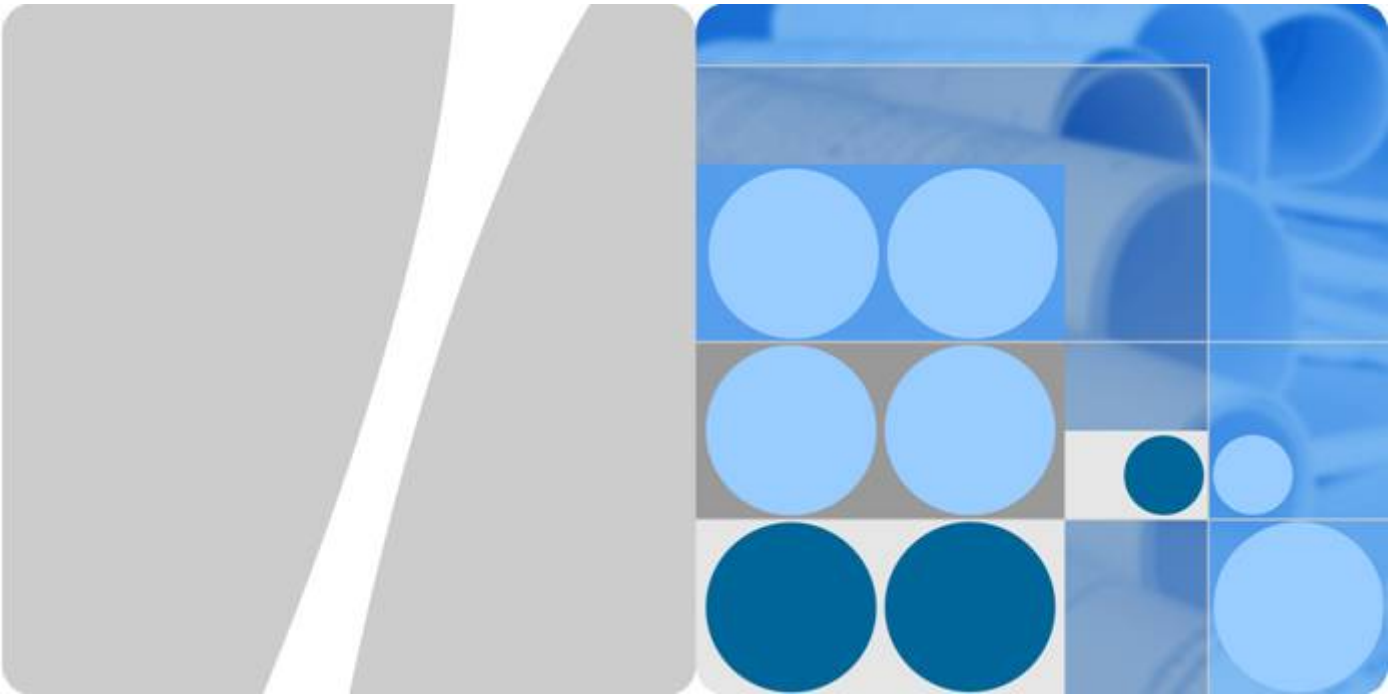


区块链服务开发指南

文档版本　01

发布日期　2018-04-19

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

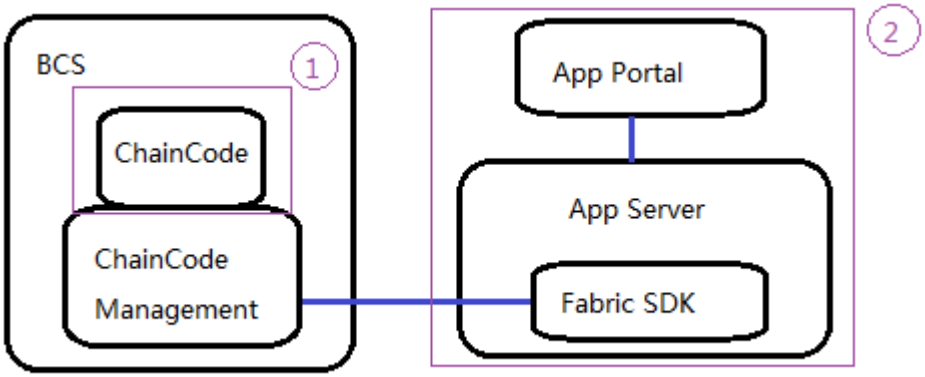
客户服务电话： 4008302118

目 录

1 概述

华为云区块链服务（Blockchain Service）面向企业及开发者，提供高性能、高可用、高安全区块链技术平台服务，帮助客户在华为云上快速、低成本的创建、部署和管理区块链应用及商业智能合约服务。

在使用区块链服务时用户需要开发自己的链代码（即智能合约）和交易应用，如下图中的①和②。



本文基于区块链身份共享的银行II类账户跨行开户的示例来介绍区块链服务开发流程。为使客户通过公网申请二类银行账户，我们需要创建容器集群并绑定弹性IP、订购区块链服务、开发安装及实例化链代码、开发配置构建并部署应用程序。

说明

如果您对业务链代码和客户端APP的设计和开发有需求，可以联系华为云区块链合作伙伴提供进一步服务，我们会结合您的业务以及华为云的优势和特点为您提供完善的解决方案，联系邮箱如下：

邮箱1：**wanglei189@huawei.com**

邮箱2：**liuzaiyao@huawei.com**

2 准备工作

需要完成如下环境准备工作，包括：创建容器集群、绑定弹性IP、创建网络存储。

表 2-1 准备工作

序号	操作指导	说明
1	创建容器集群	华为云区块链服务是基于容器所构建的集群进行部署的，区块链服务中的Peer节点、共识节点以及链代码均需要运行在容器中。此处您需要创建一个名为test的集群。
2	绑定弹性IP	通过申请弹性IP并将其绑定到弹性云服务器上，实现弹性云服务器为公网所访问的目的。
3	创建网络存储	创建网络存储用于存储数据。

3 订购区块链服务

订购区块链服务时需要您为区块链服务配置基本参数和网络节点，以便快速完成区块链服务的创建及部署。

步骤1 登录区块链服务管理控制台。

步骤2 单击页面右上角的“购买区块链服务”，在订购页面填写相关参数。

说明

- 为了保证示例Demo成功运行，请在订购页面按照如下表格填写参数。
- 若您不需要运行Demo应用，如下参数可根据您的需要进行配置。

表 3-1 参数表

参数名	参数值
区块链服务名称	test
区块链类型	私有链
容器集群	test
网络存储	选择创建好的网络存储。
节点组织	创建3个节点组织，分别为：xxx1，数量2；xxx2，数量2；xxx3，数量2。
共识策略	快速拜占庭容错共识算法
版本信息	选择最新版本
链代码管理初始密码	请自行设置。
确认密码	请自行设置。
共识节点数量	4
通道配置	创建名为testchannel的通道，并将xxx1、xxx2、xxx3节点组织添加进此通道。

步骤3 完成购买后，进入服务汇总列表，如下图：

组织状态若为“未知”，请等待1-2分钟后刷新，状态变为“运行”后，再执行后续操作。

----结束

4 链代码开发

链代码也称智能合约，是控制区块链网络中相关方相互交互的业务逻辑。链代码将业务网络交易封装在代码中，最终在一个 Docker 容器内运行。目前华为云区块链服务暂时支持Golang语言编写代码。

链代码即一个Go文件，创建好文件后进行函数开发等操作。链代码开发的更多信息，您可参阅hyperledger-fabric的链代码教程。

操作步骤

步骤1 将shim包导入您的链代码中。

说明

shim包提供了一些 API，以便您的链代码与底层区块链网络交互来访问状态变量、交易上下文、调用方证书和属性，并调用其他链代码和执行其他操作。

示例如下图：

```
package main

import (
    "fmt"
    "crypto/sha256"

    "github.com/hyperledger/fabric/core/chaincode/shim"
    sc "github.com/hyperledger/fabric/protos/peer"
)
```

步骤2 编写main函数。

说明

任何 Go 程序的起点都是 main 函数，因此该函数被用于引导/启动链代码。当对等节点部署其链代码实例时，就会执行 main 函数。

示例如下图：

```
// The main function is only relevant in unit test mode. Only included here for completeness.
func main() {

    // Create a new Smart Contract
    err := shim.Start(new(SmartContract))
    if err != nil {
        fmt.Printf("Error creating new Smart Contract: %s", err)
    }
}
```

