

# 目 录

## 区块链

产品简介

产品功能

产品架构

底层架构

产品优势

购买使用

定价说明

开发指南

构建第一个区块链应用

## 区块链

区块链技术的诞生，真正意义上开辟了为“数字化信用”赋能的通道。然而，目前区块链底层技术还不完全成熟、基础设施未全面完善，企业无法快速、高效、精准地投入生产力到基于区块链技术的各种业务场景中去，区块链技术面临的普及性问题亟待解决，值此关键时间窗口，漫道云区块链 BaaS 破茧而生。为了充分阐明和体现漫道云区块链 BaaS 能够赋予全行业的商业能力。

# 产品简介

## 简介

漫道云区块链服务平台依托漫道金融云基础设施，为企业及开发者提供一站式、高安全、简单易用的区块链服务。BaaS 区块链服务平台集成开发、管理和运维等功能，支持客户在云上快速部署联盟链和私有链网络环境。基于 BaaS 区块链服务平台，客户可以降低对区块链底层技术的获取成本，专注在区块链业务模式创新及业务应用的开发和运营之中。

BaaS 区块链服务平台不仅符合金融级别的安全合规性要求，同时还具备了漫道云完备的能力，用户在弹性、开放的云平台上能够快速构建自己的 IT 基础设施和区块链服务。BaaS 区块链服务平台集合众多区块链底层技术，目前已支持 **FISCO-BCOS** 与 **Hyperledger Fabric** 区块链底层平台，后续将支持 **Corda**、**EEA** 等不同区块链底层技术。

## 服务能力

漫道云区块链服务 BaaS，是一个企业级的区块链开放平台，可一键式快速部署接入、拥有去中心化信任机制、支持私有链与联盟链两种模式，拥有私有化部署与网络运维管理能力。我们看好区块链在金融、保险、零售、公益慈善等行业的应用潜力与前景。目前，我们在供应链金融、资金结算、电子票据、公益慈善等领域都已经有成功的解决方案落地。

# 产品特色

## 云上服务

BaaS 云上服务为用户所提供的区块链服务，遵循一个联盟一个系统的原则，不同的用户或者联盟链，不仅在逻辑上是严格隔离的，在机器硬件、网络、存储等物理资源上同样也是互相独立的系统，完全符合金融安全监管需求。BaaS 云服务遵循标准的区块链底层协议搭建，可以兼容网络协议一致的友商云平台。在多云融合的环境中，用户可以按照业务需求搭建真正的跨云平台联盟链，让用户解耦对底层技术平台的强依赖性，提升区块链平台自身的可信度。

## 私有云服务

在金融、电信、政府、能源、教育、交通等行业中，用户的核心业务需要自主可控。为了更好地满足这个诉求，BaaS 支持 TCE 私有化部署方式。TCE 是漫道云企业级私有云解决方案，经过工信部可信云认证，SLA 分别达到 99.95% 的服务可用性和 99.999% 的数据可靠性，达到金融行业高标准级别。BaaS 专有云部署方式搭建在稳健的 TCE 平台，用户可以自主管控整个 BaaS 云平台。

## 隐私保护

漫道云区块链平台采用基于数字证书的 PKI 的身份管理、多链隔离、信息加密、智能合约控制等手段保护私密信息。

基于 PKI 的身份管理：BaaS 平台采用双重身份认证机制。先通过漫道云官网完成账号验证，再进入到区块链的权限管理体系。BaaS 平台所使用的用户必须通过区块链资深用户管理中心注册才能获得相应身份证书，只有使用该安全证书签名的客户端节点才能发起交易请求或提案。

## 合约管理

鉴于智能合约开发是区块链应用的主要功能，所有区块链业务能力围绕智能合约为核心，来实现智能合约、自动触发、安全隔离、业务定义、数字协议等功能。客户需要花费大量的精力去编写和调试智能合约。为了解决这一困难，BaaS 平台提供完备的智能合约集成开发调试环境，大大缩短了用户开发周期并减轻了开发压力，以更便捷的方式辅助软件开发。

与其它平台不同，漫道云 BaaS 平台不仅可以对智能合约进行词法分析、语法检查，还专门提供了智能合约安全检查服务，对合规性和安全性进行校验，以防止类似于以太坊 DAO 安全事件的再次发生。

## 共识机制

共识机制决定了区块链的数据一致性的实现方式和适用场景，漫道云区块链目前支持超级账本原生共识机制的同时，未来也将支持用户自定义的共识插件和背书插件，方便用户根据自身业务需要进行灵活选择和切换。

## 开放机制

漫道云区块链 BaaS 是一个开放的服务平台，在支持超级账本 Fabric 的同时，我们也支持 BCOS 等优秀合作伙伴的区块链底层平台，在未来将支持 R3 Corda、企业以太坊等区块链技术，并积极关注区块链前沿科技的发展。

