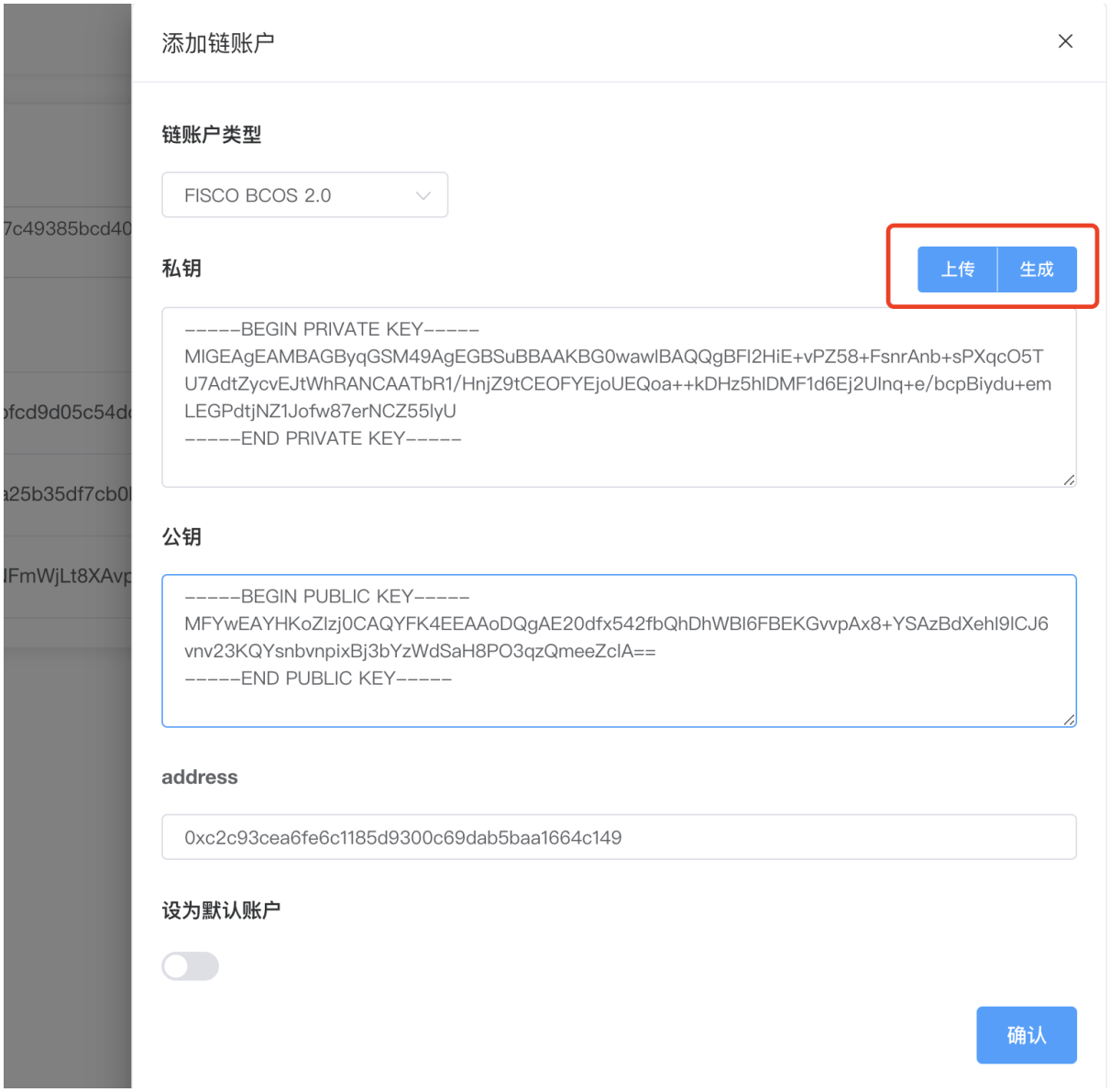
在账户管理模块，用户可查看当前UA的详细信息，包括账户名、账户公钥以及链账户详情，并对链账户进行默认链账户设置、添加链账户、删除链账户的操作。UA账户和链链账户的详细介绍可参考：链接。

如下图所示，每一种链类型的账户都会归为一类，每种链账户都必须包含一个默认账户。可点击图中圈出的红框1，列出该链类型的所有账户；点击红框2中的按钮，可选择链账户为该类型的默认账户。点击红框3将可以进行链账户添加的操作。



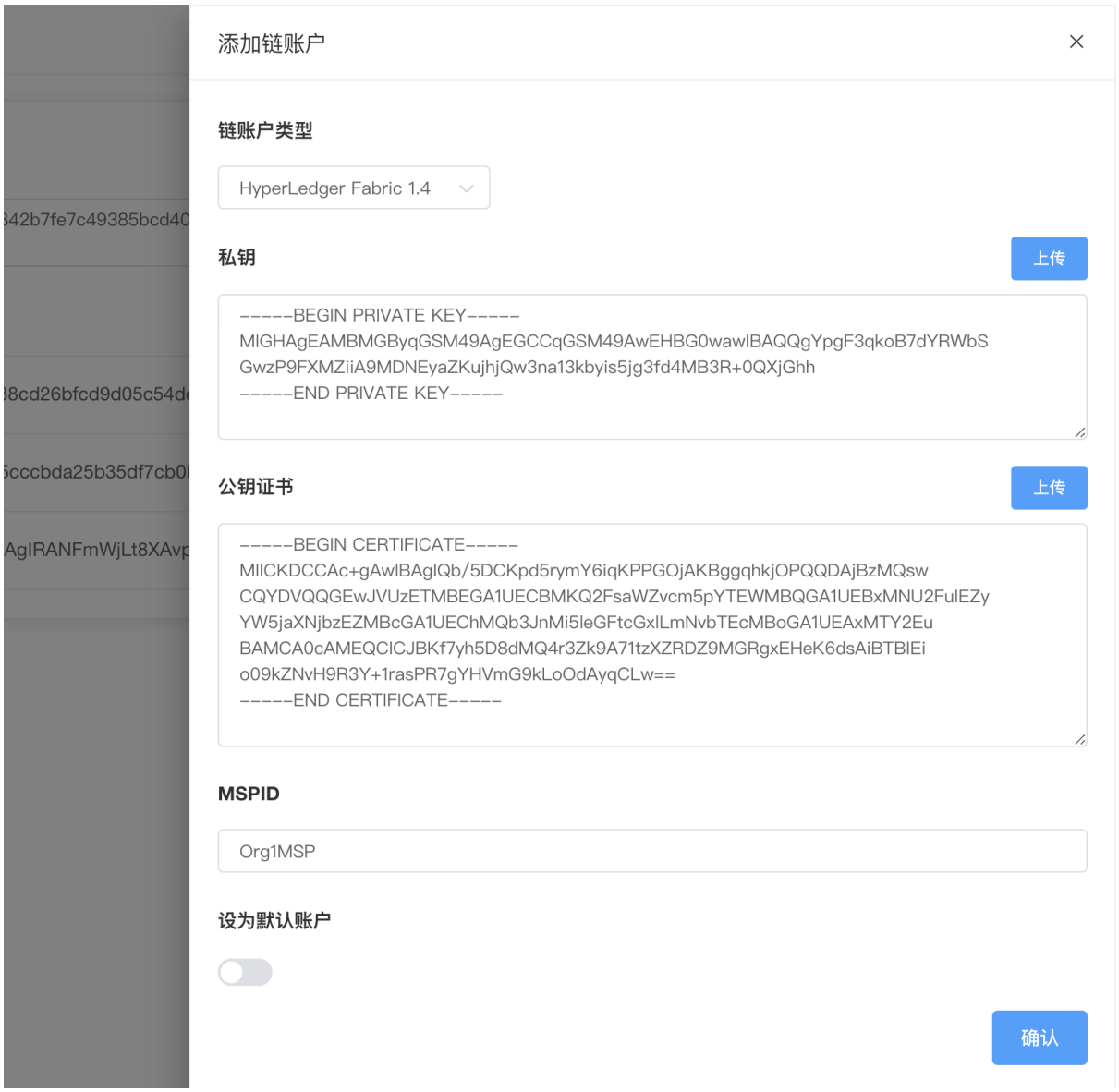
点击上图中红框3之后，将会弹出页面抽屉框，如下图所示。从下拉框选择需要添加的链账户类型，页面将会根据不同链类型生成不同的表单。

添加FISCO BCOS链类型的账户方式如下图所示，可自行选择上传私钥，系统将会自动计算公钥与地址；也可以点击红框内的生成按钮，将会自动为您生成一套私钥、公钥和地址。**注意：** FISCO BCOS账号当前只支持上传PEM格式的私钥。

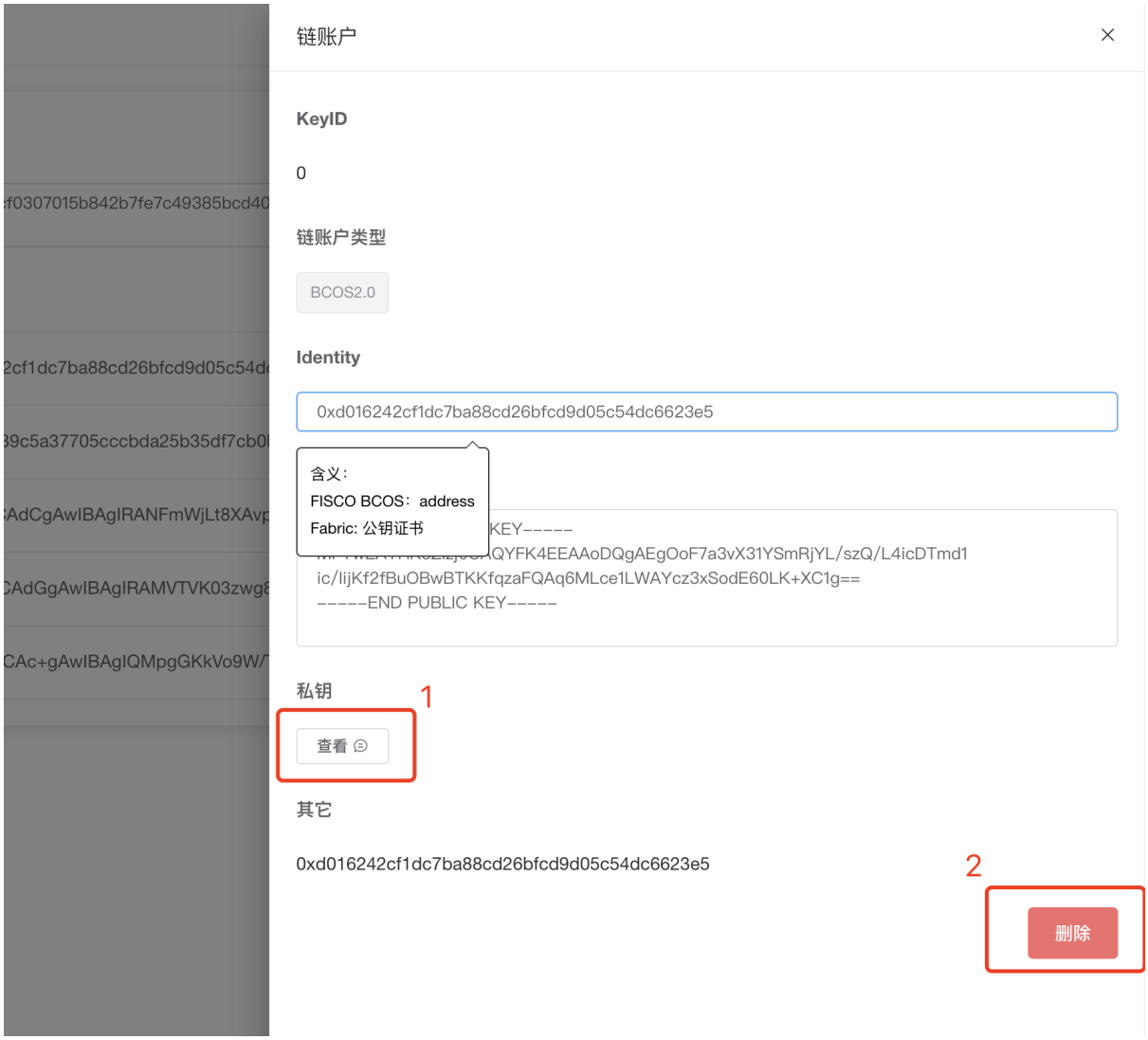


添加Hyperledger Fabric链类型的账户方式如下图所示，目前只支持自行上传私钥、公钥证书的方式添加Fabric链类型账户。**注意：** Hyperledger Fabric账号只支持上传PEM、KeyStore类型的私钥、CRT格式的证书。

MSPID为Fabric链账户所在的MembershipID，请以实际情况为准，这里以Org1MSP为例。（**注意：** 输入错误将导致不能使用该链账户发起Fabric交易调用）

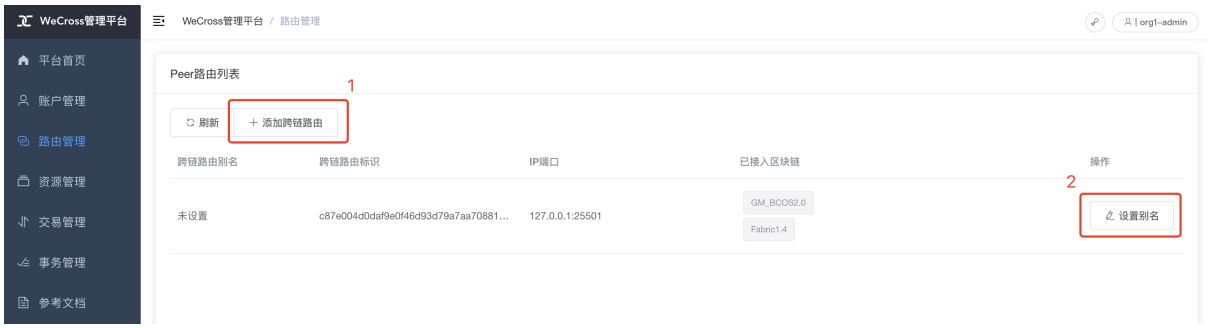


点击链账户中的任意一行，可查看该链账户的详细信息，如下图所示。可点击红框1查看私钥详细信息，点击红框2删除该链账户。



8.2.6 2.6 路由管理模块

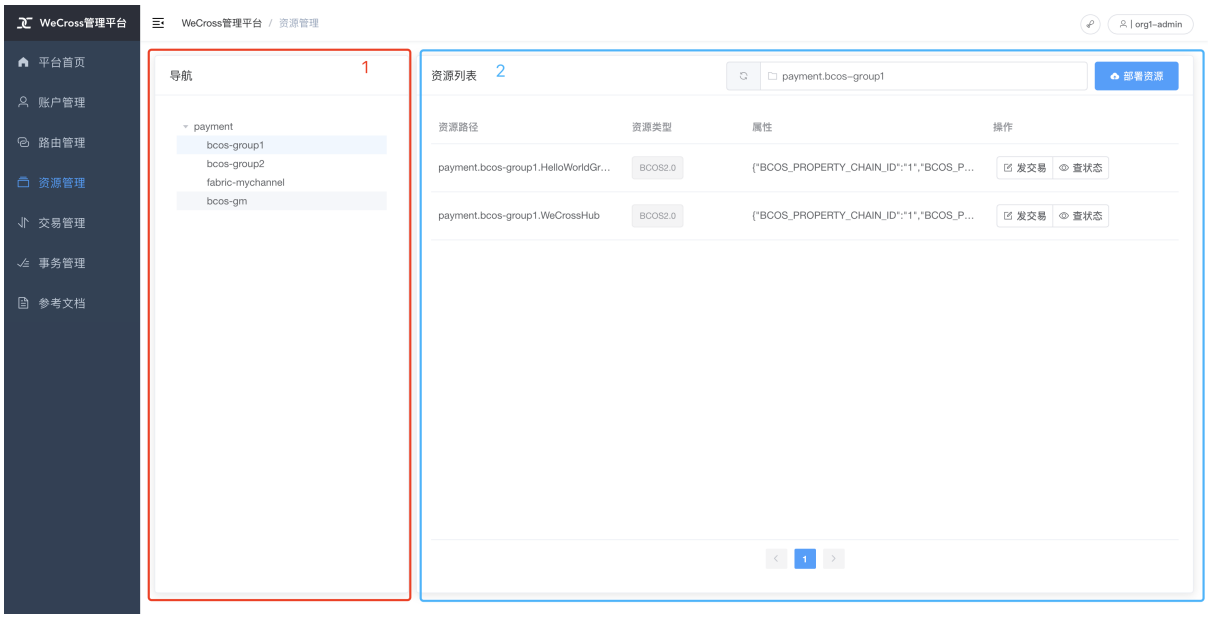
用户可通过该模块查看当前跨链路由的Peer路由信息，添加新的Peer路由信息，以及设置Peer路由的别名。值得注意的是，当一个孤立路由启动时添加Peer路由信息才有效，设置Peer路由别名仅在本机环境有效。



8.2.7 2.7 资源管理模块

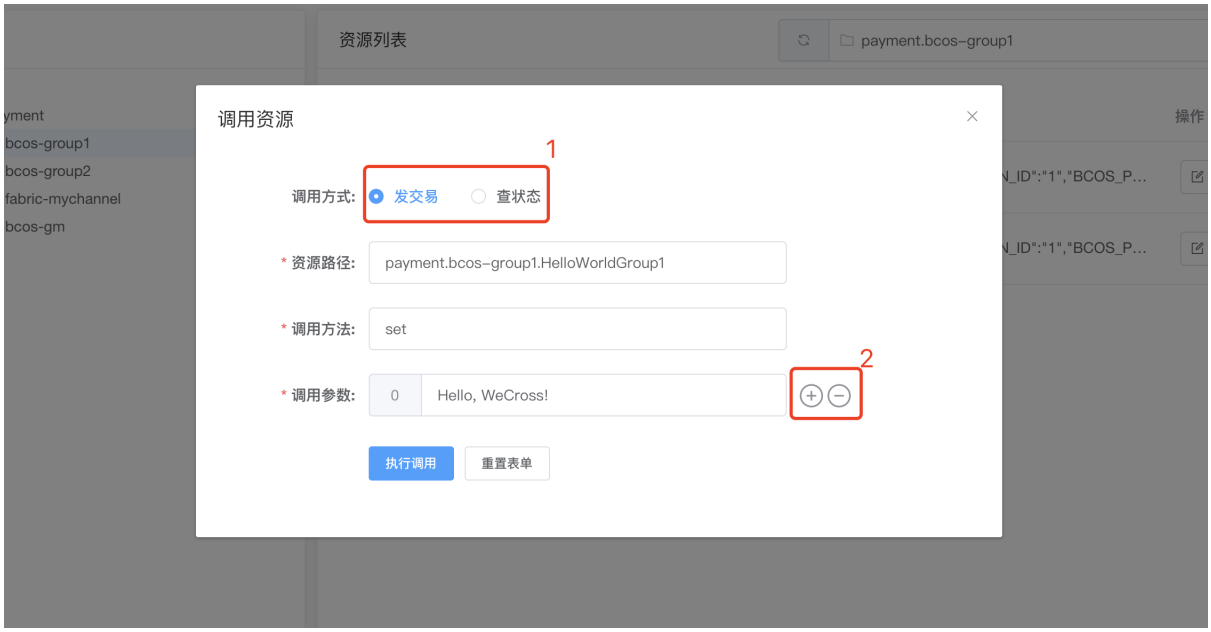
资源管理模块页面分为两大块，如下图所示。左边红框内的是资源导航，从zone到chain的树状结构，类似于Windows的资源管理界面。右边蓝框内的是资源列表视图，左边红框内的资源导航每一次点

击叶子结点表示选定某条链，右边蓝框资源列表视图都会展示出该链的所有资源。



资源列表视图展示出某条链上的所有资源，包括路径、类型、具体属性的字段，可对某个资源进行操作，包括发交易和查状态两种操作。

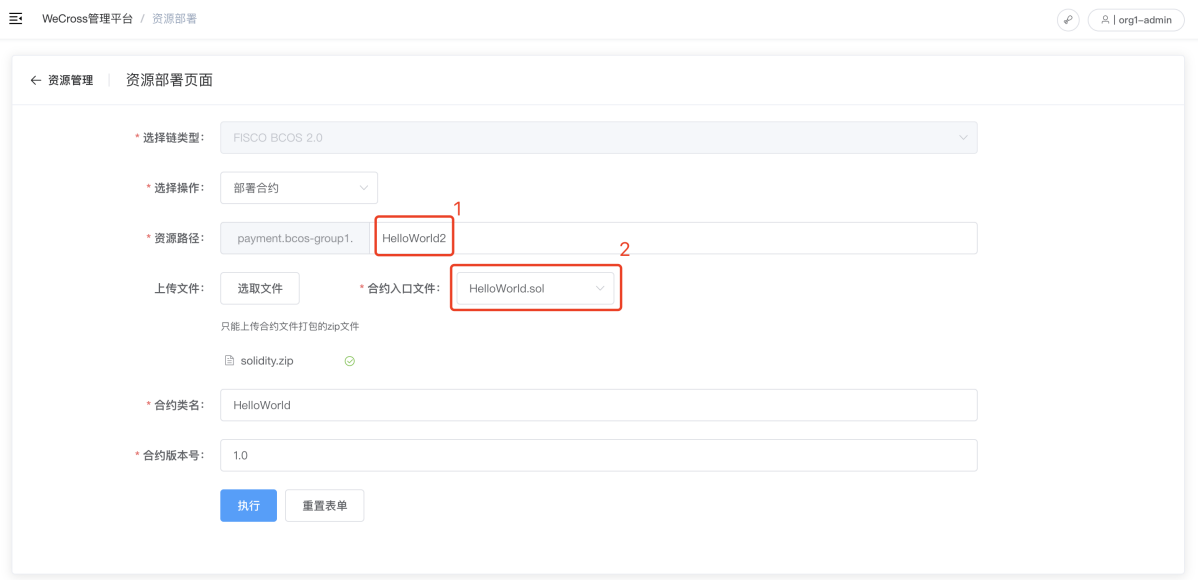
点击按钮即会弹出对话框，可进行调用该资源，如下图所示。在红框1内的单选框，若是发交易，则会发送一个交易，对资源进行写操作；若是读状态，则是对资源的状态进行读操作，不发送交易。在红框2中加减按钮可对调用参数的个数进行修改，可减少至0个，也可增加多个。



在资源管理界面点击部署合约按钮，可进入资源部署页面。资源部署页面会以点击部署合约按钮时选择的链类型改变而改变。目前总共分为两大类链类型：Hyperledger Fabric和FISCO BCOS。

FISCO BCOS 资源部署页面如下图所示，操作可分为部署合约和注册合约。在红框1内填入部署的资源命名。**注意：** 在上传合约文件是，必须上传所有合约文件打包的`ZIP`压缩包，并且必须在文件夹最外层有`.sol`或`.abi`的合约文件。

上传之后，红框2的合约入口文件下拉框会列出压缩包内最外层的所有合约文件，在确认执行部署之后，网页会自动读取合约入口文件，并解析合约之间的依赖，所以 **必须在压缩包内包含入口合约的所有依赖合约** 。在合约类名中填入合约入口文件的类名。



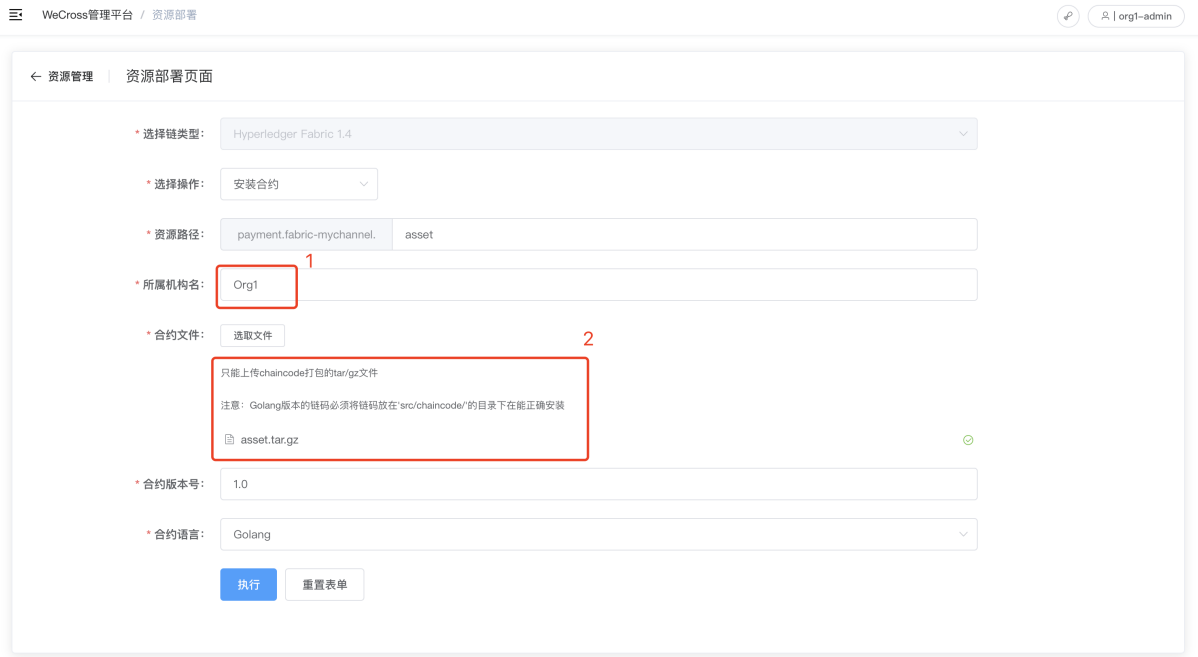
为了用户更好地体验FISCO BCOS部署流程，我们已准备打包压缩好的FISCO BCOS Solidity合约包，供用户下载体验：

Hyperledger Fabric 部署页面如下图所示，Fabric的操作分为安装合约、实例化合约以及升级合约。在安装合约时，填入的所属机构名为当前UA的Fabric1.4类型的默认链账户所属的机构。

特别注意： 由于Fabric的原因，在安装合约的时候，必须符合以下条件：

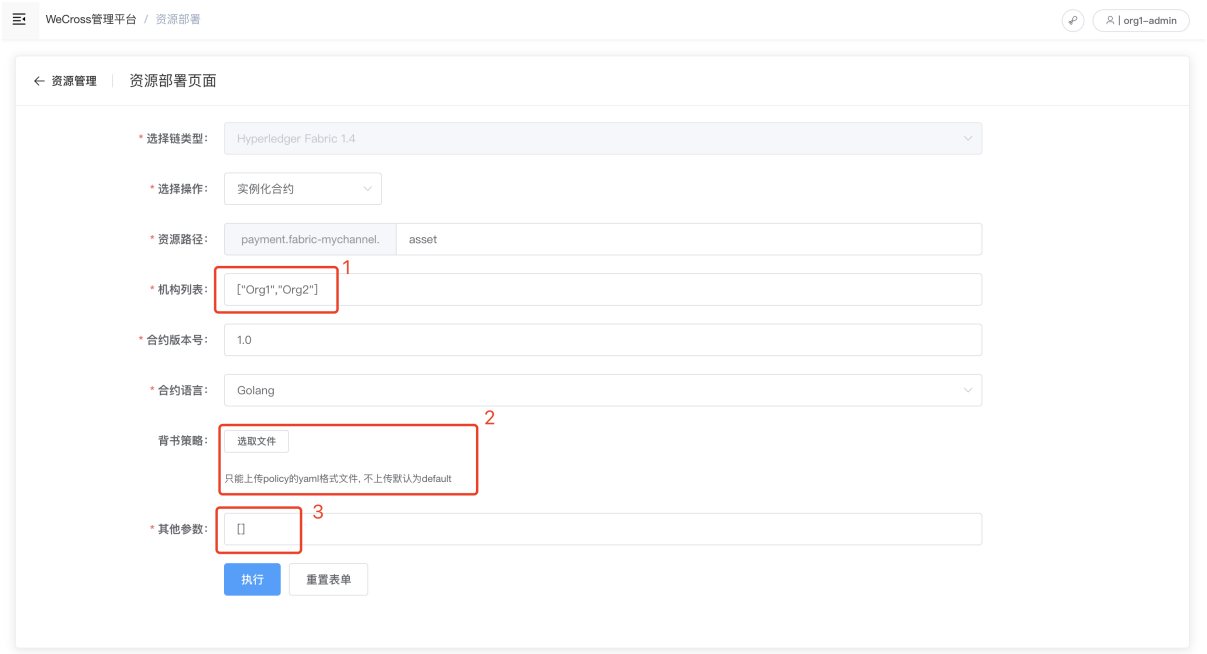
1. 上传 **GZIP** 格式压缩打包的chaincode文件，即`tar.gz`压缩格式；
2. chaincode合约 **必须** 在文件夹`src/chaincode`目录下再进行压缩打包，否则将会出现chaincode合约找不到的问题。例如图中的`asset.tar.gz`，解压缩后文件结构如下：

```
tree -L 3 src
src
└── chaincode
    └── assetSample.go
```



