**前期热身报告**

**准备环境**

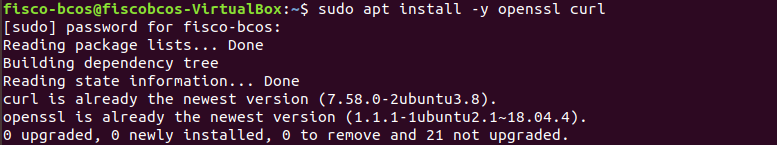
* **安装依赖**

build\_chain.sh脚本依赖于openssl, curl，使用下面的指令安装。若为CentOS，将下面命令中的apt替换为yum执行即可。macOS执行brew install openssl curl即可。

FISCO BCOS所需的必要安装和配置

先在Vbox上复制安装了老师提供的虚拟机，配置好网络，一个是NAT,一个是host-only，之前在云计算课程中配置过的，所以直接用了

sudo apt install -y openssl curl



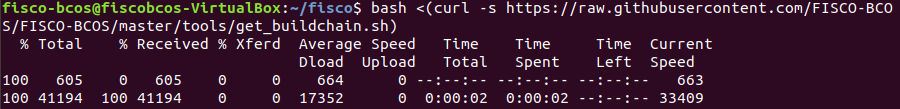
* 创建操作目录

cd ~ && mkdir -p fisco && cd fisco



* 下载build\_chain.sh脚本

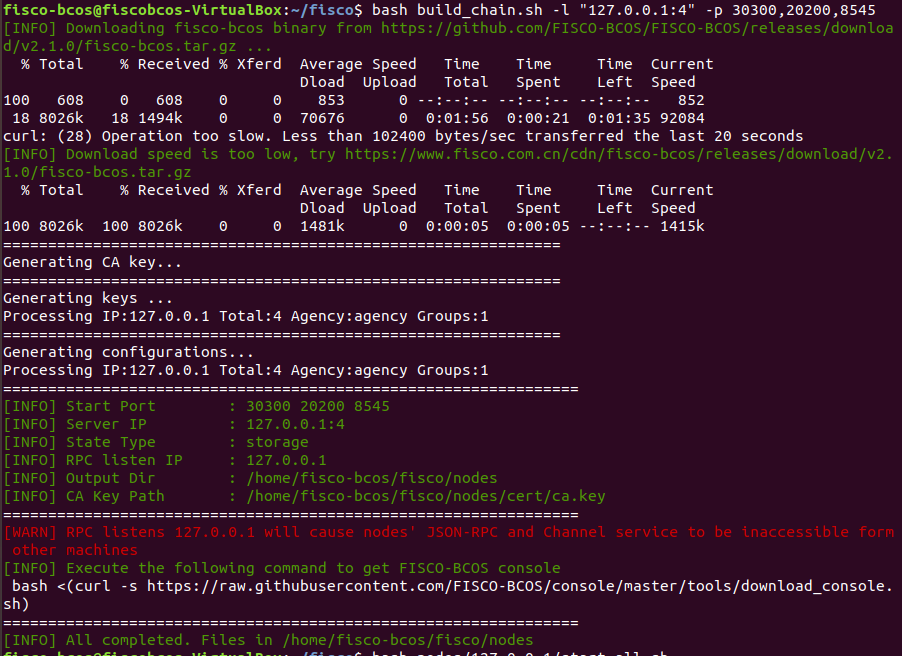
bash <(curl -s https://raw.githubusercontent.com/FISCO-BCOS/FISCO-BCOS/master/tools/get\_buildchain.sh)



**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。

bash build\_chain.sh -l "127.0.0.1:4" -p 30300,20200,8545



**注解**

* 其中-p选项指定起始端口，分别是p2p\_port,channel\_port,jsonrpc\_port，出于安全考虑jsonrpc/channel默认监听127.0.0.1，**需要外网访问请添加-i参数**。

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。

Checking fisco-bcos binary...

Binary check passed.

==============================================================

Generating CA key...

==============================================================

Generating keys ...

Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1

==============================================================

Generating configurations...

Processing IP:127.0.0.1 Total:4 Agency:agency Groups:1

================================================================

[INFO] Execute the following command to get FISCO-BCOS console

bash <(curl -s https://raw.githubusercontent.com/FISCO-BCOS/console/master/tools/download\_console.sh)

================================================================

[INFO] FISCO-BCOS Path : bin/fisco-bcos

[INFO] Start Port : 30300 20200 8545

[INFO] Server IP : 127.0.0.1:4

[INFO] State Type : storage

[INFO] RPC listen IP : 127.0.0.1

[INFO] Output Dir : /home/ubuntu16/fisco/nodes

[INFO] CA Key Path : /home/ubuntu16/fisco/nodes/cert/ca.key

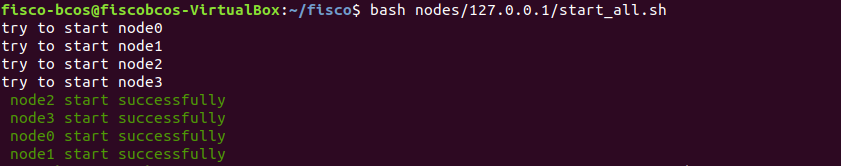
==============================================================

[INFO] All completed. Files in /home/ubuntu16/fisco/nodes

**启动FISCO BCOS链**

* 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

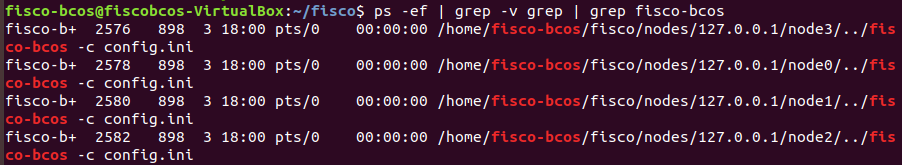
node0 start successfully

node3 start successfully

**检查进程**

* 检查进程是否启动

ps -ef | grep -v grep | grep fisco-bcos



正常情况会有类似下面的输出；如果进程数不为4，则进程没有启动（一般是端口被占用导致的）

fisco 5453 1 1 17:11 pts/0 00:00:02 /home/fisco/fisco/nodes/127.0.0.1/node0/../fisco-bcos -c config.ini

fisco 5459 1 1 17:11 pts/0 00:00:02 /home/fisco/fisco/nodes/127.0.0.1/node1/../fisco-bcos -c config.ini