## 证书管理

与传统的公有链不同，联盟链对用户身份的管理要求和隐私保护要求更高。漫道云区块链与目前国内领先的证书服务提供商中国金融认证中心（CFCA）进行深度的战略合作，支持在漫道云区块链中使用 CFCA 签发企业所需的各类证书，例如电子签名、身份认证、SSL 证书、交易监控、反欺诈等。为客户带来全平台的证书可识别性，几乎透明的权威 CA 证书使用体验以及一体化的用户与证书管理服务。

## 硬件加密

漫道云具备成熟的硬件加密能力和产品。漫道云区块链可以直接与其无缝对接，帮助银行、保险、证券等企业充分保护其数据存储安全和传输安全，提升加密解密和签名验签效率，实现密钥安全管理，帮助客户符合监管和等级保护要求。目前漫道云硬件加密支持绝大部分主流的国密算法和各类国际通用算法，例如 SM1，SM2，SM3，SM4，DES，AES，RSA。

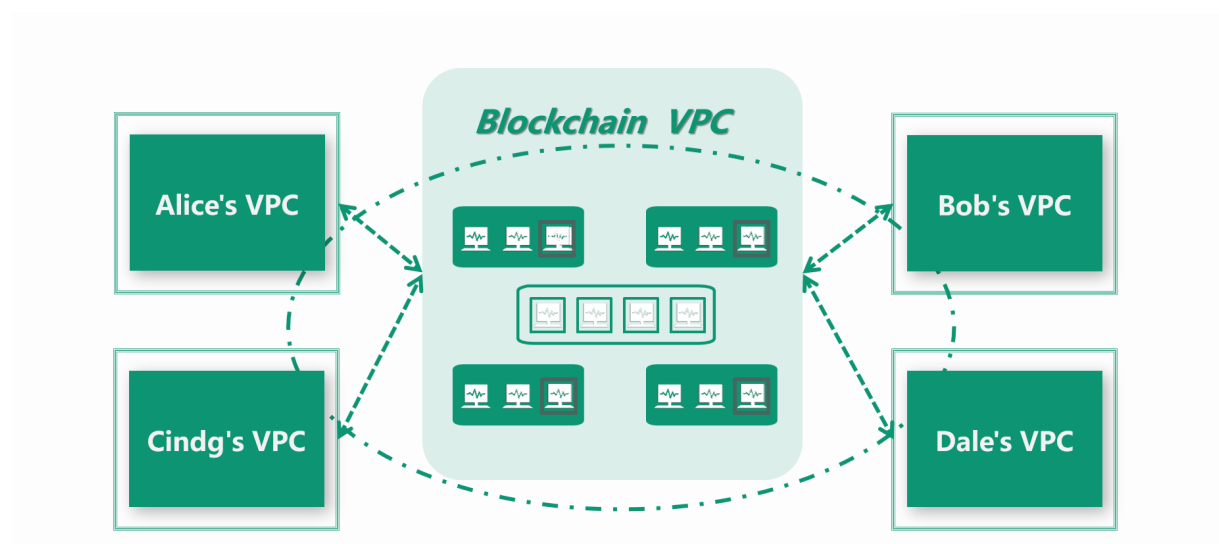
## 按需存储

区块链在不同场景下的对存储系统要求不同，漫道云区块链提供了多种存储层解决方案，以适应不同的需求。以 Hyperledger Fabric 为例，存储分为三部分，即账本数据、状态数据和历史数据。账本数据

支持使用传统块存储解决方案，例如性能更好的 CBS 云硬盘或者成本更低、运维更简便的 CFS，并支持快照镜像等备份和快速拷贝需求，方便新节点加入区块链后快速同步账本。状态数据和历史数据除了使用原生的 GoLevel DB 和 CouchDB 以外，未来可以使用漫道云端的 MongoDB 方案。

## 企业互联

云端企业用户通常拥有若干个自己的 VPC，VPC 之间天然隔离。漫道云区块链以 VPC 的形式部署和提供服务，将区块链部署于一个独立的 VPC 中，不占用用户的 VPC 配额，同时支持将区块链 VPC 与其他多个用户的 VPC 快速打通，不受限于网络地址重叠与繁琐的路由配置等因素的影响，方便用户直接通过自己的 VPC 访问自己的组织和节点，让用户无需为用户网络互联和后续拓展担心。



除此以外，BaaS 平台作为一个开放的平台，漫道云区块链对用户已有的 IT 基础设施充分保护，支持用户复用现有 VPC 和用户自有 IDC 内的基础设施，使其作为区块链的一部分节点。