因为Hyperledger Fabric的特性，安装的合约必须实例化/升级实例化才能正常地显示在资源列表中。Hyperledger Fabric 实例化/升级实例化页面如下图所示。机构列表应填入所有参与实例化机构的

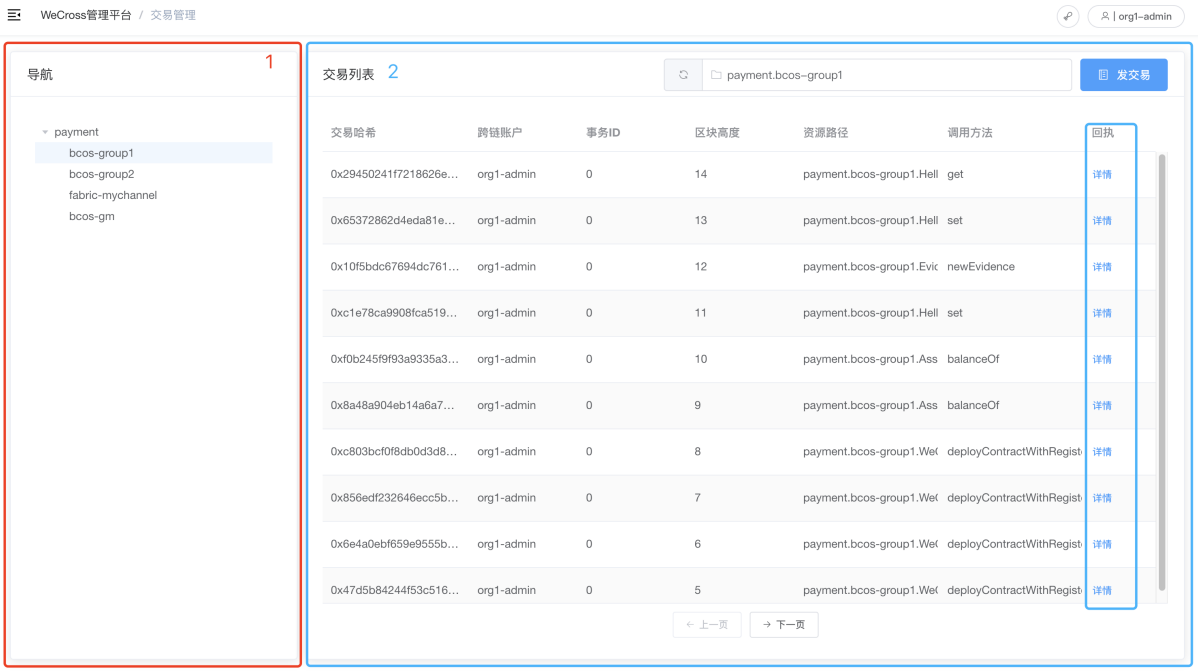
列表，必须以JSON数组的形式填入，其他参数也同理。背书策略可上传YAML格式的背书文件，若不上传则默认为空。



为了用户更好地在网页管理平台体验Hyperledger Fabric部署流程，我们已额外准备打包压缩好的Fabric部署流程，我们已准备打包压缩好的FISCO chaincode合约包，供用户下载体验：

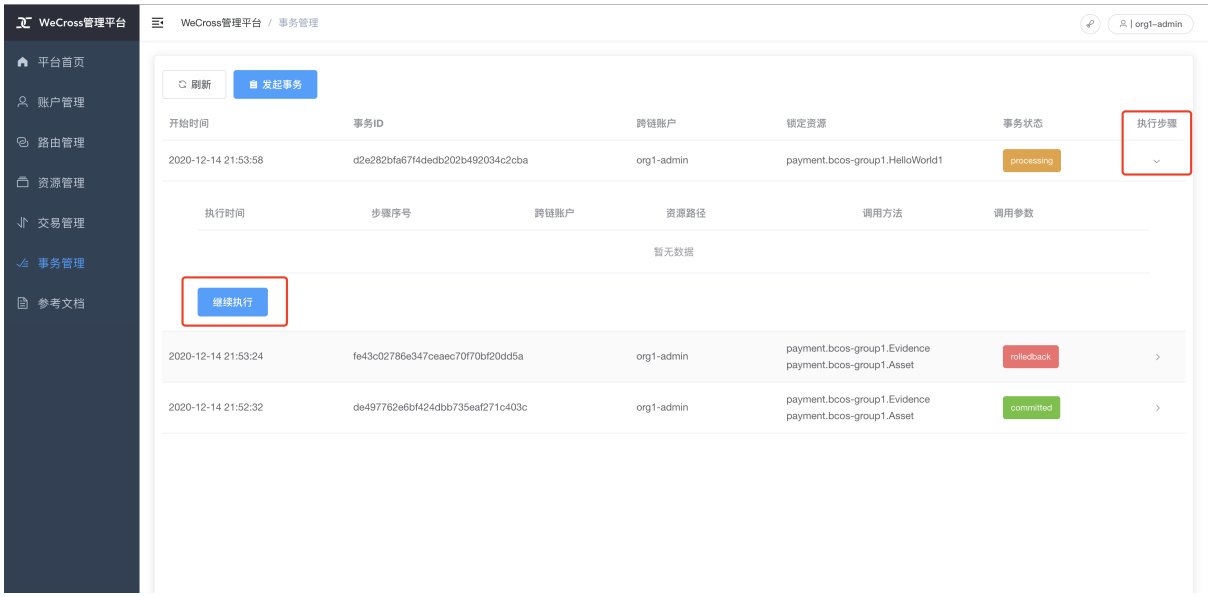
8.2.8 2.8 交易管理模块

用户可通过交易管理模块查看所有跨链资源的每一笔交易详情，进行跨链资源调用。界面如下图所示，与资源管理模块相似，分为两大块，右边蓝框1内为交易列表视图，展示所选定的链所有交易信息，包含交易哈希、账户、区块高度、资源、调用方法和回执等信息，点击蓝框2内可查看这一行交易的回执详细信息。

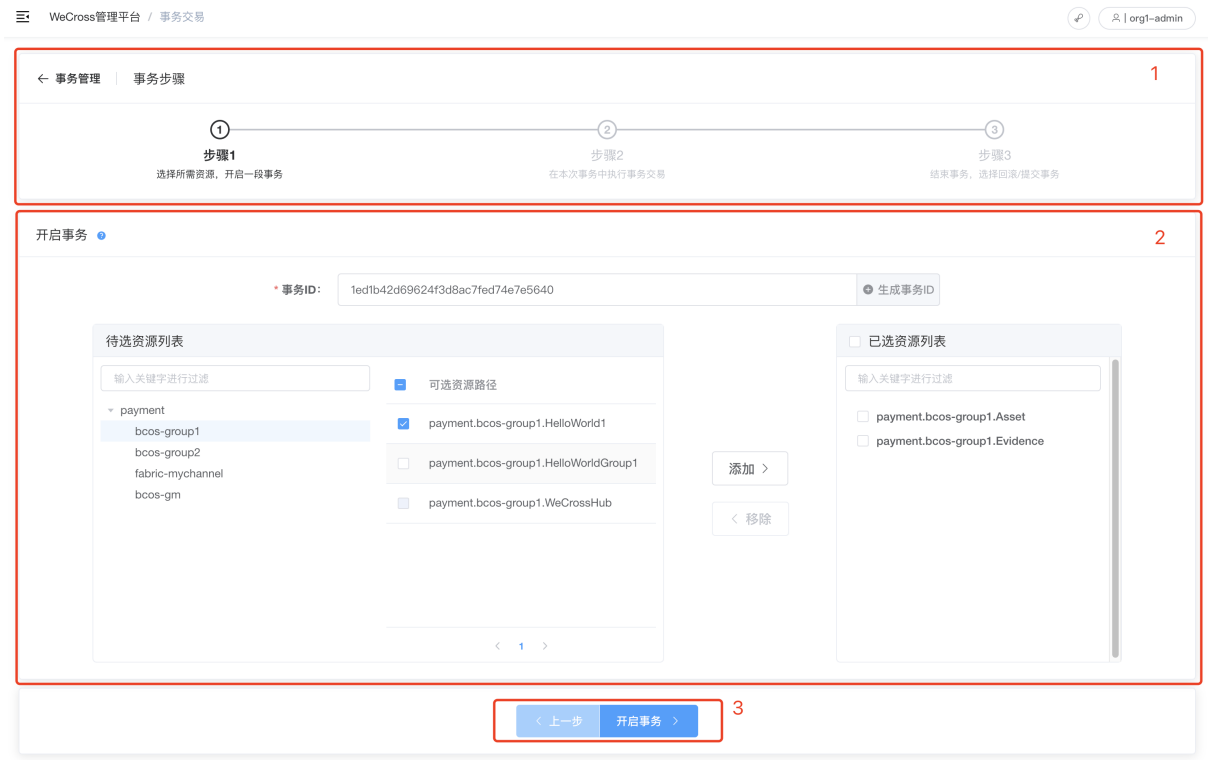


8.2.9 2.9 事务管理模块

用户可通过事务管理模块查看所有跨链事务详情，可在页面开始、执行、提交/回滚事务。界面如下图所示，界面展示了当前WeCross网络的事务列表。列表包含事务开始时间、事务ID、跨链账户、锁定资源、事务状态的字段，事务状态包含`processing`、`rolledback`和`committed`三种状态。点击执行步骤，可查看该事务的详细步骤。若状态为`processing`，则可以点击继续执行按钮，恢复到该事务上下文。

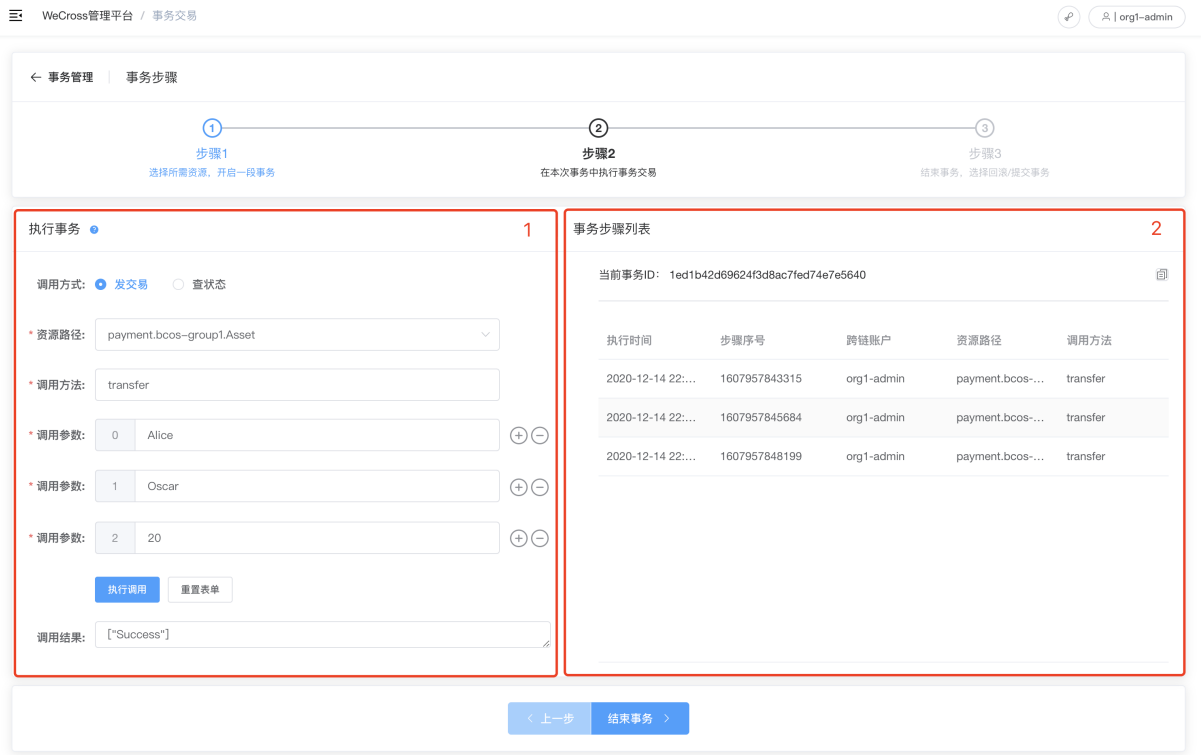


在事务管理页面点击发起事务，则可以进入事务开启页面。事务开启页面如下所示，分为三大块，红框1展示了事务步骤状态；红框2中是开启事务参数的选择，页面将自动生成一个UUID作为事务ID，使用待选资源穿梭框勾选开启事务所需要的资源，添加到右侧的已选资源列表；完成选择后点击红框3开启事务。

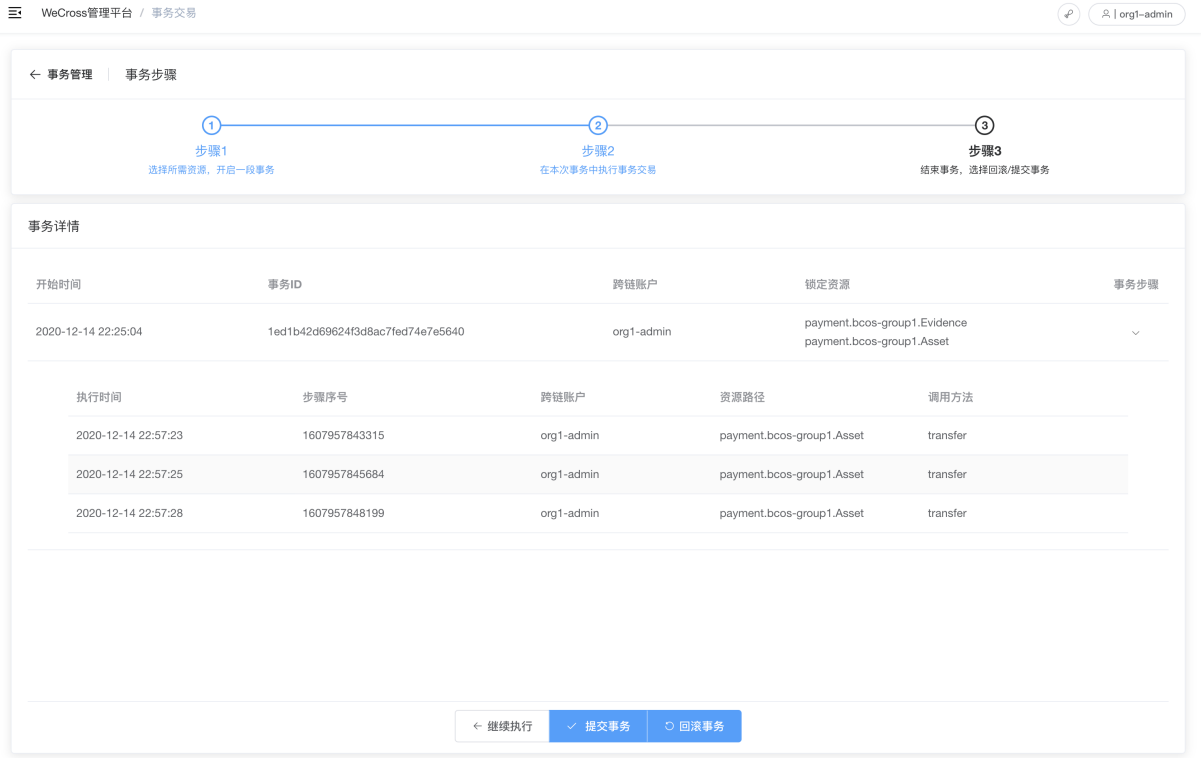


开启事务后进入执行事务步骤，界面如下图所示，左边为发交易表单，与资源管理模块的发交易类似，左边为事务步骤的列表，记录每一步的操作。 若希望结束当前事务，即提交/回滚当前事务，可点击最

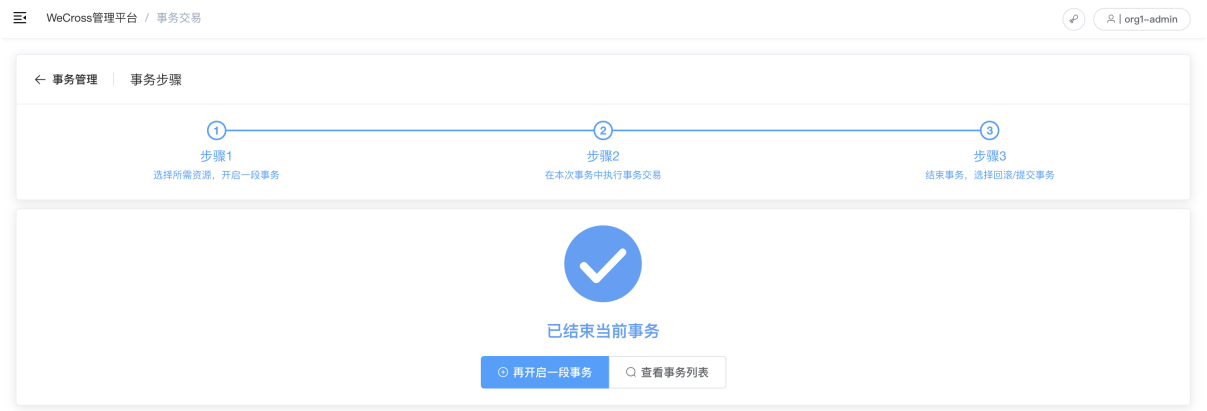
下方的结束事务按钮。



在点击结束事务后进入结束事务步骤，界面如下图所示，展示该事务的所有详细信息。可点击下方的继续执行，返回执行事务步骤，也可选择提交/回滚事务，对事务进行结束。

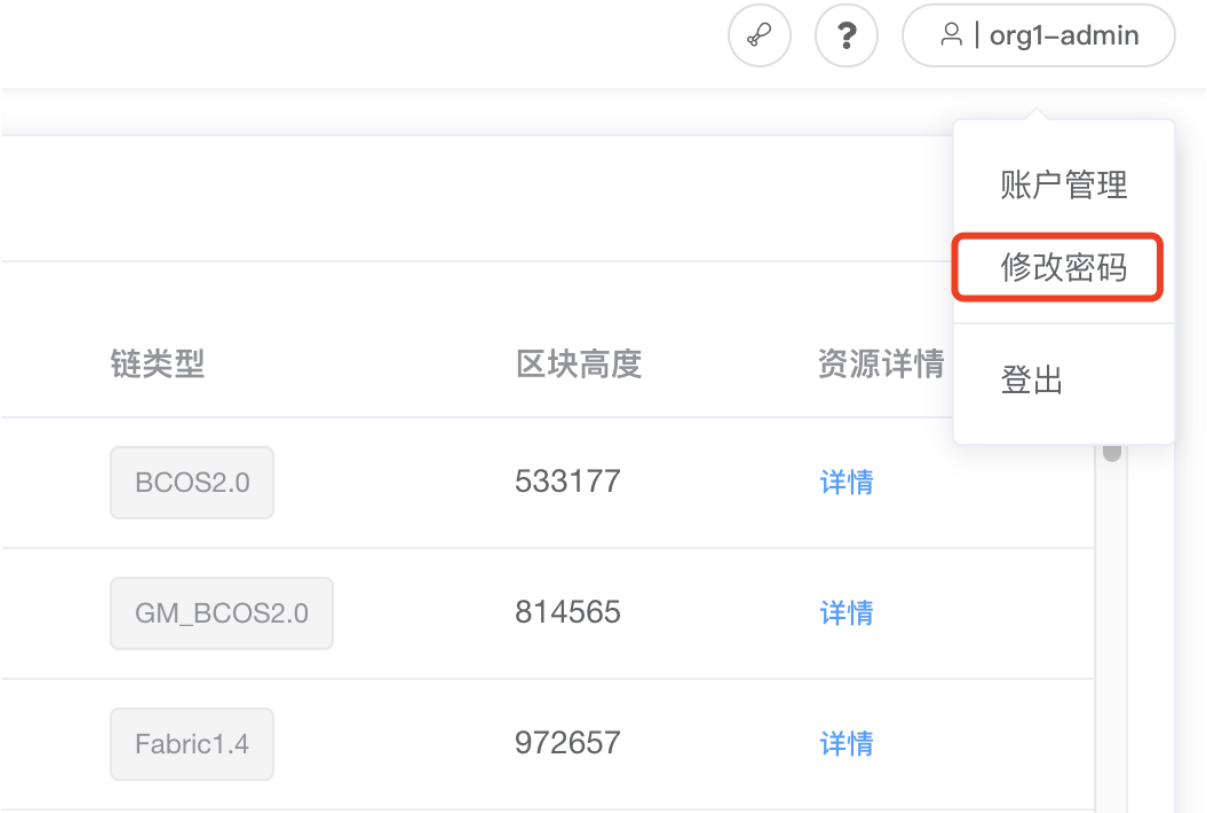


若一切正常，则会展示已结束当前事务界面。可选择点击再开启一段事务按钮再次重启事务，也可点击查看事务列表按钮，跳转到事务管理页面。



8.2.10 2.10 其他模块

可通过点击界面右上角的用户按钮，将弹出下拉框，可点击修改密码选项进入修改密码界面。



进入设置新的密码页面，输入旧密码、重复输入新密码，再输入验证码即可设置新密码。新密码不能与旧密码相同，新密码格式与注册密码一致。

通过点击界面右上角的鸡腿按钮，可打开`Contributor`下拉框。

欢迎通过以上按钮，对我们WeCross的产品进行提Issue、提Bug、提需求，也欢迎各位参与到我们的社区中，来领取Task任务，打怪升级，赢得任务奖励！:)

详情可参考：参与社区

CHAPTER 9

终端控制台

控制台是WeCross重要的客户端工具，它基于WeCross-Java-SDK与WeCross跨链路由建立连接，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，调用跨链资源，发起事务等等。

WeCross控制台源码访问链接：

- GitHub访问链接
- GitHub访问链接

9.1 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。

9.1.1 1.1 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。**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。

9.1.2 1.2 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，比如将跨链资源路径赋值给变量，初始化一个类，并用`.command`的方式访问方法。 详见：交互式命令

9.2 2. 常用命令链接

9.2.1 2.1 普通命令

- 账号操作

  - *login*：在当前控制台登录全局账号
  - *logout*：控制台登出
  - *registerAccount*：注册全局账号
  - *addChainAccount*：在当前全局账号添加一个链账号
  - *setDefaultAccount*：设定某个账号为这个链类型的交易发送默认账号
  - *listAccount*：查看当前全局账号的详细信息
- 状态查询
  - *listResources*：查看资源列表
  - *detail*：查看资源详情
  - *supportedStubs*：查看连接的router支持接入的链类型
- 资源调用
  - *call*：调用链上资源，用于查询，不触发出块
  - *sendTransaction*：发交易，用于改变链上资源，触发出块
  - *invoke*：功能等同*sendTransaction*，在跨链事务时自动转化为命令*execTransaction*
- 资源部署
  - BCOS：*bcosDeploy*、*bcosRegister*
  - Fabric：*fabricInstall*、*fabricInstantiate*、*fabricUpgrade*
- 跨链事务
  - *startTransaction*：开始两阶段事务
  - *execTransaction*：发起事务交易
  - *callTransaction*：读取事务过程中的数据
  - *commitTransaction*：提交事务，确认事务执行过程中所有的变动
  - *rollbackTransaction*：撤销本次事务的所有变更时
  - *loadTransaction*：恢复到某个正在执行的事务上下文
- 跨链转账
  - *newHTLCProposal*：创建转账提案

9.2.2 2.2 交互式命令

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

9.3 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
- `tab`：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

9.4 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。

9.5 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 。

9.5.1 5.1 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/WeBankBlockchain/WeCross-Console/releases/
↪download/resources/download_console.sh)

# 若因网络原因出现长时间下载控制台失败，可尝试以下命令：
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_
↪console.sh)
```

执行成功后，会生成`WeCross-Console`目录，结构如下：

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

9.5.2 5.2 配置控制台

控制台配置文件为 `conf/application-toml`，启动控制台前需配置

```
cd ~/wecross/WeCross-Console
# 拷贝配置sample
cp conf/application-sample.toml conf/application.toml

# 拷贝连接跨链路由所需的证书
cp ~/wecross/routers-payment/cert/sdk/* conf/ # 包含：ca.crt、node.nodeid、ssl.
↪crt、ssl.key

# 配置
vim conf/application.toml
```

配置控制台与某个router的连接

```
[connection]
    server =  '127.0.0.1:8250' # 对应router的ip和rpc端口
    sslKey = 'classpath:ssl.key'
    sslCert = 'classpath:ssl.crt'
    caCert = 'classpath:ca.crt'
    sslSwitch = 2 # disable ssl:2, SSL without client auth:1 , SSL with client and␣
↪server auth: 0
    # urlPrefix = '/' # v1.1.1新增配置，使用该配置可访问已配置urlPrefix的router跨链路由
# 可配置全局账号密码，执行命令`login`时可不输入账号密码
[login]
    username = 'username'
    password = 'password'
```

urlPrefix字段为v1.1.1新增配置，使用该配置可访问已配置urlPrefix的router跨链路由。（这里必须保证router跨链路由也配置了URL前缀，可参考链接router主配置中urlPrefix字段）

若需要进行控制台访问修改URL前置的跨链路由router，那么也需要配置控制台的配置，可参考网页管理平台配置中的配置方法。

9.5.3 5.3 启动控制台

在跨链路由已经启动的情况下，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(v1.2.1)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.

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

9.6 6. 普通命令

注： 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。

9.6.1 help

输入help或者h，查看控制台所有的命令。

```
[WeCross]> help
--------------------------------------------------------------------------------------
↪----------
quit                            Quit console.
registerAccount                 Register a Universal Account.
login                           Login SDK if you have already registered.
logout                          Logout SDK.
addChainAccount                 Add a Chain Account to your Universal Account.
setDefaultAccount               Set the chain account to be the default account␣
↪to send transaction.
supportedStubs                  List supported stubs of WeCross router.
listAccount                     List your Universal Account's information.
listLocalResources              List local resources configured by WeCross server.
listResources                   List all resources including remote resources.
detail                          Get resource information.
call                            Call constant method of smart contract.
```

(下页继续)

(续上页)