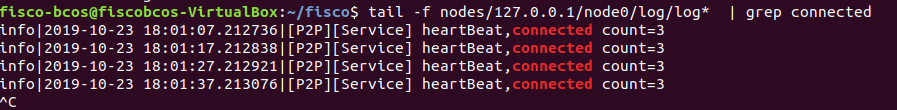
fisco 5464 1 1 17:11 pts/0 00:00:02 /home/fisco/fisco/nodes/127.0.0.1/node2/../fisco-bcos -c config.ini

fisco 5476 1 1 17:11 pts/0 00:00:02 /home/fisco/fisco/nodes/127.0.0.1/node3/../fisco-bcos -c config.ini

**检查日志输出**

* 如下，查看节点node0链接的节点数

tail -f nodes/127.0.0.1/node0/log/log\* | grep connected



正常情况会不停地输出链接信息，从输出可以看出node0与另外3个节点有链接。

info|2019-01-21 17:30:58.316769| [P2P][Service] heartBeat connected count,size=3

info|2019-01-21 17:31:08.316922| [P2P][Service] heartBeat connected count,size=3

info|2019-01-21 17:31:18.317105| [P2P][Service] heartBeat connected count,size=3

* 执行下面指令，检查是否在共识

tail -f nodes/127.0.0.1/node0/log/log\* | grep +++