步骤3 实现Init方法。

说明

Init方法在链代码首次部署到区块链网络时调用，将由部署自己的链代码实例的每个对等节点执行。此方法可用于任何与初始化、引导或设置相关的任务。

示例如下图：

```
func (s *SmartContract) Init(APIstub shim.ChaincodeStubInterface) sc.Response {
    fmt.Println("Initing chaincode")
    _, args := APIstub.GetFunctionAndParameters()

    if len(args) > 10  {
        return shim.Error("Incorrect number of arguments. Expecting 1")
    }

    return shim.Success(nil)
}
```

步骤4 实现Invoke方法。

说明

只要修改区块链的状态，就会调用 Invoke 方法。简言之，所有创建、更新和删除操作都应封装在 Invoke 方法内。因为此方法将修改区块链的状态，所以区块链 Fabric 代码会自动创建一个交易上下文，以便此方法在其中执行。对此方法的所有调用都会在区块链上记录为交易，这些交易最终被写入区块中。

示例如下图：

```
func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface) sc.Response {

    // Retrieve the requested Smart Contract function and arguments
    function, args := APIstub.GetFunctionAndParameters()

    // Route to the appropriate handler function to interact with the ledger appropriately
    if function == "creditAccountInfo" {
        return s.creditAccountInfo(APIstub, args)
    } else if  function == "authAccount" {
        return s.authAccount(APIstub, args)
    }

    return shim.Error("Invalid Smart Contract function name.")
}
```

----结束

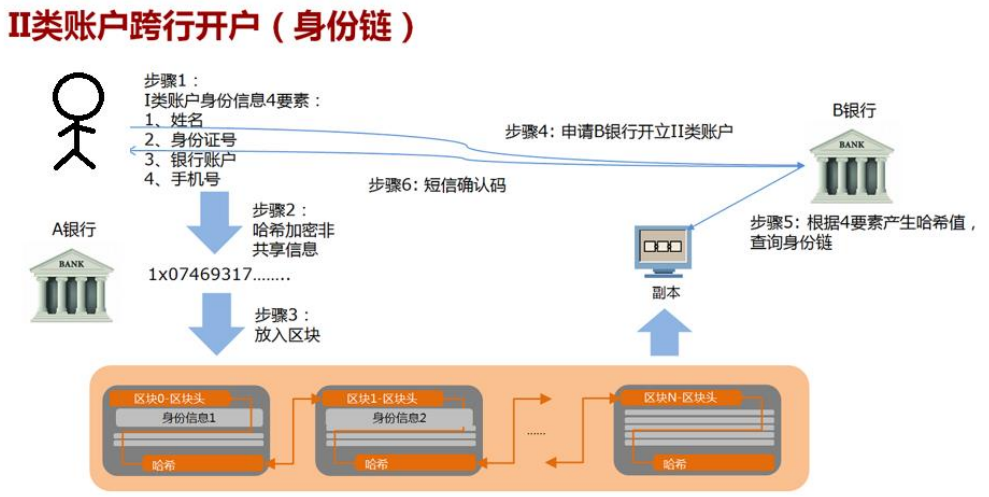
示例链代码

如下为“基于区块链身份共享的银行II类账户跨行开户”示例链代码。

fabbankid.go

- 场景描述

  用户申请A银行的一类账户，此时用户需要提交身份信息四要素（姓名、身份证号、银行账户、手机号）给A银行，A银行将信息导入系统放入区块链中。用户再去申请B银行的二类账户，由于B银行与A银行同处区块链网络，B银行只需要查询身份链即可给用户完成二类账户的开户，无需用户再提供身份信息。流程图如下。



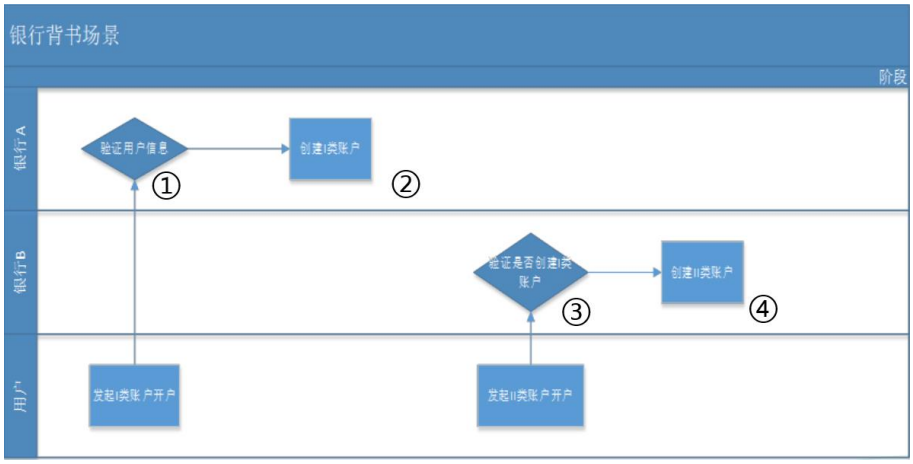
应用与链代码的详细调用过程如下：

银行选择银行登录模式成功登录系统，通过前台界面导入A银行客户数据（即I类账户身份信息4要素：姓名、身份证号、银行账号和手机号），调用后台接口FeedAccountInfo，参数校验成功之后，通过Fabric SDK调用链代码接口invoke并传入函数名creditAccountInfo及客户参数，执行链代码函数creditAccountInfo，哈希加密非共享信息，然后放入区块中。

用户选择客户登录模式成功登录系统，通过前台界面输入参数值，申请B银行开立II类账户，调用后台接口CreateType2Account，参数校验成功之后，通过Fabric SDK调用链代码接口invoke并传入函数名authAccount及账户参数，执行链代码函数authAccount，根据4要素产生哈希值，查询身份链，发送短信确认码（示例中该步省略），完成开户。

- 场景分析

  分析以上场景，我们可以拆分归纳出如下几个任务。



① 验证用户信息，作为银行的初步验证，在链下完成，由银行APP处理。

② 创建I类账户，将用户信息存入在区块链中。

③ 查询是否有I类账户，根据用户信息在区块链中查询是否在其他银行已经有I类账户。

④ 创建II类账户，根据验证结果创建II类账户。

可见银行A、B功能相同，但是部署在不同的组织中。II类账号是否存储在链代码中，主要看是否会作为多方要达成共识的数据。在本示例中，II类账户只存在本银

行内部，因此没有将其存入区块链中。所以上述中②、③可以转变成ChainCode，其他的由APP应用实现。

5 链代码安装及实例化

链代码首先安装在Peer节点上，然后在通道上进行实例化。所有通道成员都需要在将运行此链代码的每个Peer节点上安装链代码，只需在一个Peer节点上进行链代码实例化。如需使用相同的链代码，通道成员必须在链代码安装期间为链代码提供相同的名称和版本。

如下我们以示例链代码fabbankid.go的安装及实例化为例进行介绍。

安装链代码

步骤1 登录链代码管理页面。

1. 登录区块链服务管理控制台。
2. 单击服务列表操作列中的“链代码管理”，如下图。

3. 在链代码管理登录页面输入用户名及密码，登录链代码管理页面，登录成功如下图。

说明

用户名：admin，初始登录密码为您在部署区块链服务时设置的密码。

步骤2 选择 xxx1 组织的 peer-0，点击“安装链代码”，进入链代码安装配置界面。输入配置信息，如下图：

说明

SHA256摘要可在linux环境中执行命令: sha256sum fabbankid.zip生成。

步骤3 单击“确认”完成链代码安装，如下图。

----结束

实例化链代码

步骤1 链代码安装完成后，单击“实例化”，如下图。

步骤2 在实例化配置页面输入相关信息，如下图。

步骤3 单击“确认”，完成实例化。

----结束

说明

xxx2 及 xxx3 组织的 peer-0，需分别安装链代码，安装配置及步骤可参考上面 对xxx1 组织链代码安装的讲解。

6 应用程序开发配置和构建

您可以使用 Go 开发应用程序，并使用 **Hyperledger Fabric SDK Go** 中的可用 API 来调用链代码，以在区块链网络中完成事务处理。

如下为“基于区块链身份共享的银行II类账户跨行开户”示例应用。

服务端：**api-server.rar**

前端镜像：**portal.rar**

本章节以下内容均基于此示例进行讲解，您可以提前下载并按照以下描述进行操作。

服务组件证书配置

目前支持两种证书：管理员证书和用户证书。创建通道、加入通道、更新通道、安装链代码、实例化链代码、升级链代码和删除链代码需要使用管理员证书，交易和查询推荐使用用户证书。