```
invoke                          Call non-constant method of smart contract, will␣
↪auto-transfer to command execTransaction during transaction.
sendTransaction                 Call non-constant method of smart contract.
callTransaction                 Call constant method of smart contract during␣
↪transaction.
execTransaction                 Call non-constant method of smart contract during␣
↪transaction.
startTransaction                Start an xa transaction.
commitTransaction               Commit an xa transaction.
rollbackTransaction             Rollback an xa transaction.
loadTransaction                 Load a specified transaction context.
getXATransaction                Get info of specified XA transaction.
listXATransactions              List XA transactions in route.
bcosDeploy                      Deploy contract in BCOS chain.
bcosRegister                    Register contract abi in BCOS chain.
fabricInstall                   Install chaincode in fabric chain.
fabricInstantiate               Instantiate chaincode in fabric chain.
fabricUpgrade                   Upgrade chaincode in fabric chain.
genTimelock                     Generate two valid timelocks.
genSecretAndHash                Generate a secret and its hash.
newHTLCProposal                 Create a htlc transfer proposal .
checkTransferStatus             Check htlc transfer status by hash.
getCurrentTransactionID         Get Current xa Transaction ID.
WeCross.getResource             Init resource by path, and assign it to a custom␣
↪variable.
[resource].[command]            Equal to: 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：

```
[WeCross]> detail -h
-------------------------------------------------------------------------------
↪----------
Get the resource information
Usage: detail [path]
-------------------------------------------------------------------------------
↪----------
```

9.6.2 supportedStubs

显示router当前支持的插件列表。

```
[WeCross]> supportedStubs
[BCOS2.0, GM_BCOS2.0, Fabric1.4]
```

9.6.3 login

在当前控制台登录全局账号。

参数：

- username：(可选) 已注册的全局账号。
- password：(可选) 账号密码。

```
[WeCross]> login org1-admin 123456
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...

# 当在conf/application.toml已经配置[login]，可直接输入login命令进行登录，无需参数
[WeCross]> login
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...

# 当在conf/application.toml未配置[login]，也可直接输入login命令进行登录，逐行输入账号密码
[WeCross]> login
username: org1-admin
password:
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...

# 登录成功后WeCross控制台的Prompt将会加上当前登录的账号名
[WeCross.org1-admin]>
```

9.6.4 logout

在当前控制台登出账号。

```
[WeCross.org1-admin]> logout
Result: success
```

9.6.5 registerAccount

注册全局账号。

参数:

- username：(可选) 已注册的全局账号。
- password：(可选) 账号密码。

```
[WeCross]> registerAccount org2-admin 123456
Result: success

# 也可直接输入registerAccount命令进行登录，逐行输入账号密码
[WeCross]> registerAccount
tips: username can contain alphabet, digit and some special characters: [-_]
      and the length is in range [4,16].
username: test1
tips: password can contain alphabet, digit and some special characters: [@+!%*#?]
      and the length is in range [1,16].
password:
Result: success
```

9.6.6 addChainAccount

在当前全局账号添加一个链账号。

参数：

- chainType：链类型，目前包括 [Fabric1.4, BCOS2.0, GM_BCOS2.0] 三种类型。
- firstKeyPath：第一个密钥/证书路径，以填入的链类型进行区分：
  - 若是FISCO BCOS类型的链账号，第一个密钥路径则是BCOS链账号的公钥（绝对/相对）路径；
  - 若是Hyperledger Fabric类型的链账号，第一个证书路径则是Fabric链账号的证书（绝对/相对）路径；
- secondKeyPath：第二个密钥路径，以填入的链类型进行区分：
  - 若是FISCO BCOS类型的链账号，第二个密钥路径则是BCOS链账号的私钥（绝对/相对）路径；
  - 若是Hyperledger Fabric类型的链账号，第二个密钥路径则是Fabric链账号的密钥（绝对/相对）路径；
- extraData：额外数据，以填入的链类型进行区分：
  - 若是FISCO BCOS类型的链账号，额外数据则是BCOS链账号的地址（address）；
  - 若是Hyperledger Fabric类型的链账号，额外数据则是Fabric链账号所属的机构；
- isDefault：是否将新的链账号作为该类型的默认账号。

```
# 添加BCOS链账号
[WeCross.org1-admin]> addChainAccount BCOS2.0 conf/accounts/bcos_user/
↪0xd06636e57de535c8d148662189cb6c3d16e7a47b.public.pem  conf/accounts/bcos_user/
↪0xd06636e57de535c8d148662189cb6c3d16e7a47b.pem␣
↪0xd06636e57de535c8d148662189cb6c3d16e7a47b false
Result: success
Universal Account info has been changed, now auto-login again.
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...

#添加Fabric链账号，机构名请以实际为准，这里的机构名以Org1MSP为例

[WeCross.org1-admin]> addChainAccount Fabric1.4 conf/accounts/fabric_admin/account.
↪crt conf/accounts/fabric_admin/account.key Org1MSP true
Universal Account info has been changed, now auto-login again.
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

9.6.7 setDefaultAccount

设定某个账号为这个链类型的交易发送默认账号。

参数：

- chainType：链类型，目前包括 [Fabric1.4, BCOS2.0, GM_BCOS2.0] 三种类型。

- keyID：指定账号的ID，可通过*listAccount*命令获取。

```
[WeCross.org1-admin]> setDefaultAccount Fabric1.4 2
Result: success
Universal Account info has been changed, now auto-login again.
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
```

9.6.8 listAccount

查看当前全局账号的详细信息。

参数：

- -d：(可选) 可返回全局账号的更加详细的信息，包括链账号公钥/证书信息，全局账号公钥信息

```
[WeCross.org1-admin]> listAccount
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...
chainAccounts: [
  BCOS2.0 Account:
  keyID    : 0
  type     : BCOS2.0
  address  : 0xe6a68370212863821a5c209c5b66f4ba40e7c80e
  isDefault: true
  ----------
  Fabric1.4 Account:
  keyID    : 1
  type     : Fabric1.4
  MembershipID : Org1MSP
  isDefault: true
  ----------
  Fabric1.4 Account:
  keyID    : 3
  type     : Fabric1.4
  MembershipID : Org1MSP
  isDefault: false
  ----------
  Fabric1.4 Account:
  keyID    : 2
  type     : Fabric1.4
  MembershipID : Org2MSP
  isDefault: false
  ----------
]

# 加上`-d`
[WeCross.org1-admin]> listAccount -d
Universal Account:
username: org1-admin
pubKey  :␣
↪3059301306072a8648ce3d020106082a811ccf5501822d03420004a313bc3d08a93a837d81b8d38fb9172fa34c1c71d
uaID    :␣
↪3059301306072a8648ce3d020106082a811ccf5501822d03420004a313bc3d08a93a837d81b8d38fb9172fa34c1c71d
chainAccounts: [
```

(下页继续)

(续上页)

```
  BCOS2.0 Account:
  keyID    : 0
  type     : BCOS2.0
  pubKey   : -----BEGIN PUBLIC KEY-----
MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEdf3y930w4P/licg+X0qhrK7DOpCwzugj
ZS+aX511oBYULVeSihr5dnsl0uNHpIhG6q6TSYDnKF0UxqccbZKy/g==
-----END PUBLIC KEY-----

  address  : 0xe6a68370212863821a5c209c5b66f4ba40e7c80e
  isDefault: true
  ----------
  Fabric1.4 Account:
  keyID    : 1
  type     : Fabric1.4
  cert     : -----BEGIN CERTIFICATE-----
MIICKTCCAdCgAwIBAgIRAPlePqCuLU5YTOpmfTIIkicwCg... #省略
-----END CERTIFICATE-----

  MembershipID : Org1MSP
  isDefault: true
  ----------
  Fabric1.4 Account:
  keyID    : 3
  type     : Fabric1.4
  cert     : -----BEGIN CERTIFICATE-----
MIICKzCCAdGgAwIBAgIRAN44JEay7bkhHKCbD9n0kKUwCg... #省略
-----END CERTIFICATE-----

  MembershipID : Org1MSP
  isDefault: false
  ----------
  Fabric1.4 Account:
  keyID    : 2
  type     : Fabric1.4
  cert     : -----BEGIN CERTIFICATE-----
MIICKTCCAdCgAwIBAgIRAPyuc+VKfnYg+HTl5PvUn9EwCg... #省略
-----END CERTIFICATE-----

  MembershipID : Org2MSP
  isDefault: false
  ----------
]
```

9.6.9 listLocalResources

显示router配置的跨链资源。

```
[WeCross.org1-admin]> listLocalResources
path: payment.bcos.HelloWorld, type: BCOS2.0, distance: 0
path: payment.bcos.htlc, type: BCOS2.0, distance: 0
path: payment.bcos.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.bcos.evidence, type: BCOS2.0, distance: 0
path: payment.bcos.ledger, type: BCOS2.0, distance: 0
path: payment.bcos.asset, type: BCOS2.0, distance: 0
total: 6
```

9.6.10 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。

```
[WeCross.org1-admin]> listResources
path: payment.fabric.WeCrossHub, type: Fabric1.4, distance: 1
path: payment.bcos.htlc, type: BCOS2.0, distance: 0
path: payment.fabric.ledger, type: Fabric1.4, distance: 1
path: payment.bcos.WeCrossHub, type: BCOS2.0, distance: 0
path: payment.fabric.htlc, type: Fabric1.4, distance: 1
path: payment.bcos.evidence, type: BCOS2.0, distance: 0
path: payment.bcos.ledger, type: BCOS2.0, distance: 0
path: payment.fabric.sacc, type: Fabric1.4, distance: 1
path: payment.fabric.evidence, type: Fabric1.4, distance: 1
path: payment.bcos.asset, type: BCOS2.0, distance: 0
path: payment.bcos.HelloWorld, type: BCOS2.0, distance: 0
path: payment.fabric.asset, type: Fabric1.4, distance: 1
total: 12
```

9.6.11 detail

查看跨链资源的详细信息。

参数:

- path：跨链资源标识。

```
[WeCross.org1-admin]> detail payment.bcos.HelloWorld
ResourceDetail{
 path='payment.bcos.HelloWorld',
 distance=0',
 stubType='BCOS2.0',
 properties={
  BCOS_PROPERTY_CHAIN_ID=1,
  BCOS_PROPERTY_GROUP_ID=1
 },
 checksum='c77f0ac3ead48d106d357ffe0725b9761bd55d3e27edd8ce669ad8b470a27bc8'
}

[WeCross.org1-admin]> detail payment.fabric.sacc
ResourceDetail{
 path='payment.fabric.sacc',
 distance=1',
 stubType='Fabric1.4',
 properties={
  ORG_NAMES=[
   Org1,
   Org2
  ],
  PROPOSAL_WAIT_TIME=300000,
  CHAINCODE_VERSION=1.0,
  CHANNEL_NAME=mychannel,
  CHAINCODE_NAME=sacc
 },
 checksum='058b239a9f10dc2b1154e28910861053c376be61cbbfd539b71f354b85ed309b'
}
```

注意： 若在properties中isTemporary字段为true，那么说明该资源有可能还未注册到WeCross，也有可能该资源不存在。

9.6.12 call

调用智能合约的方法，不涉及状态的更改，不发交易。

参数：

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

```
[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [Hello, World!]
# 在事务状态下，call命令可自动转变成命令 callTransaction
[Transaction running: 575905db7aab412bb476db8b89a4c042]
[WeCross.org1-admin]> call payment.bcos.HelloWorld get
Result: [test2]
```

9.6.13 sendTransaction

调用智能合约的方法，会更改链上状态，需要发交易。

参数：

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

```
[WeCross.org1-admin]> sendTransaction payment.bcos.HelloWorld set "Hello, WeCross!"
Txhash  : 0x4e3fe01371cf2a304b0782b554399bf8643cd50e5ea9c522d9c846ee38ec00dd
BlockNum: 16
Result  : []
```

9.6.14 invoke

在非事务状态时，与命令*sendTransaction*功能一致；在事务状态时，与命令*execTransaction*功能一致

参数：

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

```
# 非事务状态时，invoke相当于命令 sendTransaction
[WeCross.org1-admin]> invoke payment.bcos.HelloWorld set test1
Txhash  : 0x5b925655f1668357845c1846450b47ab92bbe1f96cdeaf9e2db98aa834536da1
BlockNum: 35
Result  : []

[WeCross.org1-admin]> startTransaction payment.bcos.HelloWorld
Result: success!
Transaction ID is: 575905db7aab412bb476db8b89a4c042
# 事务状态时，invoke相当于命令 execTransaction
[Transaction running: 575905db7aab412bb476db8b89a4c042]
[WeCross.org1-admin]> invoke payment.bcos.HelloWorld set test2
Txhash  : 0x3d22c863570696c9eed121f94bd6794865286ac85c5c9778b14777b2a8338bff
BlockNum: 37
Result  : []
```

(下页继续)

(续上页)

```
[Transaction running: 575905db7aab412bb476db8b89a4c042]
[WeCross.org1-admin]> callTransaction payment.bcos.HelloWorld get
Result: [test2]
```

9.6.15 bcosDeploy

FISCO BCOS 合约部署命令，成功返回部署的合约地址，失败返回错误描述

参数：

- Path: 跨链资源标示，用于标记部署的合约资源
- Source file path: 部署的合约路径，支持绝对路径/相对路径
- Class Name: 部署的合约名
- Verseion: 部署合约版本号，**注意：同一合约版本号唯一，再次部署相同的版本号会失败**

示例：

```
[WeCross.org1-admin]> bcosDeploy payment.bcos.HelloWorld contracts/solidity/
↪HelloWorld.sol HelloWorld 2.0
Result: 0xc3c72dce00c1695e8f50696f22310375e3348e1e
```

9.6.16 bcosRegister

FISCO BCOS 注册已有合约为跨链资源，成功返回`Success`，失败返回错误描述

参数：

- Path: 跨链资源标示，用于标记注册的合约资源
- Source file path: 合约的ABI文件路径或者合约源码路径
- Contract address: 注册的合约地址
- Version: 合约版本号，**注意：同一部署合约版本号唯一，再次部署相同版本号的合约会失败**

示例：

```
[WeCross.org1-admin]> bcosRegister payment.bcos.HelloWorld contracts/solidity/
↪HelloWorld.sol 0xc3c72dce00c1695e8f50696f22310375e3348e1e HelloWorld 3.0
Result: success
```

9.6.17 fabricInstall

Fabric 安装链码命令，安装后需fabricInstantiate来启动链码

参数：

- path：跨链资源标识。
- sourcePath：链码工程所在目录，支持绝对路径和WeCross-Console的conf目录内的相对路径
- version：指定一个版本，fabricInstantiate时与此版本对应
- language：指定一个链码语言，支持GO_LANG和JAVA

```
# 在Org1中部署sacc合约
[WeCross.org1-admin]> fabricInstall payment.fabric.sacc Org1 contracts/chaincode/
↪sacc 2.0 GO_LANG
path: classpath:contracts/chaincode/sacc
Result: Success

# 将Fabric的默认账号切换至Org2-admin的账号，以便于我们部署合约到Org2中
# 注意：此示例中，Org2-admin账号的keyID为2，可通过listAccount命令查询
[WeCross.org1-admin]> setDefaultAccount Fabric1.4 2
Result: success
Universal Account info has been changed, now auto-login again.
Result: success
================================================================================================
Universal Account:
username: org1-admin
pubKey  : 3059301306...
uaID    : 3059301306...

# 在Org2中部署sacc合约
[WeCross.org1-admin]> fabricInstall payment.fabric.sacc Org2 contracts/chaincode/
↪sacc 2.0 GO_LANG
path: classpath:contracts/chaincode/sacc
Result: Success
```

9.6.18 fabricInstantiate

Fabric 启动（实例化）已安装的链码。此步骤前需先用fabricInstall向指定机构安装链码。

参数：

- path：跨链资源标识。
- orgNames：链码被安装的的机构列表
- sourcePath：链码工程所在目录，支持绝对路径和WeCross-Console的conf目录内的相对路径
- version：指定一个版本，与fabricInstall时的版本对应
- language：指定一个链码语言，支持GO_LANG和JAVA
- policy：指定背书策略文件，设置default为所有endorser以OR相接
- initArgs：链码初始化参数

```
[WeCross.org1-admin]> fabricInstantiate payment.fabric.sacc ["Org1","Org2"]␣
↪contracts/chaincode/sacc 1.0 GO_LANG default ["a","10"]
Result: Instantiating... Please wait and use 'listResources' to check. See router
↪'s log for more information.
```

启动时间较长（1min左右），可用listResources查看是否已启动，若长时间未启动，可查看router的日志进行排查。

9.6.19 fabricUpgrade

Fabric 升级已启动的链码逻辑，不改变已上链的数据。此步骤前需先用`fabricInstall`向指定机构安装另一个版本的链码。

参数：

- path：跨链资源标识。
- orgNames：链码被安装的的机构列表

- sourcePath：链码工程所在目录，支持绝对路径和WeCross-Console的conf目录内的相对路径
- version：指定一个版本，与fabricInstall时的版本对应
- language：指定一个链码语言，支持GO_LANG和JAVA
- policy：指定背书策略文件，设置default为OR所有endorser
- initArgs：链码初始化参数

```
[WeCross.org1-admin]> fabricUpgrade payment.fabric.sacc ["Org1","Org2"] contracts/
↪chaincode/sacc 2.0 GO_LANG default ["a","10"]
Result: Upgrading... Please wait and use 'detail' to check the version. See router
↪'s log for more information.
```

升级时间较长（1min左右），可用`detail payment.fabric.sacc`查看版本号，若长时间升级完成，可查看router的日志进行排查。

9.6.20 genTimelock

跨链转账辅助命令，根据时间差生成两个合法的时间戳。

参数：

- interval：时间间隔

```
[WeCross.org1-admin]> genTimelock 300
timelock0: 1607245965
timelock1: 1607245665
```

9.6.21 genSecretAndHash

跨链转账辅助命令，生成一个秘密和它的哈希。

```
[WeCross.org1-admin]> genSecretAndHash
hash   : cdbfc235be9aff715967119a25b03f49e7103480fec459a610d2efe51ff35fad
secret: 266a85d058afbf743c541f8da4add95d54f02cad27acd90d8b2155947524d303
```

9.6.22 newHTLCProposal

新建一个基于哈希时间锁合约的跨链转账提案，该命令由两条链的资金转出方分别执行。

参数：

- path：跨链转账资源标识。
- args：提案信息，包括两条链的转账信息。
  - hash： 唯一标识，提案号，
  - secret： 提案号的哈希原像
  - role： 身份，发起方-true，参与方-false。发起方需要传入secret，参与方secret传null。
  - sender0：发起方的资金转出者在对应链上的地址
  - receiver0：发起方的资金接收者在对应链上的地址
  - amount0：发起方的转出金额
  - timelock0：发起方的超时时间
  - sender1：参与方的资金转出者在对应链上的地址
  - receiver1：参与方的资金接收者在对应链上的地址

  - amount1：参与方的转出金额
  - timelock1：参与方的超时时间，小于发起方的超时时间

```
[WeCross.org1-admin]> newHTLCProposal payment.bcos.htlc␣
↪bea2dfec011d830a86d0fbeeb383e622b576bb2c15287b1a86aacdba0a387e11␣
↪9dda9a5e175a919ee98ff0198927b0a765ef96cf917144b589bb8e510e04843c true␣
↪0x4305196480b029bbecb071b4b68e95dfef36a7b7␣
↪0x2b5ad5c4795c026514f8317c7a215e218dccd6cf 700 2000010000 Admin@org1.example.com␣
↪User1@org1.example.com 500 2000000000

Txhash: 0xf9b50871c0b05830af9c251d786d2ea93e6f98282e33b6d4c530ed0fddde2f25
BlockNum: 18
Result: create a htlc proposal successfully
```

9.6.23 checkTransferStatus

根据提案号（Hash）查询htlc转账状态。

参数：

- path：跨链资源标识。
- method：合约方法名。
- hash：转账提案号。

```
[WeCross.org1-admin]> checkTransferStatus payment.bcos.htlc␣
↪bea2dfec011d830a86d0fbeeb383e622b576bb2c15287b1a86aacdba0a387e11
status: ongoing!
```

9.6.24 startTransaction

写接口，开始两阶段事务。

参数：

- path_1 ... path_n：参与事务的资源路径列表，路径列表中的资源会被本次事务锁定，锁定后仅限本事务相关的交易才能对这些资源发起写操作，非本次事务的所有写操作都会被拒绝。

```
[WeCross.org1-admin]> startTransaction payment.bcos.evidence payment.fabric.
↪evidence
Result: success!
Transaction ID is: 024f0ff81cda4b938a0b48805be9eb61
# 系统自动生成一串事务ID，并在下次Prompt加上事务标识表示事务正在进行中

[Transaction running: 024f0ff81cda4b938a0b48805be9eb61]
[WeCross.org1-admin]>
```

9.6.25 execTransaction

写接口，发起事务交易。

参数：

- path：资源路径。
- method：接口名，同sendTransaction。需要注意的是，该接口需要在合约中配套以_revert结尾的回滚接口。
- args：参数，同sendTransaction。

```
[Transaction running: 024f0ff81cda4b938a0b48805be9eb61]
[WeCross.org1-admin]> execTransaction payment.bcos.evidence newEvidence evidence1␣
↪Jerry
Txhash  : 0xcb4ee8b1b83f6855e7583c75bec35c29004ee446c93a7bcfe92b5d7156572235
BlockNum: 28
Result  : [true]

[Transaction running: 024f0ff81cda4b938a0b48805be9eb61]
[WeCross.org1-admin]> execTransaction payment.fabric.evidence newEvidence␣
↪evidence2 Tom
Txhash  : bb955ee814b12ebc19a068c8d3c10afbc7aadc6c62225dac51fc5c496049c36d
BlockNum: 14
Result  : [Success]
```

9.6.26 callTransaction

读接口，查询事务中的数据。

参数：

- path：资源路径。
- method：接口名，同sendTransaction。
- args：参数，同sendTransaction。

```
[Transaction running: 024f0ff81cda4b938a0b48805be9eb61]
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence1
Result: [Jerry]
```

9.6.27 commitTransaction

写接口，提交事务，确认事务执行过程中所有的变动。

参数：

- path_1 ... path_n：(可选) 用于提交事务的路径列表。

```
[WeCross.org1-admin]> commitTransaction payment.bcos.evidence payment.fabric.
↪evidence

# 也可以直接输入`commitTransaction`命令，输入y提交这次事务，输入n则取消
[Transaction running: 024f0ff81cda4b938a0b48805be9eb61]
[WeCross.org1-admin]> commitTransaction
Transaction running now, transactionID is: 024f0ff81cda4b938a0b48805be9eb61
Are you sure commit it now?(y/n)  y
Committing transaction: 024f0ff81cda4b938a0b48805be9eb61...

Result: success!
# 结束事务后，Prompt不再显示事务标识
[WeCross.org1-admin]>
```

9.6.28 rollbackTransaction

写接口，撤销本次事务的所有变更时。

参数：

- path_1 ... path_n：用于回滚事务的路径列表。

```
# 查看开始前的状态
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence0
Result: []

# 开始事务
[WeCross.org1-admin]> startTransaction payment.bcos.evidence
Result: success!
Transaction ID is: e27ddd9fc4564256b20e6329f9ce9969

# 执行事务
[Transaction running: e27ddd9fc4564256b20e6329f9ce9969]
[WeCross.org1-admin]> execTransaction payment.bcos.evidence newEvidence evidence0␣
↪WeCross
Txhash  : 0x5851a927a33b673f76b9fe3a57767ddd3d920230d54d7fee92400789a0bd551c
BlockNum: 31
Result  : [true]

# 读事务数据
[Transaction running: e27ddd9fc4564256b20e6329f9ce9969]
[WeCross.org1-admin]> callTransaction payment.bcos.evidence queryEvidence evidence0
Result: [WeCross]

# 回滚事务
# 可以直接输入`rollbackTransaction`命令，输入y回滚这次事务，输入n则取消，也可以输入全资源路径
[Transaction running: e27ddd9fc4564256b20e6329f9ce9969]
[WeCross.org1-admin]> rollbackTransaction
Transaction running now, transactionID is: e27ddd9fc4564256b20e6329f9ce9969
Are you sure rollback transaction now?(y/n)  y
Rollback transaction: e27ddd9fc4564256b20e6329f9ce9969...

Result: success!

# 查看事务回滚后的状态，和开始前保持一致
[WeCross.org1-admin]> call payment.bcos.evidence queryEvidence evidence0
Result: []
```

9.6.29 getXATransaction

读接口，查询事务信息。

参数:

- transactionID：事务ID，待提交事务的ID
- path_1 ... path_n：参与事务的资源路径列表

```
[WeCross.org1-admin]> getXATransaction 024f0ff81cda4b938a0b48805be9eb61 payment.
↪fabric.evidence
XATransactionResponse{
 xaTransactionID='024f0ff81cda4b938a0b48805be9eb61',
 username='org1-admin',
 status='committed',
 startTimestamp=1607246930,
 commitTimestamp=1607247480,
 rollbackTimestamp=0,
 paths=[
  payment.fabric.evidence,
  payment.bcos.evidence
 ],
 xaTransactionSteps=[
  XATransactionStep{
```

(下页继续)

(续上页)

```
    xaTransactionSeq=1607247002498,
    username='org1-admin',
    path='payment.fabric.evidence',
    timestamp=1607247002,
    method='newEvidence',
    args='[
     "evidence2",
     "Tom"
    ]'
   }
 ]
}
```

9.6.30 listXATransactions

读接口，查询所给个数的最新的事务。

参数：

- size：指定查询个数，范围在[1-1024]

```
[WeCross.org1-admin]> listXATransactions 4
Result: [ {
  "xaTransactionID" : "e27ddd9fc4564256b20e6329f9ce9969",
  "username" : "org1-admin",
  "status" : "rolledback",
  "timestamp" : 1607247643,
  "paths" : [ "payment.bcos.evidence" ]
}, {
  "xaTransactionID" : "024f0ff81cda4b938a0b48805be9eb61",
  "username" : "org1-admin",
  "status" : "committed",
  "timestamp" : 1607246930,
  "paths" : [ "payment.bcos.evidence", "payment.fabric.evidence" ]
}, {
  "xaTransactionID" : "65aaedc68bc24126ad48a794e08bb7d1",
  "username" : "org1-admin",
  "status" : "committed",
  "timestamp" : 1607246797,
  "paths" : [ "payment.bcos.evidence" ]
}, {
  "xaTransactionID" : "46299d812d2f42eaad64989daf94fe6f",
  "username" : "org1-admin",
  "status" : "committed",
  "timestamp" : 1607246405,
  "paths" : [ "payment.bcos.HelloWorld" ]
} ]
```

9.6.31 loadTransaction

恢复到某个正在执行的事务上下文。

参数：

- transactionID：恢复的事务ID。
- path_1 . . . path_n：正在执行的事务中资源路径列表。

```
# 查询最新事务，事务正处于进行中
[WeCross.org1-admin]> listXATransactions 1
Result: [ {
  "xaTransactionID" : "0a359201499047f8a99a6a978b9a7f77",
  "username" : "org1-admin",
  "status" : "processing",
  "timestamp" : 1607249155,
  "paths" : [ "payment.bcos.HelloWorld" ]
} ]

[WeCross.org1-admin]> loadTransaction 0a359201499047f8a99a6a978b9a7f77 payment.
↪bcos.HelloWorld
Load transaction success!
# 恢复到事务上下文
[Transaction running: 0a359201499047f8a99a6a978b9a7f77]
[WeCross.org1-admin]>
```

9.6.32 getCurrentTransactionID

获取当前控制台的事务ID

```
[Transaction running: 0a359201499047f8a99a6a978b9a7f77]
[WeCross.org1-admin]> getCurrentTransactionID
There is a Transaction running now, ID is: 0a359201499047f8a99a6a978b9a7f77.
```

9.7 7. 交互式命令

9.7.1 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
[WeCross.org1-admin]> myResource = WeCross.getResource payment.bcos.HelloWeCross

# 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
[WeCross.org1-admin]> path = payment.bcos.HelloWeCross

[WeCross.org1-admin]> myResource = WeCross.getResource path
```

9.7.2 [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tab键可以看到能够访问的所有命令
[WeCross.org1-admin]> myResource.
myResource.call                 myResource.detail
myResource.sendTransaction
```

9.7.3 detail

```
[WeCross.org1-admin]> myResource.detail
ResourceDetail{
 path='payment.bcos.HelloWorld',
 distance=0',
 stubType='BCOS2.0',
 properties={
  BCOS_PROPERTY_CHAIN_ID=1,
  BCOS_PROPERTY_GROUP_ID=1,
  HelloWorld=0x9bb68f32a63e70a4951d109f9566170f26d4bd46
 },
 checksum='0x888d067b77cbb04e299e675ee4b925fdfd60405241ec241e845b7e41692d53b1'
}
```

9.7.4 call