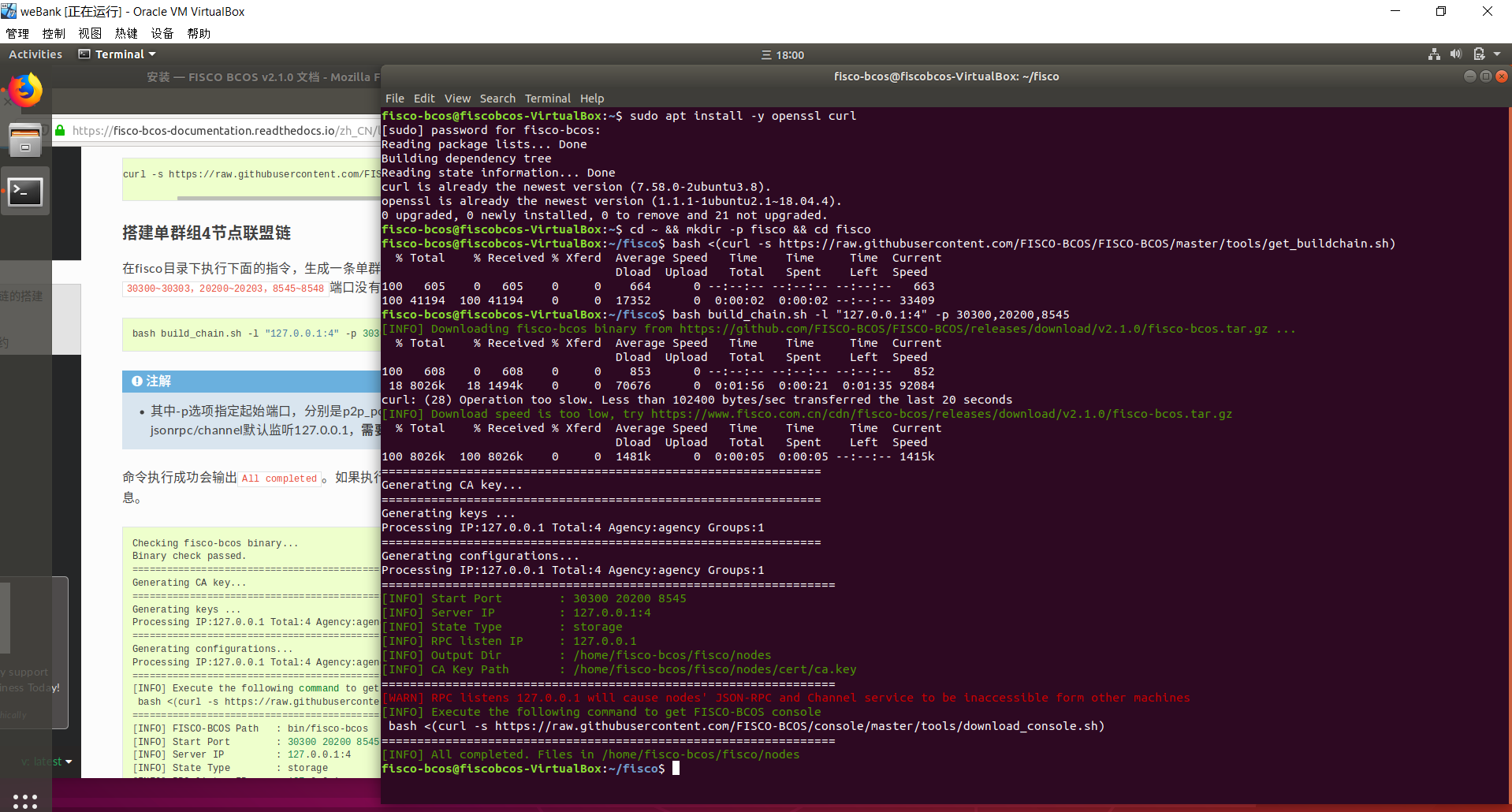
正常情况会不停输出++++Generating seal，表示共识正常。

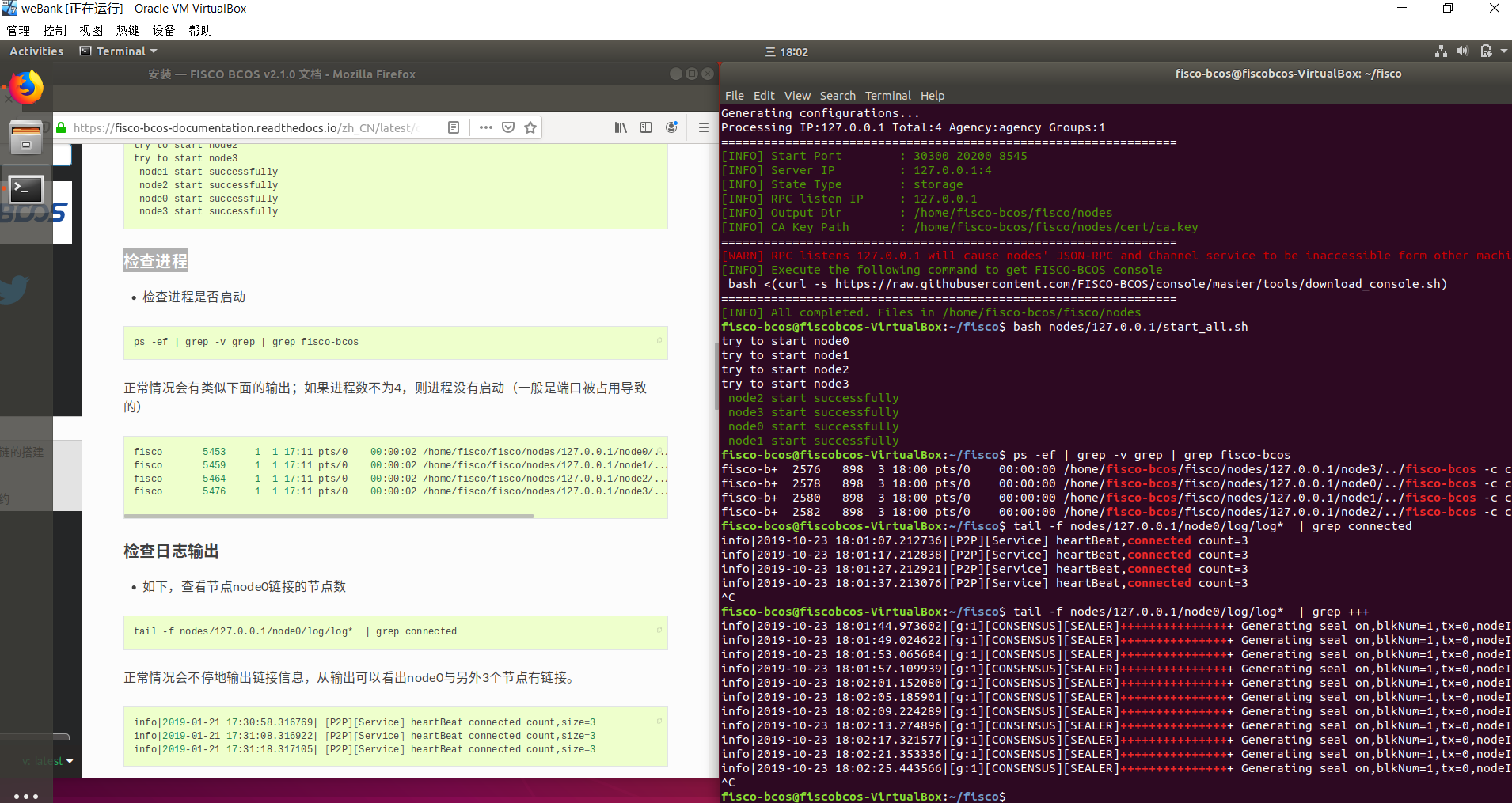
info|2019-01-21 17:23:32.576197| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++Generating seal on,blkNum=1,tx=0,myIdx=2,hash=13dcd2da...

info|2019-01-21 17:23:36.592280| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++Generating seal on,blkNum=1,tx=0,myIdx=2,hash=31d21ab7...

info|2019-01-21 17:23:40.612241| [g:1][p:264][CONSENSUS][SEALER]++++++++++++++++Generating seal on,

上述过程：





## 配置及使用控制台

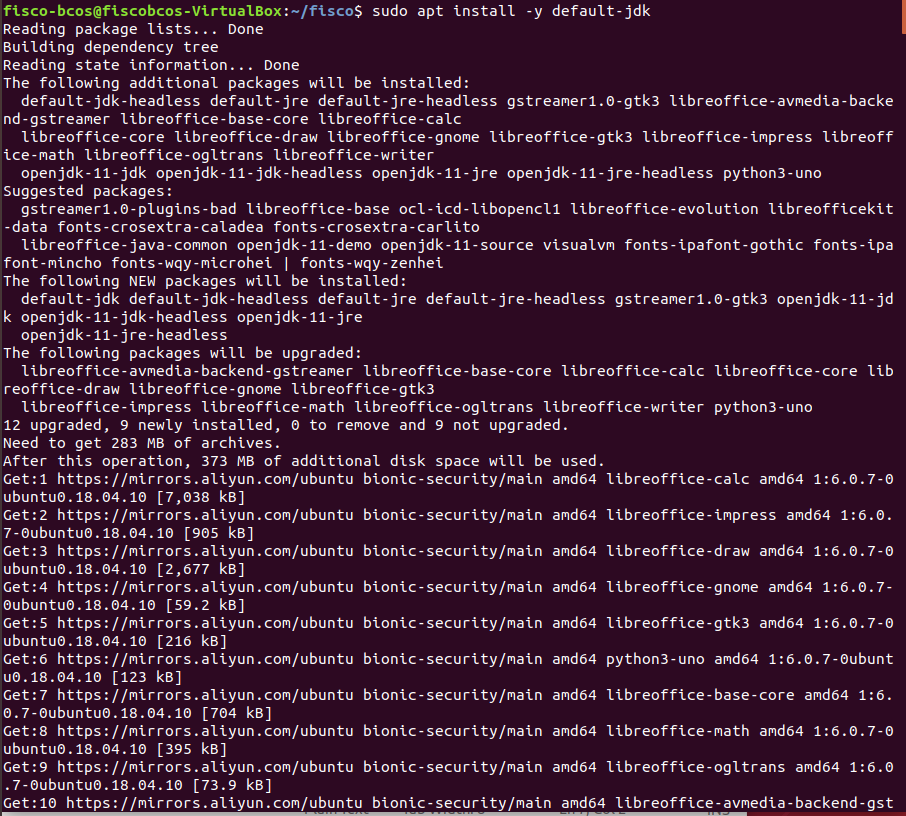
在控制台通过Web3SDK链接FISCO BCOS节点，实现**查询区块链状态、部署调用合约**等功能，能够快速获取到所需要的信息。控制台指令详细介绍[参考这里](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/console.html)。