步骤1 应用程序开发人员需要到区块链服务管理下载对应服务的证书，如图：

步骤2 将证书文件压缩包下载到api-server/src/api-server/conf/crypto目录，并解压，完成后目录应类似如下结构：

```
SIA1000117344:/home/paas/code/baas-demos/src/api-server/conf/crypto # ll
total 64
drwxr-xr-x 4 paas users  4096 Apr 16 15:11 4fa097b85bc0d65271f9fe9dce98e6370e344c13.peer-4fa097b85bc0d65271f9fe9dce98e6370e344c13.default.svc.cluster.local
drwxr-xr-x 4 paas users  4096 Apr 16 15:12 62ffe38ae8e41ac33953b7c861309b7f4391c5d6.peer-62ffe38ae8e41ac33953b7c861309b7f4391c5d6.default.svc.cluster.local
drwxr-xr-x 4 paas users  4096 Apr 16 15:08 91201e2dc724320ab0d19e27f4ed5660e002f81d.orderer-91201e2dc724320ab0d19e27f4ed5660e002f81d.default.svc.cluster.local
drwxr-xr-x 4 paas users  4096 Apr 16 15:11 d4b95a85435a089cb70c1ec3c0de9a3f64c28aca.peer-d4b95a85435a089cb70c1ec3c0de9a3f64c28aca.default.svc.cluster.local
```

此例中，编译时我们默认将证书拷贝放置在/opt/gopath/src/github.com/hyperledger/api-server/conf/crypto路径下。

----结束

Fabric SDK 下载与配置

步骤1 单击“更多 > 下载SDK配置”，如下图。

步骤2 配置SDK文件参数，如下图。

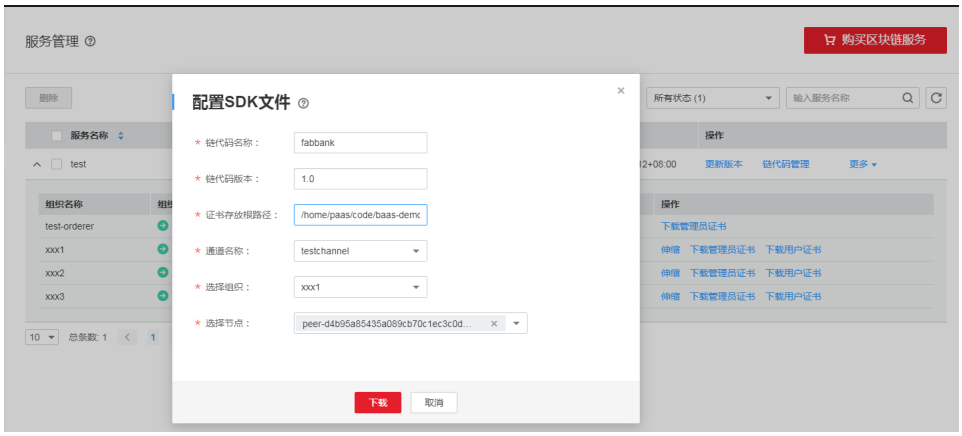


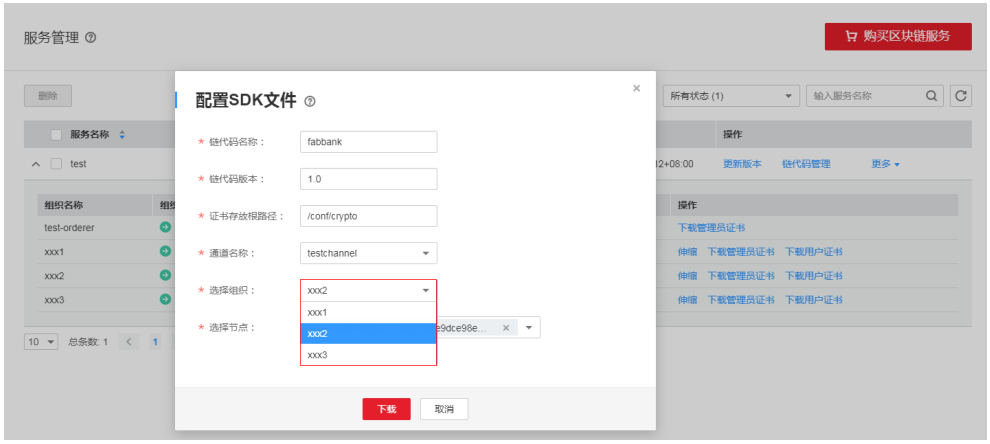
表 6-1 参数表

参数名称	参数值
链代码名称	fabbank
链代码版本	1.0
证书存放根路径	/opt/gopath/src/github.com/hyperledger/api-server/conf/crypto
通道名称	testchannel
选择组织	xxx1
选择节点	节点使用关联出来的默认配置

步骤3 单击下载。下载后的文件命名为test-sdk-config.zip。

步骤4 下载后解压出test-sdk-config.yaml文件，重命名为“组织名.yaml”，例如xxx1.yaml。

步骤5 组织xxx1的Fabric SDK下载完成后，需再用同样的方法，下载其他组织的SDK。下载后解压并重命名为“对应组织名.yaml”。



SDK最终目录结构如下：

```
SIA1000117344:/home/paas/code/tmp/baas-demos/src/api-server/conf # ll
total 32
-rw-r--r-- 1 paas users  307 Apr 17 09:53 app.conf
drwxr-xr-x 6 paas users 4096 Apr 17 09:52 crypto
-rwxr--r-- 1 paas users 7356 Apr 17 10:03 xxx1.yaml
-rwxr--r-- 1 paas users 7356 Apr 17 10:03 xxx2.yaml
-rwxr--r-- 1 paas users 7356 Apr 17 10:03 xxx3.yaml
SIA1000117344:/home/paas/code/tmp/baas-demos/src/api-server/conf #
```

----结束

应用程序开发

用户需要自行开发自己的业务逻辑。开发过程中可通过Fabric的API来调用链代码执行事务处理。也可参考我们提供的示例应用中的bank-account-demo/api-server/src/api-server/controllers/transaction/transaction.go文件进行开发。

应用程序配置

配置bank-account-demo/api-server/src/api-server/conf/app.conf文件中链代码的chaincode_id及通道的channel_id值，需与安装链代码时一致。

说明

示例应用中的app.conf为默认配置，不需要修改。

应用程序构建

进入bank-account-demo/api-server/build 目录执行命令：bash -x build.sh，生成应用程序服务端镜像 api-server.tar.gz。

```
root@HGH1000050619:~/bank-account-demo/api-server/build# bash -x build.sh
```

```
root@HGH1000050619:~/bank-account-demo/api-server/build# ls ../release/
api-server  api-server.tar.gz
```

portal前端请直接下载我们提供的镜像使用。

7 部署应用程序服务端与前端

上传镜像

将构建成功的服务端镜像包api-server.tar.gz与前端镜像包demo-chaincode-portal2.tar.gz传到部署节点上（可为任意能访问外网的机器，本例以华为云上申请的弹性云服务器为例）

步骤1 申请购买华为云弹性云服务器。如何申请可参考**弹性云服务器帮助中心**。

说明

本例以CentOS操作系统的弹性云服务器为例。

步骤2 登录服务器执行如下命令，安装Docker。

yum install docker

说明

- 其他操作系统请自行安装Docker，例如ubuntu可以执行apt-get install docker安装。
- 弹性云服务器操作系统建议选择CentOS或者ubuntu，其他个别操作系统可能默认不支持使用包管理软件进行软件安装。

步骤3 上传镜像，如下图。

- 上传api-server.tar.gz

```
SIA1000117345:/home/paas/code/baas-demos/release # docker load -i api-server.tar.gz
a75caa09eb1f: Loading layer [==================================================>]   105 MB/105 MB
e38b8aef9521: Loading layer [==================================================>] 24.72 MB/24.72 MB
faccc7315fd9: Loading layer [==================================================>] 7.994 MB/7.994 MB
52c175f1a4b1: Loading layer [==================================================>] 146.3 MB/146.3 MB
2a55a2194a6c: Loading layer [==================================================>]   164 MB/164 MB
1654abf914f4: Loading layer [==================================================>] 304.7 MB/304.7 MB
e1e44e9665b9: Loading layer [==================================================>]  2.56 kB/2.56 kB
6bfc813a3812: Loading layer [==================================================>] 5.632 kB/5.632 kB
7970b53a928c: Loading layer [==================================================>] 24.51 MB/24.51 MB
2e6633ef00a2: Loading layer [==================================================>] 4.096 kB/4.096 kB
990617d1ae5a: Loading layer [==================================================>] 30.72 MB/30.72 MB
f92337a56ff2: Loading layer [==================================================>] 224.3 kB/224.3 kB
SIA1000117345:/home/paas/code/baas-demos/release #
```