```
[WeCross.org1-admin]> myResource.call get
Result: [Hello, World!]
```

9.7.5 sendTransaction

```
[WeCross.org1-admin]> myResource.sendTransaction set "Hello, WeCross!"
Result: []

[WeCross.org1-admin]> myResource.call get
Result: [Hello, WeCross!]
```

CHAPTER 10

账户服务

10.1 跨链账户

10.1.1 概念

不同类型的链有不同的账户，WeCross将其进行统一管理，抽象出**跨链账户**：

- 缩写：UA（Universal Account）
- 定义
  - **跨链账户：WeCross中用户的身份**
  - 链账户：向不同类型的链发交易时，给交易签名的实际账户
- 作用：对于不同类型的链，都采用跨链账户发送交易
- 效果：对于不同类型的链，上链的交易都能映射回对应的跨链账户
- 限制：一个跨链账户可以持有多个链账户，一个链账户被一个跨链账户持有

10.1.2 原理

- 用户通过登录的方式获得跨链账户的身份
- 用户在跨链账户下可添加不同类型的链账户
- 通过互签的方式，生成UA Proof，建立跨链账户与链账户的映射
- UA Proof在router间同步，在获取交易发送者时，可结合UA Proof获取发交易的跨链账户
- 相关逻辑由WeCross Account Manager进行管理

10.1.3 举例

一个跨链账户包括以下信息：

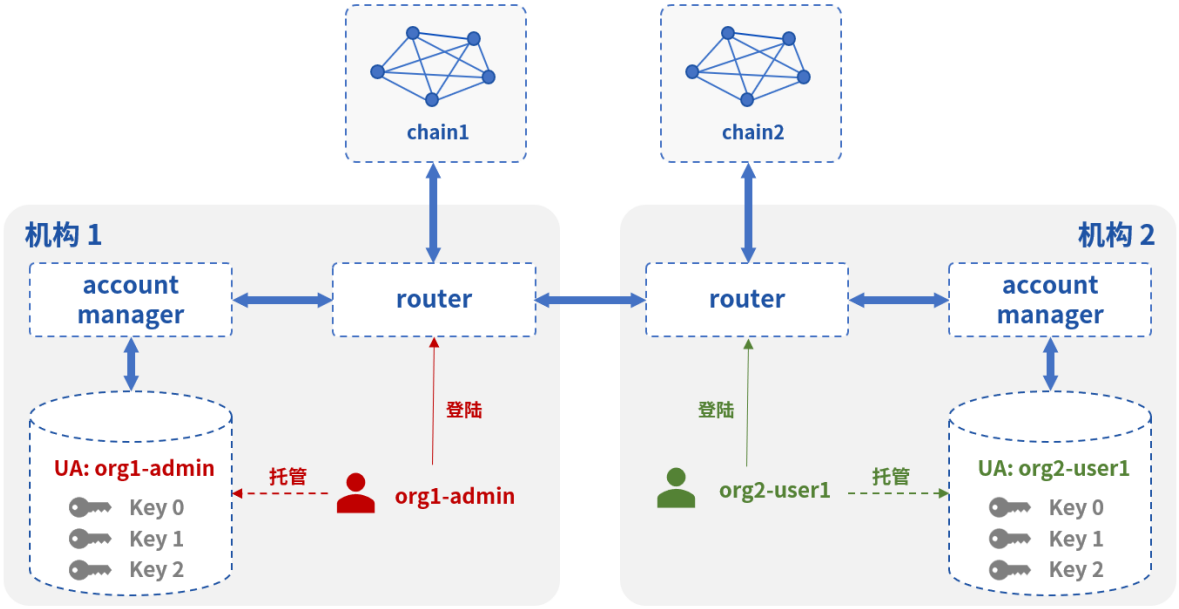
- UA名：org1-admin
- 公私钥：aaaa、bbbb

- 链账户
  - FISCO BCOS 2.0
    - 0：公钥、私钥、地址、UA Proof、默认账户
    - 1：公钥、私钥、地址、UA Proof、非默认账户
  - FISCO BCOS 2.0 国密
    - 2：公钥、私钥、地址、UA Proof、非默认账户
    - 3：公钥、私钥、地址、UA Proof、默认账户
  - Hyperledger Fabric1.4：
    - 4：证书、私钥、MSPID、UA Proof、默认账户
    - 5 . . .

10.2 WeCross Account Manager

10.2.1 概念

- 作用：管理自己机构的跨链账户
- 架构
  - Router选择自己机构的Account Manager进行连接
  - Account Manager部署在机构内网中，外部router无法访问
- 操作
  - Router连接Account Manager后，用户在Router上进行登录
  - 登陆后，可采用此身份向跨链网络中的任意资源发交易，无论是否是此router连接的资源
- 限制
  - Router限制：Router只有连接了相应机构的Account Manager，才能启动
  - 用户限制：跨链账户数据由某个Account Manager管理，用户只能在连接了自身机构WeCross Account Manager的Router上才能登录成功



10.2.2 部署

下载

参考：下载程序

配置

- 配置与Router连接的公私钥

```
cd WeCross-Account-Manager
cp ~/wecross/routers-payment/cert/sdk/* ./conf #从生成的router目录拷贝
```

- 生成自身加密的公私钥

```
bash create_rsa_keypair.sh -d conf/ # 在conf下生成：rsa_private.pem  rsa_public.pem
```

- 写配置

```
cp conf/application-sample.toml conf/application.toml
vim conf/application.toml
```

内容为

```
[service] # Account Manager启动的service配置，Router连接至此service
    address = '0.0.0.0'
    port = 8340
    sslKey = 'classpath:ssl.key'
    sslCert = 'classpath:ssl.crt'
    caCert = 'classpath:ca.crt'
    sslOn = true

[admin] # 机构的admin跨链账户，第一次启动按此配置创建
    username = 'org1-admin'
    password = '123456'

[auth] # 登录后颁发登录令牌（jwt）的配置，name可修改为自身机构名
    name = 'org1'
    expires = 18000 # 5 h
    noActiveExpires = 600 # 10 min

[encrypt] # 自身加密公私钥配置
    privateKeyFile = 'classpath:rsa_private.pem'
    publicKeyFile = 'classpath:rsa_public.pem'

[db] # 数据库连接配置
    # 可在'?'后面增加其他JDBC连接参数
    # Note: 对于MySQL 8.0+，默认开启useSSL=false，若MySQl部署在远端，应该正确配
置MySQL的SSL选项，使用SSL
    url = 'jdbc:mysql://localhost:3306/wecross_account_manager?useSSL=false'
    username = 'jimmy'
    password = 'abc123'
    # 用于加密数据库中的敏感数据，若未配置则不进行加密
    encryptKey = 'rIBJD38jqSMR@CSM'

[ext] # 拓展配置
    allowImageAuthCodeEmpty = true # 是否允许空验证码，设为false后控制台无法使用
```

- 启动

```
bash start.sh # 停止：bash stop.sh
```

10.2.3 账户操作

用户可通过控制台和网页管理平台进行操作

- 登录、登出、注册
- 添加链账户、设置默认账户、删除链账户

可直接查阅控制台或网页管理平台相关部分的说明

CHAPTER 11

脚本

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(如：~/wecross-demo/routers-payment/127.0.0.1-8250-25500/)，本章节将对这些脚步做详细介绍。

11.1 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
```

11.2 停止脚本

stop.sh

停止脚本stop.sh用于停止WeCross服务。

```
bash stop.sh
```

11.3 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 <your pwd>
    -z  <generate tar packet>       [Optional]   default no
    -T  <enable test mode>          [Optional]   default no. Enable test resource.
    -h  call for help
e.g
    bash build_wecross.sh -n payment -l 127.0.0.1:8250:25500
    bash build_wecross.sh -n payment -f ipfile
    bash build_wecross.sh -n payment -f ipfile -c ./ca/
```

- **-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
```

- **-c**：可选，指定跨链路由基于某个路径下的ca证书生成
- **-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

11.4 添加新接入链脚本

add_chain.sh

脚本add_chain.sh用于在router中创建特定区块链的连接配置

```
Usage:
    -t <type>                           [Required] type of chain, BCOS2.0 or GM_
↪BCOS2.0 or Fabric1.4
    -n <name>                           [Required] name of chain
    -d <dir>                            [Optional] generated target_directory,␣
↪default conf/stubs/
    -h                                  [Optional] Help
```

- **-t**：连接类型，按照插件选择，如BCOS2.0或Fabric1.4
- **-n**：连接名，账户名称
- **-d**：连接配置目录，默认生成在conf/chains/下

不同的链有不同的操作方法，具体操作请查看（操作后，请重启router，让router重启加载配置）：

- BCOS2.0 接入配置
- Fabric1.4 接入配置

11.5 系统合约工具脚本

deploy_system_contract.sh

deploy_system_contract.sh脚本用于部署或者更新代理合约/桥接合约。

帮助信息:

```
$ bash deploy_system_contract.sh -h

Usage:
    -c <chain name>                     [Required] chain name
    -u <upgrade>                        [Optional] upgrade proxy/hub contract if␣
↪proxy/hub contract has been deployed, default deploy proxy/hub contract
    -t <type>                           [Required] type of chain, BCOS2.0 or GM_
↪BCOS2.0 or Fabric1.4
    -P <proxy contract>                 [Optional] upgrade/deploy operation on␣
↪proxy contract
    -H <hub contract>                   [Optional] upgrade/deploy operation on hub␣
↪contract
    -h                                  [Optional] Help
e.g
    bash deploy_system_contract.sh -t BCOS2.0    -c chains/bcos -P
    bash deploy_system_contract.sh -t BCOS2.0    -c chains/bcos -H
    bash deploy_system_contract.sh -t BCOS2.0    -c chains/bcos -u -P
    bash deploy_system_contract.sh -t BCOS2.0    -c chains/bcos -u -H
    bash deploy_system_contract.sh -t Fabric1.4  -c chains/fabric -P
    bash deploy_system_contract.sh -t Fabric1.4  -c chains/fabric -H
    bash deploy_system_contract.sh -t Fabric1.4  -c chains/fabric -u -P
    bash deploy_system_contract.sh -t Fabric1.4  -c chains/fabric -u -H
```

- **-t**：操作类型，根据链的类型选择，支持BCOS2.0、GM_BCOS2.0、Fabric1.4三种类型
- **-c**：链配置路径，链配置位于chains目录下
- **-u**：更新合约操作，默认情况下为部署合约操作，只有在合约已经部署时才可以更新
- **-P**：操作对象为代理合约
- **-H**：操作对象为桥接合约
- **-h**：帮助信息

11.6 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
```

(下页继续)

(续上页)

```
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。
- **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。
- **n**：生成节点证书。
- **t**：指定cert.cnf的路径

11.7 升级脚本

已经部署的环境需要升级至最新版本时可以使用升级脚本，包括upgrade_wecross.sh、upgrade_console.sh、upgrade_account_manager.sh三个工具，分别升级对应服务。**注意：升级的版本与部署的版本必须保持兼容，才能够进行升级操作，各个版本之间的兼容性参见WeCross程序版本对应版本的兼容性章节。**

- **upgrade_wecross.sh**

升级Router

```
$ bash upgrade_wecross.sh -h

Usage:
    -s <upgrade dir>                    [Optional] the router's upgrade dir
    -d <router dir>                     [Optional] the router's deploy dir
    -h                                  [Optional] Help
e.g
    bash upgrade_wecross.sh -d ~/wecross-demo/router/127.0.0.1-8250-25500/
```

参数:

- -s: 待升级的安装包的目录，默认为当前目录
- -d: 已部署环境的目录

- **upgrade_console.sh**

升级控制台

```
$ bash upgrade_console.sh -h

Usage:
    -s <upgrade dir>                    [Optional] the WeCross Console's upgrade␣
↪dir
    -d <console's dir>                  [Optional] the WeCross Console's deploy dir
    -h                                  [Optional] Help
e.g
    bash upgrade_console.sh -d ~/wecross-demo/WeCross-Console
```

参数:

- -s: 待升级的安装包的目录，默认为当前目录
- -d: 已部署环境的目录

- **upgrade_account_manager.sh**

```
$ bash upgrade_account_manager.sh -h

升级`Account-Manager`

Usage:
    -s <upgrade dir>                    [Optional] the WeCross Account Manager's␣
↪upgrade dir
    -d <account manager's dir>          [Optional] the WeCross Account Manager's␣
↪deploy dir
    -h                                  [Optional] Help
e.g
    bash upgrade_account_manager.sh -d ~/wecross-demo/WeCross-Account-Manager
```

参数:

- -s: 待升级的安装包的目录，默认为当前目录
- -d: 已部署环境的目录

CHAPTER 12

配置

本节描述WeCross Router的配置。

12.1 配置结构

WeCross Router的配置位于conf目录下，分为：

- 主配置（wecross.toml）：配置Router连接等信息
- 链配置（chains/<chain_name>/stub.toml）：配置连接至对应区块链、链上资源
- 账户配置（accounts/<account_name>/account.toml）：配置可用于发交易账户的公私钥等信息
- 区块头验证配置（verifier.toml）：配置对应链的区块验证公钥或证书路径，**该配置为可选配置**

配置的目录结构如下：

```
# 这是conf目录下标准的配置结构，Router配置连接了两条链，分别叫bcos和fabric
.
├── log4j2.xml    // 日志配置文件，无需更改
├── accounts
│   ├── bcos_user1
│   │   └── account.toml  // 账户配置
│   └── fabric_user1
│       └── account.toml  // 账户配置
├── chains
│   ├── bcos
│   │   └── stub.toml     // 链配置
│   └── fabric
│       └── stub.toml     // 链配置
├── verifier.toml         // 区块头验证配置
└── wecross.toml          // 主配置
```

12.2 主配置

主配置为 conf/wecross.toml，配置示例如下：

```
[common]
    zone = 'payment'
    visible = true
    enableAccessControl = false

[chains]
    path = 'classpath:chains'

[rpc] # rpc ip & port
    address = '127.0.0.1'
    port = 8250
    caCert = 'classpath:ca.crt'
    sslCert = 'classpath:ssl.crt'
    sslKey = 'classpath:ssl.key'
    threadNum = 16
    threadQueueCapacity = 10000
    sslSwitch = 2  # disable ssl:2, SSL without client auth:1 , SSL with client␣
↪and server auth: 0
    webRoot = 'classpath:pages'
    mimeTypesFile = 'classpath:conf/mime.types' # set the content-types of a file
    # urlPrefix = '/wecross' # v1.1.1新增配置，使用该配置可收敛所有请求URL前缀

[p2p]
    listenIP = '0.0.0.0'
    listenPort = 25500
    caCert = 'classpath:ca.crt'
    sslCert = 'classpath:ssl.crt'
    sslKey = 'classpath:ssl.key'
    peers = ['127.0.0.1:25501']
    threadNum = 16
    threadQueueCapacity = 100000

[account-manager]
    server =  '127.0.0.1:8250'
    admin = 'org1-admin'
    password = '123456' # This field is not used at v1.0.0, just for future
    sslKey = 'classpath:ssl.key'
    sslCert = 'classpath:ssl.crt'
    caCert = 'classpath:ca.crt'
    maxTotal = 200
    maxPerRoute = 8
    allowNameToken = false

#[[htlc]]
#    selfPath = 'payment.bcos.htlc'
#    account1 = 'bcos_default_account'
#    counterpartyPath = 'payment.fabric.htlc'
#    account2 = 'fabric_default_account'
```

跨链服务配置有五个配置项，分别是[common]、[chains]、[rpc]、[p2p]以及[test]，各个配置项含义如下：

- [common] 通用配置
  - zone：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
  - enableAccessControl：布尔；是否开启权限控制，开启后，仅admin账户可访问所有资源，普通账户需admin账户在网页管理台中为其授权后才可访问相应资源

- `[chains]` 链配置
  - path：字符串；链配置的根目录；WeCross从该目录下去加载各个链的配置
- `[rpc]` RPC配置
  - address：字符串；RPC服务监听地址，通常设置为本机IP地址
  - port：整型；WeCross Router的RPC端口；WeCross Java SDK调用Router的端口
  - caCert：字符串；WeCross Router根证书路径
  - sslCert：字符串；WeCross Router证书路径
  - sslKey：字符串；WeCross Router私钥路径
  - threadNum（可选）：整型，rpc线程数，默认16
  - threadQueueCapacity（可选）：整型，任务队列容量，默认10000
  - sslSwitch：整型，SL加密配置，0：双向验证，1：验Router证书，0：无验证
  - webRoot：字符串，网页管理平台页面存放位置
  - mimeTypesFile：字符串网页管理平台的content-type映射文件存放位置
  - urlPrefix: 用于收敛请求URL，若不配置则默认为空，若配置则必须配置正确，支持数字英文和特殊符号(-, _)，长度1-18
    - 若需要进行控制台/网页管理平台访问修改URL前置的跨链路由router，可参考控制台配置中的`urlPrefix`字段，参考网页管理平台配置中的配置方法。
- `[p2p]` 组网配置
  - listenIP：字符串；P2P服务监听地址；一般为'0.0.0.0'
  - listenPort：整型；P2P服务监听端口；WeCross Router之间交换消息的端口
  - caCert：字符串；WeCross Router根证书路径
  - sslCert：字符串；WeCross Router证书路径
  - sslKey：字符串；WeCross Router私钥路径
  - peers：字符串数组；peer列表；需要互相连接的WeCross Router列表
  - threadNum（可选）：p2p线程数，默认16
  - threadQueueCapacity（可选）：任务队列容量，默认10000
- `[account-manager]` 跨链账户服务配置
  - server：字符串，需要连接的Account Manager的IP:Port
  - admin：字符串，此Router连接的Account Manager的admin账户名，需与Account Manager中的配置保持一致
  - password：字符串保留字段，目前无用
  - sslKey：字符串；WeCross Router根证书路径
  - sslCert：字符串；WeCross Router证书路径
  - caCert：字符串；WeCross Router私钥路径
  - maxTotal（可选）：整型，连接Account Manager的连接池maxTotal参数，默认200
  - maxPerRoute（可选）：整型，连接Account Manager的连接池maxPerRoute参数，默认8
  - allowNameToken (可选)：布尔类型，用于适配外部登录系统，设置为true后router取消对登录token的校验，此时删除header中的Authorization，在url中加上`&wecross-name-token=xxx`，即可直接将xxx作为登陆者id，在大多数场景下此配置应为false
- `[htlc]` htlc配置（可选）

  - selfPath：本地配置的htlc合约资源路径
  - account1：可调用本地配置的htlc合约的账户
  - counterpartyPath：本地配置的htlc合约的对手方合约路径
  - account2：可调用对手方htlc合约的账户

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，其中链配置的根目录默认为chains。

12.3 链配置

链配置是Router连接每个区块链的配置：

- 指定链名

在chains/<chain_name>/stub.toml目录下，通过目录名<chain_name>指定链名。

- 区块链链接信息

在stub.toml中配置与区块链交互所需链接的信息。

- 跨链资源（可选）

在stub.toml中配置需要参与跨链的资源。

WeCross启动后会在wecross.toml中所指定的chains的根目录下去遍历所有的一级目录，目录名即为chain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Hyperledger Fabric。

FISCO BCOS访问链接： GitHub访问链接， Gitee访问链接

Hyperledger Fabric访问链接： GitHub访问链接， Gitee访问链接

配置FISCO BCOS

请参考：FISCO BCOS 2.0插件配置

配置Fabric

请参考：Fabric 1.4插件配置

12.4 区块头验证配置

区块头验证配置为 conf/verifier.toml，配置示例如下：

```
[verifiers]
    [verifiers.payment.bcos-group1]
        chainType = 'BCOS2.0'
        # 填写所有共识节点的公钥
        pubKey = [
            'b949f25fa39a6b3797ece30a2a9e025...',
            '...'
        ]
    [verifiers.payment.bcos-group2]
        chainType = 'BCOS2.0'
```

(下页继续)

(续上页)

```
        # 填写所有共识节点的公钥
        pubKey = [
            'b949f25fa39a6b3797ece3132134fa3...',
            '...'
        ]
    [verifiers.payment.bcos-gm]
        chainType = 'GM_BCOS2.0'
        pubKey = [
            'ecc094f00b11a0a5cf616963e313218...'
        ]
    [verifiers.payment.fabric-mychannel]
        chainType = 'Fabric1.4'
        # 填写所有机构CA的证书路径
        [verifiers.payment.fabric-mychannel.endorserCA]
            Org1MSP = 'classpath:verifiers/org1CA/ca.org1.example.com-cert.pem'
            Org2MSP = 'classpath:verifiers/org2CA/ca.org2.example.com-cert.pem'
        [verifiers.payment.fabric-mychannel.ordererCA]
            OrdererMSP = 'classpath:verifiers/ordererCA/ca.example.com-cert.pem'
```

区块头验证配置有一个主要配置项：

- `[verifiers]`：定义文件Map入口
  - `[verifiers.zone.chain]`：zone.chain为指定某个待验证的链类型，若配置该链则必须配置全；
    - chainType：字符串；定义为待验证的链的类型，若配置错误会报错；
    - 其余字段：根据链类型，配置不同的字段
  - pubKey：字符串数组；仅适用于`BCOS2.0`和`GM_BCOS2.0`类型的链，填入该链类型的共识节点公钥数组，配置必须符合以下条件：
    - 公钥均以字符串形式填入数组中
    - 必须包含所有共识节点的公钥信息
  - `[verifiers.zone.chain.endorserCA]`：Map类型；仅适用于`Fabric1.4`类型的链，填入该类型链的所有机构CA的MSP签名证书路径，其中Map的key是各个机构的MSPID，value是证书路径
  - `[verifiers.zone.chain.ordererCA]`：Map类型；仅适用于`Fabric1.4`类型的链，填入该类型链的所有Orderer机构颁发的MSP签名证书路径，其中Map的key是各个Orderer的MSPID，value是证书路径

区块头验证配置注意事项：

- 若未写`verifier.toml`文件，则默认不进行所有链的区块头验证；
- 若已配置`verifier.toml`文件，但未设置某条链的验证配置，则默认不进行该链的区块头验证；
- Fabric链的`endorserCA`和`ordererCA`必须同时配置；
- 若配置中漏了某个公钥和证书，则会导致验证不通过。

CHAPTER 13

跨链应用开发(SDK)

13.1 WeCross Java SDK开发应用

WeCross 跨链路由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。

13.1.1 环境要求

重要:

- java版本

  要求 JDK8或以上

- WeCross服务部署

  参考 部署指南

13.1.2 Java应用引入SDK

通过gradle或maven引入SDK到java项目

- gradle:

```
compile ('com.webank:wecross-java-sdk:1.2.1')
```

- maven:

```
<dependency>
    <groupId>com.webank</groupId>
    <artifactId>wecross-java-sdk</artifactId>
    <version>1.2.1</version>
</dependency>
```

13.1.3 配置SDK

WeCorss Java SDK 主要包括以下配置选项：

- 路由网络配置
- SSL连接配置

配置步骤

1. 将跨链路由router-${zone}/cert/sdk/目录下的证书和密钥拷贝到项目的classpath目录；
2. 在项目的classpath目录下新建文件 application.toml，默认配置如下：

```
[connection]
    server =  '127.0.0.1:8250'
    sslKey = 'classpath:ssl.key'
    sslCert = 'classpath:ssl.crt'
    caCert = 'classpath:ca.crt'
    sslSwitch = 2 # disable ssl:2, SSL without client auth:1 , SSL with client and
↪server auth: 0
```

1. 修改application.toml中server，使用跨链路由实际监听的IP和端口。

配置解读

在application.toml配置文件中，[connection]配置跨链路由的连接信息，具体包括以下配置项：

- server: 跨链路由监听的IP和端口；
- sslKey: 客户端私钥路径
- sslCert: 客户端证书路径
- caCert: CA证书路径
- sslSwitch: SSL的模式开关，0: 双向验证；1: 只验证服务端；2: 关闭SSL。

13.1.4 使用方法

示例代码如下：

```
try {
    // 初始化 Service
    WeCrossRPCService weCrossRPCService = new WeCrossRPCService();

    // 初始化Resource
    WeCrossRPC weCrossRPC = WeCrossRPCFactory.build(weCrossRPCService);

    weCrossRPC.login("username", "password").send(); // 需要有登录态才能进一步操作
    Resource resource = ResourceFactory.build(weCrossRPC, "payment.bcos.
↪HelloWecross"); // RPC服务，资源的path

    // 用初始化好的resource进行调用
    String[] callRet = resource.call("get");    // call 接口函数名 参数列表
    System.out.println((Arrays.toString(callRet)));

    // 用初始化好的resource进行调用
    String[] sendTransactionRet = resource.sendTransaction("set", "Tom"); //
↪sendTransaction 接口函数名 参数列表
```

(下页继续)

(续上页)

```
    System.out.println((Arrays.toString(sendTransactionRet)));

} catch (WeCrossSDKException e) {
    System.out.println("Error: " + e);
}
```

13.2 WeCross Java SDK API

SDK API分为两大类型，一种是对跨链路由RPC接口调用的封装，一种是资源接口。

13.2.1 API列表

- RPC封装接口

```
    RemoteCall<StubResponse> supportedStubs();

    RemoteCall<AccountResponse> listAccount();

    RemoteCall<ResourceResponse> listResources(Boolean ignoreRemote);

    RemoteCall<ResourceDetailResponse> detail(String path);

    RemoteCall<TransactionResponse> call(String path, String method, String...␣
↪args);

    RemoteCall<TransactionResponse> sendTransaction(String path, String method,␣
↪String... args);

    RemoteCall<TransactionResponse> invoke(String path, String method, String...␣
↪args);

    RemoteCall<TransactionResponse> callXA(
            String transactionID, String path, String method, String... args);

    RemoteCall<TransactionResponse> sendXATransaction(
            String transactionID, String path, String method, String... args);

    RemoteCall<XAResponse> startXATransaction(String transactionID, String[]␣
↪paths);

    RemoteCall<XAResponse> commitXATransaction(String transactionID, String[]␣
↪paths);

    RemoteCall<XAResponse> rollbackXATransaction(String transactionID, String[]␣
↪paths);

    RemoteCall<XATransactionResponse> getXATransaction(String transactionID,␣
↪String[] paths);

    RemoteCall<CommandResponse> customCommand(String command, String path, Object..
↪. args);

    RemoteCall<XATransactionListResponse> listXATransactions(int size);

    RemoteCall<UAResponse> register(String name, String password) throws␣
↪WeCrossSDKException;
```

(下页继续)

(续上页)

```
    RemoteCall<UAResponse> login(String name, String password);

    RemoteCall<UAResponse> logout();

    RemoteCall<UAResponse> addChainAccount(String type, ChainAccount chainAccount);

    RemoteCall<UAResponse> setDefaultAccount(String type, ChainAccount␣
↪chainAccount);

    RemoteCall<UAResponse> setDefaultAccount(String type, Integer keyID);

    String getCurrentTransactionID();
```

- 资源接口

```
    Resource ResourceFactory.build(WeCrossRPC weCrossRPC, String path, String␣
↪account)

    boolean isActive();

    ResourceDetail detail();

    String[] call(String method);

    String[] call(String method, String... args);

    String[] sendTransaction(String method);

    String[] sendTransaction(String method, String... args);
```

13.2.2 RPC封装接口解析

supportedStubs

显示router当前支持的插件列表。

参数

- 无

返回值

- `StubResponse` - 响应包
  - `version`: `String` - 版本号
  - `errorCode`: `int` - 状态码
  - `message`: `String` - 错误消息
  - `data`: `Stubs` - 支持的插件列表

java示例

```
// 初始化RPC实例
WeCrossRPCService weCrossRPCService = new WeCrossRPCService();
WeCrossRPC weCrossRPC = WeCrossRPCFactory.build(weCrossRPCService);

// 调用RPC接口，目前只支持同步调用
StubResponse response = weCrossRPC.supportedStubs().send();
```