### 准备依赖

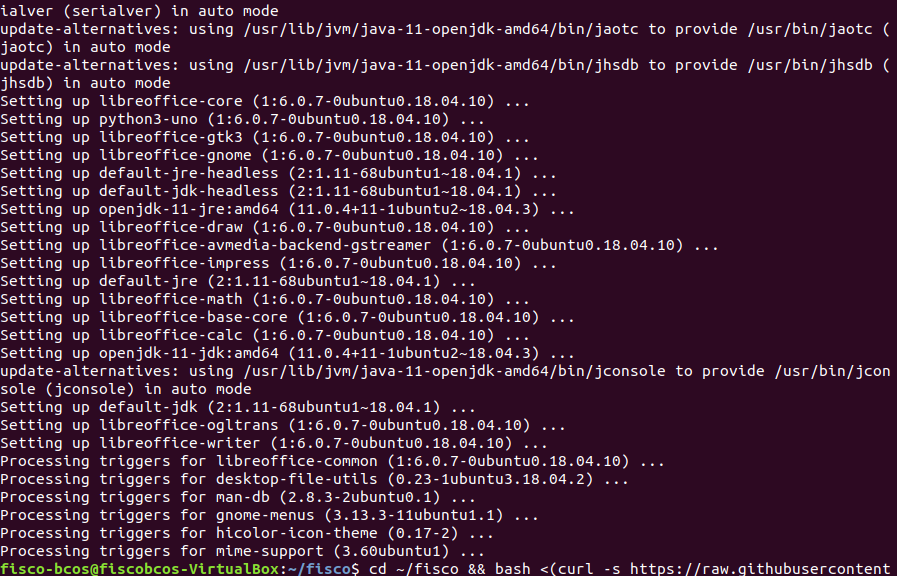
* 安装openjdk

Ubuntu使用下面命令安装Java，CentOS请手动安装，macOS执行brew cask install java安装。

sudo apt install -y default-jdk

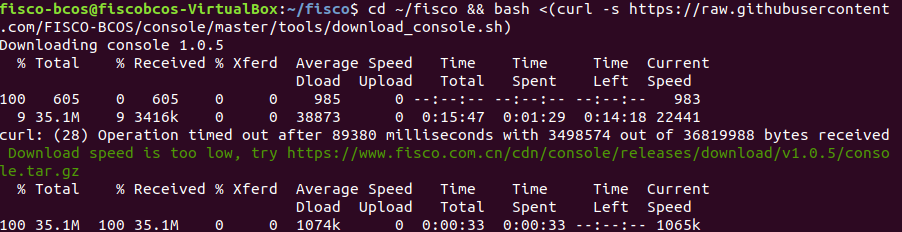


……



* 获取控制台并回到fisco目录

cd ~/fisco && bash <(curl -s https://raw.githubusercontent.com/FISCO-BCOS/console/master/tools/download\_console.sh)



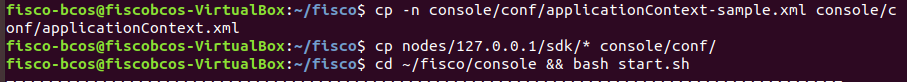
* 拷贝控制台配置文件

若节点未采用默认端口，请将文件中的20200替换成节点对应的channle端口。

cp -n console/conf/applicationContext-sample.xml console/conf/applicationContext.xml