## 网络管理

BaaS 平台依托于漫道云基础设施，提供高速、低延迟的区块链网络。为区块链各个节点间提供无阻塞、无超载的高可靠性的通信，且通信保护符合金融业安全级别标准。

漫道区块链的网络遵照联盟链的准入机制，采用多组织联合、区域自治的方式、多链隔离等技术，实现了网络既可以灵活的拓展，又可以实现自适应的管理模式。

## 联盟链治理

联盟采用邀请准入的机制，仅对成员开放。成员可邀请新成员加入联盟。联盟中的成员可自由发展成多个区块链网络。在网络中的每一个业务通道，参与成员需经过投票准入机制，以获得区块链上的读写权限、参与记账的权限。

联盟链针对企业级的管理和监控需求，对用户的权限、角色，和各种共识策略、访问策略进行了全方位的增强。BaaS 为用户提供了完善的联盟链管理机制，通过灵活的准入机制与权限策略管理，实现了对不同用户访问权限的控制。



## 产品功能

功能	功能描述
概览	展示用户加入的联盟、 区块链网络相关的统计信息， 以及产品动态和常见问题。
联盟	展示用户创建的或者加入的联盟详细信息， 支持用户新建联盟，邀请成员加入联盟等操作。
区块链网络	展示用户创建的或者加入的区块链网络详细信息， 支持用户新建区块链网络， 邀请组织加入网络等操作。 区块链网络支持联盟链和私有链两种形态。
事件中心	事件流转与处理中心， 用户可以在事件中心查看和处理联盟加入邀请、 区块链网络加入邀请，通道准入投票等事件。
自动化部署	支持自主自动化部署和配置区块链。
通道概览	区块数量 / 生成速率 / 列表与详情，节点数量 / 列表，交易数量 / 频次，智能合约数量。
通道管理	区块链服务器上所有的通道信息， 可以添加新的通道和查看各个通道的详情。
合约管理	添加新合约到某一通道上， 并为合约组织上的节点安装合约和初始化合约。

功能	功能描述
节点管理	节点信息和节点加入不同通道和合约的管理。
策略管理	合约背书策略的管理。
证书管理	权威CA接入，区块链网络中证书的管理，包括证书申请、吊销、补发等功能。
组织查看	查看区块链上的各个组织信息。
日志查看	记录用户在 MBaaS 控制台的关键行为操作，帮助用户查看其操作日志，了解历史操作动态。
监控告警	支持多级事件告警，系统主动向注册邮箱推送告警信息，支持管理页面查询告警事件。

# 产品架构



- 漫道云区块链服务 BaaS，提供两种模式的服务：漫道公有云金融专区和漫道专有云。用户可以根据实际业务需求，购买金融云上的区块链服务。BaaS 也支持将系统搭建在自建数据中心或私有云上。
- BaaS 构建于 Docker 和 Kubernetes 之上，自身具备极高的可靠性和扩展性，同时与漫道云其他产品能力完全打通，用户无需担

心数据膨胀和性能等问题，同时客户可以根据自身的需求，灵活地选择将整个区块链网络或者部分节点构建于 BaaS 之上，BaaS 提供多种互联互通能力，支持客户充分使用现有 IT 基础设施，并连接周边生态和业务合作伙伴。

- BaaS 目前支持 Hyperledger Fabric，BCOS等区块链底层平台。BaaS 提供多种用户层接口，包括 API 和多语言 SDK，并进一步优化服务能力，提供一站式的合约编辑、调试、部署、运行环境，大限度降低区块链系统门槛。同时，针对各个行业的特性，BaaS 提供实践和高度定制化的解决方案，帮助各行业客户快速落地业务场景。

# 底层架构

## FISCO BCOS 概述

**FISCO BCOS** 平台是金融区块链合作联盟（深圳）（以下简称：金融链盟）开源工作组以金融业务实践为参考样本，深度定制的安全可控、适用于金融行业且完全开源的区块链底层平台。目前 FISCO BCOS 已有数十应用落地，百级参与机构，千级社区成员，并且仍在与日俱增中。其影响力遍及以支付、对账、交易清结算、资产数字化、供应链金融、数据存证、征信、场外市场等为代表的金融领域，以及司法存证、文化版权、娱乐游戏、社会管理、政务服务等领域。