注 - 之后的java示例，会省去初始化WeCrossRPC的步骤。

listAccount

查看当前全局账号的详细信息。

参数

- 无

返回值

- `AccountResponse` - 响应包
  - `version`: `String` - 版本号
  - `errorCode`: `int` - 状态码
  - `message`: `String` - 错误消息
  - `data`: `UniversalAccount` - 账号详细信息

java示例

```
AccountResponse response = weCrossRPC.listAccounts().send();
```

listResources

显示router配置的跨链资源。

参数

- `ignoreRemote`: `Boolean` - 是否忽略远程资源

返回值

- `ResourceResponse` - 响应包
  - `version`: `String` - 版本号
  - `errorCode`: `int` - 状态码
  - `message`: `String` - 错误消息
  - `data`: `Resources` - 配置的资源列表

java示例

```
    ResourceResponse response = weCrossRPC.listResources(true).send();
```

detail

获取资源详情。

参数

- path: String - 跨链资源标识

返回值

- ResourceDetailResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: ResourceDetail - 资源详情

java示例

```
    ResourceDetailResponse response = weCrossRPC.detail("payment.bcos.HelloWeCross
↪").send();
```

call(无参数)

调用智能合约，不更改链状态，不发交易。

参数

- path: String - 跨链资源标识
- method: String - 调用的方法

返回值

- TransactionResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: Receipt - 调用结果

java示例

```
    TransactionResponse transactionResponse =
        weCrossRPC
            .call(
                "payment.bcos.HelloWeCross","get")
            .send();
```

call(带参数)

调用智能合约，不更改链状态，不发交易。

参数

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

返回值

- TransactionResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: Receipt - 调用结果

java示例

```
    TransactionResponse transactionResponse =
        weCrossRPC
            .call(
                "payment.bcos.HelloWeCross","get","key")
            .send();
```

sendTransaction(无参数)

调用智能合约，会改变链状态，发交易。

参数

- path: String - 跨链资源标识
- method: String - 调用的方法

返回值

- TransactionResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: Receipt - 调用结果

java示例

```
TransactionResponse transactionResponse =
    weCrossRPC
        .sendTransaction(
            "payment.bcos.HelloWeCross","set")
        .send();
```

sendTransaction(带参数)

invoke(无参数)

调用智能合约，会改变链状态，发交易；在非事务状态下，与sendTransaction接口一致；在事务状态下，与sendXATransaction接口一致。

参数

- path: String - 跨链资源标识
- method: String - 调用的方法

返回值

- TransactionResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: Receipt - 调用结果

java示例

```
TransactionResponse transactionResponse =
    weCrossRPC
        .invoke(
            "payment.bcos.HelloWeCross","set")
        .send();
```

sendTransaction(带参数)

调用智能合约，会改变链状态，发交易。在非事务状态下，与sendTransaction接口一致；在事务状态下，与sendXATransaction接口一致。

参数

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

返回值

- TransactionResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: Receipt - 调用结果

java示例

```
    TransactionResponse transactionResponse =
        weCrossRPC
            .invoke(
                "payment.bcos.HelloWeCross","set","value")
            .send();
```

callXA

获取事务中的状态数据，不发交易

参数

- transactionID:String - 事务ID
- path: String - 跨链资源标识
- method: String - 调用的方法
- args : String... - 可变参数列表

返回值

- TransactionResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: Receipt - 调用结果

java示例

```
    TransactionResponse transactionResponse =
        weCrossRPC
            .callXA(
                "payment.bcos.evidence","queryEvidence","key1")
            .send();
```

sendXATransaction

执行事务，发交易

参数

- transactionID:String - 事务ID
- path: String - 跨链资源标识
- method: String - 调用的方法
- args : String... - 可变参数列表

返回值

- TransactionResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: Receipt - 调用结果

java示例

```
    TransactionResponse transactionResponse =
        weCrossRPC
            .sendXATransaction(
                "0001","payment.bcos.evidence","newEvidence","key1","evidence1")
            .send();
```

startXATransaction

开始事务，锁定事务相关资源，发交易

参数

- transactionID:String - 事务ID
- paths: String[] - 参与该事务的链路径列表

返回值

- XAResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: int - 调用结果

java示例

```
XAResponse xaResponse =
    weCrossRPC
        .startXATransaction(
            "0001", new String[]{"payment.bcos", "payment.fabric"},)
        .send();
```

commitXATransaction

提交事务，释放事务相关资源，发交易

参数

- transactionID:String - 事务ID
- paths: String[] - 参与该事务的链路径列表

返回值

- XAResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: int - 调用结果

java示例

```
XAResponse xaResponse =
    weCrossRPC
        .commitTransaction(
            "0001", new String[]{"payment.bcos", "payment.fabric"},)
        .send();
```

rollbackXATransaction

回滚事务，释放事务相关资源，发交易

参数

- transactionID:String - 事务ID
- paths: String[] - 参与该事务的链路径列表

返回值

- XAResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: int - 调用结果

java示例

```
XAResponse xaResponse =
    weCrossRPC
        .rollbackTransaction(
            "0001", new String[]{"payment.bcos", "payment.fabric"},)
        .send();
```

getXATransaction

获取事务详情，不发交易

参数

- transactionID:String - 事务ID
- paths: String[] - 参与该事务的链路径列表

返回值

- XATransactionResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: String - 事务信息

java示例

```
XATransactionResponse xaTransactionResponse =
    weCrossRPC
        .getTransactionInfo(
            "0001", new String[]{"payment.bcos", "payment.fabric"},)
        .send();
```

listXATransactions

获取事务列表，不发交易

参数

- `size`: `int` - 获取事务个数

返回值

- `XATransactionListResponse` - 响应包
  - `version`: `String` - 版本号
  - `errorCode`: `int` - 状态码
  - `message`: `String` - 错误消息
  - `data`: `String[]` - 事务信息列表

java示例

```
XATransactionListResponse xaTransactionListResponse =
    weCrossRPC
        .getTransactionIDs(
            "payment.bcos.evidence", "bcos_user1", 0)
        .send();
```

getCurrentTransactionID

获取当前SDK正处的事务ID，不发交易

参数

- 无

返回值

- `String` - 当前SDK正处的事务ID

java示例

```
String transactionID = weCrossRPC.getCurrentTransactionID();
```

customCommand

自定义命令

参数

- command: String - 命令名称
- path: String - 跨链资源标识
- args: Object... - 可变参数

返回值

- CommandResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: String - 调用结果

java示例

```
CommandResponse commandResponse =
    weCrossRPC
        .customCommand(
            "deploy", "payment.bcos.evidence", "Evidence")
        .send();
```

register

注册一个UniversalAccount账号

参数

- name: String - 账号名
- password: String - 账号密码

返回值

- UAResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: UAReceipt - 注册账号信息

java示例

```
UAResponse uaResponse =
    weCrossRPC
        .register(
            "org1-admin", "123456").send();
```

login

登录一个UniversalAccount账号

参数

- name: String - 账号名
- password: String - 账号密码

返回值

- UAResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: UAReceipt - 登录账号信息

java示例

```
UAResponse uaResponse =
    weCrossRPC
        .login(
            "org1-admin", "123456").send();
```

addChainAccount

添加一个链账号。

参数

- type: String - 链类型
- chainAccount: ChainAccount - 链账号

返回值

- UAResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: UAReceipt - 添加结果

java示例

```
    UAResponse uaResponse =
        weCrossRPC
            .addChainAccount(
                "BCOS2.0", chainAccount).send();
```

setDefaultAccount

添加一个链账号。

参数

- type: String - 链类型
- keyID: Integer - 链的KeyID

返回值

- UAResponse - 响应包
  - version: String - 版本号
  - errorCode: int - 状态码
  - message: String - 错误消息
  - data: UAReceipt - 设置结果

java示例

```
    UAResponse uaResponse =
        weCrossRPC
            .setDefaultAccount(
                "BCOS2.0", 1).send();
```

13.2.3 资源接口解析

ResourceFactory.build

初始化一个跨链资源

参数

- weCrossRPC : WeCrossRPC - RPC实例
- path: String - 跨链资源标识

返回值

- Resource - 跨链资源实例

java示例

```
    // 初始化RPC实例
    WeCrossRPCService weCrossRPCService = new WeCrossRPCService();
    WeCrossRPC weCrossRPC = WeCrossRPCFactory.build(weCrossRPCService);

    // 初始化资源实例
    Resource resource = ResourceFactory.build(weCrossRPC, path);
```

注 - 之后的java示例，会省去初始化Resource的步骤。

isActive

获取资源状态，`true`:可达，`false`:不可达。

参数

- 无

返回值

- `bool` - 资源状态

java示例

```
    bool status = resource.isActive();
```

detail

获取资源详情。

参数

- 无

返回值

- `ResourceDetail` - 资源详情

java示例

```
    ResourceDetail detail = resource.detail();
```

call(无参数)

调用智能合约，不更改链状态，不发交易。

参数

- method: String - 调用的方法

返回值

- String[] - 调用结果

java示例

```
    String[] result = resource.call("get");
```

call(带参数)

调用智能合约，不更改链状态，不发交易。

参数

- method: String - 调用的方法
- args: String... - 可变参数列表

返回值

- String[] - 调用结果

java示例

```
    String[] result = resource.call("get", "key");
```

sendTransaction(无参数)

调用智能合约，会改变链状态，发交易。

参数

- method: String - 调用的方法

返回值

- String[] - 调用结果

java示例

```
    String[] result = resource.sendTransaction("set");
```

sendTransaction(带参数)

调用智能合约，会改变链状态，发交易。

参数

- `method`: `String` - 调用的方法
- `args`: `String...` - 可变参数列表

返回值

- `String[]` - 调用结果

java示例

```
    String[] result = resource.sendTransaction("set", "value");
```

13.3 WeCross Go SDK开发应用

WeCross 跨链路由向外部暴露了所有的UBI接口，开发者可以在Go程序项目中引入Go-SDK实现这些接口的快速调用。

13.3.1 环境要求

重要:

- go版本

  要求 go1.18或以上 - WeCross服务部署 参考 部署指南

13.3.2 Go项目中引入SDK

在终端中输入命令`go get github.com/WeBankBlockchain/WeCross-Go-SDK`下载WeCross-Go-SDK。

然后在Go项目中引用即可。

```
import "github.com/WeBankBlockchain/WeCross-Go-SDK"
```

13.3.3 SDK配置说明

如果使用Go-SDK调用RPC接口，需要首先实现SDK的网络配置与SSL连接配置。

配置步骤如下：

1. 将跨链路由`router-${zone}/cert/sdk/`目录下的证书和密钥拷贝到项目的`classpath`目录；
2. 在项目的`classpath`目录下新建文件 `application.toml`，默认配置如下：

```
[connection]
    server =  '127.0.0.1:8250'
    sslKey = 'classpath:ssl.key'
    sslCert = 'classpath:ssl.crt'
    caCert = 'classpath:ca.crt'
    sslSwitch = 2 # disable ssl:2, SSL without client auth:1 , SSL with client and
↪server auth: 0
```

1. 修改application.toml中server，使用跨链路由实际监听的IP和端口。

配置解读： 在application.toml配置文件中，[connection]配置跨链路由的连接信息，具体包括以下配置项：

- server: 跨链路由监听的IP和端口；
- sslKey: 客户端私钥路径，可以使用相对路径与绝对路径；
- sslCert: 客户端证书路径，可以使用相对路径与绝对路径；
- caCert: CA证书路径，可以使用相对路径与绝对路径；
- sslSwitch: SSL的模式开关，0: 双向验证；1: 只验证服务端；2: 关闭SSL。

注意： 当sslSwitch的值为2时，sslKey，sslCert以及caCert属性所指的文件路径可以不存在（即文件可以找不到），但不能略去这几个属性。

13.3.4 使用方法

详细的API接口说明请查阅WeCross Go SDK API。

简易Go程序日志系统

本SDK提供了一种简易的日志记录系统，方便用户进行快速程序开发，用户可以选择是否开启，并且自定义设置日志系统的输出格式与输出文件，示例代码如下：

```
func main() {
        // 将标准输出添加进日志系统的输出
        logger.AddStdOutLogSystem(logger.Info)
        // 自定义一个日志输出系统
        logger.AddNewLogSystem("./", "test.log", log.LstdFlags, logger.Debug)

        // 在你的go程序中定义不同的日志标签
        testLogTag := logger.NewLogger("quickStart")
        // 使用这些标签进行日志填写
        testLogTag.Infoln("Log here as you wish.")
        testLogTag.Warnf("Use the log level you like, warn level is: %d", logger.
↪Warn)

        // 日志消息可能需要等待一定时间才能刷入标准输出， 使用flush可以强制刷新
        // 此处为了在标准输出中显示测试结果使用Flush，实际使用时无需用户自己去Flush
        logger.Flush()
}
```

日志文件./test.log输出如下：

```
2022/11/20 16:39:42 [quickStart] Log here as you wish.
2022/11/20 16:39:42 [quickStart] Use the log level you like, warn level is: 2
```

注意： WeCross-Go-SDK中RPC服务与资源调用产生的日志信息是默认添加且无法关闭的，如不能满足场景需求，建议用户在自己的go程序中使用其它日志系统。

调用RPC服务与资源封装接口

示例代码如下：

```
func main() {
        // 首先创建RPC服务并设置存放配置文件的文件目录classpath
        // classpath下应该放置application.toml
        rpcService := service.NewWeCrossRPCService()
        rpcService.SetClassPath("./tomldir") // 如不设置classpath,默认为当前程序运行目录

        err := rpcService.Init()
        if err != nil {
                panic(err)
        }

        // 初始化WeCrossRPC模块绑定RPC服务
        weCrossRPC := rpc.NewWeCrossRPCModel(rpcService)

        // 使用WeCrossRPC模块构造不同的RPC命令
        // 此处为用户登录
        call, err := weCrossRPC.Login("username", "password")
        if err != nil {
                panic(err)
        }

        // RPC执行并返回结果
        res, err := call.Send()
        if err != nil {
                panic(err)
        }
        fmt.Printf("The response is: %s\n", res.ToString())

        // 对RPC结果response更加复杂的处理,需要知道不同RPC返回的response data的数据类型
        // 更多RPC指令以及所对应的response data类型可查阅官方文档中的WeCross-Go-SDK技术文档
        data, ok := res.Data.(*response.UAReceipt)
        if !ok {
                panic("type is not right")
        }
        fmt.Printf("Universal Account info: %s\n", data.UniversalAccount.
↪ToString())

        // 使用资源封装接口前请一定先保证用户为登录态
        // 使用NewResource绑定资源的path
        testResource := resource.NewResource(weCrossRPC, "payment.bcos.HelloWecross
↪")

        // 更多资源相关API请查阅官方文档
        // 检查资源是否可达
        if !testResource.IsActive() {
                panic("the path is not active")
        }

        var callRes, sendRes []string

        // 使用绑定的资源进行调用
        callRes, err = testResource.Call("get")
        if err != nil {
                panic(err)
        }
        fmt.Println("Call Result:")
        fmt.Println(callRes)
```

(下页继续)

(续上页)

```
        // 使用绑定的资源进行调用
        sendRes, err = testResource.SendTransaction("set", "Leo")
        if err != nil {
                panic(err)
        }
        fmt.Println("SendTransaction Result:")
        fmt.Println(sendRes)
}
```

注意： 更多RPC命令，资源操作命令与Response的Data类型可查阅WeCross Go SDK API。

13.4 WeCross Go SDK API

GO SDK API分为两大类型，一种是对跨链路由RPC接口调用的封装，一种是资源接口。

13.4.1 API列表

- RPC封装接口

```
type WeCrossRPC interface {
        Test() *RemoteCall
        SupportedStubs() *RemoteCall
        QueryPub() *RemoteCall
        QueryAuthCode() *RemoteCall
        ListAccount() *RemoteCall
        ListResources(ignoreRemote bool) *RemoteCall
        Detail(path string) *RemoteCall
        Call(path, method string, args ...string) *RemoteCall
        SendTransaction(path, method string, args ...string) *RemoteCall
        Invoke(path, method string, args ...string) *RemoteCall
        CallXA(transactionID, path, method string, args ...string) *RemoteCall
        SendXATransaction(transactionID, path, method string, args ...string)␣
↪*RemoteCall
        StartXATransaction(transactionID string, paths []string) *RemoteCall
        CommitXATransaction(transactionID string, paths []string) *RemoteCall
        RollbackXATransaction(transactionID string, paths []string) *RemoteCall
        GetXATransaction(transactionID string, paths []string) *RemoteCall
        CustomCommand(command string, path string, args ...any) *RemoteCall
        ListXATransactions(size int) *RemoteCall
        Register(name, password string) (*RemoteCall, *common.WeCrossSDKError)
        Login(name, password string) (*RemoteCall, *common.WeCrossSDKError)
        Logout() *RemoteCall
        AddChainAccount(chainType string, chainAccount account.ChainAccount)␣
↪*RemoteCall
        SetDefaultAccount(chainType string, chainAccount account.ChainAccount,␣
↪keyID int) *RemoteCall
        GetCurrentTransactionID() string
}
```

- 资源接口

```
func NewResource(weCrossRPC rpc.WeCrossRPC, path string) *Resource {}
func (rsc *Resource) Check() *common.WeCrossSDKError {}
func (rsc *Resource) IsActive() bool {}
func (rsc *Resource) Detail() (*resources.ResourceDetail, *common.WeCrossSDKError)
↪{}
```

(下页继续)

(续上页)

```
func (rsc *Resource) Call(method string, args ...string) ([]string, *common.
↪WeCrossSDKError) {}
func (rsc *Resource) SendTransaction(method string, args ...string) ([]string,␣
↪*common.WeCrossSDKError) {}
```

13.4.2 数据类型解析

common.WeCrossSDKError

数据结构

```
type WeCrossSDKError struct {
        Code    ErrorCode
        Content string
}
```

类型说明

Go-SDK中的错误类型，实现了error接口类型。

rpc.RemoteCall

数据结构

```
type RemoteCall struct {
        weCrossService service.WeCrossService
        httpMethod     string
        uri            string
        responseType   response.ResponseType
        request        *types.Request
}
```

类型说明

RPC远程调用类型，通过Send函数实现远程调用并返回结果。

types.Response

数据结构

```
type Response struct {
        Version   string             // 版本号
        ErrorCode common.ErrorCode   // 状态码
        Message   string             // 错误消息
        Data      Data               // 调用结果
}
```

类型说明

RPC服务调用返回结果，其中Data为实现ToString函数的接口类型，针对不同的RPC命令返回的类型不同。

response.NullResponse

数据结构

```
type NullResponse struct {
}
```

类型说明

用于测试RPC时的返回类型，为一空结构体。

response.Stubs

数据结构

```
type Stubs struct {
        StubTypes []string `json:"stubTypes"`
}
```

类型说明

RPC命令SupportedStubs的返回类型，包含支持的stub种类。

response.Pub

数据结构

```
type Pub struct {
        Pub string `json:"pub"`
}
```

类型说明

RPC命令QueryPub的返回类型，获取公钥，前端用于对敏感数据加密。

response.AuthCodeReceipt

数据结构

```
type AuthCodeReceipt struct {
        ErrorCode common.ErrorCode `json:"errorCode"`
        Message   string           `json:"message"`
        AuthCode  *AuthCodeInfo    `json:"authCode"`
}

type AuthCodeInfo struct {
        RandomToken string `json:"randomToken"`
        ImageBase64 string `json:"imageBase64"`
}
```

类型说明

RPC命令QueryAuthCode的返回类型，获取验证码，用于用户注册与登录。

response.UAReceipt

数据结构

```
type UAReceipt struct {
        ErrorCode        common.ErrorCode          `json:"errorCode"`
        Message          string                    `json:"message"`
        Credential       string                    `json:"credential"`
        UniversalAccount *account.UniversalAccount `json:"universalAccount"`
}
```

类型说明

涉及Universal Account改变的RPC命令的返回类型，包含了当前状态下的Universal Account信息。

account.UniversalAccount

数据结构

```
type UniversalAccount struct {
        username      string
        password      string
        pubKey        string
        secKey        string
        uaID          string
        chainAccounts []ChainAccount
}
```

类型说明

RPC命令ListAccount的返回类型，包含了当前状态Universal Account信息以及其所拥有的各类型链账户信息。

resources.Resources

数据结构

```
type Resources struct {
        ResourceDetails []*ResourceDetail `json:"resourceDetails"`
}
```

类型说明

RPC命令ListResources的返回类型，包含了当前所有可用的资源信息。

resources.ResourceDetail

数据结构

```
type ResourceDetail struct {
        Path       string                 `json:"path"`
        Distance   int                    `json:"distance"`
        StubType   string                 `json:"stubType"`
        Properties map[string]interface{} `json:"properties"`
        CheckSum   string                 `json:"checkSum"`
}
```

类型说明

RPC命令Detail的返回类型，包含查询到的指定资源的所有细节信息。

response.TXReceipt

数据结构

```
type TXReceipt struct {
        ErrorCode   common.ErrorCode `json:"errorCode"`
        Message     string           `json:"message"`
        Hash        string           `json:"hash"`
        ExtraHashes []string         `json:"extraHashes"`
        BlockNumber int              `json:"blockNumber"`
        Result      []string         `json:"result"`
}
```

类型说明

在发起交易信息后返回的交易回执信息。

RawXAResponse

数据结构

```
type RawXAResponse struct {
        Status             common.StatusCode         `json:"status"`
        ChainErrorMessages []*eles.ChainErrorMessage `json:"chainErrorMessages"`
}

type ChainErrorMessage struct {
        Path    string `json:"path"`
        Message string `json:"message"`
}
```

类型说明

与事务状态有关的RPC命令返回类型。

response.RawXATransactionListResponse

数据结构

```
type RawXATransactionListResponse struct {
        XaList      []*xa.XA       `json:"xaList"`
        NextOffsets map[string]int `json:"nextOffsets"`
        Finished    bool           `json:"finished"`
}

type XA struct {
        XaTransactionID string   `json:"xaTransactionID"`
        UserName        string   `json:"username"`
        Status          string   `json:"status"`
        TimeStamp       int      `json:"timestamp"`
        Paths           []string `json:"paths"`
}
```

类型说明

RPC命令`ListXATransactions`的返回类型，包含了历史发起的事务的相关状态与信息。

response.RawXATransactionResponse

数据结构

```
type RawXATransactionResponse struct {
        XaResponse    *RawXAResponse    `json:"xaResponse"`
        XaTransaction *xa.XATransaction `json:"xaTransaction"`
}

type XATransaction struct {
        XaTransactionID    string              `json:"xaTransactionID"`
        Username           string              `json:"username"`
        Status             string              `json:"status"`
        StartTimestamp     int64               `json:"startTimestamp"`
        CommitTimestamp    int64               `json:"commitTimestamp"`
```

(下页继续)

(续上页)

```
        RollbackTimestamp  int64                `json:"rollbackTimestamp"`
        Paths              []string             `json:"paths"`
        XaTransactionSteps []*XATransactionStep `json:"xaTransactionSteps"`
}

type XATransactionStep struct {
        XaTransactionSeq int64  `json:"xaTransactionSeq"`
        Username         string `json:"username"`
        Path             string `json:"path"`
        Timestamp        int64  `json:"timestamp"`
        Method           string `json:"method"`
        Args             string `json:"args"`
}
```

类型说明

RPC命令GetXATransaction的返回类型,包含了事物的当前状态与历史步骤。

response.StringResponse

数据结构

```
type StringResponse string
```

类型说明

RPC命令CustomCommand的返回类型，为一长字符串，供用户自行解析处理。

13.4.3 RPC封装接口解析

Test

RPC服务测试函数，测试成功返回一空类型。

参数

- 无

返回值Data类型

- response.NullResponse

用例

```
        // 初始化 RPC 实例
        rpcService := service.NewWeCrossRPCService()
        rpcService.Init()
        weCrossRPC := rpc.NewWeCrossRPCModel(rpcService)
```

(下页继续)

(续上页)

```
        // 调用RPC接口，目前只支持同步调用
        call := weCrossRPC.Test()
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.NullResponse) // 类型断言
```

注 - 之后的go示例，会省去初始化WeCrossRPC的步骤。所有RPC命令的返回结果的Data类型详细信息可在前文数据类型解析中查看。

SupportedStubs

显示router当前支持的插件列表。

参数

- 无

返回值Data类型

- response.Stubs

用例

```
        call := weCrossRPC.SupportedStubs()
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.Stubs)
```

QueryPub

获取公钥，前端用于对敏感数据加密

参数

- 无

返回值Data类型

- response.Pub

用例

```
        call := weCrossRPC.QueryPub()
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.Pub)
```

QueryAuthCode

获取验证码，登录、注册时使用

参数

- 无

返回值Data类型

- response.AuthCodeReceipt

用例

```
        call := weCrossRPC.QueryAuthCode()
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.AuthCodeReceipt)
```

ListAccount

查看当前全局账号的详细信息。

参数

- 无

返回值Data类型

- account.UniversalAccount

用例

```
        call := weCrossRPC.ListAccount()
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*account.UniversalAccount)
```

ListResources

显示router配置的跨链资源。

参数

- `ignoreRemote`: `bool` - 是否忽略远程资源

返回值Data类型

- resources.Resources

用例

```
        call := weCrossRPC.ListResources(true)
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*resources.Resources)
```

Detail

获取资源详情。

参数

- path: string - 跨链资源标识

返回值Data类型

- resources.ResourceDetail

用例

```
        call := weCrossRPC.Detail("payment.bcos.HelloWeCross")
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*resources.ResourceDetail)
```

Call

调用智能合约，不更改链状态，不发交易；在事务状态下，与CallXA接口一致。

参数

- path: string - 跨链资源标识
- method: string - 调用的方法
- args : ...string - 可变参数列表

返回值Data类型

- response.TXReceipt

用例

```
        call := weCrossRPC.Call("payment.bcos.HelloWeCross","get","key")
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.TXReceipt)
```

SendTransaction

调用智能合约，会改变链状态，发交易。

参数

- path: string - 跨链资源标识
- method: string - 调用的方法
- args : ...string - 可变参数列表

返回值Data类型

- response.TXReceipt

用例

```
        call := weCrossRPC.SendTransaction("payment.bcos.HelloWeCross","set","Leo",
↪"awesome")
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.TXReceipt)
```

Invoke

调用智能合约，会改变链状态，发交易；在非事务状态下，与SendTransaction接口一致；在事务状态下，与SendXATransaction接口一致。

参数

- path: string - 跨链资源标识
- method: string - 调用的方法
- args : ...string - 可变参数列表

返回值Data类型

- response.TXReceipt

用例

```
        call := weCrossRPC.Invoke("payment.bcos.HelloWeCross","set","Leo","awesome
↪")
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.TXReceipt)
```

CallXA

获取事务中的状态数据，不发交易

参数

- transactionID:string - 事务ID
- path: string - 跨链资源标识
- method: string - 调用的方法
- args : ...string - 可变参数列表

返回值Data类型

- response.TXReceipt

用例

```
        call := weCrossRPC.CallXA("0001","payment.bcos.evidence","queryEvidence",
↪"key1")
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.TXReceipt)
```

SendXATransaction

执行事务，发交易

参数

- transactionID:string - 事务ID
- path: string - 跨链资源标识
- method: string - 调用的方法
- args : ...string - 可变参数列表

返回值Data类型

- response.TXReceipt

用例

```
        call := weCrossRPC.SendXATransaction("0001","payment.bcos.evidence",
↪"queryEvidence","key1","evidence1")
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.TXReceipt)
```

StartXATransaction

开始事务，锁定事务相关资源，发交易

参数

- transactionID:string - 事务ID
- paths: []string - 参与该事务的链路径列表

返回值Data类型

- response.RawXAResponse

用例

```
        call := weCrossRPC.StartXATransaction("0001",[]string{"payment.bcos",
↪"payment.fabric"})
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.RawXAResponse)
```