* 配置控制台证书

cp nodes/127.0.0.1/sdk/\* console/conf/



**重要**

* 如果控制台配置正确，但是在CentOS系统上启动控制台出现如下错误：

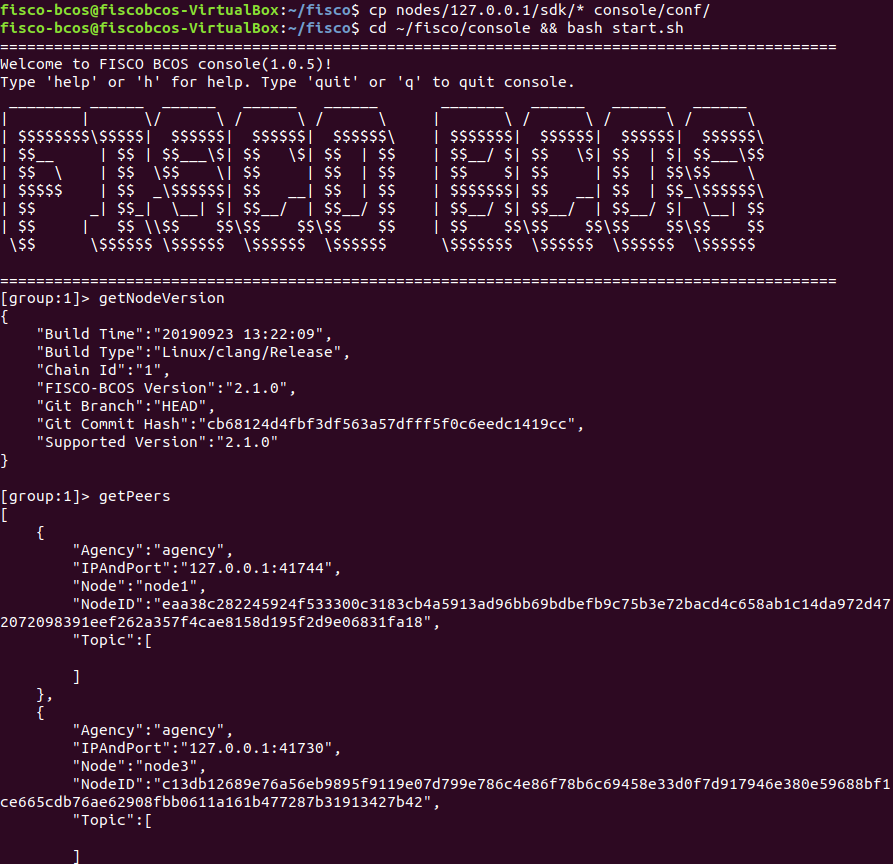
Failed to connect to the node. Please check the node status and the console configuration.

是因为使用了CentOS系统自带的JDK版本(会导致控制台与区块链节点认证失败)，请从 [OpenJDK官网](https://jdk.java.net/java-se-ri/8) 或 [Oracle官网](https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) 下载并安装Java 8或以上版本(具体安装步骤 [参考附录](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/console.html#java) )，安装完毕后再启动控制台。

### 启动控制台

* 启动

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



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

各字段解释：

getBlockNumber Query the number of most recent block.

getNodeVersion Query the current node version.

getPeers Query peers currently connected to the client.

|  |  |  |
| --- | --- | --- |
|  | | |
|  | | |
|  | | [group:1]> getNodeVersion | | |
|  | | { | | |
|  | | "Build Time":"20190923 13:22:09",节点创建时间 | | |
|  | | "Build Type":"Linux/clang/Release",节点类型 | | |
|  | | "Chain Id":"1",节点号 | | |
|  | | "FISCO-BCOS Version":"2.1.0",FISCO版本号 | | |
|  | | "Git Branch":"HEAD", | | |
|  | | "Git Commit Hash":"cb68124d4fbf3df563a57dfff5f0c6eedc1419cc", | | |
|  | | "Supported Version":"2.1.0" | | |
|  | | } | | |
|  | |  | | |
|  | | [group:1]> getPeers 返回已经连接的p2p节点信息 | | |
|  | | [ | | |
|  | | { | | |
|  | | "Agency":"agency",代理 | | |
|  | | "IPAndPort":"127.0.0.1:41744",节点连接的IP和端口 | | |
|  | | "Node":"node1",节点 | | |
|  | | 节点ID "NodeID":"eaa38c282245924f533300c3183cb4a5913ad96bb69bdbefb9c75b3e72bacd4c658ab1c14da972d472072098391eef262a357f4cae8158d195f2d9e06831fa18", | | |
|  | | "Topic":[ | | |
|  | | 节点关注的topic信息 | | |
|  | | ] | | |
|  | | }, | | |
|  | | { | | |
|  | | "Agency":"agency", | | |
|  | | "IPAndPort":"127.0.0.1:41730", | | |
|  | | "Node":"node3", | | |
|  | | "NodeID":"c13db12689e76a56eb9895f9119e07d799e786c4e86f78b6c69458e33d0f7d917946e380e59688bf1ce665cdb76ae62908fbb0611a161b477287b31913427b42", | | |
|  | | "Topic":[ | | |
|  | |  | | |
|  | | ] | | |
|  | | }, | | |
|  | | { | | |
|  | | "Agency":"agency", | | |
|  | | "IPAndPort":"127.0.0.1:41738", | | |
|  | | "Node":"node2", | | |
|  | | "NodeID":"5d3e3c3813f67b1d790530a57b78ea7aebc9205d173152d9e897c90f0d993bd86969fe16b6e6a8d25463c0c39dfefa90a060537850cde8cfe8007c7ab749b51f", | | |
|  | | "Topic":[ | | |
|  | |  | | |
|  | | ] | | |
|  | | } | | |
|  | | ] | | |
|  | |  | | |
|  | | [group:1]> deploy HelloWorld | | |
|  | | contract address: 0xe94ea5f7604e0612ff94f0cb88f6d2208a84f718 | | |
|  | |  | | |
|  | | [group:1]> getBlockNumber区块高度 | | |
|  | | 1 | | |
|  | |  | | |
|  | | [group:1]> call HelloWorld 0xb3c223fc0bf6646959f254ac4e4a7e355b50a344 get | | |
|  | | The contract address is incorrect. | | |
|  | |  | | |
|  | | [group:1]> call HelloWorld 0xe94ea5f7604e0612ff94f0cb88f6d2208a84f718 get | | |
|  | | Hello, World! | | |
|  | |  | | |
|  | | [group:1]> getBlockNumber | | |
|  | | 1 | | |
|  | |  | | |
|  | | [group:1]> call HelloWorld 0xe94ea5f7604e0612ff94f0cb88f6d2208a84f718 set "Hello,FISCO BCOS" | | |
|  | | transaction hash: 0x0a099fe4cb860a9d5155cae8e82bb63b4a9ae748addc1f55537819f3df98f402 | | |
|  | |  | | |
|  | | [group:1]> getBlockNumber | | |
|  | | 2 | | |
|  | |  | | |
|  | | [group:1]> call HelloWorld 0xe94ea5f7604e0612ff94f0cb88f6d2208a84f718 get | | |
|  | | Hello,FISCO BCOS | | |
|  | |  | | |
|  | | [group:1]> quit退出 | | |
|  | | |  | |