- 上传demo-chaincode-portal2.tar.gz

```
SIA1000117345:/home/paas/code/baas-demos/release # docker load -i demo-chaincode-portal.tar.gz
c01c63c6823d: Loading layer [==================================================>] 129.3 MB/129.3 MB
d9a5f9b8d5c2: Loading layer [==================================================>] 45.45 MB/45.45 MB
2f9128310b77: Loading layer [==================================================>] 126.8 MB/126.8 MB
63866df00998: Loading layer [==================================================>] 332.2 MB/332.2 MB
bf3841becf9d: Loading layer [==================================================>] 352.8 kB/352.8 kB
0da372da714b: Loading layer [==================================================>] 133.6 kB/133.6 kB
f558212cfa0b: Loading layer [==================================================>] 47.07 MB/47.07 MB
d0cd31697952: Loading layer [==================================================>] 4.183 MB/4.183 MB
88d583cdaab3: Loading layer [==================================================>] 2.048 kB/2.048 kB
d78cd5d6a6a7: Loading layer [==================================================>] 597.6 MB/597.6 MB
SIA1000117345:/home/paas/code/baas-demos/release #
```

----结束

部署应用

服务端部署

步骤1 先执行如下命令启动容器（此例以容器部署方式为例）

docker run -p 8080:8080 -it api-server:latest sh

步骤2 容器启动后，需要先配置hosts域名解析规则，添加对应的order和peer节点的域名及EIP。

1. 从Fabric SDK的yaml文件中，取出order和peer的域名、EIP及端口。

   说明

   - Order及peer的域名可通过搜索关键字“url: grpcs://”查找，“url: grpcs://”后的内容即为域名和端口。

     例如：url: grpcs://orderer-60949f3c8d323e30df78f5bb2cb1f1bb436a8aa6-0.orderer-60949f3c8d323e30df78f5bb2cb1f1bb436a8aa6.default.svc.cluster.local:30810，其中order域名为orderer-60949f3c8d323e30df78f5bb2cb1f1bb436a8aa6-0.orderer-60949f3c8d323e30df78f5bb2cb1f1bb436a8aa6.default.svc.cluster.local，端口为30810。

   - EIP即为订购区块链服务对应容器集群绑定的弹性IP，也可以从yaml配置文件中通过搜索eip关键字查找到对应的EIP。

2. 执行命令配置hosts，如下为参考命令（需要替换命令中的域名及EIP为实际值）

```
echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-0.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-1.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-2.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-3.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-680103b6495409146bc326f7159087d46fab3371-0.peer-680103b6495409146bc326f7159087d46fab3371.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-890da32946030f71a3df2b79685f4f252211bc66-0.peer-890da32946030f71a3df2b79685f4f252211bc66.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e-0.peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e.default.svc.cluster.local" >>/etc/hosts
```

   此例中需配上4个orderer节点的域名及3个peer节点的域名，修改好的文件可参考如下图示：

```
SIA1000117344:/home/paas/code/tmp/baas-demos/release # docker run -p 8080:8080 -it api-server:latest sh
# echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-0.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-1.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
# # echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-2.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-3.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-680103b6495409146bc326f7159087d46fab3371-0.peer-680103b6495409146bc326f7159087d46fab3371.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-890da32946030f71a3df2b79685f4f252211bc66-0.peer-890da32946030f71a3df2b79685f4f252211bc66.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e-0.peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e.default.svc.cluster.local" >>/etc/hosts# # # #
# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      282ff280044c
10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-0.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local
10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-1.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local
10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-2.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local
10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-3.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local
10.154.77.240 peer-680103b6495409146bc326f7159087d46fab3371-0.peer-680103b6495409146bc326f7159087d46fab3371.default.svc.cluster.local
10.154.77.240 peer-890da32946030f71a3df2b79685f4f252211bc66-0.peer-890da32946030f71a3df2b79685f4f252211bc66.default.svc.cluster.local
10.154.77.240 peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e-0.peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e.default.svc.cluster.local
# 
```

步骤3 执行./api-server启动服务。

```
SIA1000117344:/home/paas/code/tmp/baas-demos/release # docker run -p 8080:8080 -it api-server:latest sh
# echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-0.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-1.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
# # echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-2.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-3.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-680103b6495409146bc326f7159087d46fab3371-0.peer-680103b6495409146bc326f7159087d46fab3371.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-890da32946030f71a3df2b79685f4f252211bc66-0.peer-890da32946030f71a3df2b79685f4f252211bc66.default.svc.cluster.local" >>/etc/hosts
echo "10.154.77.240 peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e-0.peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e.default.svc.cluster.local" >>/etc/hosts# # # #
# cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      282ff280044c
10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-0.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local
10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-1.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local
10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-2.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local
10.154.77.240 orderer-3846714c24f22ab46a71a08291948a2833b7e38c-3.orderer-3846714c24f22ab46a71a08291948a2833b7e38c.default.svc.cluster.local
10.154.77.240 peer-680103b6495409146bc326f7159087d46fab3371-0.peer-680103b6495409146bc326f7159087d46fab3371.default.svc.cluster.local
10.154.77.240 peer-890da32946030f71a3df2b79685f4f252211bc66-0.peer-890da32946030f71a3df2b79685f4f252211bc66.default.svc.cluster.local
10.154.77.240 peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e-0.peer-3efc55e46535e814cf90fa985fe1a5a6e2fa1a1e.default.svc.cluster.local
# ls
api-server  conf
# ./api-server
2018/04/17 02:35:17 [I] [asm_amd64.s:2337] http server Running on http://0.0.0.0:8080
```

输出如上截图中的日志，表明服务端启动成功。

----结束

前端部署

步骤1 执行如下命令启动容器。

```
docker run -p 4200:4200 -e API_SERVER_IP=127.0.0.1 -e API_SERVER_PORT=8080 -it demo-chaincode-portal2:latest sh
```

步骤2 容器启动后，进入容器执行./start.sh启动服务。

```
paas@SIA1000117345:~/code/baas-demos> sudo docker run -p 4200:4200 -e API_SERVER_IP=10.180.43.175 -e API_SERVER_PORT=8080 -it demo-chaincode-portal2:latest sh
# ./start.sh

> baasbankdemo@1.0.0 start /opt/bankdemo
> ng serve --proxy-config proxy.conf.js

** NG Live Development Server is listening on 0.0.0.0:4200, open your browser on http://localhost:4200/ **
 10% building modules 3/3 modules 0 active[HPM] Proxy created: /api  ->  http://10.180.43.175:8080/blockchain
[HPM] Proxy rewrite rule created: "^/api" ~> ""                                                                  Date: 2018-04-16T11:24:41.376Z
Hash: f520a3d8776d51979fd6
Time: 25051ms
chunk {inline} inline.bundle.js (inline) 5.79 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 456 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 950 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 651 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 10.4 MB [initial] [rendered]

webpack: Compiled successfully.
```

输出如上截图中的日志，表明服务启动成功。

----结束

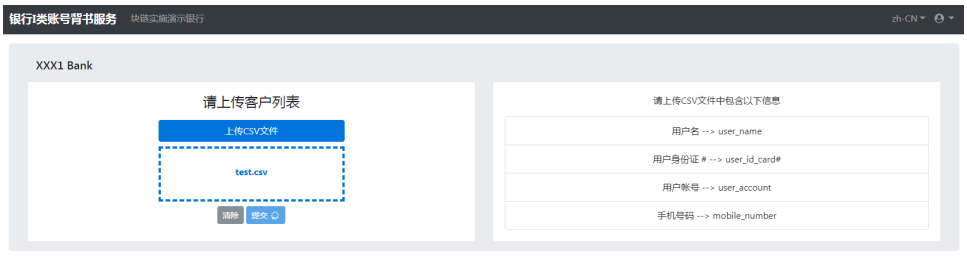
8 访问应用调试业务

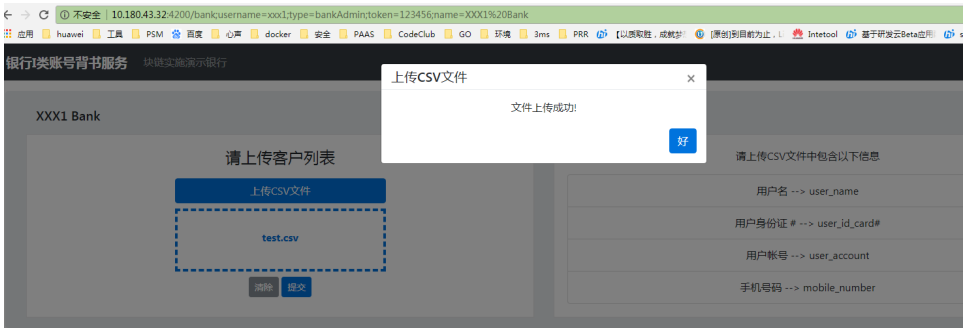
生成客户信息

步骤1 api-server 和 demo-chaincode-portal2 成功运行后，通过 **http://EIP: Port** 来访问 portal。