CommitXATransaction

提交事务，释放事务相关资源，发交易

参数

- transactionID:string - 事务ID
- paths: []string - 参与该事务的链路径列表

返回值Data类型

- response.RawXAResponse

用例

```
        call := weCrossRPC.CommitXATransaction("0001",[]string{"payment.bcos",
↪"payment.fabric"})
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.RawXAResponse)
```

RollbackXATransaction

回滚事务，释放事务相关资源，发交易

参数

- transactionID:string - 事务ID
- paths: []string - 参与该事务的链路径列表

返回值Data类型

- response.RawXAResponse

用例

```
        call := weCrossRPC.RollbackXATransaction("0001",[]string{"payment.bcos",
↪"payment.fabric"})
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.RawXAResponse)
```

GetXATransaction

获取事务详情，不发交易

参数

- `transactionID:string` - 事务ID
- `paths: []string` - 参与该事务的链路径列表

返回值Data类型

- response.RawXATransactionResponse

用例

```
        call := weCrossRPC.GetXATransaction("0001",[]string{"payment.bcos",
↪"payment.fabric"})
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.RawXATransactionResponse)
```

ListXATransactions

获取事务列表，不发交易

参数

- `size: int` - 获取事务个数

返回值Data类型

- response.RawXATransactionListResponse

用例

```
        call := weCrossRPC.ListXATransactions(5)
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.RawXATransactionListResponse)
```

CustomCommand

自定义命令，返回一长字符串，需用户自行解析。

参数

- command: string - 命令名称
- path: string - 跨链资源标识
- args: ...any - 可变参数

返回值Data类型

- response.StringResponse

用例

```
        call := weCrossRPC.CustomCommand("deploy", "payment.bcos.evidence",
↪"Evidence")
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.StringResponse)
```

Register

注册一个UniversalAccount账号，如果账户密码不符合规范会直接报错。

参数

- name: string - 账号名
- password: string - 账号密码

返回值Data类型

- response.UAReceipt

用例

```
        call,err := weCrossRPC.Register("org1-admin", "123456")
        if err != nil {
                panic(err)
        }
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.UAReceipt)
```

Login

登录一个UniversalAccount账号，如果账户密码不符合规范会直接报错。

参数

- name: string - 账号名
- password: string - 账号密码

返回值Data类型

- response.UAReceipt

用例

```
        call,err := weCrossRPC.Login("org1-admin", "123456")
        if err != nil {
                panic(err)
        }
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.UAReceipt)
```

Logout

退出当前UniversalAccount账号。

参数

- 无

返回值Data类型

- response.UAReceipt

用例

```
        call := weCrossRPC.Logout()
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.UAReceipt)
```

AddChainAccount

添加一个链账号。

参数

- type: string - 链类型
- chainAccount: account.ChainAccount - 链账号

返回值Data类型

- response.UAReceipt

用例

```
        call := weCrossRPC.AddChainAccount("BCOS2.0", chainAccount)
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.UAReceipt)
```

SetDefaultAccount

设置一个链账号为某一链类型的默认账号。优先使用chainAccount进行设置，若为nil，则使用KeyID。

参数

- type: string - 链类型
- chainAccount: account.ChainAccount - 链账号
- keyID: int - 链的KeyID

返回值Data类型

- response.UAReceipt

用例

```
        call := weCrossRPC.SetDefaultAccount( "BCOS2.0", nil ,1)
        rsp, _ := call.Send()
        data,_ := rsp.Data.(*response.UAReceipt)
```

GetCurrentTransactionID

返回Go-SDK当前维护的事务ID。

参数

- 无

返回值

- transactionID: string - 当前事物的ID号

用例

```
        txID := weCrossRPC.GetCurrentTransactionID()
```

13.4.4 资源接口解析

NewResource

初始化一个跨链资源。

参数

- weCrossRPC : rpc.WeCrossRPC - RPC实例
- path: string - 跨链资源标识

返回值

- Resource - 跨链资源实例

用例

```
        // 初始化 RPC 实例
        rpcService := service.NewWeCrossRPCService()
        rpcService.Init()
        weCrossRPCModel := rpc.NewWeCrossRPCModel(rpcService)

        // 初始化资源实例
        testResource := resource.NewResource(weCrossRPCModel, "payment.bcos-group1.
↪HelloWorldGroup1")
```

注 - 之后的Go示例，会省去初始化Resource的步骤。

Check

检查当前资源是否初始化成功。

参数

- 无

返回值

- common.WeCrossSDKError - 错误信息

用例

```
        err := testResource.Check()
        if err != nil {
                panic(err)
        }
```

IsActive

获取资源状态，true:可达，false:不可达。

参数

- 无

返回值

- `bool` - 资源状态

用例

```
        status := testResource.IsActive()
```

Detail

获取资源详情。

参数

- 无

返回值

- `resources.ResourceDetail` - 资源详情
- `common.WeCrossSDKError` - 错误信息

用例

```
        resourceDetail, err := testResource.Detail()
```

Call

调用智能合约，不更改链状态，不发交易。

参数

- `method`: `string` - 调用的方法
- `args`: `...string` - 可变参数列表

返回值

- `[]string` - 调用结果

用例

```
        result := resource.Call("get", "key")
```

SendTransaction

调用智能合约，会改变链状态，发交易。

参数

- method: string - 调用的方法
- args: ...string - 可变参数列表

返回值

- []string - 调用结果

用例

```
        result := resource.SendTransaction("set", "Leo", "awesome")
```

CHAPTER 14

合约跨链调用(Interchain)

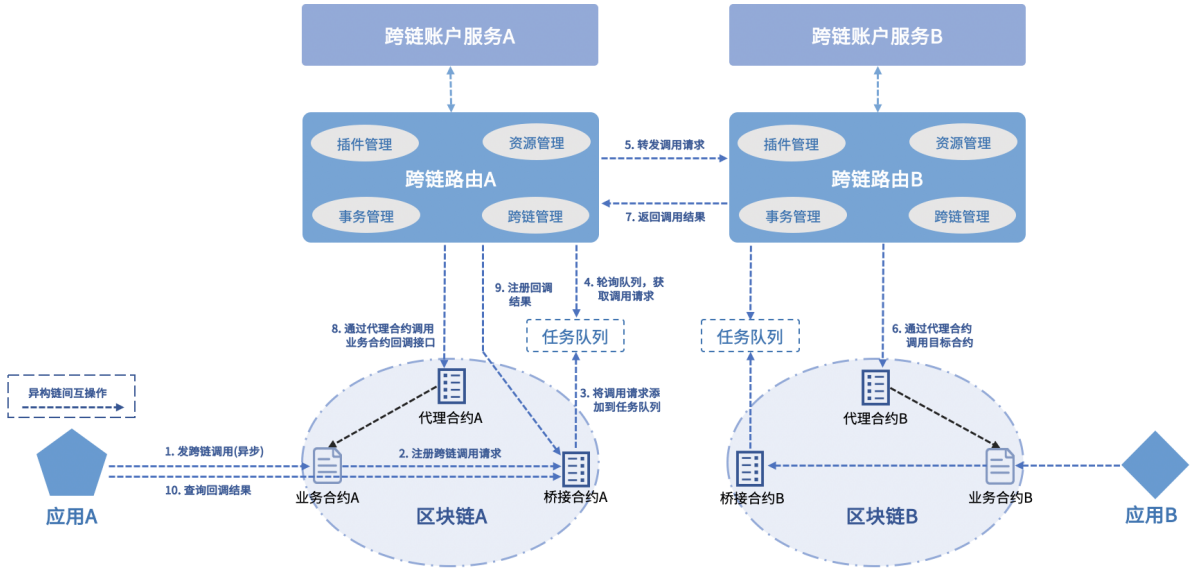
WeCross支持由合约发起跨链调用，即可在源链的智能合约中发起对其它链资源的调用。

14.1 原理解析

WeCross提供了两个系统合约，分别是代理合约和桥接合约。代理合约是WeCross调用链上其它合约的统一入口，桥接合约则负责保存链上的跨链调用请求。跨链路由通过轮询桥接合约的任务队列，获取跨链调用请求，然后完成请求的转发和处理。

合约跨链调用具体流程如下：

1. 源链的业务合约调用源链的桥接合约，注册跨链请求以及回调接口
2. 源链的跨链路由轮询源链的桥接合约，获取跨链请求
3. 源链的跨链路由解析跨链请求，完成对目标链的调用
4. 源链的跨链路由将目标链返回的结果作为参数，调用源链的回调接口
5. 源链的跨链路由将回调接口的调用结果注册到源链的桥接合约



14.2 桥接合约

WeCross提供了Solidity版本和Golang版本的桥接合约。

Solidity版本合约下载地址： GitHub访问链接 Gitee访问链接

Golang版本合约下载地址： GitHub访问链接 Gitee访问链接

- Solidity版本

```
/** 供业务合约调用，注册跨链调用请求
 *
 *  @param _path      目标链合约的路径
 *  @param _method    调用方法名
 *  @param _args      调用参数列表
 *  @param _callbackPath    回调的合约路径
 *  @param _callbackMethod 回调方法名
 *  @return 跨链请求的唯一ID
 */
function interchainInvoke(
        string memory _path,
        string memory _method,
        string[] memory _args,
        string memory _callbackPath,
        string memory _callbackMethod
) public returns(string memory uid)

/** 供用户调用，查询回调的调用结果
 *
 *  @param _uid    跨链请求的唯一ID
 *  @return        字符串数组：[事务ID, 事务Seq, 错误码, 错误消息, 回调调用结果的JSON序列化]
 */
function selectCallbackResult(
    string memory _uid
) public view returns(string[] memory)
```

- Golang版本

```
/**
 *  @param [path, method, args, callbackPath, callbackMethod]
 *  @return 跨链请求的唯一ID
 */
func (h *Hub) interchainInvoke(
    stub shim.ChaincodeStubInterface,
    args []string
) peer.Response

/**
 *  @param  uid
 *  @return 字符串数组：[事务ID, 事务Seq, 错误码, 错误消息, 回调调用结果的JSON序列化]
 */
func (h *Hub) selectCallbackResult(
    stub shim.ChaincodeStubInterface,
    args []string
) peer.Response
```

重要:

- 调用的目标链的接口定义必须匹配: `string[] func(string[] args)`
- 回调函数的接口定义必须匹配: `string[] func(bool state, string[] result)`，state表示调用目标链是否成功，result是调用结果

- 实现跨链调用的业务合约编写规范可参考示例合约，GitHub访问： Solidity版 和 Golang版
- Gitee访问： Solidity版 和 Golang版

14.3 操作示例

WeCross控制台提供了两种语言版本的跨链调用示例合约，示例合约的接口包括：

```
init(): 传入本链的桥接合约地址进行初始化

interchain(): 跨链调用的发起接口，其内部调用了桥接合约的interchainInvoke接口

get(): 获取data

set(): 更新data

callback(): 使用跨链调用的结果更新data
```

两个示例合约联动过程：A链的示例合约发起一个跨链调用，调用B链的示例合约的set接口，更新B链的data，然后触发回调，调用A链的callback接口并更新A链的data。

通过上述方式，一次控制台调用就能完成两条链数据的更新。

14.3.1 前期准备

以下操作示例涉及 FISCO BCOS 和 Hyperledger Fabric 两条链，整个跨链网络的搭建和部署请参考快速入门章节。

14.3.2 部署跨链调用示例合约

完成环境搭建后，在WeCross控制台执行以下命令：

```
# 登录
[WeCross]> login org1-admin 123456

# 部署BCOS链示例合约
[WeCross.org1-admin]> bcosDeploy payment.bcos.interchain contracts/solidity/
↪InterchainSample.sol InterchainSample 1.0

Result: 0xb9fe7fd54b0c595c40a6417ba35908f310c0aee9

# 切换Fabric默认账户
[WeCross.org1-admin]> setDefaultAccount Fabric1.4 2

# 安装链码
[WeCross.org1-admin]> fabricInstall payment.fabric.interchain Org2 contracts/
↪chaincode/interchain 1.0 GO_LANG

path: classpath:contracts/chaincode/interchain
Result: Success

# 切换Fabric默认账户
[WeCross.org1-admin]> setDefaultAccount Fabric1.4 1

# 安装链码
[WeCross.org1-admin]> fabricInstall payment.fabric.interchain Org1 contracts/
↪chaincode/interchain 1.0 GO_LANG
```

(下页继续)

(续上页)

```
path: classpath:contracts/chaincode/interchain
Result: Success

# 实例化链码
[WeCross.org1-admin]> fabricInstantiate payment.fabric.interchain ["Org1","Org2"]␣
↪contracts/chaincode/interchain 1.0 GO_LANG default []

# 等待实例化完成
```

14.3.3 查询桥接合约地址

因为示例合约需要调用桥接合约，所以先查询桥接合约地址，以用于初始化示例合约。

在BCOS链的跨链路由根目录下执行命令：

```
# 进入跨链路由根目录
cd  ~/wecross-demo/routers-payment/127.0.0.1-8250-25500

# 非国密链，其中chains/bcos是链的路径，在conf目录下可查看
java -cp 'conf/:lib/*:plugin/*' com.webank.wecross.stub.bcos.normal.preparation.
↪HubContractDeployment getAddress chains/bcos

# 国密链
java -cp 'conf/:lib/*:plugin/*' com.webank.wecross.stub.bcos.guomi.preparation.
↪HubContractDeployment getAddress chains/bcos

# 结果
WeCrossHub address: 0xb00b0a913f2c4b6bc9e7a588061a8bc55d07afe1
```

因此BCOS链的桥接合约地址为`0xb00b0a913f2c4b6bc9e7a588061a8bc55d07afe1`。而Fabric链的桥接合约名字固定为`WeCrossHub`，无需查询。

14.3.4 初始化示例合约

在WeCross控制台执行以下命令：

```
# 登录
[WeCross]> login org1-admin 123456

# 初始化BCOS示例合约
[WeCross.org1-admin]> sendTransaction payment.bcos.interchain init␣
↪0xb00b0a913f2c4b6bc9e7a588061a8bc55d07afe1

# 初始化Fabric示例合约，其中mychannel是channel名
[WeCross.org1-admin]> sendTransaction payment.fabric.interchain init mychannel␣
↪WeCrossHub
```

14.3.5 发起跨链调用

在WeCross控制台执行以下命令：

```
# 登录
[WeCross]> login org1-admin 123456

# 查看示例合约原始状态
```

(下页继续)

(续上页)

```
[WeCross.org1-admin]> call payment.fabric.interchain get

Result: [["Talk is cheap, show me the code."]]

[WeCross.org1-admin]> call payment.bcos.interchain get

Result: [["Talk is cheap, show me the code."]]

# BCOS发起跨链调用
# 命令解析：调用示例合约的interchain接口，该接口触发跨链调用
# 参数列表：[目标链资源路径] [目标接口] [调用参数] [回调资源路径] [回调接口]
[WeCross.org1-admin]> sendTransaction payment.bcos.interchain interchain payment.
↪fabric.interchain set "Hello world" payment.bcos.interchain callback

Txhash  : 0x840355ed53de047594f0a5778312edb2f2fe93908eb5efb6336cf535805f5c26
BlockNum: 1898
Result  : [1]

# 查询示例合约，发现两条链的数据都发生了变化
[WeCross.org1-admin]> call payment.bcos.interchain get

Result: [["Hello world"]]

[WeCross.org1-admin]> call payment.fabric.interchain get

Result: [["Hello world"]]

# 根据跨链调用的返回值，查询调用回调的结果
[WeCross.org1-admin]> call payment.bcos.WeCrossHub selectCallbackResult 1

Result: [[ "0", "0", "0", "Success", "[\"Hello world\"]" ]]


# Fabric发起跨链调用
[WeCross.org1-admin]> sendTransaction payment.fabric.interchain interchain payment.
↪bcos.interchain set "Hello WeCross" payment.fabric.interchain callback

Txhash  : cf7eda25f1c0515b68d702ed495fdbbefed6bdcfd4a3bc68aaab315631d3d102
BlockNum: 2386
Result  : [1]

# 查询示例合约
[WeCross.org1-admin]> call payment.bcos.interchain get

Result: [[ "Hello WeCross" ]]

[WeCross.org1-admin]> call payment.fabric.interchain get

Result: [[ "Hello WeCross" ]]

# 查询调用回调的结果
[WeCross.org1-admin]> call payment.fabric.WeCrossHub selectCallbackResult 1

Result: [["0","0","0","Success","[ \"Hello WeCross\" ]"]]
```

CHAPTER 15

区块链接入开发(Stub)

本章节首先介绍WeCross Stub的设计原理，在此基础上给出区块链接入WeCross的开发流程，用户可以根据教程实现一个区块链的WeCross Stub插件，通过插件接入WeCross。

注解:

- Java编程语言

  WeCross使用Java实现，区块链需要支持Java版本的SDK，要求开发人员具备Java开发能力。

- Gradle构建工具

  WeCross Java组件使用Gradle构建，假定用户能够使用Gradle。

15.1 Stub设计

15.2 Stub开发

Stub开发分为两个部分：

- 系统合约
- Java插件

15.2.1 系统合约

系统合约包括代理合约(WeCrossProxy)和桥接合约(WeCrossHub)，代理合约是WeCross调用该链其它合约的统一入口，桥接合约用于记录跨链调用请求，以配合跨链路由实现合约跨链调用。

代理合约

功能点：

- 合约调用入口

- 事务管理

示例：

Solidity版：

- GitHub访问链接
- Gitee访问链接

Golang版：

- GitHub访问链接
- Gitee访问链接

以Solidity合约为例，接口列表：

```
/** 读接口，调用业务合约（不需要脏读，即无事务ID，读事务中资源）
 *
 *  @param _XATransactionID  事务ID
 *  @param _path    目标资源路径
 *  @param _func    调用方法
 *  @param _args    调用参数
 *  @return 调用结果
 */
function constantCall(
    string memory _XATransactionID,
    string memory _path,
    string memory _func,
    bytes memory _args
) public returns(bytes memory)


/** 写接口，调用业务合约
 *
 *  @param _uid  交易ID，用于去重
 *  @param _XATransactionID  事务ID
 *  @param _XATransactionSeq  事务序列号
 *  @param _path    目标资源路径
 *  @param _func    调用方法
 *  @param _args    调用参数
 *  @return 调用结果
 */
function sendTransaction(
    string memory _uid,
    string memory _XATransactionID,
    uint256 _XATransactionSeq,
    string memory _path,
    string memory _func,
    bytes memory _args
) public returns(bytes memory)


/** 开启事务
 *
 *  @param _XATransactionID  事务ID
 *  @param _selfPaths  参与事务的己链资源列表
 *  @param _otherPaths 参与事务的它链资源列表
 *  @return 执行结果
 */
function startXATransaction(
    string memory _xaTransactionID,
    string[] memory _selfPaths,
    string[] memory _otherPaths
```

(下页继续)

(续上页)

```
) public returns(string memory)


/** 提交事务
 *
 *  @param _XATransactionID  事务ID
 *  @return 执行结果
 */
function commitXATransaction(
    string memory _xaTransactionID
) public returns(string memory)


/** 回滚事务
 *
 *  @param _XATransactionID  事务ID
 *  @return 执行结果
 */
function rollbackXATransaction(
  string memory _xaTransactionID
) public returns(string memory)


/** 获取事务列表
 *
 *  @param _index 下标
 *  @param _size  数量
 *  @return 事务列表详情，JSON格式
 */
function listXATransactions(
    string memory _index,
    uint256 _size
) public view returns (string memory)


/** 获取事务详情
 *
 *  @param _xaTransactionID 事务ID
 *  @return 事务详情，JSON格式
 */
function getXATransaction(
    string memory _xaTransactionID
) public view returns(string memory)
```

桥接合约

功能点：

- 注册跨链调用请求
- 查询跨链调用回调结果

示例：

Solidity版：

- GitHub访问链接
- Gitee访问链接

Golang版：

- GitHub访问链接

- Gitee访问链接

以Solidity合约为例，接口列表：

```
/** 供业务合约调用，注册跨链调用请求
 *
 *  @param _path      目标链合约的路径
 *  @param _method    调用方法名
 *  @param _args      调用参数列表
 *  @param _callbackPath    回调的合约路径
 *  @param _callbackMethod 回调方法名
 *  @return 跨链请求的唯一ID
 */
function interchainInvoke(
        string memory _path,
        string memory _method,
        string[] memory _args,
        string memory _callbackPath,
        string memory _callbackMethod
) public returns(string memory uid)


/** 供跨链路由调用，获取跨链调用请求
 *
 *  @param _num    获取数量
 *  @return        请求列表，JSON格式
 */
funct
function getInterchainRequests(
    uint256 _num
) public view returns(string memory)


/** 供跨链路由调用，更新跨链任务处理进度
 *
 *  @param _index  新下标
 */
function updateCurrentRequestIndex(uint256 _index) public


/** 供跨链路由调用，注册回调结果
 *
 *  @param _uid  跨链请求的唯一ID
 *  @param _tid  事务ID
 *  @param _seq  事务序列号
 *  @param _errorCode 回调错误码
 *  @param _errorMsg  回调错误详情
 *  @param _result    回调结果
 */
function registerCallbackResult(
    string memory _uid,
    string memory _tid,
    string memory _seq,
    string memory _errorCode,
    string memory _errorMsg,
    string[] memory _result
) public


/** 供用户调用，查询回调的调用结果
 *
 *  @param _uid    跨链请求的唯一ID
 *  @return        字符串数组：[事务ID, 事务Seq, 错误码, 错误消息, 回调调用结果的JSON序列化]
```

(下页继续)

(续上页)

```
 */
function selectCallbackResult(
    string memory _uid
) public view returns(string[] memory)
```

15.2.2 Java插件

Java核心组件

- StubFactory
  - 组件实例化工厂类
- Account
  - 区块链账户，用于交易签名
- Connection
  - 调用区块链SDK接口，与区块链交互
- Driver
  - 交易、交易回执、区块等与区块链相关数据的编解码
  - 实现Stub的基础接口
  - 调用Connection对象的发送入口与区块链交互

StubFactory

- 功能描述:
  - 添加@Stub注解，定义插件类型
  - 提供Account、Connection、Driver实例化入口

重要: @Stub定义了插件类型，添加@Stub注解的插件才能被Wecross Router识别加载!

- 接口定义

```
public interface StubFactory {
    public Driver newDriver();
    public Connection newConnection(String path);
    public Account newAccount(Map<String, Object> properties);
}
```

- 接口列表
  - newDriver
    - 实例化Driver对象
  - newConnection
    - 实例化Connection对象
      - path:配置文件路径，配置文件名称默认stub.toml
  - newAccount
    - 实例化Account对象
      - properties:Account对象实例化的参数

- FISCO-BCOS StubFactory示例

```
/** @Stub注解，插件类型：BCOS2.0 */
@Stub("BCOS2.0")
public class BCOSStubFactory implements StubFactory {
    @Override
    public Driver newDriver() {
        Driver driver = new BCOSDriver();
        /** 其他逻辑 */
        return driver;
    }
    @Override
    public Connection newConnection(String path) {
        Connection connection = new BCOSConnection();
        /** 解析配置文件，初始化 BCOSConnection */
        return connection;
    }
    @Override
    public Account newAccount(Map<String, Object> properties) {
        Account account = new BCOSAccount();
        /** 根据properties参数，初始化 BCOSAccount */
        return account;
    }
}
```

15.2.3 Account

- 功能描述
  - 交易签名
  - 链账户签名、验签
- 接口定义

```
public interface Account {
    String getName();
    String getType();
    String getIdentity();
    int getKeyID();
    boolean isDefault();
}
```

- 接口列表
  - getName
    - 链账户名称，自定义
  - getType
    - 链账户类型，与Stub类型保持一致
  - getIdentify
    - 链账户标记符，通常为链账户公钥
  - isDefault
    - 当前账户是否为默认链账户
  - getKeyID
    - 链账户KeyID

Connection

- 功能描述
  - 解析配置文件，初始化区块链JavaSDK，参考下面配置文件小节
  - 为Driver提供统一的发送接口，与区块链进行交互
  - 获取链上的资源列表
- 接口定义

```
public interface Connection {
    Response send(Request request);
    List<ResourceInfo> getResources();
}
```

接口列表

- getResources
  - 获取区块链上的资源列表

```
/** 资源对象 */
public class ResourceInfo {
    /** 资源名称 */
    private String name;
    /** 资源类型，用户自定义 */
    private String stubType;
    /** 资源属性 */
    private Map<Object, Object> properties = new HashMap<Object, Object>();
}
```

- send
  - 发送接口
    * Request request: 请求对象，包括请求类型、请求内容

```
public class Request {
    // 请求类型，自定义类型
    private int type;
    // 请求内容，序列化的请求参数
    private byte[] data;
}
```

    * Response response: 返回对象，包括返回状态、描述信息、返回内容

```
public class Response {
    // 返回状态码
    private int errorCode;
    // 描述信息
    private String errorMessage;
    // 返回内容，序列化的返回参数
    private byte[] data;
}
```

FISCO-BCOS BCOSConnection示例:

```
// Request type定义，自定义
public class BCOSRequestType {
    // 查询操作
    public static final int CALL = 1000;
    // 发送交易
    public static final int SEND_TRANSACTION = 1001;
```

(下页继续)

(续上页)

```
    // 获取块高
    public static final int GET_BLOCK_NUMBER = 1002;
    // 获取区块
    public static final int GET_BLOCK_BY_NUMBER = 1003;
    // 获取交易证明
    public static final int GET_TRANSACTION_PROOF = 1004;
}

// Connection定义各个类型消息的处理方式
public class BCOSConnection implements Connection {
    /** 发送入口，区分消息类型，调用区块链RPC接口 */
    @Override
    public Response send(Request request) {
        switch (request.getType()) {
            /** 查询 */
            case BCOSRequestType.CALL:
                /** call请求 */
                break;
            /** 发送交易 */
            case BCOSRequestType.SEND_TRANSACTION:
                /** sendTransaction请求 */
                break;
            /** 获取区块头 */
            case BCOSRequestType.GET_BLOCK_NUMBER:
                /** 获取区块高度请求 */
                break;
            /** 获取块高 */
            case BCOSRequestType.GET_BLOCK_BY_NUMBER:
                /** 获取区块 */
                break;
            /** 获取交易证明 */
            case BCOSRequestType.GET_TRANSACTION_PROOF:
                /** 获取交易证明 */
                break;
        }
    }
}
```

- 配置文件 配置文件主要包括区块链JavaSDK初始化需要的参数，也可以包含其他的一些附加信息，由用户自定义。配置默认位于chains/目录，可以配置多个stub，每个stub位于单独的子目录，配置文件名称stub.toml。

```
# 目录结构, conf/chains/stub名称/
conf/chains/
        └── bcos # stub名称: bcos
            └── stub.toml # stub.toml配置文件
            # 其他文件列表，比如：证书文件
```

stub.toml解析流程可以参考FISCO-BCOS Stub stub.toml解析

FISCO-BCOS stub.toml示例

```
[common]    # 通用配置
    name = 'bcos' # 名称，必须项
    type = 'BCOS2.0' # 必须项，插件类型，与插件@Stub注解定义的类型保持一致


[chain]     # FISCO-BCOS 属性
    groupId = 1 # default 1
    chainId = 1 # default 1
```

(下页继续)

(续上页)

```
[channelService]    # FISCO-BCOS JavaSDK配置
    caCert = 'ca.crt'
    sslCert = 'sdk.crt'
    sslKey = 'sdk.key'
    timeout = 300000  # 超时时间
    connectionsStr = ['127.0.0.1:20200', '127.0.0.1:20201', '127.0.0.1:20202'] # 连
接列表
```