### 使用控制台获取信息

*# 获取客户端版本*

[group:1]> getNodeVersion

{

"Build Time":"20190121 06:21:05",

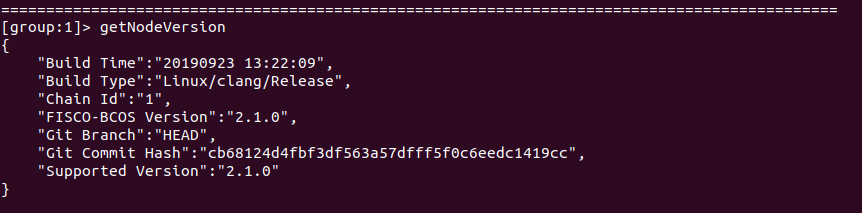
"Build Type":"Linux/clang/Debug",

"FISCO-BCOS Version":"2.0.0",

"Git Branch":"master",

"Git Commit Hash":"c213e033328631b1b8c2ee936059d7126fd98d1a"

}



*# 获取节点链接信息*

[group:1]> getPeers

[

{

"IPAndPort":"127.0.0.1:49948",

"NodeID":"b5872eff0569903d71330ab7bc85c5a8be03e80b70746ec33cafe27cc4f6f8a71f8c84fd8af9d7912cb5ba068901fe4131ef69b74cc773cdfb318ab11968e41f",

"Topic":[]

},

{

"IPAndPort":"127.0.0.1:49940",

"NodeID":"912126291183b673c537153cf19bf5512d5355d8edea7864496c257630d01103d89ae26d17740daebdd20cbc645c9a96d23c9fd4c31d16bccf1037313f74bb1d",

"Topic":[]

},

{

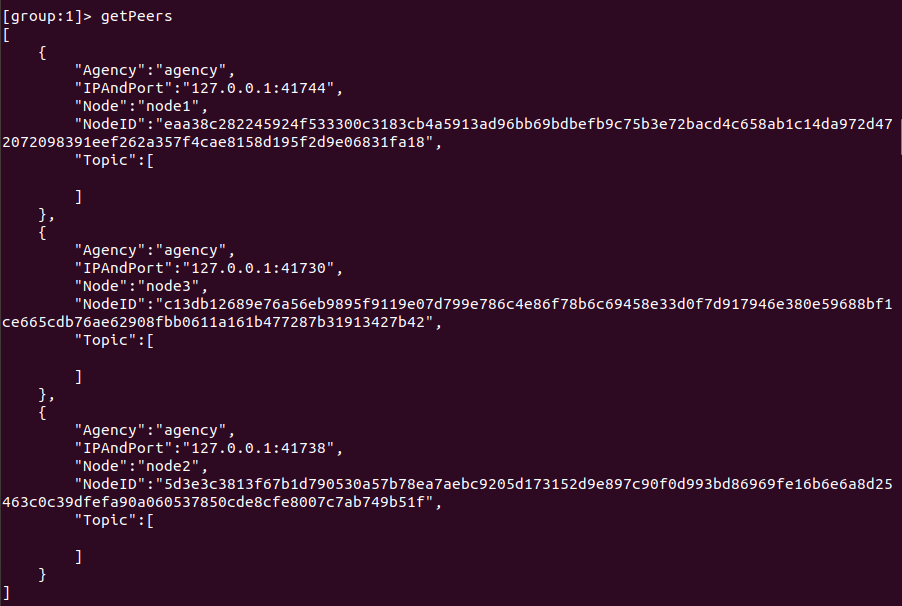
"IPAndPort":"127.0.0.1:49932",

"NodeID":"db75ab16ed7afa966447c403ca2587853237b0d9f942ba6fa551dc67ed6822d88da01a1e4da9b51aedafb8c64e9d208d9d3e271f8421f4813dcbc96a07d6a603",

"Topic":[]

}

]



## 部署及调用HelloWorld合约

### HelloWorld合约

HelloWorld合约提供两个接口，分别是get()和set()，用于获取/设置合约变量name。合约内容如下:

pragma solidity ^0.4.24;

contract HelloWorld {

string name;

function HelloWorld() {

name = "Hello, World!";

}

function get()constant returns(string) {

return name;

}

function set(string n) {

name = n;

}

}

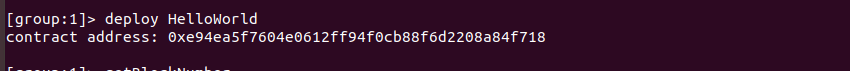
### 部署HelloWorld合约

为了方便用户快速体验，HelloWorld合约已经内置于控制台中，位于控制台目录下solidity/contracts/HelloWorld.sol，参考下面命令部署即可。

*# 在控制台输入以下指令 部署成功则返回合约地址*

[group:1]> deploy HelloWorld

contract address:0xb3c223fc0bf6646959f254ac4e4a7e355b50a344



### 调用HelloWorld合约

*# 查看当前块高*

[group:1]> getBlockNumber

1

*# 调用get接口获取name变量 此处的合约地址是deploy指令返回的地址*

[group:1]> call HelloWorld 0xb3c223fc0bf6646959f254ac4e4a7e355b50a344 get

Hello, World!