## 特色与优势

**FISCO BCOS 平台** 基于价值联盟、安全可信、业务可行、自主可控、高效可用、智能监管、灵活配置七大理理念进行设计。针对联盟链商业级生产面临的“高安全行、高性能、高可用性、业务落地、合法合规”五座大山提供以下解决方案：

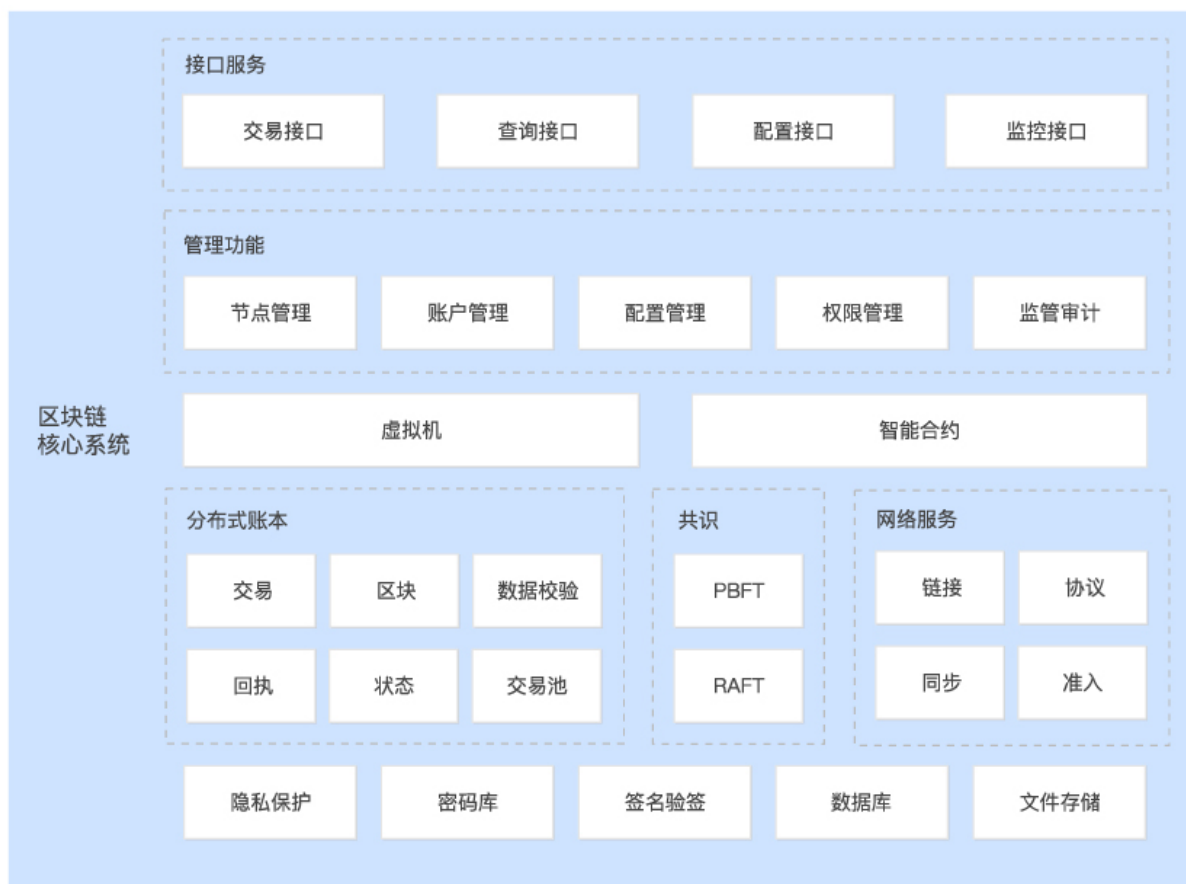
- 在安全性方面，FISCO BCOS 通过节点准入控制、可靠的密钥管理、灵活的权限控制，在应用、存储、网络、主机层实现全面的安全保障。
- 在性能优化方面，FISCO BCOS 优化网络通信模型，采用拜占庭容错的共识机制，结合多链架构和跨链交互方案，解决并发访

问和热点帐户的性能痛点，从而满足金融级高频交易场景需求。

- 在可用性方面，FISCO BCOS 设计为 7×24小时 运行，达到金融级高可用性，通过简化建链过程、适应多种环境的部署方式、全局配置更新实现了高可用性。
- 在业务落地方面，FISCO BCOS 提供各种开发接口，方便程序员编写和调用智能合约。
- 在监管方面，FISCO BCOS 支持监管和审计机构作为观察节点加入联盟链，获取实时数据进行监管审计。

## 系统框架

## 架构图



## 开源社区

FISCO BCOS 开源社区 : <https://github.com/fisco-bcos>

FISCO BCOS 文档地址 : [https://fisco-bcos-documentation.readthedocs.io/zh\\_CN/latest/docs/community/index.html](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/community/index.html)

## 常见问题

FISCO BCOS FAQ 文档 : [https://fisco-bcos-documentation.readthedocs.io/zh\\_CN/latest/docs/community/FAQ.html](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/community/FAQ.html)



## 产品优势

### 全流程运维服务

- 满足用户个性化需求，一站式快速交付定制 BaaS 服务。
- 支持底层区块链平台的版本选择与补丁更新、升级。

### 一体化监控

- 集群全天候、实时多维度监控。
- 支持自定义、多渠道告警。

### 底层区块链产品

- 兼容性：底层区块链平台源代码开放，兼容社区版本。
- 标准化：数据迁移标准化，迁移成本可控。
- 定制化：定制化安全特性，支持国密算法；定制化共识框架，支持 RAFT、BFT 等共识算法。