Driver

- 功能描述
  - 发送交易
  - 状态查询
  - 查询块高
  - 查询区块
  - 获取交易证明
  - 交易、区块编解码
  - 验证交易
  - 查询资源列表
- 接口定义

```
public interface Driver {
    interface Callback {
        void onTransactionResponse(
                TransactionException transactionException, TransactionResponse␣
↪transactionResponse);
    }

    ImmutablePair<Boolean, TransactionRequest> decodeTransactionRequest(Request␣
↪request);

    List<ResourceInfo> getResources(Connection connection);

    void asyncCall(
            TransactionContext context,
            TransactionRequest request,
            boolean byProxy,
            Connection connection,
            Driver.Callback callback);

    void asyncSendTransaction(
            TransactionContext context,
            TransactionRequest request,
            boolean byProxy,
            Connection connection,
            Driver.Callback callback);

    interface GetBlockNumberCallback {
        void onResponse(Exception e, long blockNumber);
    }

    void asyncGetBlockNumber(Connection connection, GetBlockNumberCallback␣
↪callback);
```

(下页继续)

(续上页)

```
    interface GetBlockCallback {
        void onResponse(Exception e, Block block);
    }

    void asyncGetBlock(
            long blockNumber, boolean onlyHeader, Connection connection,␣
↪GetBlockCallback callback);

    interface GetTransactionCallback {
        void onResponse(Exception e, Transaction transaction);
    }

    void asyncGetTransaction(
            String transactionHash,
            long blockNumber,
            BlockManager blockManager,
            boolean isVerified,
            Connection connection,
            GetTransactionCallback callback);

    interface CustomCommandCallback {
        void onResponse(Exception error, Object response);
    }

    void asyncCustomCommand(
            String command,
            Path path,
            Object[] args,
            Account account,
            BlockManager blockManager,
            Connection connection,
            CustomCommandCallback callback);

    byte[] accountSign(Account account, byte[] message);

    boolean accountVerify(String identity, byte[] signBytes, byte[] message);
}
```

- 接口列表:
  - asyncCall
  - asyncSendTransaction 状态查询/发送交易
    - TransactionContext context
      - 请求上下文，交易的上下文，包含交易的附属信息
    - TransactionRequest request
    - boolean byProxy
      - 是否通过代理合约查询状态/发送交易
    - Connection connection
      - 发送请求
    - Driver.Callback callback
      - 回调返回
  - asyncGetBlockNumber 获取区块高度
    - Connection connection
      - 发送请求

        - GetBlockNumberCallback callback
            - 回调返回
    - asyncGetBlock 获取区块
        - long blockNumber
            - 区块高度
        - boolean onlyHeader
            - 是否只获取区块头
        - Connection connection
            - 发送请求
        - GetBlockCallback callback
            - 回调返回
    - asyncGetTransaction 获取交易，并且对交易进行合法性验证
        - String transactionHash
            - 交易hash
        - long blockNumber
            - 区块高度
        - BlockManager blockManager
            - 区块管理对象，用于获取区块信息，可以使用区块头部的状态信息校验交易是否合法
        - boolean isVerified
            - 是否校验交易
        - Connection connection
            - 发送请求
        - GetTransactionCallback callback
            - 回调返回
    - asyncCustomCommand 用户自定义其他接口
        - String command
            - 命令
        - Path path
            - 资源
        - Object[] args
            - 参数列表
        - Account account
            - 账户
        - BlockManager blockManager
            - 区块管理对象
        - Connection connection
            - 发送请求
        - CustomCommandCallback callback
            - 回调返回

- accountSign 链账户Account对消息进行签名，返回序列化之后的签名对象
  - Account account
    - 签名账户
  - byte[] message
    - 代签名的消息
- accountVerify 链账户验签
  - String identity
  - byte[] signBytes
    - 签名对象，accountSign的返回值
  - byte[] message
    - 签名的原始消息

15.3 开发模板

WeCross提供一个Java模板工程，加快用户开发WeCross Stub的速度，用户仅需要进行少量的修改。

获取：

```
git clone https://github.com/WeBankBlockchain/WeCross-Stub-Dev-Template.git
```

目录结构：

```
WeCross-Stub-Dev-Template
├── README.md
├── build.gradle
└── src
    ├── main
    │   ├── java
    │   │   └── wecross
    │   │       └── stub
    │   │           └── demo                    ## Java核心组件，参考上文各个组件的介绍
    │   │               ├── DemoAccount.java      # Account
    │   │               ├── DemoConnection.java   # Connection
    │   │               ├── DemoDriver.java       # Driver
    │   │               └── DemoStubFactory.java  # StubFactory
    │   └── resources
    └── test
        ├── java
        │   └── wecross
        │       └── stub
        │           └── demo
        │               └── DemoStubTest.java
        └── resources
```

编译：

```
cd WeCross-Stub-Dev-Template
bash gradlew build

$ tree -L 1 dist/apps
dist/apps
└── WeCross-Stub-Dev-Template-1.0.0-SNAPSHOT.jar
```

15.4 参考链接

WeCross-BCOS-Stub

WeCross-Fabric-Stub

WeCross文档

CHAPTER 16

JSON-RPC API

WeCross提供了Java-SDK与Go-SDK，方便Java项目与Go项目直接引入，其它语言的项目则可通过调用JSON-RPC API完成跨链开发。

16.1 状态码

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

- errorCode: 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。

状态码及其对应的含义如下:

16.2 RPC接口列表

16.2.1 pub

获取公钥，前端用于对敏感数据加密

接口URL

http://127.0.0.1:8250/auth/pub

请求方式

GET

Content-Type

application/json

请求Query参数

空

请求Body参数

空

成功响应示例


```
{
  "version": "1.0",
  "errorCode": 0,
  "message": "success",
  "data": {
    "errorCode": 0,
    "message": "success",
    "pub": "MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAiJ0u2c/
↪Xze3Jv+beltOfoaavZtatVmldwctq3ItZZ5w60whzKiNdrsFcQZqj2PszLbwiYsC4fEFKQIcaTiHCXDSQ1Km25ay8/
↪c+NKprl/
↪ruevGB1pXDnQNZhQqoaghfzijTX2bl6DJqCnuSV46sCKgKfyKm3PNPcsMUxYWC1283an3cviUvdSnZyXHN++T2Otw77EVm5
↪9vUA+jqR8ljqKJ7Q6bLVW3/xuqAN542U8a3dvPY3RVAkLVxgnl7UIfQl5PcBIxd4W3NZM6da/
↪ZGmp76MAq/
↪hxpceoU7DmQntkP9mX9ExtzcUTNFTm+LERwR530YRx4P7QB3EAAujEklZrlhXVwNa3phXnZRJWm4nuJ3qQB0I2VIw9q247a
↪r2o3Y+pCO/
↪2eBSDllsYSwCMRcgFwGvmutSD5dLes+zFZusxTRZ6vVnnnob+fOZ0NAdEDG9QY4UZoUxMjqSqM2db9jQ67QlcuMuEsc7uQ7
↪UIjvFKnw7XnvGWcT/hKTPKYbgkqOJ/KQ05DoF/W3VHU+inPMCAwEAAQ=="
  }
}
```


16.2.2 authCode

获取验证码，登录、注册时使用

接口URL

http://127.0.0.1:8250/auth/authCode

请求方式

GET

Content-Type

application/json

请求Query参数

空

请求Body参数

空

成功响应示例


```
{
  "version": "1.0",
  "errorCode": 0,
  "message": "success",
  "data": {
    "errorCode": 0,
    "message": "success",
    "authCode": {
      "randomToken":
↪"ad4b480b9585eaee7368a8260e28a198119bb88073f6f3b1aa03ede49ef1214e",
      "imageBase64":
↪"iVBORw0KGgoAAAANSUhEUgAAAJsAAAA8CAIAAAD+Gl+NAAADQUlEQVR42u3cMW7kMAwFUB1im9S5xQLpUi9yhhxiq82Vttw
↪kn2lXi6N9u/zL/
↪FjNqN2Pc6yvq2hIJyJOkyxf20M0XQNPkxvPUVXa0Ugl6gpmi2AaIZIa4pmGrOgqEWEnc1fNFuKZkvRbCmaLUVTdJ4mvllRt/
↪ffP/c9Rc/b25/XQwf/
↪8OXXE9gF+dgB8m5P0dCoxLic1zVF2ZzBUT2vo1zUbZIMghrTdRrR7WqHc+KiMs4UlYvuQ9Mxorj6o4se2EDUQ7LRGQSJUVu
↪q5so8ik+6yoWBdUTGBpsMlFk/
↪LmI0oV7Lip9A+AUTHfi9Rmj3NwUR6W995Z2xd56E2OiXDdR2WKnl1LEW/
↪bYm3gG3lwXj3W1RAU1ffWcFcxzrn1YuOuTj4oX47pa1CJ66oIWES/
↪0JlNYURy1dZdGsXjUCoy1khlQ0aeuq4tKXFbrBexQb5sQV3rFrnRk5yYqiHjrL9bSou/
↪VyKbT1iklTlL36wFFkSg9kCiozgpoZdUG4gIJPpoEjsj6/
↪COW6Xoj2MUVrAhqJaa6ySg4RokhSAxWVg7tL8qt8Q6oCI4UPe3c8N7hjre6qEXhXlbj1c1e6MgoxKwrdtVF5T6iPaB2j4iC
↪oy1QTy/
↪eKkF4aw0lDieopWhgkd7Qc4DW5qYZo50fRaAlkkvFs4cY9yUg58Xjf8Fuk3BndUJrPeJDN5moaYBKh5GK/
↪13H7rh1ifbstCCJvAyp1vZHKwMmkhQdLeo4QOWi1pyd2aoX6oOKLow68aw7XtS9yD7LGP0G/
↪T53V67Tc1kAAAAASUVORK5CYII="
    }
  }
}
```


16.2.3 register

用户注册

接口URL

http://127.0.0.1:8250/auth/register

请求方式

POST

Content-Type

application/json

请求Body参数


```
{
  "version": "1",
```


(下页继续)

(续上页)

```
  "data": "eEG9fFyTDU2RHjp/kVMTCANqrbQACm0tjGdE+UQgvniGT/
↪+xzrDDKOPpIPMPhFTs4rqOAaCHAmzzTP26i72e4l1a+YvRo1lqARxtofDMPD9ku7yaM8xz47bCz5d+9P9E/
↪i9lZiKZ1fsv1qgdTw74/Sbixh7KhPeqUUajh4v3cu4/4b57/4NKoHUAr1AlMDE1/
↪N7wcuRlWQposZLpMQrPd9uLuviWFw+l7b7ugT/VsVPIuM8K6qo7ubeMoH269jp+1/
↪tYNzqbG2bAi2uoFXSYelcETM3ew8zVxJZEGAeqgklNWSFOjKAeZSvkIceQzVH4wxVT/
↪b6+hiH7Q+hQiowK6Yd7AcHNm/
↪mCkXKdZIH87NaACWVimWowQZvrIrmINgESuMMQo60iZJc+pU46O0118WXNSeBnlgUf7LeVUz37sOn00O6rNH1/
↪ov2z7LmUo1XdCTOB24qj6Pl9040NMWWV/
↪mh8Ck+0cThhf+IKmpdS+Hx4cvPChM6mSZDI5reQdoe+Ay1ABLIAEERLwDM3Oa3IAnWaG1hzihsWh5daTi3Xo3jICX08aTy6
↪"
}
```

成功响应示例

```
{
  "version": "1",
  "errorCode": 0,
  "message": "success",
  "data": {
    "errorCode": 0,
    "message": "success",
    "universalAccount": {
      "username": "shareong",
      "pubKey":
↪"3059301306072a8648ce3d020106082a811ccf5501822d034200044f7f2e394493742fa58bf17b22ed73fd92125be9
↪",
      "uaID":
↪"3059301306072a8648ce3d020106082a811ccf5501822d034200044f7f2e394493742fa58bf17b22ed73fd92125be9
↪"
    }
  }
}
```

16.2.4 login

登录接口

接口URL

http://127.0.0.1:8250/auth/login

请求方式

POST

Content-Type

application/json

请求Body参数

```
{
  "version": "1",
  "data": "LvZLmoOgui6NAoTNuDz4T9rv5rmvFAzji+87EOm39MhfK/
↪sXeUZ0WqLPoLYHL8kabVKOAvSdzskGHUYc84O88hO1bN7aUc6RYjw2e2dJdf1Bqe0MnGRccxTEZU37mwA1JcWbpzL9yv0w6
↪3FaHhZ0YMRx9LsdD4/lsrBg3Qk0D2/
↪V1PTlt+KG32PJvwB6EsCDjQYFUhHkOfBQQZ6uZVWWcJkJQtSQ0NwFKaaQ4nUUP+dueUGD+1dfYuer1mOhjiuwuQNoMBcr8m
↪nw57+L3M01tf6YthuQV7QVDUdFZjP5q2bcX70217BYivAGJar4w6WlEhPAJTnu8ovl+DEGBNWPnhOBdtKffUW51iqtiooQf
↪+4issYzjniqu6Lf7Lx2iiejzTk10csQ/
↪DjDN1Hs1gHv4NZ7s1n8ESM7uL1Hqu2fOKIzMbctoNzVMeBidxBtdrz0g/keUMBzzN/j4meq6kDvxc/
↪FaVI0TWCuOZ5diRD/
↪+dGcOELh2eEhTdNknE67ekx1oY7RleyObulRexV3gu1C6X4PtcxfnsnfD04sGkO73zyTepUu7cE="
}
```


成功响应示例


```
{
        "version": "1",
        "errorCode": 0,
        "message": "success",
        "data": {
                "errorCode": 0,
                "message": "success",
                "credential": "Bearer␣
↪eyJpYXRtaWxsIjoxNjA2OTkyMDc5NzU0LCJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJhdWQiOiJzaGFyZW9uZyIsIm5iZiI6MTYwNjk5MjA3OSwiaXNzIjoib3JnMSIsImV4cCI6MTYwNzAxMDA3OSwiaWF0Ijo
↪9_IhhFwBajPrucruAx05noBUzFrHc7Xfl2fUObhdkX8",
                "universalAccount": {
                        "username": "shareong",
                        "uaID":
↪"3059301306072a8648ce3d020106082a811ccf5501822d034200044f7f2e394493742fa58bf17b22ed73fd92125be9c
↪",
                        "pubKey":
↪"3059301306072a8648ce3d020106082a811ccf5501822d034200044f7f2e394493742fa58bf17b22ed73fd92125be9c
↪",
                        "isAdmin": false
                }
        }
}
```


16.2.5 logout

登出接口

接口URL

http://127.0.0.1:8250/auth/logout

请求方式

POST

Content-Type

application/json

请求Header参数

请求Body参数

```
{}
```

成功响应示例

```
{
        "version": "1.0",
        "errorCode": 0,
        "message": "success",
        "data": {
                "errorCode": 0,
                "message": "success"
        }
}
```

16.2.6 listResources

获取指定区块链的资源列表

接口URL

http://127.0.0.1:8250/sys/listResources?path=payment.bcos&offset=0&size=1

请求方式

GET

Content-Type

application/json

请求Query参数

请求Header参数

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "total": 13,
                "resourceDetails": [
                        {
                                "path": "payment.bcos.HelloWorld",
                                "distance": 0,
                                "stubType": "BCOS2.0",
                                "properties": {
```

(下页继续)

(续上页)

```
                                        "BCOS_PROPERTY_CHAIN_ID": "1",
                                        "BCOS_PROPERTY_GROUP_ID": "1"
                                },
                                "checksum": null
                        }
                ]
        }
}
```

16.2.7 call

基于只读方式调用资源，若资源路径path=zone.chain.name，则访问路径为：`resource/zone/chain/name/call`

接口URL

http://127.0.0.1:8250/resource/payment/bcos/HelloWorld/call

请求方式

POST

Content-Type

application/json

请求Header参数

请求Body参数

```
{
        "version": "1",
        "path": "payment.bcos.HelloWorld",
        "data": {
                "method": "get",
                "args": [],
                "options": {
                        "XA_TRANSACTION_ID": "c7ba79423f9b4cacb5e2ee52d07f5831"
                }
        }
}
```

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "errorCode": 0,
                "message": "success",
                "hash": "",
```

(下页继续)

(续上页)

```
                "blockNumber": 0,
                "result": [
                        "Hello World"
                ]
        }
}
```

16.2.8 sendTransaction

基于发交易的方式调用资源，若资源路径path=zone.chain.name，则访问路径为：`resource/zone/chain/name/sendTransaction`

接口URL

http://127.0.0.1:8250/resource/payment/bcos/HelloWorld/sendTransaction

请求方式

POST

Content-Type

application/json

请求Header参数

请求Body参数

```
{
        "version": "1",
        "data": {
                "method": "set",
                "args": [
                        "Hello WeCross"
                ],
                "options": {
                        "XA_TRANSACTION_ID": "c7ba79423f9b4cacb5e2ee52d07f5831",
                        "XA_TRANSACTION_SEQ": 1607330421667
                }
        }
}
```

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "errorCode": 0,
                "message": "success",
                "hash": "",
```

(下页继续)

(续上页)

```
                "blockNumber": 0,
                "result": [
                        "Hello World"
                ]
        }
}
```

16.2.9 listTransactions

获取指定区块链的交易列表

接口URL

http://127.0.0.1:8250/trans/listTransactions?path=payment.bcos&blockNumber=10&offset=0&size=2

请求方式

GET

Content-Type

application/json

请求Query参数

请求Header参数

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "nextBlockNumber": 8,
                "nextOffset": 0,
                "transactions": [
                        {
                                "txHash":
↪"0xd9a1f1415826c0a8b891ddb6998b21b581d81c3cbbf624cbaa6664cfffff747e",
                                "blockNumber": 10
                        },
                        {
                                "txHash":
↪"0x11420a62d2f81dae948148a47d5ee04983cf319122b192f8acc751084be2e015",
                                "blockNumber": 9
                        }
                ]
        }
}
```

16.2.10 getTransaction

根据块高和哈希获取交易详情

接口URL

http://127.0.0.1:8250/trans/getTransaction?path=payment.bcos&txHash=0xba938113bdbae8e57dcf68f96b1edd37f288959ecc

请求方式

GET

Content-Type

application/json

请求Query参数

请求Header参数

成功响应示例


```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "path": "payment.bcos.HelloWorld",
                "username": "shareong",
                "blockNumber": 1673,
                "txHash":
↪"0xba938113bdbae8e57dcf68f96b1edd37f288959ecc0e51c0b409901c27dabafc",
                "xaTransactionID": "c7ba79423f9b4cacb5e2ee52d07f5831",
                "xaTransactionSeq": 1607330421667,
                "method": "set",
                "args": [
                        "Hello WeCross"
                ],
                "result": [],
                "byProxy": true,
                "txBytes":
↪"eyJoYXNoIjoiMHhiYTkzODExM2JkYmFlOGU1N2RjZjY4Zjk2YjFlZGQzN2YyODg5NTllY2MwZTUxYzBiNDA5OTAxYzI3ZG
↪",
                "receiptBytes":
↪"eyJ0cmFuc2FjdGlvbkhhc2giOiIweGJhOTM4MTEzYmRiYWU4ZTU3ZGNmNjhmOTZiMWVkZDM3ZjI4ODk1OWVjYzBlNTFjMG
↪"
        }
}
```


16.2.11 startXATransaction

开始事务

接口URL

http://127.0.0.1:8250/xa/startXATransaction

请求方式

POST

Content-Type

application/json

请求Header参数

请求Body参数

```
{
    "version":"1",
    "data":{
        "xaTransactionID":"1c199e6d72744133992c139c3f12ba07",
        "paths":[
            "payment.fabric.mycc",
            "payment.bcos.HelloWorld"
        ]
    }
}
```

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "status": 0,
                "chainErrorMessages": []
        }
}
```

16.2.12 commitXATransaction

提交事务，注意：提交和回滚事务只需要传入参与该事务的区块链的路径就可以了，无需传入所有参与事务的资源

接口URL

http://127.0.0.1:8250/xa/commitXATransaction

请求方式

POST

Content-Type

application/json

请求Header参数

请求Body参数

```
{
        "version": "1",
        "data": {
                "xaTransactionID": "1c199e6d72744133992c139c3f12ba17",
                "paths": [
                        "payment.fabric",
                        "payment.bcos"
                ]
        }
}
```

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "status": 0,
                "chainErrorMessages": []
        }
}
```

16.2.13 rollbackXATransaction

回滚事务，注意：提交和回滚事务只需要传入参与该事务的区块链的路径就可以了，无需传入所有参与事务的资源

接口URL

http://127.0.0.1:8250/xa/rollbackXATransaction

请求方式

POST

Content-Type

application/json

请求Header参数

请求Body参数

```
{
        "version": "1",
        "data": {
                "xaTransactionID": "1c199e6d72744133992c139c3f12ba08",
                "paths": [
                        "payment.fabric",
                        "payment.bcos"
                ]
        }
}
```

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "status": 0,
                "chainErrorMessages": []
        }
}
```

16.2.14 listXATransactions

获取事务列表，结果根据事务的开启时间排序，同时返回是否已经获取完毕，以及下一次请求的偏移

接口URL

http://127.0.0.1:8250/xa/listXATransactions

请求方式

POST

Content-Type

application/json

请求Header参数

请求Body参数

```
{
        "version": "1",
        "data": {
                "size": 2,
                "offsets": {}
```

(下页继续)

(续上页)

```
        }
}
```

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "xaList": [
                        {
                                "xaTransactionID":
↪"1c199e6d72744133992c139c3f12ba08",
                                "username": "shareong",
                                "status": "rolledback",
                                "timestamp": 1607333274,
                                "paths": [
                                        "payment.bcos.HelloWorld",
                                        "payment.fabric.mycc"
                                ]
                        },
                        {
                                "xaTransactionID":
↪"1c199e6d72744133992c139c3f12ba17",
                                "username": "shareong",
                                "status": "committed",
                                "timestamp": 1607332955,
                                "paths": [
                                        "payment.bcos.HelloWorld",
                                        "payment.fabric.mycc"
                                ]
                        }
                ],
                "nextOffsets": {
                        "payment.bcos": 15,
                        "payment.fabric": 9
                },
                "finished": false
        }
}
```

16.2.15 getXATransaction

根据事务ID获取事务详情

接口URL

http://127.0.0.1:8250/xa/getXATransaction

请求方式

POST

Content-Type

application/json

请求Header参数

请求Body参数

```
{
        "version": 1,
        "data": {
                "xaTransactionID": "1c199e6d72744133992c139c3f12ba10",
                "paths": [
                        "payment.bcos",
                        "payment.fabric"
                ]
        }
}
```

成功响应示例

```
{
        "version": "1",
        "errorCode": 0,
        "message": "Success",
        "data": {
                "xaList": [
                        {
                                "xaTransactionID":
↪"1c199e6d72744133992c139c3f12ba08",
                                "username": "shareong",
                                "status": "rolledback",
                                "timestamp": 1607333274,
                                "paths": [
                                        "payment.bcos.HelloWorld",
                                        "payment.fabric.mycc"
                                ]
                        },
                        {
                                "xaTransactionID":
↪"1c199e6d72744133992c139c3f12ba17",
                                "username": "shareong",
                                "status": "committed",
                                "timestamp": 1607332955,
                                "paths": [
                                        "payment.bcos.HelloWorld",
                                        "payment.fabric.mycc"
                                ]
                        }
                ],
                "nextOffsets": {
                        "payment.bcos": 15,
                        "payment.fabric": 9
                },
                "finished": false
        }
}
```

CHAPTER 17

应用场景

17.1 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。

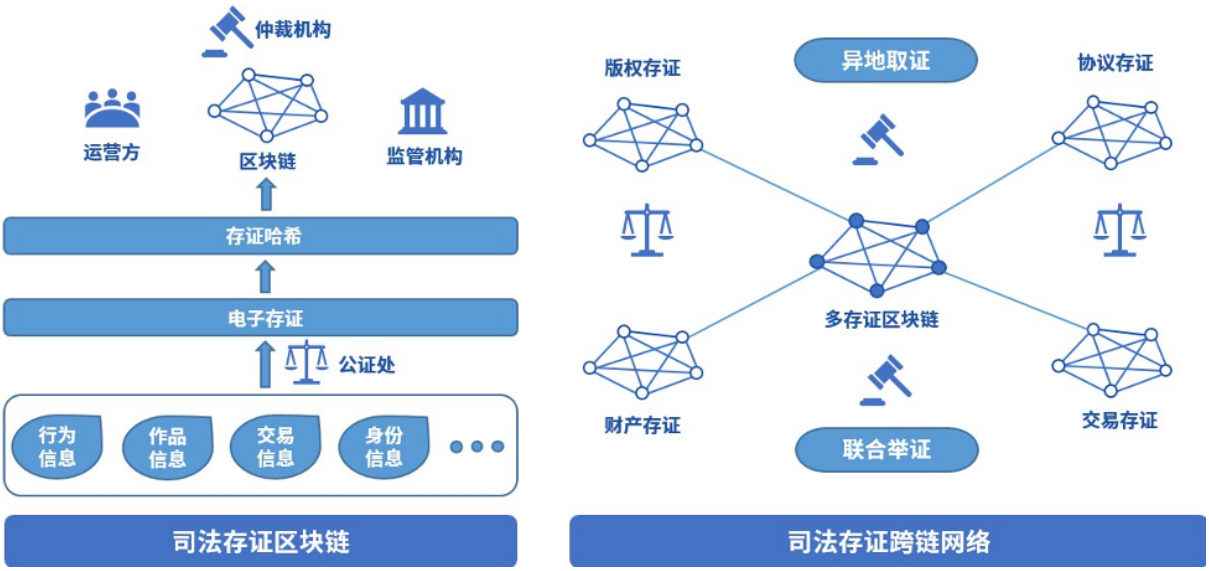
伴随着区块链在金融领域落地应用的飞速增长，多元化的数字资产场景和区块链应用带来了区块链资产相互隔离的问题，不同数字资产业务彼此搭建的区块链上的数字资产无法安全可信地实现互通，区块链上存在的数字资产价值越来越大，跨链的需求愈发迫切。

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。

17.2 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。

17.3 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。

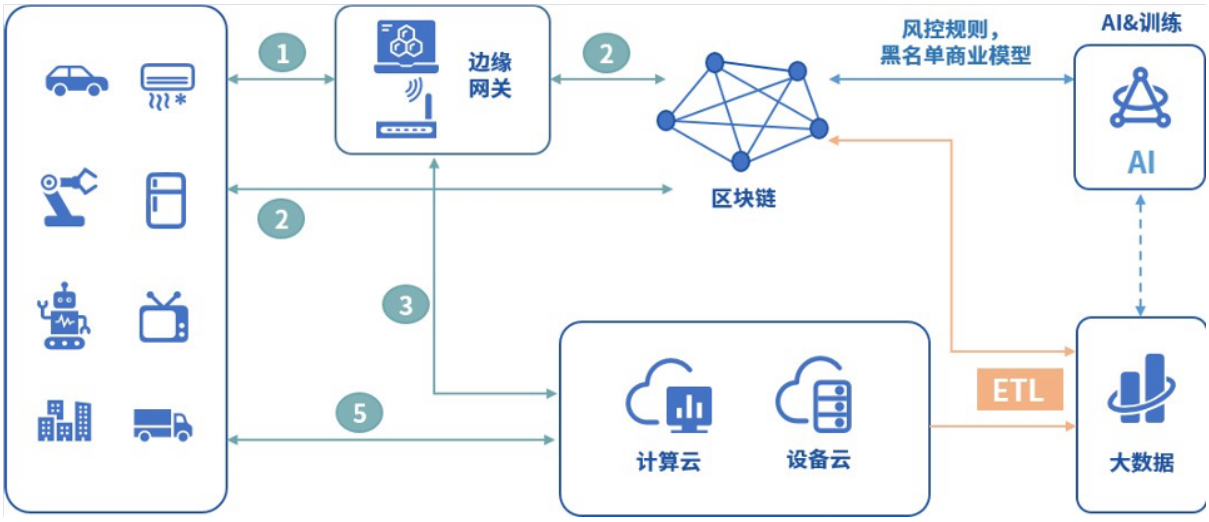
身份认证正向跨地域的方向发展，不同地域、业务和基于不同区块链平台的身份认证产品之间尚不能互认的现状造成信息的鸿沟，导致身份和资质等数据仍然局限在小范围的地域和业务内，无法互通。

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。

17.4 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 18

FAQ

18.1 部署问题

18.1.1 1. 问题：下载速度过慢/连接不上github

回答 我们提供了多种下载方式，可尝试使用国内资源下载，点击此处查看。

18.1.2 2. 问题：Demo中Fabric报错

在搭建Demo时报错

```
========== ERROR !!! FAILED to execute End-2-End Scenario ==========

ERROR !!!! Test failed
```

回答 Fabric 的demo和机器上的Fabric网络冲突了，尝试用demo目录下的`clear.sh`用脚本清理机器上已有的Fabric网络。

18.1.3 3. 问题：macOS用户出现“无法验证开发者”的情况

在部署demo时，**macOS用户**若出现“**无法打开**”，“**无法验证开发者**”的情况。

回答 macOS对下载的包权限要求较为严格，必须同一个进程下载的才可执行，可采用如下方法解决：

```
# 清理环境
cd ~/wecross-demo/ && bash clear.sh && cd ~ && rm -rf demo
# 执行下载demo包
bash <(curl -sL https://github.com/WeBankBlockchain/WeCross/releases/download/
↪resources/download_demo.sh)

# 若出现网络原因长时间无法下载，可执行以下命令下载demo包
bash <(curl -sL https://gitee.com/WeBank/WeCross/raw/master/scripts/download_demo.
↪sh)

# 进入demo文件夹，执行构建逻辑
cd demo && bash build.sh
```

18.1.4 4. 问题：用户使用MySQL 8.0+ 社区版本时，出现WeCross-Account-Manager启动错误的情况

在部署1.0.0版本的demo时，若出现`WeCross-Account-Manager`连接不上MySQL 8.0+版本时，可能有以下原因：

- MySQL配置错误，登录账号没有本地或远程访问某个数据库的权限；
- MySQL 8.0+版本默认开启SSL验证连接导致的SSL连接问题；
- MySQL 8.0+版本使用强密码检查插件；

可以做以下检查进行规避：

- 检查MySQL配置、账号可正确访问；
- （推荐）正确配置MySQL的SSL选项；
- 尝试将MySQL的SSL插件关闭后再连接，或者修改 `WeCross-Account-Manager` 的配置文件 `conf/application.toml` 的JDBC URL后面增加关闭SSL选项L：

```
#该项将在 1.1.0 版本统一使用
url = 'jdbc:mysql://localhost:3306/wecross_account_manager?useSSL=false'
```

- 关闭强密码选项，或者尝试使用较低版本的MySQL。

18.2 非技术问题

Q：WeCross和FISCO BCOS是什么关系？

A：FISCO BCOS是区块链平台，WeCross是跨链平台，WeCross可对接各种类型的区块链，包括FISCO BCOS

Q：WeCross适用于哪些应用场景？

A：WeCross适用于所有需要链间互操作的场景，包括数字资产跨链、司法仲裁跨域、个体数据跨域授权、物联网跨平台联动

Q：使用WeCross跨链需要改造原有的区块链平台或应用系统吗？

A：不需要，WeCross使用非侵入模型，不需要改造原有区块链平台和系统

Q：WeCross支持哪些区块链平台？是否支持公链？

A：参考本文档《跨链接入》章节

Q：WeCross是否支持平行扩展？

A：支持在线平行扩展

Q：WeCross最多支持多少区块链同时参与跨链？

A：WeCross本身不设区块链数量的硬限制，参与跨链的区块链数量仅受机器配置的限制

Q：WeCross支持哪几种跨链模式，中继链、侧链、平行链都支持吗？

A：中继链、侧链和平行链等模式在WeCross中均有相应方案支持

Q：WeCross是否支持主备切换或者多活架构？

A：WeCross默认是多活架构，支持主备架构

Q：WeCross是否支持审计和监管？

A：支持

Q：WeCross支持国密吗？

A：支持

Q：WeCross支持哪些事务/一致性模型?