*# 查看当前块高，块高不变，因为get接口不更改账本状态*

[group:1]> getBlockNumber

1

*# 调用set设置name*

[group:1]> call HelloWorld 0xb3c223fc0bf6646959f254ac4e4a7e355b50a344 set "Hello, FISCO BCOS"

0x21dca087cb3e44f44f9b882071ec6ecfcb500361cad36a52d39900ea359d0895

*# 再次查看当前块高，块高增加表示已出块，账本状态已更改*

[group:1]> getBlockNumber

2

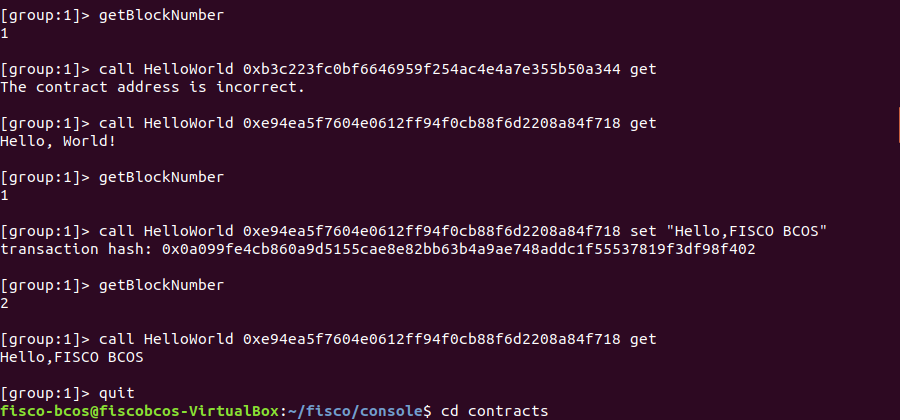
*# 调用get接口获取name变量，检查设置是否生效*

[group:1]> call HelloWorld 0xb3c223fc0bf6646959f254ac4e4a7e355b50a344 get

Hello, FISCO BCOS

*# 退出控制台*

[group:1]> quit



**注：**

1. 部署合约还可以通过deployByCNS命令，可以指定部署的合约版本号，使用方式[参考这里](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/console.html#deploybycns)。
2. 调用合约通过callByCNS命令，使用方式[参考这里](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/console.html#callbycns)。

接下来是构建第一个区块链应用：

# 构建第一个区块链应用

本章将会介绍一个基于FISCO BCOS区块链的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](https://solidity.readthedocs.io/en/latest/)。

**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。

**重要**

请参考 [安装文档](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/installation.html) 完成FISCO BCOS区块链的搭建和控制台的下载工作，本教程中的操作假设在该文档搭建的环境下进行。

## 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](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/smart_contract.html#crud)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

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 |*

*// |---------------------|-------------------|*

*//*

*// 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 asset\_value) **public** returns(int256){

int256 ret\_code = 0;

int256 ret= 0;

uint256 temp\_asset\_value = 0;

*// 查询账户是否存在*

(ret, temp\_asset\_value) = select(account);

**if**(ret != 0) {

Table table = openTable();

Entry entry = table.newEntry();

entry.set("account", account);

entry.set("asset\_value", int256(asset\_value));

*// 插入*

**int** count = table.insert(account, entry);

**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, asset\_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\_account, to\_account, amount);

**return** ret\_code;

}

Entry entry1 = table.newEntry();

entry1.set("account", to\_account);

entry1.set("asset\_value", int256(to\_asset\_value + amount));

*// 更新接收账户*

table.update(to\_account, entry1, table.newCondition());

emit TransferEvent(ret\_code, from\_account, to\_account, amount);

**return** ret\_code;

}

}

**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](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/manual/smart_contract.html#crud)。

## 合约编译

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合约文件编译为Java文件。

控制台提供了编译工具，可以将Asset.sol合约文件存放在console/contracts/solidity目录。利用console目录下提供的sol2java.sh脚本进行编译，操作如下：

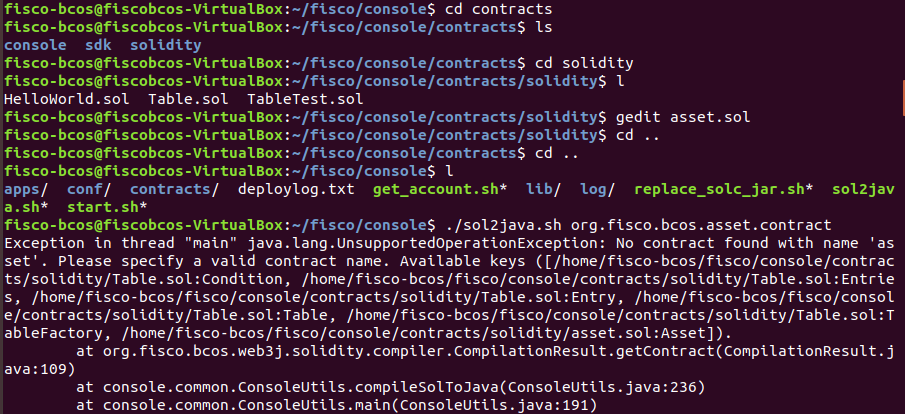
*# 切换到fisco/console/目录*

$ cd ~/fisco/console/

*# 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

这里遇到了一个问题，就是一开始我编译合约后报错说找不到asset合约，检查多次之后发现是把文件名写错了，把Asset.sol写成了asset.sol所以一直编译不过：



更改之后删除asset.sol后成功编译：



运行成功之后，将会在console/contracts/sdk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