## 高性能

- 15节点的区块链网络中实现单通道超过7000TPS。

## 网络扩展性

- 允许节点、成员动态加入/退出区块链网络。

## 基础设施弹性

- 支持物理基础设施无缝扩展，不中断现有业务。
- 支持多种智能合约开发语言，包括 `go`、`nodeJS` 等

## 丰富的配套工具

- 智能合约开发 IDE：支持智能合约在线编辑、调试和部署。
- 权威CA：集成 CFCA 等权威证书认证中心。
- 证书管理：提供统一的证书管理中心，支持一站式证书申请、审批与吊销。
- 合约管理：管理智能合约，实现智能合约的部署、查看和升级。
- 用户管理：帮助组织更好的管理用户，实现用户身份认证、权限管理等功能。
- 区块链浏览器：区块与交易信息统计、浏览工具，实现区块链数据的可视化管理。



## 购买使用

## 申请体验说明

目前区块链为内测体验，参与体验者可自行申请账户进行体验，也可用以下公共账户进行体验。

## 定价说明

漫道云 BaaS 提供了在云中的一键式部署的区块链网络资源，允许您挑选多种节点配置来部署您的 BaaS 实例。后续随着业务量的变化，您还可以动态添加组织、节点、扩容存储空间以及升级节点配置等调整您的 BaaS 规格。

漫道云 BaaS 提供包年包月的预付费计费模式，具体请见定价信息。

## 定价详情

- 计费模式：包年包月。
- 付款方式：预付费。
- 计费单位：元/月，至少使用一个月。
- 调整实例节点配置：随时可升级节点配置，扩容存储空间，升配无限制。
- 包年包月模式下费用计算：

说明：

区块链服务实例的价格随时间变动，下文中的数值仅作参考。

例如：您在广州三区，购买一套区块链服务实例，实例中包含共识服务、1个网络组织，2个网络节点，节点证书选用默认 CA 生成，节点配置为一台标准型 S2 的4核8G的云服务器，系统盘为50G普通云硬盘，数据盘选择200G云硬盘，购买时长为1个月。用户所需所支付的费用计算如下：

- 共识服务与节点证书（默认 CA）：免费。
- 网络组织与节点： $4750\text{元/个/月} * 1 * 2\text{个} = 9500\text{元/月}$ 。
- 标准型 S2 的4核8G： $360\text{元/台/月} * 2\text{台} = 720\text{元/月}$ 。
- 系统盘50G免费，数据盘为200G云硬盘： $0.3\text{元/GB/月} * 200\text{G} * 2\text{台} = 120\text{元/月}$ 。

故总费用为： $9500+720+120 = 10340$  元/月。

# 开发指南

# 构建第一个区块链应用

本章将会介绍一个基于 `FISCO BCOS区块链底层` 业务应用场景开发全过程，从业务场景分析，到合约的设计实现，然后介绍合约编译以及如何部署到区块链，最后介绍一个应用模块的实现，通过我们提供的Web3SDK实现对区块链上合约的调用访问。

本教程要求用户熟悉Linux操作环境，具备Java开发的基本技能，能够使用Gradle工具，熟悉Solidity语法[\[合约开发教程\]](#)。通过学习教程，你将会了解到以下内容：

1. 如何将一个业务场景的逻辑用合约的形式表达
2. 如何将Solidity合约转化成Java类
3. 如何配置Web3SDK
4. 如何构建一个应用，并集成Web3SDK到应用工程
5. 如何通过Web3SDK调用合约接口，了解Web3SDK调用合约接口的原理

教程中会提供示例的完整项目源码，用户可以在此基础上快速开发自己的应用。

## 示例应用需求

区块链天然具有防篡改，可追溯等特性，这些特性决定其更容易受金融领域的青睐，本文将会提供一个简易的资产管理的开发示例，并最终实现以下功能：

- 能够在区块链上进行资产注册



- 能够实现不同账户的转账
- 可以查询账户的资产金额

## 合约设计与实现

在区块链上进行应用开发时，结合业务需求，首先需要设计对应的智能合约，确定合约需要储存的数据，在此基础上确定智能合约对外提供的接口，最后给出各个接口的具体实现。

### 存储设计

FISCO BCOS提供[合约CRUD接口](#)开发模式，可以通过合约创建表，并对创建的表进行增删改查操作。针对本应用需要设计一个存储资产管理的表 `t_asset`，该表字段如下：

- account: 主键，资产账户(string类型)
- asset\_value: 资产金额(uint256类型)

其中account是主键，即操作 `t_asset` 表时需要传入的字段，区块链根据该主键字段查询表中匹配的记录。 `t_asset` 表示例如下：

account	asset_value
Alice	10000
Bob	20000

## 接口设计

按照业务的设计目标，需要实现资产注册，转账，查询功能，对应功能的接口如下：

```
// 查询资产金额
function select(string account) public constant
returns(int256, uint256)
// 资产注册
function register(string account, uint256 amount) public returns(int256)
// 资产转移
function transfer(string from_asset_account, string to_asset_account, uint256 amount) public returns(int256)
```