步骤2 选择银行登录模式，银行： XXX1 Bank，用户名：xxx1，密码：password。

步骤3 登录成功后，上传客户信息文件**test.csv**，在示例系统中生成客户信息。



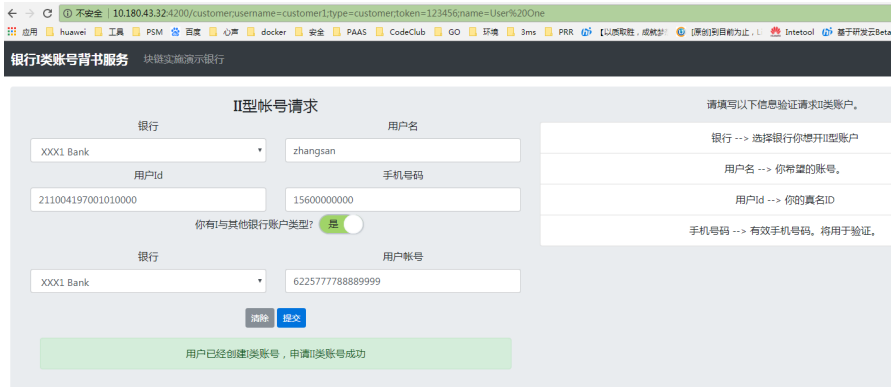


----结束

申请二类账户

步骤1 退出系统银行登录模式，选择客户登录模式，用户名：customer1，密码：password。

步骤2 登录成功后，申请二类账户，输入信息后，点击“提交”，申请成功。



----结束

9 SDK 使用

9.1 国密加密 SDK 使用

9.1.1 概述

国密加密可提高加密强度和加解密性能，很多行业加密准入门槛都要求支持国密算法等。华为云区块链服务提供国密加密算法SDK及加密工具供您使用，方便您进行开发。

下载 SDK

1.8.5版本**SDK包下载**。

1.9.2版本**SDK包下载**。

注意

- 请选择与本地编译环境相一致版本的包：比如本地使用的go编译器为1.8.5（可以使用go version来查看go编译器的版本），则下载1.8.5版本的SDK包；如果本地使用的go编译器为1.9.2，则下载1.9.2版本的SDK包。如果没有一致的版本请联系客服人员获取特定版本的SDK下载包。
- 国密SDK涵盖了普通SDK的所有功能，并在此基础上增加了对国密算法的支持。

将下载的压缩包解压后，得到3个目录，分别为bin、src和pkg，目录的功能如下表：

目录	说明
bin	存放私钥加密工具encrypt_tool，具体用法请参考**9.1.3 私钥加密工具**。
src	存放空的源码文件，用于用户使用静态库。
pkg	存放静态库代码。

9.1.2 SDK 的使用

安装依赖库文件

国密SDK具有依赖库，依赖的库文件位于SDK库的src目录中，具体路径为：src/github.com/hyperledger/fabric-sdk-go/sm/lib/，将里面的两个动态库文件拷贝至/usr/local/include/openssl/目录下（如果没有该目录，请自行创建该目录），然后设置如下环境变量：

```
export LD_LIBRARY_PATH=/usr/local/include/openssl:$LD_LIBRARY_PATH
```

注意

如果用户环境中缺少ltdl库，则需要安装libtool工具，具体安装方法可以参考linux操作系统包管理工具（比如：ubuntu系统可以使用：apt-get install libtool来安装，或者下载源码进行编译安装：**https://www.gnu.org/software/libtool/**）。

安装 SDK

将下载的压缩包解压到用户的$GOPATH目录下。

开发 Client 程序

国密SDK与FabricSDK的使用方式一致，如果用户使用我们提供的私钥加密工具进行了私钥加密，则开发人员需要按照如下方式调用SDK，来设置解密私钥用的用户口令（与用户使用私钥加密工具时使用的口令要一致）：

1. 按如下方式导入API包：
   ```
   import "github.com/hyperledger/fabric-sdk-go/api/apiuserpass"
   ```
2. 按如下方式调用函数：
   ```
   err := apiuserpass.SetUserPassword("123456789")
   if err != nil {
   fmt.Println("fail to set user password")
   }
   ```

   说明

   如上代码中的加密口令仅为示例，您需要根据自己的业务需求设置自己的加密口令。

修改配置文件

用户在运行客户端程序前，需要设置配置文件和证书文件。配置文件中需要关注的地方有两个：一是设置正确的签名算法，另一个是设置正确的证书文件路径。

- 如果要使用ECDSA签名算法，配置文件中需要设置为下图：

```
BCCSP:
  security:
   enabled: true
   default:
    provider: "SW"
   hashAlgorithm: "SHA2"
   softVerify: true
   ephemeral: false
   level: 256
```

- 如果使用国密签名算法，配置文件中需要设置为下图：

```
BCCSP:
  security:
   enabled: true
   default:
    provider: "SM"
   hashAlgorithm: "SM3"
   softVerify: true
   ephemeral: false
   level: 256
```

- 另外，需要关注下配置文件中证书文件的路径，包括MSP路径和tlsca证书路径，如下图所示：

```
organizations:
 Org1:    peer组织名称
   mspid: Org1MSP

   # Needed to load users crypto keys and certs for this org (absolute path or relative to global
   crypto path, DEV mode)
   cryptoPath:  /path/to/peer/admin/msp    peer组织msp证书目录


 ordererorg:   orderer组织名称
   # Membership Service Provider ID for this organization
   mspID: "OrdererOrg"

   # Needed to load users crypto keys and certs for this org (absolute path or relative to global
   crypto path, DEV mode)
   cryptoPath: /path/to/orderer/admin/msp    orderer组织msp证书目录
```

```
orderers:
  orderer0.example.com:
    ...
    tlsCACerts:    orderer组织: tls/ca.crt路径
      # Certificate location absolute path
      path: /path/to/ordererertlsfile

peers:
  peer0.org1.example.com:
    ...
    tlsCACerts:    peer组织: tls/ca.crt路径
      # Certificate location absolute path
      path: /path/to/peertlscafile
```

9.1.3 私钥加密工具

该工具为命令行工具，接收6个参数，分别是：明文的msp私钥文件路径，明文的tls私钥文件路径，msp私钥文件密文路径，tls私钥文件密文路径，私钥类型，用户口令，如下：

flag	含义
-pmsp	明文msp私钥文件的路径
-ptls	明文tls私钥文件的路径
-cmsp	加密后msp私钥文件的路径
-ctls	加密后tls私钥文件的路径
- password	用户口令，8~32个英文字符
- keytype	私钥文件的类型（国密还是ECDSA）： sm：表示国密 sw：表示ECDSA 租户根据是否使用国密服务来进行设置。

假设编译出的私钥加密工具为encrypt_tool，当前要加密的msp私钥文件是plainMSPKey，要加密的tls私钥文件是plainMSPKey，输出的加密后的msp私钥文件是cipherMSPKey，输出的加密后的tls私钥文件是cipherTLSKey，用户口令是123456789：

- 加密国密私钥的用法如下：

```
./encrypt_tool -pmsp ./plainMSPKey -ptls ./plainTLSKey -cmsp ./cipherMSPKey -ctls ./
cipherTLSKey -keytype sm -password 123456789
```

- 加密ECDSA私钥的用法如下：

```
./encrypt_tool -pmsp ./plainMSPKey -ptls ./plainTLSKey -cmsp ./cipherMSPKey -ctls ./
cipherTLSKey -keytype sw -password 123456789
```