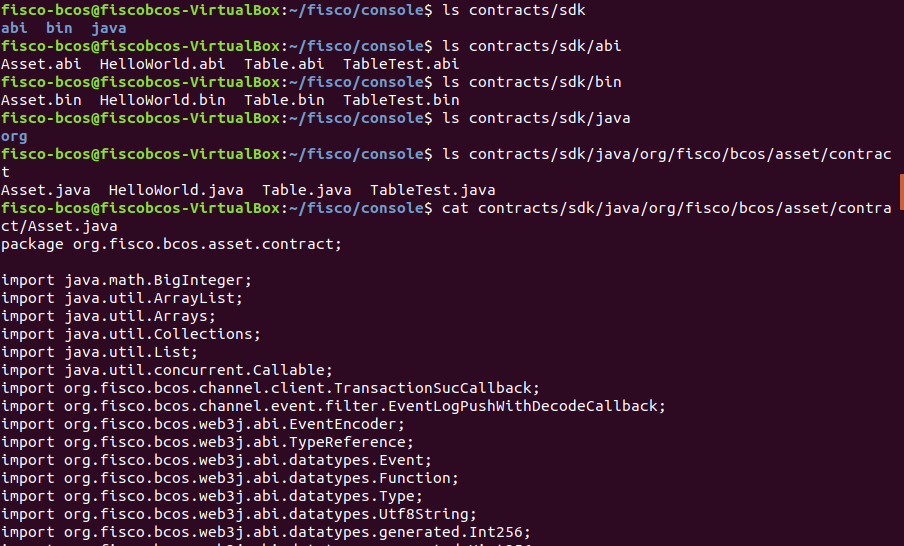
| |--asset

| |--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：

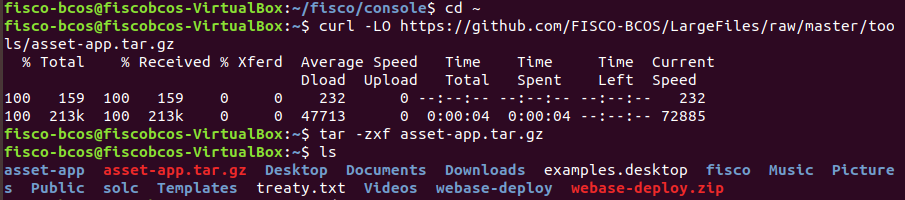
*# 获取Java工程项目压缩包*

$ 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

| | |-- AssetClient.java

| | |-- contract // 放置Java合约类

| | |-- Asset.java

| |-- test

| |-- resources // 存放代码资源文件

| |-- applicationContext.xml // 项目配置文件

| |-- 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

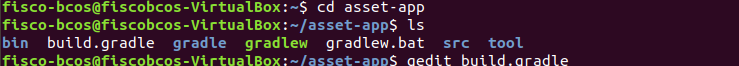
| |-- Asset.sol

| |-- 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：2.1.0')

### 证书与配置文件

* 区块链节点证书配置

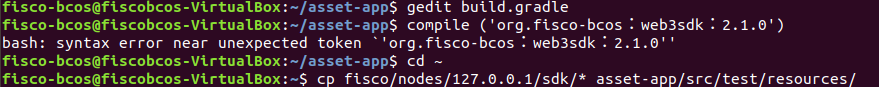
拷贝区块链节点对应的SDK证书

*# 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](https://fisco-bcos-documentation.readthedocs.io/zh_CN/latest/docs/sdk/java_sdk.html#spring)。

## 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** ClassPathXmlApplicationContext("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。

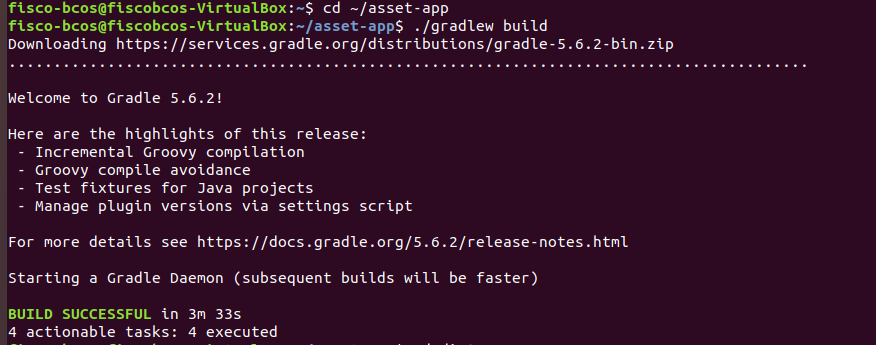
* 编译

*# 切换到项目目录*

$ cd ~/asset-app

*# 编译项目*

$ ./gradlew build



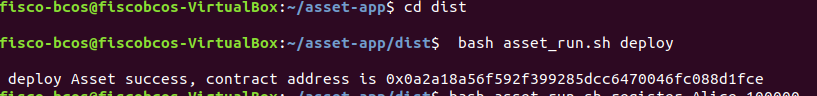
编译成功之后，将在项目根目录下生成dist目录。dist目录下有一个asset\_run.sh脚本，简化项目运行。现在开始一一验证本文开始定下的需求。部署Asset.sol合约

*# 进入dist目录*

$ cd dist

$ bash asset\_run.sh deploy

Deploy Asset succesfully, contract address is 0xd09ad04220e40bb8666e885730c8c460091a4775



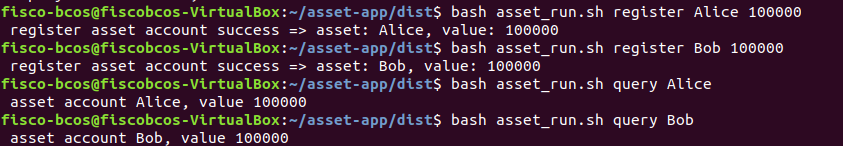
* 注册资产

$ 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



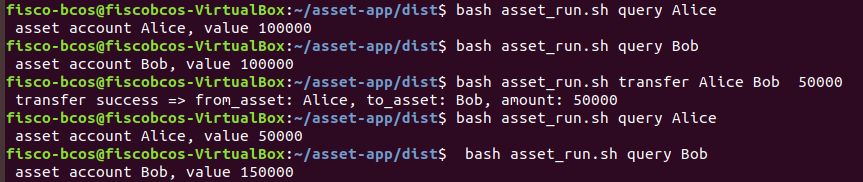
* 查询资产

$ 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

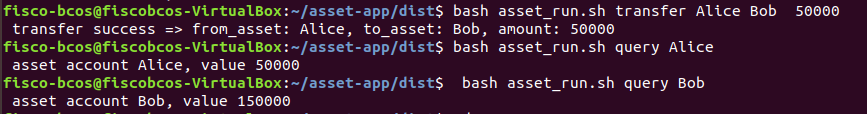
Transfer successfully => from\_account: Alice, to\_account: Bob, amount: 50000

$ 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。以上过程：