## 完整源码

```
# 切换到fisco/console/tools目录
$ cd ~/fisco/console/tools/
```

```
cat > ./contracts/Asset.sol << EOF
pragma solidity ^0.4.24;

import "./Table.sol";

contract Asset {
    // event
```

```
    event RegisterEvent(int256 ret, string account, uint256 asset_value);
```

```
    event TransferEvent(int256 ret, string from_account, string to_account, uint256 amount);
```

```
    constructor() public {
```

```
        // 构造函数中创建t_asset表
```

```
        createTable();
```

```
    }
```

```
    function createTable() private {
```

```
        TableFactory tf = TableFactory(0x1001);
```

```
        // 资产管理表, key : account, field : asset_value
```

```
        // | 资产账户(主键) | 资产金额
```

```
|
```

```
        // | ----- | -----
```

```
-----|
```

```
        // | account | asset_value
```

```
e |
```

```
        // | ----- | -----
```

```
-----|
```

```
        //
```

```
        // 创建表
```

```
        tf.createTable("t_asset", "account", "asset_value");
```

```
    }
```

```
    function openTable() private returns(Table)
```

```
{
```

```

        TableFactory tf = TableFactory(0x1001);
        Table table = tf.openTable("t_asset");
        return table;
    }

    /*
    描述 : 根据资产账户查询资产金额
    参数 :
        account : 资产账户

    返回值 :
        参数一 : 成功返回0, 账户不存在返回-1
        参数二 : 第一个参数为0时有效, 资产金额
    */
    function select(string account) public constant returns(int256, uint256) {
        // 打开表
        Table table = openTable();
        // 查询
        Entries entries = table.select(account,
            table.newCondition());
        uint256 asset_value = 0;
        if (0 == uint256(entries.size())) {
            return (-1, asset_value);
        } else {
            Entry entry = entries.get(0);
            return (0, uint256(entry.getInt("asset_value")));
        }
    }
}

```

```

    /*
    描述 : 资产注册
    参数 :
        account : 资产账户
        amount : 资产金额
    返回值 :
        0 资产注册成功
        -1 资产账户已存在
        -2 其他错误
    */
    function register(string account, uint256 as
set_value) public returns(int256){
        int256 ret_code = 0;
        int256 ret= 0;
        uint256 temp_asset_value = 0;
        // 查询账号是否存在
        (ret, temp_asset_value) = select(accoun
t);
        if(ret != 0) {
            Table table = openTable();

            Entry entry = table.newEntry();
            entry.set("account", account);
            entry.set("asset_value", int256(asse
t_value));
            // 插入
            int count = table.insert(account, en
try);
            if (count == 1) {

```

```

        // 成功
        ret_code = 0;
    } else {
        // 失败? 无权限或者其他错误
        ret_code = -2;
    }
} else {
    // 账户已存在
    ret_code = -1;
}

emit RegisterEvent(ret_code, account, as
set_value);

return ret_code;
}

```

/\*

描述 : 资产转移

参数 :

from\_account : 转移资产账户

to\_account : 接收资产账户

amount : 转移金额

返回值 :

0 资产转移成功

-1 转移资产账户不存在

-2 接收资产账户不存在

-3 金额不足

-4 金额溢出

-5 其他错误

```

    */
    function transfer(string from_account, string to_account, uint256 amount) public returns(int256) {
        // 查询转移资产账户信息
        int ret_code = 0;
        int256 ret = 0;
        uint256 from_asset_value = 0;
        uint256 to_asset_value = 0;

        // 转移账户是否存在?
        (ret, from_asset_value) = select(from_account);
        if(ret != 0) {
            ret_code = -1;
            // 转移账户不存在
            emit TransferEvent(ret_code, from_account, to_account, amount);
            return ret_code;
        }

        // 接受账户是否存在?
        (ret, to_asset_value) = select(to_account);
        if(ret != 0) {
            ret_code = -2;
            // 接收资产的账户不存在
            emit TransferEvent(ret_code, from_account, to_account, amount);

```

```

        return ret_code;
    }

    if(from_asset_value < amount) {
        ret_code = -3;
        // 转移资产的账户金额不足
        emit TransferEvent(ret_code, from_account, to_account, amount);
        return ret_code;
    }

    if (to_asset_value + amount < to_asset_value) {
        ret_code = -4;
        // 接收账户金额溢出
        emit TransferEvent(ret_code, from_account, to_account, amount);
        return ret_code;
    }

    Table table = openTable();

    Entry entry0 = table.newEntry();
    entry0.set("account", from_account);
    entry0.set("asset_value", int256(from_asset_value - amount));
    // 更新转账账户
    int count = table.update(from_account, entry0, table.newCondition());
    if(count != 1) {