A：参考本文档《跨链事务》章节

Q：目前有哪些落地应用场景使用了WeCross?

A：参考本文档《应用场景》章节

Q：使用WeCross遇到了问题应该去哪里咨询?

A：参考本文档《社区》章节

Q：使用WeCross有哪些好处?

A：WeCross完全开源，支持多种区块链平台，简单易用，性能高

Q：WeCross跟竞品对比，有哪些优势?

A：WeCross支持的跨链模型较多，不局限于中继链、侧链或平行链，且完全开源，支持多种区块链平台

CHAPTER 19

社区（跨链兴趣小组）

19.1 加入社区 (っ•ω•)っ

如果你有志于成为跨链协作基础设施建造者，欢迎加入WeCross社区！

你的每一个Issue，每一次PR，每一行代码，每一回“吐槽”，都是WeCross大步向前的铿锵印记。

19.1.1 跨链兴趣小组（NEW！）

FISCO BCOS跨链技术专项兴趣小组(Cross-Chain Special Interest Group，CC-SIG)正式筹建并面向开源社区招募志同道合的伙伴。

CC-SIG工作围绕跨链协作平台WeCross展开，致力于解决FISCO BCOS和其它异构链互联互通问题。兴趣小组将秉承社区对跨链的核心诉求，持续追踪技术发展与行业动态；以WeCross为基础，开展后续跨链功能研发与项目维护工作，不断探索如何构建更高效、更安全、更易用的跨链平台。

目前，CC-SIG小组已举办首次月度Meetup线上会议，经过开源社区伙伴的积极参与和踊跃讨论，确定了小组的重点工作方向。兴趣小组重点工作包括但不限于以下方向：

- 跨链基础架构设计与优化；
- 跨链平台配套工具的完善；
- 丰富区块链平台的接入；
- 跨链对数据传输和存储的可扩展性探索。

欢迎大家踊跃参与CC-SIG，详情请看 跨链兴趣小组 CC-SIG，查看更多关于CC-SIG的信息。

19.1.2 社区资源

社区是促进WeCross稳步发展的关键支撑，是推进WeCross研发进程的中坚力量，也是汇聚了所有开发者集体智慧的大本营。WeCross的开发、维护、运营都由社区协作完成。

社区资源来自开发者，用于开发者，如果你遇到问题，这些资源可以带你快速找到答案。社区也欢迎你在解决问题之后，将经验化成资源回馈社区，以飨更多开发者。

- 交流群：如在实操方面遇到阻碍或想和开发者们随时随地交流，那么加群、加群、加群，群里的“老司机”很乐意为你提供解答和支持。

- 公众号：及时掌握项目动态、了解版本更新信息、阅读来自社区的精彩文章，以及查阅近期活动安排。

- 社区活动：社区不定期举办各类型交流活动，你可以来Meetup上和社区开发者深度讨论技术，极速提高技术能力；可以花36小时在Hackathon赛场上和开发者们进行代码的创意角逐，收获一份“秃然”的革命友谊；可以参加WeCross社区例会，为开发进程建言献策，并找到合适的切入点，一起参与WeCross项目的建设。

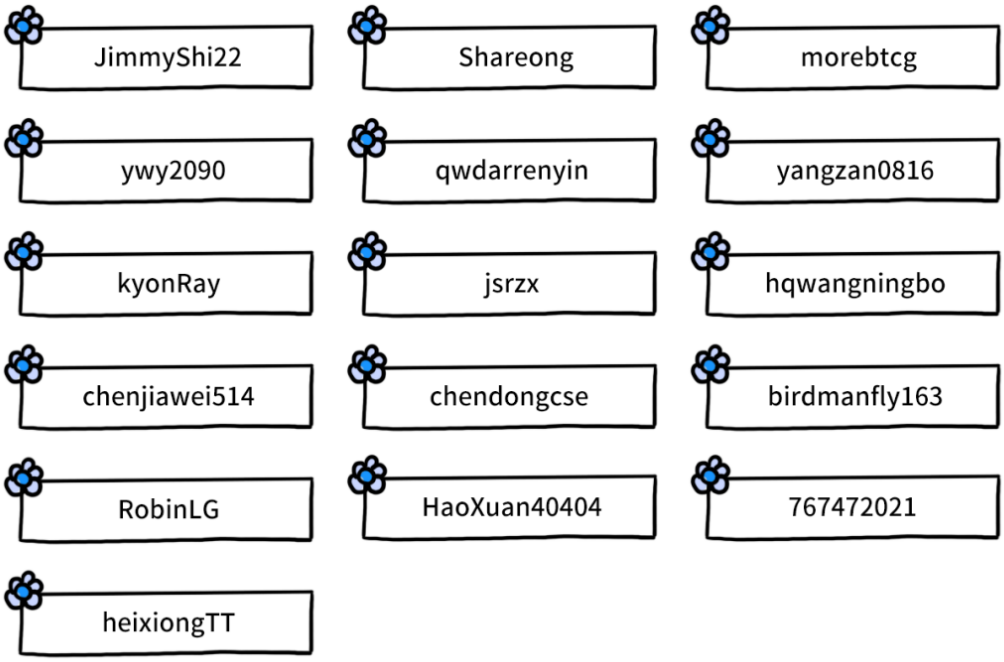
19.1.3 社区治理

每一次陪伴都值得感激。社区感谢每一位见证WeCross成长的开发者和用户，并为每一个贡献者提供专属的纪念品(神秘.JPG)。同时，参与社区活动也有机会获得礼品等福利。

> 目前，我们在WeCross Issues中设置了低难度（Task-L）和中难度（Task-M）两个梯度的Task，其中标注为help wanted的Issue均为可认领Task。
>
> 完成代码，提交PR并合入后，将根据任务难度等级获得积分，累计一定数量积分可换取神秘纪念品！详情请关注：社区活动

每一份付出都值得铭记。社区公开维护一个贡献榜记录曾参与和帮助WeCross建设的社区贡献者。榜单见证了每一位贡献者成长的轨迹，也留下了WeCross不断迭代发展的烙印。

WeCross的开发、维护、运营都由社区协作完成，感谢在各个版本开发过程中积极参与和贡献的每一位小伙伴。



如有遗漏，欢迎联系社区小助手反馈。

每一个约定都值得坚守。为了让用户的诉求得到清晰而充分地表达，为了让开发者更高效便捷地参与WeCross建设，社区协作制定了PR规范，代码规范，测试规范，改进提案(CIP)规范等社区规约，欢迎大家一起遵守和完善。

正如WeCross白皮书所言，“风起于青萍之末，一场围绕区块链技术的变革正在徐徐拉开帷幕。与一个具备无限潜力的趋势共同成长，现在，正是最好的时节”。期待更多开发者和WeCross社区并肩同行！

19.1.4 参与社区

- 报告错误和功能请求：如你发现错误、发现不合理的地方需要优化，或想提出功能请求，可以提交Issue，如果是小问题修复，欢迎顺手提个PR。
- 领取可以完成的任务：社区在Issue中提供了一些容易上手的任务，标记为`help wanted`标签，你可以从中领取合适的任务，尝试完成开发，并按照贡献代码指引提交代码。
- 提交改进提案CIP：如果社区的任务你已经游刃有余，那么做点有挑战的事吧！你可以从WeCross设计的层面提出思考，然后在Issue中提一个改进提案CIP，和社区开发者分享、讨论你的提案，并在社区统一共识后，一起完成开发。
- 分享开发心得：他山之石可以攻玉，经过充分沉淀的你可以写文章，分享任何你觉得有助于其他开发者的经验和知识，文章会署名发表在微众银行区块链公众号。

19.2 跨链兴趣小组 CC-SIG

跨链兴趣小组（Cross-Chain Special Interest Group），是一个围绕WeCross的跨链爱好者聚集地，其中不乏跨链meetup分享，前沿技术探索，跨链项目开发落地等内容。参与CC-SIG，加入圈子，在提问中获取帮助，在讨论中提升自己，在思维碰撞中让自己的项目更强，在开发者的帮助下让自己的工作更完美，在meetup分享时实现个人信仰。

19.2.1 基本介绍

- 开源社区：社区化
- 定位：围绕WeCross展开
- 目的：开放共建，汇聚技术，完善社区，成就自我

19.2.2 活动

小组内正不定期开展活动 :star:

- Project：跨链领域的多个创新项目，持续开发中
- Meetup：聚焦主题，邀请开发者分享原理，相互学习讨论
- 日常交流：提问回答，技术讨论，跨链新闻动态

Project

跨链领域的多个创新项目，持续开发中。分为四个阶段，组员可选择阶段参与

- 方案讨论：讨论需求，讨论初步方案
- 方案设计：敲定具体落地实现方案并拆分成task
- 开发阶段：领取task进行开发
- 发布阶段：测试、发布

Meetup

聚焦主题，邀请开发者分享原理，相互学习讨论。

可选方向（持续更新中）：

1. 跨链平台总体架构设计
2. 跨链交互
    - 资源抽象：IPath
    - 跨链路由的实现
    - 代理合约设计与实现
3. 统一账户
    - 账户抽象：一级账户与二级账户
    - 账户服务的实现
4. 接入区块链
    - 区块链插件设计与实现（Stub）
    - 接入FISCO BCOS 的实现
    - 接入Hyperledger Fabric 1.4 的实现
5. 跨链互信机制
    - 跨链互信机制
    - FISCO BCOS 的实现
    - Hyperledger Fabric 的实现
6. 跨链事务机制
    - 哈希时间锁定（HTLC） 原理及实现
    - 通用事务框架（XA） 原理及实现

7. 跨链开发教程
    - 跨链应用开发（SDK）
    - 合约跨链调用（Interchain）
    - 区块链接入开发（Stub）
8. 工具使用分享
    - 网页管理平台的使用
    - 终端控制台的使用

日常交流

Do as you like :)

19.2.3 参与方式

- **获取帮助**
    - 群里提问
    - 提meetup想听啥
    - 参加meetup
- **提升自己**
    - 参加meetup：获取新知识，学习提升，拓展人际关系
    - 参加project：获取实践经验，提升技术能力，积累个人经历
- **让自己的项目更强**
    - 参加meetup：相识跨链技术圈，获取研究前沿，碰撞新idea
    - 发起project：借助社区经验快速起步，汇聚力量，有效开发；借助社区平台提高知名度，吸引更多用户和开发者
    - 关注已有project：获取技术新进展
- **让从事的工作更完美**
    - 提需求：向小组提需求，社区会快速支持，实现完整功能，形成官方组件，进而反馈相关项目
    - 提供初步实现：能让社区更快的形成官方组件，同时成为核心开发者，有效积累个人经历
- **实现个人信仰**
    - meetup分享：在meetup上分享观点，传播理念
    - 发起project：借助已有代码快速搭建基础功能，借助社区平台提高知名度，借助社区力量快速开发，借助社区资源吸引用户与开发者

添加小助手微信，了解加入方式。

19.3 交流 挑战 成长

跨链技术专项兴趣小组(Cross-Chain Special Interest Group，CC-SIG)正式筹建并面向开源社区招募志同道合的伙伴。

CC-SIG工作围绕跨链协作平台WeCross展开，致力于解决FISCO BCOS和其它异构链互联互通问题。兴趣小组将秉承社区对跨链的核心诉求，持续追踪技术发展与行业动态；以WeCross为基础，开展后续跨链功能研发与项目维护工作，不断探索如何构建更高效、更安全、更易用的跨链平台。

19.3.1 长期活动

- 社区成员交流
- 分析跨链行业热点问题
- 介绍WeCross项目开发进度
- 剖析WeCross技术原理
- 制定WeCross技术路线和版本规范

Task挑战

目前，我们在WeCross Issues中设置了低难度（Task-L）和中难度（Task-M）两个梯度的Task，其中标注为help wanted的Issue均为可认领Task。

你可以在Issue中评论“挑战”，认领该Task。同一个Task可以同时被多人认领，每位开发者可同时领取多个Task。

接下来进入最重要的一步——完成代码，提交PR。开发者可自行或组队完成Task，按照PR规范提交代码并合入后，视为挑战成功。如果某个Task被多位开发者认领，则首位成功合入的参赛者视为挑战成功。

- Task-L （初阶任务），只需了解某个模块一小部分知识并使用较少代码量即可完成，挑战成功可获得积分；
- Task-M （中阶任务），需要了解某个模块多种知识完成，挑战功可获得大量积分；
- Task-S (高阶任务)，Do a big deal，一起开发一个新的Stub，期待好汉揭榜！这是一个极具挑战的任务，如果感兴趣，请留言，社区将协助你完成开发。详情请看：链接

完成Task可获得相应积分，积分可自由兑换社区礼品。

19.3.2 参与方式

- 关注社区通知获取报名渠道

19.4 操作指引

非常感谢能有心为WeCross贡献代码！

社区开发者可以针对代码中的BUG、不足和功能点发起Pull Request(PR)来申请对代码的修改。

社区将牢记每一份来自社区成员的贡献，并且会奖励每一位贡献者。

19.4.1 发起提案

改进提案(CIP，CrossChain Improvement Proposals)，是一种遵循一定规范的特殊Issue，用于提出和讨论WeCross的新需求、新特性和新功能。

CIP会对WeCross一个或多个模块的功能特性、接口规范、协议设计和逻辑流程等实现细节进行描述，可以看作是WeCross的设计蓝图。WeCross的项目维护和版本迭代都是通过社区基于CIP讨论和开发完成的。

如何发起CIP

要发起一个优秀的CIP，你可以借鉴以下的思路：

- 尽可能描述清楚你的动机，它或许不是一个痛点，但很有必要性，那么请让其他的成员也能明白你的想法。你可以通过**背景-拟解决的问题-使用的场景**这样的结构对你的CIP进行阐述。
- 尽可能的完善你的方案，这并不是必选项，但是却能帮助其他成员快速理解并一起参与你的方案设计。你可以在CIP中描述模块架构、功能列表、协议和接口设计、数据结构以及必要的时序图。
- 当CIP涉及重大功能模块的更改，或者增加了新的特性，那么兼容性和安全分析以及版权声明是不可或缺的，这样整个社区才有信心和底气一起推进该CIP的执行进程。

CIP工作流

一个CIP从提出到结束，会有很多的社区成员参与，具体的工作流如下：

1. 提案阶段(Proposal)：发起人按照CIP模板，填写CIP的功能简介，提交Issue，由社区审核。
2. 方案阶段(Draft)：社区认可该CIP提案后，为该提案赋予`Proposal`标签，然后发起人进行CIP补充。期间发起人可以与其他社区成员一起进行多轮沟通，共同完善方案。
3. 采纳阶段(Accepted)：发起人完成CIP完善后，社区决定是否采纳该CIP，如果发现还有需要优化的地方，会重新回到Draft阶段。如果确定采纳，社区会将此CIP纳入版本规划，发起者可以一起参与开发。
4. 结束(Final)：CIP功能随版本发布后，将该CIP标识为结束状态。

社区由衷感谢每一个CIP发起者，并会提供专属的纪念品。

19.4.2 代码分支策略

贡献代码之前，你需要了解WeCross项目版本控制策略。WeCross采用的是git-flow的工作流程：

- master：最新的稳定分支
- dev：待发布的稳定分支
- feature-xxx：一个正在开发xxx特性分支
- bugfix-xxx：一个正在修复xxx Bug的分支

19.4.3 代码贡献流程

无论是修复Bug还是开发新特性，都是基于以下流程：

- Fork项目仓库到个人仓库
- 确定好你的工作基础，拉取相应分支并创建一个新分支
- 编写代码以及单元测试，确保所有测试都已通过
- 提交你的工作，具体格式详见之后的PR规范
- CI系统会自动测试所有的提交请求，包括单元测试和集成测试；如果CI报错，请根据错误提示完成更改
- CI通过后，你的PR将会由社区进行审核，他们可能会提出一些修改建议：
  - 完成更改后，需要重新审核你的PR
  - 如果PR过期，你可以使用GitHub的`update branch`按钮
  - 如果存在冲突，你可以在本地通过Rebase或者Merge解决，然后强制推送到你的PR分支；你不需要仅为解决冲突而重新审核，但是如果有重大更改，则应该申请重新审核。
- 审核通过后会合并你的PR，感谢你的贡献

19.4.4 如何快速参与

如果你已经对WeCross有一定了解，并且希望能够为WeCross贡献代码，Help Wanted Issues为你列出了很多合适的开发任务，每一项都标注了难度级别，详见Task挑战介绍。当然里面也会有一些极具挑战的任务，期待极客的你迎难而上，马到功成！

如果你想搞点大事情，例如添加新的功能组件、改变目前的使用方式或者进行较大的项目优化，在实践之前请务必确保你已经在Issue区发起了一个改进提案CIP，并经过了社区的充分讨论和设计。

19.4.5 PR规范

为了让你的工作快速得到其它社区成员的理解和认可，每个PR需要你遵循一些比较粗略的约定，例如表达清楚你更改了什么，以及更改的原因。PR的Comment栏建议包含如下内容：

```
<子系统/模块>: <标注当前PR更改了什么>
<空白行>
<详细描述为什么要这样更改>
<空白行>
Signed-off-by: <你的名字> <你的电子邮箱>
```

19.4.6 代码规范

WeCross作为开源项目，需要严格把控代码质量，增强其可读性和可扩展性，以便于其它的社区成员理解WeCross代码，并快速地加入项目的开发和维护。

代码风格

WeCross项目代码采用Google Java风格，基于Gradle插件google-java-format-gradle-plugin完成代码的格式化，提交PR之前需要执行如下命令格式化代码：

```
bash gradlew goJF
```

代码质量

WeCross项目发布新版本之前会对代码进行安全扫描和渗透测试。为了避免反复的漏洞修复和检测，在提交PR之前需要使用安全工具扫描代码，并解决报告里的中高危漏洞以及不规范的编程习惯。推荐使用Sonarlint代码扫描工具，主流的IDE包括Idea和Eclipse都有对应的插件集成。

19.4.7 测试规范

为了保证代码的正确性，以及约束后续PR不会影响之前的代码逻辑，对于任何有逻辑改动的PR，都需要添加对应的单元测试。如果该PR为项目新增了功能点，那么还需要添加相应的集成测试以覆盖该功能点。

单元测试

以类为界限测试该类中所有方法的正确性，如果涉及到其它类的调用，可以使用单元测试工具Mockito模拟其它类的功能。在编写单元测试时，不仅要关注正确执行的代码流程，还要确保所有异常情况的处理都符合你的预期。务必重视单元测试，从效果上而言，单元测试就像是能执行的文档，描述了在各种条件调用这段代码时，你所期望这段代码完成的功能和效果。

集成测试

单元测试的作用收敛在了单个模块或函数，但是 WeCross是复杂的大型项目，还需要集成测试来保证所有模块组合之后整个项目依然符合预期。WeCross中的集成测试由CI触发，会搭建复杂的跨链网络，测试WeCross所有的功能点和周边组件的正确性。

CHAPTER 20

微众区块链开源生态

20.1 FISCO-BCOS 适用于金融行业的区块链底层平台

git地址：https://github.com/FISCO-BCOS

gitee地址：https://gitee.com/FISCO-BCOS

文档地址： https://fisco-bcos-documentation.readthedocs.io/

20.2 WeIdentity 基于区块链的实体身份标识及可信数据交换解决方案

git地址：https://github.com/WeBankFinTech/WeIdentity

gitee地址：https://gitee.com/WeBank/WeIdentity

文档地址：https://weidentity.readthedocs.io/

20.3 WeEvent 基于区块链的分布式事件驱动架构

git地址：https://github.com/WeBankFinTech/WeEvent

gitee地址：https://gitee.com/WeBank/WeEvent

文档地址：https://weevent.readthedocs.io/

20.4 WeBase 区块链中间件平台

git地址：https://github.com/WeBankFinTech/WeBASE

gitee地址：https://gitee.com/WeBank/WeBASE

文档地址：https://webasedoc.readthedocs.io/

20.5 WeCross 区块链跨链协作平台

git地址：https://github.com/WeBankBlockchain/WeCross

gitee地址：https://gitee.com/WeBank/WeCross

文档地址：https://wecross.readthedocs.io/

20.6 WeDPR 即时可用，场景式隐私保护高效解决方案套件和服务

git地址：https://github.com/WeBankBlockchain/WeDPR-Lab-Core

文档地址：https://wedpr-lab.readthedocs.io/

20.7 Data-Stash 数据仓库组件

git地址：https://github.com/WeBankBlockchain/Data-Stash

gitee地址：https://gitee.com/WeBankBlockchain/Data-Stash

文档地址：https://data-doc.readthedocs.io/zh_CN/stable/docs/WeBankBlockchain-Data-Stash/index.html

20.8 Data-Export 数据导出组件

git地址：https://github.com/WeBankBlockchain/Data-Export

gitee地址：https://gitee.com/WeBankBlockchain/Data-Export

文档地址：https://data-doc.readthedocs.io/zh_CN/stable/docs/WeBankBlockchain-Data-Export/index.html

20.9 Data-Reconcile 数据对账组件

git地址：https://github.com/WeBankBlockchain/Data-Reconcile

gitee地址：https://gitee.com/WeBankBlockchain/Data-Reconcile

文档地址：https://data-doc.readthedocs.io/zh_CN/stable/docs/WeBankBlockchain-Data-Reconcile/index.html

20.10 Liquid 智能合约编程语言软件

git地址：https://github.com/WeBankBlockchain/liquid

gitee地址：https://gitee.com/WeBankBlockchain/liquid

文档地址： https://liquid-doc.readthedocs.io/

20.11 cargo-liquid Liquid智能合约辅助开发工具

git地址：https://github.com/WeBankBlockchain/liquid

gitee地址：https://gitee.com/WeBankBlockchain/cargo-liquid

20.12 Governance-Account 账户治理组件

git地址：https://github.com/WeBankBlockchain/Governance-Account

gitee地址：https://gitee.com/WeBankBlockchain/Governance-Account

文档地址：https://governance-doc.readthedocs.io/zh_CN/latest/docs/WeBankBlockchain-Governance-Acct/index.html

20.13 Governance-Authority 权限治理组件

git地址：https://github.com/WeBankBlockchain/Governance-Authority

gitee地址：https://gitee.com/WeBankBlockchain/Governance-Authority

文档地址：https://governance-doc.readthedocs.io/zh_CN/latest/docs/WeBankBlockchain-Governance-Auth/index.html

20.14 Governance-Key 私钥管理组件

git地址：https://github.com/WeBankBlockchain/Governance-Key

gitee地址：https://gitee.com/WeBankBlockchain/Governance-Key

文档地址：https://governance-doc.readthedocs.io/zh_CN/latest/docs/WeBankBlockchain-Governance-Key/index.html

20.15 Governance-Cert 证书管理组件

git地址：https://github.com/WeBankBlockchain/Governance-Cert

gitee地址：https://gitee.com/WeBankBlockchain/Governance-Cert

文档地址：https://governance-doc.readthedocs.io/zh_CN/latest/docs/WeBankBlockchain-Governance-Cert/index.html

20.16 Truora 可信预言机服务

git地址：https://github.com/WeBankBlockchain/Truora

gitee地址：https://gitee.com/WeBankBlockchain/Truora

文档地址：https://truora.readthedocs.io/

20.17 SmartDev-Contract 智能合约库组件

git地址：https://github.com/WeBankBlockchain/SmartDev-Contract

gitee地址：https://gitee.com/WeBankBlockchain/SmartDev-Contract

文档地址：https://smartdev-doc.readthedocs.io/zh_CN/latest/docs/WeBankBlockchain-SmartDev-Contract/index.html

20.18 SmartDev-SCGP 合约编译插件

git地址：https://github.com/WeBankBlockchain/SmartDev-SCGP

gitee地址：https://gitee.com/WeBankBlockchain/SmartDev-SCGP

文 档 地 址 ：https://smartdev-doc.readthedocs.io/zh_CN/latest/docs/WeBankBlockchain-SmartDev-SCGP/index.html

20.19 SmartDev-Scaffold 应用开发脚手架

git地址：https://github.com/WeBankBlockchain/SmartDev-Scaffold

gitee地址：https://gitee.com/WeBankBlockchain/SmartDev-Scaffold

文 档 地 址 ：https://smartdev-doc.readthedocs.io/zh_CN/latest/docs/WeBankBlockchain-SmartDev-Scaffold/index.html