9.1.4 附录

fabric-sdk-client/go依赖的第三方包目录：

序号	包名
1	vendor/github.com/Knetic/govaluate
2	vendor/github.com/cloudflare/cfssl
3	vendor/github.com/fsnotify/fsnotify
4	vendor/github.com/golang/groupcache
5	vendor/github.com/golang/mock
6	vendor/github.com/golang/protobuf
7	vendor/github.com/hashicorp/hcl
8	vendor/github.com/magiconair/properties
9	vendor/github.com/miekg/pkcs11
10	vendor/github.com/mitchellh/mapstructure

序号	包名
11	vendor/github.com/pelletier/go-toml
12	vendor/github.com/pkg/errors
13	vendor/github.com/spf13/afero
14	vendor/github.com/spf13/cast
15	vendor/github.com/spf13/jwalterweatherman
16	vendor/github.com/spf13/pflag
17	vendor/github.com/spf13/viper
18	vendor/golang.org/x/crypto
19	vendor/golang.org/x/net
20	vendor/golang.org/x/sync
21	vendor/golang.org/x/sys
22	vendor/golang.org/x/text
23	vendor/google.golang.org/genproto/googleapis
24	vendor/google.golang.org/grpc
25	vendor/gopkg.in/yaml.v2

9.2 同态加密 SDK 使用

华为云区块链服务提供同态加密算法SDK供您使用，方便您进行开发。

9.2.1 概述

同态加密是一类具有特殊自然属性的加密方法，与一般加密算法相比，同态加密除了基本加密外，还能实现密文间的多种计算功能，对于保护信息的安全具有重要意义。利用同态加密技术可以实现无密钥方对密文的计算，密文计算无须经过密钥方，既可以减少通信代价，又可以转移计算任务，可平衡各方的计算代价。利用同态加密技术可以实现让解密方只能获知最后的结果，而无法获得每一个密文的消息，可以提高信息的安全性。

我们提供客户端 SDK库和Chaincode库，该库主要用于交易类的密文运算服务，达到用户交易的隐私保护。

- SDK库：用于在client端提供加法同态功能和生成交易金额的证明信息。
- 同态加密链代码IDChaincode.go：在同态加密的场景下，用户在部署应用前需要下载安装并且实例化此链代码至区块链服务。
- Chaincode库：提供零知识证明功能，用于在密文条件下，校验用户交易的证明，并生成交易后的数据，使背书者无需解密用户交易的数据，达到余额范围的判断。

资源下载

- 1.8.5版本**SDK包下载**
- 1.9.2版本**SDK包下载**
- 同态加密链代码下载 **IDChaincode.go**
- Chaincode库接口文件下载 **api_ahe_cc.tar.gz**

注意

- 需要选择与本地编译环境相一致版本的包。比如本地使用的go编译器为1.8.5（可以使用go version来查看go编译器的版本），则下载1.8.5版本的SDK包；如果本地使用的go编译器为1.9.2，则下载1.9.2版本的SDK包。如果没有一致的版本请联系客服人员获取特定版本的SDK下载包。
- 使用同态加密库需要提前安装好国密SDK。
- api_ahe_cc.tar.gz包仅用于本地编译。

9.2.2 SDK 的使用

安装同态库 SDK

将下载的压缩包解压到用户的GOPATH目录中即可。

开发 app 应用与链代码

请参考下面章节的接口说明进行具体的应用和链代码（智能合约）的开发。

app客户端的典型逻辑过程是：

1. 注册用户
2. 初始化余额
3. 发起交易

说明

- 其中，注册用户时可以调用密钥生成函数为用户生成公私钥。
- 初始化余额时可以调用初始余额准备函数生成具有隐私保护的初始余额信息。
- 交易时可以调用交易准备函数生成具有隐私保护的交易数据。

链码端对应的逻辑过程是：

1. 保存用户公钥与地址的映射关系
2. 验证初始余额的有效性并生成初始交易
3. 验证交易数据的有效性并生成交易结果

说明

- 链码端可以通过调用初始余额校验函数来验证初始余额的有效性。
- 调用交易校验函数来验证交易数据的有效性。
- 具体的开发过程可以参考下述接口说明。

9.2.3 SDK 库接口

提供给用户静态库文件，用于用户开发应用时使用同态加密功能。

库的引用路径为：import "ahe/PSW/api/ahelib"

GenerateKey

- **接口原型**

func GenerateKey(pwd string) (privKeyStr string, pubKeyStr string, err error)

- **功能描述**

生成同态加密的密钥对。

- **输入说明**

参数名	类型	描述	是否必须
pwd	string	用于加密保护生成的同态私钥串，采用AES-128加密。	否

- **输出说明**

参数名	类型	描述
privKeyStr	string	同态加密私钥，已用pwd加密，如果pwd为空字符串，则返回为私钥的明文串。
pubKeyStr	string	同态加密公钥串
err	error	错误

- **注意事项**

保护密钥的复杂度取决于用户实际的应用场景，作为底层库的调用，不去严格限制。

Encypt

- **接口原型**

func Encypt (secretNumStr string, pubKeyStr string) (ciphertext string, err error)

- **功能描述**

同态加密。

- **输入说明**

参数名	类型	描述	是否必须
secretNumStr	string	需要加密的数值，只支持大于等于0的正整数，如果数字有小数，需要扩大倍数传入。	是
pubKeyStr	String	同态加密的公钥	是

- **输出说明**

参数名	类型	描述
ciphertext	string	加密后的数据
err	error	返回错误

- **注意事项**

Secret Number 不能小于0。

Decrypt

- **接口原型**

  func Decrypt(ciphertext string, privKeyStr string, pwd string) (plainText *big.Int, err error)

- **功能描述**

同态解密。

- **输入说明**

参数名	类型	描述	是否必须
ciphertext	string	需要解密的密文	是
privKeyStr	string	同态加密的私钥串（被pwd加密保护	是
pwd	string	用于保护私钥的字符串	否

- **输出说明**

参数名	类型	描述
plainText	*big.int	解密后的数据
err	error	返回错误

- **注意事项**

无。

Add

- **接口原型**

  func Add(cipher1, cipher2 string) (cipher string, err error)

- **功能描述**

同态解密。

- **输入说明**

参数名	类型	描述	是否必须
cipher1	string	被加密的密文1	是
cipher2	string	被加密的密文2	是

- **输出说明**

参数名	类型	描述
cipher	string	相加后的数据
err	error	返回错误

- **注意事项**

无。

InitBalance

- **接口原型**

  func InitBalance(balanceStr string, pubKey string) (string, error)

- **功能描述**

初始化余额。

- **输入说明**

参数名	类型	描述	是否必须
pubKey	string	公钥串	是
balanceStr	string	初始余额，只支持大于等于0的正整数，如果数字有小数，需要扩大倍数传入	是

- **输出说明**

参数名	类型	描述
balanceInfo	string	交易金额信息
err	error	返回错误

- **注意事项**

无。

PrepareTxInfo

- **接口原型**

  func PrepareTxInfo(cipherBalanceA string, transNumStr string, pubKeyA, pubKeyB string, privKeyA string,pwd string) (string, error)

- **功能描述**

  交易准备。

- **输入说明**

参数名	类型	描述	是否必须
cipherBalanceA	string	A的当前余额（密文），取链上当前A的余额	是
transNumStr	string	转账金额（明文）	是
pubKeyA	string	A的公钥串	是
pubKeyB	string	B的公钥串	是
PrivKeyA	string	A的私钥串	是
pwd	string	保护的字符串	否

- **输出说明**

参数名	类型	描述
Txinfo	string	交易准备数据
err	error	返回错误

- **注意事项**

无。

9.2.4 Chaincode 库接口

该静态库集成在BCS服务中。用户在开发链代码时，可以使用我们提供的API接口文件对开发中的链码进行本地编译。

先将API接口文件下载（下载链接参见**资源下载**）并解压到本地的GOPATH目录中，按照后续的链码示例代码来引用同态库。当链码开发完成后，将链码安装到BCS中时，链码会自动链接到BCS中的库代码，实现对链码端同态加密库的调用。

链码中调用同态加密库的引用路径为：import "ahe/PSW/api/ChainCode"

注意

请确保使用该引用路径，否则链码端调用同态加密库会失败。

ValidateInitBalance

- **接口原型**

func ValidateInitBalance(BalanceInfo, PubKey string) (InitBalance string, err error)

- **功能描述**

校验sdk.InitBalance生成的balanceinfo中余额证明的有效性。

- **输入说明**

参数名	类型	描述	是否必须
BalanceInfo	string	初始余额数据	是
Pubkey	string	余额加密的公钥信息	是

- **输出说明**

参数名	类型	描述	是否必须
InitBalance	string	初始余额密文	是
err	error	错误信息	是

- **处理说明**

  验证余额有效后，返回余额的密文。

- **注意事项**

这里用户余额真实性由用户的app逻辑保证，Chaincode端无法验证该用户的真实金额，只能验证该金额是大于0的范围。

ValidateTxInfo

- **接口原型**

ValidateTxInfo(txInfo, cipherBalanceA, cipherBalanceB string)
(newCipherBalanceA,newCipherBalanceB,newCipherTxA,newCipherTxB string,err error)