```



```

        ret_code = -5;
        // 失败? 无权限或者其他错误?
        emit TransferEvent(ret_code, from_ac
count, to_account, amount);
        return ret_code;
    }

    Entry entry1 = table.newEntry();
    entry1.set("account", to_account);
    entry1.set("asset_value", int256(to_asse
t_value + amount));
    // 更新接收账户
    table.update(to_account, entry1, table.n
ewCondition());

    emit TransferEvent(ret_code, from_accoun
t, to_account, amount);

    return ret_code;
}
}
EOF

```

注： `Asset.sol` 合约的实现需要引入FISCO BCOS提供的一个系统合约接口文件 `Table.sol`，该系统合约文件中的接口由FISCO BCOS底层实现。当业务合约需要操作CRUD接口时，均需要引入该接口合约文件。 `Table.sol` 合约详细接口[参考这里](#)。

## 合约转换

上一小节，我们根据业务需求设计了合约 `Asset.sol` 的存储与接口，给出了完整实现，但是Java程序无法直接调用Solidity合约，需要先将Solidity合约文件转换为Java文件。

控制台提供了这种转换的工具，可以将 `Asset.sol` `Table.sol` 两个合约文件存放在 `console/tools/contracts` 目录，利用`console/tools`目录下提供的 `sol2java.sh` 脚本进行转换，操作如下：

```
# 编译合约，后面指定一个Java的包名参数，可以根据实际项目
# 路径指定包名
$ ./sol2java.sh org.fisco.bcos.asset.contract
```

运行成功之后，将会在`console/tools`目录生成`java`、`abi`和`bin`目录，如下所示。

```
| -- abi // 生成的abi目录，存放solidity合约编译生成的abi文件
|   | -- Asset.abi
|   | -- Table.abi
| -- bin // 生成的bin目录，存放solidity合约编译生成的bin文件
|   | -- Asset.bin
|   | -- Table.bin
| -- contracts // 存放solidity合约源码文件，将需要编译的合约拷贝到该目录下
|   | -- Asset.sol // 拷贝进来的Asset.sol合约，依赖Table.sol
|   | -- Table.sol // 默认提供的系统CRUD合约接口文件
| -- java // 存放编译的包路径及Java合约文件
|   | -- org
```

```
| --fisco
| --bcos
| --asset
| --contract
| --Asset.java //
Asset.sol合约生成的Java文件
| --Table.java //
Table.sol合约生成的Java文件
|-- sol2java.sh
```

java目录下生成了 `org/fisco/bcos/asset/contract/` 包路径目录，该目录下包含 `Asset.java` 和 `Table.java` 两个文件，其中 `Asset.java` 是Java应用调用 `Asset.sol` 合约需要的文件。

`Asset.java` 的主要接口：

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

public class Asset extends Contract {
    // Asset.sol合约 transfer接口生成
    public RemoteCall<TransactionReceipt> transfer(String from_account, String to_account, BigInteger amount);
    // Asset.sol合约 register接口生成
    public RemoteCall<TransactionReceipt> register(String account, BigInteger asset_value);
    // Asset.sol合约 select接口生成
    public RemoteCall<Tuple2<BigInteger, BigInteger>> select(String account);
```

```

// 加载Asset合约地址，生成Asset对象
public static Asset load(String contractAddress, Web3j web3j, Credentials credentials, ContractGasProvider contractGasProvider);

// 部署Assert.sol合约，生成Asset对象
public static RemoteCall<Asset> deploy(Web3j web3j, Credentials credentials, ContractGasProvider contractGasProvider);
}

```

其中load与deploy函数用于构造Asset对象，其他接口分别用来调用对应的solidity合约的接口，详细使用在下文会有介绍。

## SDK配置

我们提供了一个Java工程项目供开发使用，首先获取Java工程项目：

```

# 获取Java工程项目压缩包
$ cd ~
$ curl -LO https://github.com/FISCO-BCOS/LargeFiles/raw/master/tools/asset-app.tar.gz
# 解压得到Java工程项目asset-app目录
$ tar -zxf asset-app.tar.gz

```

asset-app项目的目录结构如下：

```

|-- build.gradle // gradle配置文件
|-- gradle

```

```
| | -- wrapper
| | -- gradle-wrapper.jar // 用于下载Gradle的
相关代码实现
| | -- gradle-wrapper.properties // wrapper
所使用的配置信息，比如gradle的版本等信息
|-- gradlew // Linux或者Unix下用于执行wrapper命令的
Shell脚本
|-- gradlew.bat // windows下用于执行wrapper命令的批
处理脚本
|-- src
| | -- main
| | | -- java
| | | | -- org
| | | | | -- fisco
| | | | | | -- bcos
| | | | | | | -- asset
| | | | | | | -- client
// 放置客户端调用类
| | | | | | | -- A
ssetClient.java
| | | | | | | -- contract
// 放置Java合约类
| | | | | | | -- A
sset.java
| | -- test
| | -- resources // 存放代码资源文件
| | -- applicationContext.xml // 项目配
置文件
| | -- ca.crt // 区块链ca证书
| | -- node.crt // 区块链ca证书
```

```
| -- node.key // 节点证书
| -- contract.properties // 存储部署合约地址的文件
| -- log4j.properties // 日志配置文件
| -- contract //存放solidity约文件
| -- Asset.sol
| -- Table.sol
|
|-- tool
    |-- asset_run.sh // 项目运行脚本
```

## 项目引入Web3SDK

项目的 `build.gradle` 文件已引入Web3SDK，不需修改。其引入方法介绍如下：

- Web3SDK引入了以太坊的solidity编译器相关jar包，因此在 `build.gradle` 文件需要添加以太坊的远程仓库：

```
repositories {
    maven {
        url "http://maven.aliyun.com/nexus/content/groups/public/"
    }
    maven { url "https://dl.bintray.com/ethereum/maven/" }
    mavenCentral()
}
```

- 引入Web3SDK jar包

```
compile ('org.fisco-bcos:web3sdk:2.0.0-rc2')
```

## 证书与配置文件

- 区块链节点证书配置

拷贝区块链节点对应的SDK证书

```
# 进入~目录
# 拷贝节点证书到项目的资源目录
$ cd ~
$ cp fisco/nodes/127.0.0.1/sdk/* asset-app/src/test/resources/
```

- applicationContext.xml

**注意：**如果搭链时设置的rpc\_listen\_ip为127.0.0.1或者0.0.0.0，channel\_port为20200，则 `applicationContext.xml` 配置不用修改。若区块链节点配置有改动，需要同样修改配置 `applicationContext.xml`，具体请参考[SDK使用文档](#)。

## 业务开发

我们已经介绍了如何在自己的项目中引入以及配置Web3SDK，本节介绍如何通过Java程序调用合约，同样以示例的资产管理说明。

asset-app项目已经包含示例的完整源码，用户可以直接使用，现在介绍核心类 `AssetClient` 的设计与实现。

`AssetClient.java`：通过调用 `Asset.java` 实现对合约的部署与调用，路径 `/src/main/java/org/fisco/bcos/asset/client`，初始化以及调用流程都在该类中进行。

- 初始化

初始化代码的主要功能为构造Web3j与Credentials对象，这两个对象在创建对应的合约类对象(调用合约类的deploy或者load函数)时需要使用。

```
// 函数initialize中进行初始化
ApplicationContext context = new ClassPathXmlApp
licationContext("classpath:applicationContext.xml");
Service service = context.getBean(Service.class);
service.run();

ChannelEthereumService channelEthereumService =
new ChannelEthereumService();
channelEthereumService.setChannelService(service);
// 初始化Web3j对象
Web3j web3j = Web3j.build(channelEthereumService, 1);
// 初始化Credentials对象
Credentials credentials = Credentials.create(Keys.createEcKeyPair());
```



- 构造合约类对象

可以使用deploy或者load函数初始化合约对象，两者使用场景不同，前者适用于初次部署合约，后者在合约已经部署并且已知合约地址时使用。

```
// 部署合约
Asset asset = Asset.deploy(web3j, credentials, new StaticGasProvider(gasPrice, gasLimit)).send();
// 加载合约地址
Asset asset = Asset.load(contractAddress, web3j, credentials, new StaticGasProvider(gasPrice, gasLimit));
```

- 接口调用

使用合约对象调用对应的接口，处理返回结果。

```
// select接口调用
Tuple2<BigInteger, BigInteger> result = asset.select(assetAccount).send();
// register接口调用
TransactionReceipt receipt = asset.register(assetAccount, amount).send();
// transfer接口
TransactionReceipt receipt = asset.transfer(fromAssetAccount, toAssetAccount, amount).send();
```

# 运行

至此我们已经介绍使用区块链开发资产管理应用的所有流程并实现了功能，接下来可以运行项目，测试功能是否正常。

- 编译

```
# 切换到项目目录
$ cd ~/asset-app
# 编译项目
$ ./gradlew build
```

编译成功之后，将在项目根目录下生成 `dist` 目录。dist目录下有一个 `asset_run.sh` 脚本，简化项目运行。现在开始——验证本文开始定下的需求。

- 部署 `Asset.sol` 合约

```
# 进入dist目录
$ cd dist
$ bash asset_run.sh deploy
Deploy Asset succesfully, contract address is 0xd09ad04220e40bb8666e885730c8c460091a4775
```

- 注册资产

```
$ bash asset_run.sh register Alice 100000
Register account succesfully => account: Alice,
```

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

- 查询资产

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

- 资产转移

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

**总结：**至此，我们通过合约开发，合约编译，SDK配置与业务开发构建了一个基于FISCO BCOS联盟区块链的应用。