- **功能描述**

校验PrepareTxInfo生成的Txinfo中交易证明的有效性。

- **输入说明**

参数名	类型	描述	是否必须
txinfo	string	交易证明数据 **说明** 交易信息数据包含有交易密文数据和交易证明数据，该信息来源于sdk.PrepareTxInfo返回的txinfo信息。	是
cipherBalanceA	string	A当前的交易余额（密文）	是
cipherBalanceB	string	B当前的交易余额（密文）	是

- **输出说明**

参数名	类型	描述
newCipherBalanceA	string	交易后待更新的A的余额
newCipherBalanceB	string	交易后待更新的B的余额
newCipherTxA	string	交易金额（A的同态加密公钥加密）
newCipherTxB	string	交易金额（B的同态加密公钥加密）
err	error	错误

- **注意事项**

这里A是转账方，B是收款方。

9.2.5 IDChaincode

用于保存用户的公钥和账户，新生成用户的同态密钥对时，需要将公钥注册到IDchaincode上，便于后续根据账户能查询到收款方的同态公钥。链代码IDChaincode.go的下载请参见资源下载。

注册 Register

账户地址是通过公钥hash计算转16进制字符串得到。

查询 Query

使用账户地址查询公钥。

9.2.6 示例 Appdemo

为了说明同态加密库的具体使用方法，提供一个应用示例代码和对应的链码示例代码。该应用的主要功能是实现用户间相互转账，同时使用同态加密库保护用户的转账交易信息。

该应用的使用包括三个步骤：注册用户（同时会初始化用户余额），用户间转账，查询用户余额。

应用使用命令行的方式进行业务操作，具体过程如下。

注册

```
Usage:
  appdemo register [flags]
Flags:
  -C, --channel string       channel id (default "mychannel")
  -c, --config string        configuration file path (default "./config_test.yaml")
  -I, --idcc string          idendity chaincode name (default "IDChaincode")
  -i, --initbalance string   init initbalance
  -o, --orgid string         organization id (default "org1")
  -p, --protectpwd string    protect pwd
  -T, --txcc string          transaction chaincode name (default "TxChaincode")
  -u, --userid string        user id
./appdemo register -u A -p test -i 100
```

- 为用户生成一对同态公私钥

这里假设有个用户表示userid，用于区分用户，新用户注册的时候为这个用户生成一对公私钥，这里demo为每个用户将密钥对写到了本地的${userid}.data文件。

```
privKeyStr, pubKeyStr, err := pswapi_sdk.GenerateKey(propwd)
check(err)
fmt.Println("key is nil")
userdata.PubKey = pubKeyStr
userdata.PriKey = privKeyStr
```

- 注册公钥

使用fabric SDK的接口向IDChaincode 注册，参数为生成的公钥。

说明

注册公钥是为了使大家的同态公钥共享，这里利用了区块链本身链码特性完成该功能。主要作用是在转账的时候，使用转账者的公钥加密交易金额，保护交易隐私，只有私钥持有者才可以解密交易内容。

```
res, err := sdk_client.Invoke(setup, "Register", [][]byte{[]byte(userdata.PubKey),
[]byte(senderAddr)})
if err != nil {
     fmt.Println("Fail to register user pk ", err.Error())
} else {
     addrByte := res[0].ProposalResponse.GetResponse().Payload
     fmt.Println("Register addr: ", string(addrByte))
  }
```

- 注册初始余额
  a. 使用sdk.InitBalance来初始余额，进行加密生成初始化balanceinfo
  b. 发送balanceinfo到交易Chaincode

```
balanceInfo, err := pswapi_sdk.InitBalance(initbalance, userdata.PubKey) check(err)
setup.ChainCodeID = txchaincode
_, err = sdk_client.Invoke(setup, "init", [][]byte{[]byte(userdata.PubKey), []byte(balanceInfo)})
 if err != nil {
    fmt.Println("Initbalance error for user: ", senderAddr, err.Error())
 } else {
    fmt.Println("init balance successfully: ", senderAddr)
 }
 check(err)
```

交易

```
Usage:
  appdemo transaction [flags]
Flags:
  -b, --AddrB string        B' addr
  -t, --Tx string           Transaction Num
  -C, --channel string      channel id (default "mychannel")
  -c, --config string       configuration file path (default "./config_test.yaml")
  -I, --idcc string         idendity chaincode name (default "IDChaincode")
  -o, --orgid string        organization id (default "org1")
  -p, --protectpwd string   protect pwd
  -T, --txcc string         transaction chaincode name (default "TxChaincode")
  -u, --userid string       user id
./appdemo transaction -u A -p test -i 100 -b -t 10
```

查询

```
Usage:
  appdemo querybalance [flags]
Flags:
  -C, --channel string      channel id (default "mychannel")
  -c, --config string       configuration file path (default "./config_test.yaml")
  -I, --idcc string         idendity chaincode name (default "IDChaincode")
  -o, --orgid string        organization id (default "org1")
  -p, --protectpwd string   protect pwd
  -T, --txcc string         transaction chaincode name (default "TxChaincode")
  -u, --userid string       user id
./appdemo querybalance -p test -u A
```

查询代码：

1. 调用sdk提供的InitBalance接口初始化余额
2. 调用fabirc sdk发送交易

```
get balance
 setup.ChainCodeID = txchaincode
 transRec := sdk_client.TransRecord{}

 fmt.Println("query balance")

 resps, err := sdk_client.Query(setup, "QueryBalance", [][]byte{[]byte(addrA)})
 if err != nil {
  fmt.Println("Fail to query balance :", err.Error())
  return err
 }

 err = json.Unmarshal(resps[0].ProposalResponse.GetResponse().Payload, &transRec)
 if err != nil {
  fmt.Println("unmarshal query result error: ", err.Error())
  return err
 }
```

```
decrypt balance

 curbalance, err := pswapi_sdk.Decrypt(transRec.Balance, userdata.PriKey, propwd)
 if err != nil {
  fmt.Println("sdk Decrypt error: ", err.Error())
  return err
```

```
}

fmt.Println("current balance:" + curbalance.String())
```

9.2.7 示例 Transaction Chaincode

交易链码是用户实现其业务逻辑的链码，这里给出的示例代码配置应用示例代码完成用户间的转账操作。在转账数据的验证过程中使用同态加密库对密文交易数据进行合法性校验，确保没有非法操作。示例中实现了余额初始化，余额查询，转账交易三个功能函数，具体功能实现参考如下。

Init 初始化余额

当用户注册成功后，需要初始化该用户的余额，默认为0。

- **输入说明**

参数名	类型	描述	是否必须
PubKey	string	公钥	是
Balance	string	初始金额	是

- **处理**

  调用ValidateInitBalance，校验balance范围有效性，完成余额背书。

```
PubKey := string(args[0])
BalanceInfo := string(args[1])
//PubKey := string(args[2])

hashPubkey, err := t.calcAddr(PubKey)
logger.Debug("encrypt initial balance")
//validate balance
cipherBalance,err := pswapi_cc.ValidateInitBalance(BalanceInfo,PubKey)
 if err != nil {
    logger.Error("fail to Validate InitBalance ")
    return shim.Error("fail toValidate InitBalance")
 }
```

- **输出说明**

参数名	类型	描述
newCipherBalance	string	余额密文

QueryBalance 查询余额

查询余额功能比较简单，根据key来查询value，即根据账户地址来查询当前余额，具体可见transaction_demo的queryBalance接口。

queryBalance接口参数如下：

-	参数名	类型	描述
输入	addr	string	账户地址
输出	cipherbalance	string	账户余额密文

-	参数名	类型	描述
输出	err	error	错误信息

Transfer 转账

- **输入说明**

参数名	类型	描述	是否必须
AddrA	string	A-转账者地址	是
AddrB	string	B-收账者地址	是
txinfo	String	交易信息PrepareTxInfo	是

- **处理说明**

a. 根据账户地址获取账本中A,B的当前余额cipherBalanceAKeyABlock, cipherBalanceBKeyBBlock。

b. 校验证明有效性。

```
newCipherBalanceA, newCipherBalanceB, newCipherTxA, newCipherTxB, err :=
pswapi_cc.ValidateTxInfo(txInfo, cipherBalanceAKeyABlock, cipherBalanceBKeyBBlock)
 if err != nil {
    logger.Error("fail to validate trans information")
    return shim.Error("fail to validate trans information")
 }
```

c. 输出更新后的余额（密文）

业务的账本内容需要用户定制，将上面加密的金额合入到用户的账本中保存，demo中定义了一个存储结构，保存完后通过json序列化为一个交易记录对象进行保存。

```
type TransRecord struct {
    FromAddr string
    ToAddr   string
    TXType   string
    Balance  string
    TX       string
    remark   string
 }
```

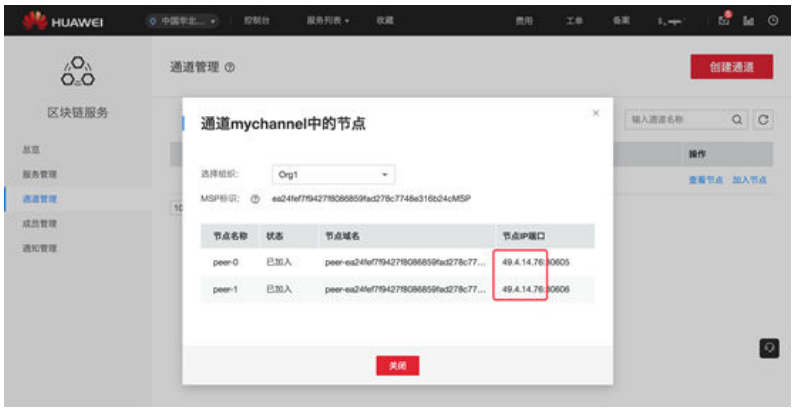
9.3 Fabric-sdk-java 使用

本章节指导说明成功运行一个SDK测试用例，支持app通过sdk直连华为云的BCS实例。github上提供了使用说明，该指导文档以官方说明为基础做简单说明，不同版本使用方式可能有细微差别，仅供参考。

github地址：**https://github.com/hyperledger/fabric-sdk-java**

使用步骤

步骤1 在华为云上已经订购了BCS，并且保证BCS实例所属集群的公网IP能够ping通，IP可通过单击"通道管理 > 查看节点"来获取，如下图。如果不能ping通，请在安全组里面**添加ICMP规则**。



步骤2 下载fabric-SDK-java

```
git clone git@github.com:hyperledger/fabric-sdk-java.git
```

步骤3 配置mvn。

步骤4 修改配置文件。

说明

官方示例仅提供单元测试，所有相关配置均写死在代码中，正式开发时可以从network-config.yaml配置文件读取。此处仅说明如何修改以成功运行测试用例。

修改文件：src\test\java\org\hyperledger\fabric\sdk\testutils\TestConfig.java,具体修改内容可参考**TestConfig.java**

- 修改100行左右peer、orderer的IP、port、mspid等信息即可运行。
- 当前华为云只支持grcps方式，ip和port可以通过下载SDK配置文件来确定，ip为bcs实例所属的集群公网ip，而peer的port一般从30605开始。具体可参考下图以及相关字段说明。

```
//////
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg1.mspid", "ea24fef7f9427f8086859fad278c7748e316b24cMSP");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg1.domname", "org1.example.com");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg1.ca_location", "http://" + LOCALHOST + ":30605");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg1.caName", "ca0");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg1.peer_locations", "peer0.org1.example.com@grpcs://" +
        LOCALHOST + ":30605, peer1.org1.example.com@grpcs://" + LOCALHOST + ":30606");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg1.orderer_locations", "orderer.example.com@grpcs://" +
        LOCALHOST + ":30805");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg1.eventhub_locations", "peer0.org1.example.com@grpcs://" +
        LOCALHOST + ":30705,peer1.org1.example.com@grpcs://" + LOCALHOST + ":30706");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg2.mspid", "Org2MSP");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg2.domname", "org2.example.com");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg2.ca_location", "http://" + LOCALHOST + ":8054");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg2.peer_locations", "peer0.org2.example.com@grpcs://" +
        LOCALHOST + ":30610,peer1.org2.example.com@grpcs://" + LOCALHOST + ":30611");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg2.orderer_locations", "orderer.example.com@grpcs://" +
        LOCALHOST + ":30805");
defaultProperty(INTEGRATIONTESTS_ORG + "peerOrg2.eventhub_locations", "peer0.org2.example.com@grpcs://" +
        LOCALHOST + ":30710, peer1.org2.example.com@grpcs://" + LOCALHOST + ":30711");
```

说明

  - peerOrg1.mspid：ea24fef7f9427f8086859fad278c7748e316b24cMSP（根据sdk配置内容修改，对应sdk的yaml文件的mspid）；
  - peerOrg1.peer_locations："peer0.org1.example.com@grpcs://" + LOCALHOST + ":30605, peer1.org1.example.com@grpcs://" + LOCALHOST + ":30606" (注意：LOCALHOST需要改为bcs所属集群公网ip，另外必须使用grpcs，具体端口号参考sdk配置yaml文件的url) 如：private static final String LOCALHOST = "49.4.14.76";
  - peerOrg1.orderer_locations："orderer.example.com@grpcs://" + LOCALHOST + ":30805"，这里orderer端口号默认是30805，具体以yaml文件的orderers的url里的端口为准，url一般为 grpcs://orderer-xxx-0.orderer-xxx.default.svc.cluster.local:30805;
  - peerOrg1.eventhub_locations："peer0.org1.example.com@grpcs://" + LOCALHOST + ":30705,peer1.org1.example.com@grpcs://" + LOCALHOST + ":30706"，这里的evenHub端口一般也是从30705开始，具体端口号以yaml文件的eventUrl为准；

- 关于TLS配置在函数 private Properties getEndPointProperties(final String type, final String name) {} 把server.crt，修改成ca.crt。

其他相关配置基本在该文件及后面的测试用例文件，可根据需要自行修改。

步骤5 证书拷贝。

目录：src\test\fixture\sdkintegration\e2e-2Orgs\channel 删除原始证书，拷贝自己的证书。

步骤6 把peer和orderer的根证书安装到jdk，否则会报以下错误。其中peer和orderer的根证书在tls/ca.crt,如果使用intellij运行，那么请参考**intellij安装证书**把tls/ca.crt安装到对应的jdk目录下面。如果使用命令行方式运行，请参考**jdk安装证书**。

```
unable to find valid certification path to requested target
```

步骤7 测试用例。

官方提供了完成的测试用例 **End2endIT.java**

该用例包含了create channel、join channel、insta chaincode、instantitate chaincode、invoke、query等所有相关操作，该配置是按照fabric官方环境搭建指导文档配置。运行在我们的环境会有一些小问题，如果仅需要invoke和query功能，可参考下面自己实现的用例。

demo代码：**SendTx.java**，把SendTx.java文件放到src\test\java\org\hyperledger\fabric\sdkintegration下面。该测试用例测试的是example02的chaincode，若测试该chaincode修改最前面的一些参数信息即可运行。若测试其他chaincode，修改invoke()、query()函数的fcn及args即可。 完成修改后run setup()函数即可进行测试。

说明

reconstructChannel()函数为初始化SDK的client及channel。

----结束

常见问题

1. 出现如下回显信息表示jdk缺少BCS实例节点的根证书，需要把tls/ca.crt安装到jdk里。

```
unable to find valid certification path to requested target
```

2. 出现如下回显信息表示sdk缺少证书，需要在创建相应的目录，并把msp、tls的证书拷贝到所创建的目录。

```
java.lang.RuntimeException: Missing cert file for: peer-
ea24fef7f9427f8086859fad278c7748e316b24c-0.peer-
ea24fef7f9427f8086859fad278c7748e316b24c.default.svc.cluster.local. Could not find at
location: fabric-sdk-java/src/test/fixture/sdkintegration/e2e-2Orgs/v1.0/crypto-config/
peerOrganizations/peer-ea24fef7f9427f8086859fad278c7748e316b24c.default.svc.cluster.local/
peers/peer-ea24fef7f9427f8086859fad278c7748e316b24c-0.peer-
ea24fef7f9427f8086859fad278c7748e316b24c.default.svc.cluster.local/tls/ca.crt
```

3. 出现如下回显信息一般是因为TestConfig.java的配置不对，没有把grpc设置为grpcs，或者ip、port与sdk的yaml文件的ip、port不一致，解决方式请参考上文描述。

```
First received frame was not SETTINGS. Hex dump for first 5 bytes
```