

中山大学计算机学院本科生实验报告

(2021 年秋季学期)

课程名称：区块链原理与技术

年级	2019 级	专业（方向）	软件工程
学号	18327034	姓名	胡泽钊
电话	13724330119	Email	huzzh3@foxmail.com
开始日期	2022-01-04	完成日期	2022-01-06

0 前言

本次大作业仅由本人一人完成，本人个人信息如下：

姓名：胡泽钊

学号：18327034

专业：软件工程

年级：2019 级

同时，程序的演示视频我已经上传至 `github` 了：

- `github` 地址为：https://github.com/huzzh3/fisco_bcos_supply_finance
- 视频位置为代码仓库的 `report/run.mp4`

1 项目背景

假设我们现在处于传统的交易环境下，某核心车企向轮胎公司购买了一批轮胎，由于该车企资金短缺，故该车企向轮胎公司承诺，一年后归还 1000 万元，此时轮胎公司有 1000 万的应收账款。

某天，该轮胎公司也出现了资金短缺的问题，它拿着车企给的 1000 万的单据向银行融资 500 万，银行认可该车企的还款能力，故同意向轮胎公司借款 500 万元。

但是又过了一个月，该轮胎公司需要向轮毂公司购买一批轮毂，由于该轮胎公司经营不善，其向轮毂公司承诺一年后归还 500 万元，此时轮毂公司也有了 500 万的应收账款。

可是轮毂公司也发生了资金短缺的问题，它拿着轮胎公司的 500 万钱款单据向银行借款，但是银行不认可轮胎公司的经营能力，不予轮毂公司贷款，可是此时，轮胎公司手头上有着 1000 万的车企的欠款，按理来说，其应该具备还款能力，可由于传统的交易模式下，各种交易环节不透明，使得交易信用无法向下游公司传递，这也就给交易带来很多额外的成本。

故我们希望通过区块链的技术，能够搭建一种新型的供应链金融，使得核心企业（车企）的交易信用能够向下游传递，也就是说，下游企业之间能够转让车企的欠款，使得下游企业可以凭借车企的交易信用向银行融资。

2 方案设计

2.1 私有链的搭建以及新节点的加入

2.1.0 准备实验环境

首先，安装依赖：

```
sudo apt install -y curl openssl
```

之后，创建操作目录，下载安装脚本：

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

# 下载建链脚本
curl -#LO https://github.com/FISCO-BCOS/FISCO-BCOS/releases/download/v3.0.0-rc1/build_chain.sh && chmod u+x build_chain.sh
```

2.1.1 搭建私有链

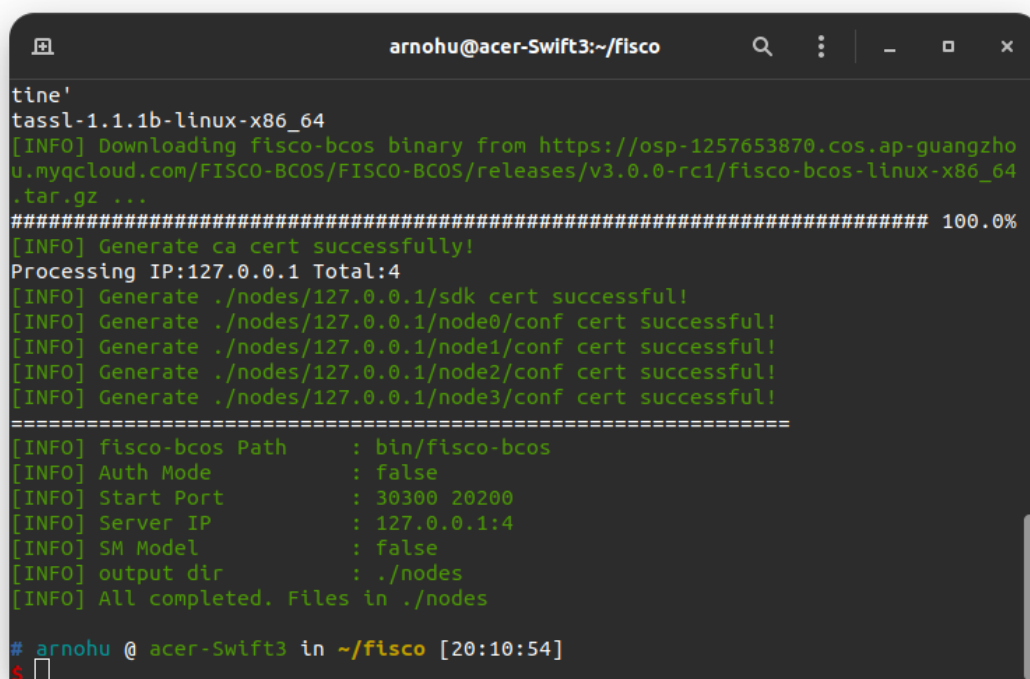
2.1.1.1 搭建Air版本FISCO BCOS联盟链

在 `fisco` 目录下执行以下指令，生成一条单群组 4 节点的 `FISCO` 链：

```
bash build_chain.sh -l 127.0.0.1:4 -p 30300,20200
```

其中 30300 是 `p2p` 监听端口，20200 是 `rpc` 监听端口。

当终端显示以下页面的时候，就表示私有链搭建完毕：



```
arnohu@acer-Swift3:~/fisco
tine'
tassl-1.1.1b-linux-x86_64
[INFO] Downloading fisco-bcos binary from https://osp-1257653870.cos.ap-guangzho
u.myqcloud.com/FISCO-BCOS/FISCO-BCOS/releases/v3.0.0-rc1/fisco-bcos-linux-x86_64
.tar.gz ...
##### 100.0%
[INFO] Generate ca cert successfully!
Processing IP:127.0.0.1 Total:4
[INFO] Generate ./nodes/127.0.0.1/sdk cert successful!
[INFO] Generate ./nodes/127.0.0.1/node0/conf cert successful!
[INFO] Generate ./nodes/127.0.0.1/node1/conf cert successful!
[INFO] Generate ./nodes/127.0.0.1/node2/conf cert successful!
[INFO] Generate ./nodes/127.0.0.1/node3/conf cert successful!
=====
[INFO] fisco-bcos Path      : bin/fisco-bcos
[INFO] Auth Mode           : false
[INFO] Start Port          : 30300 20200
[INFO] Server IP           : 127.0.0.1:4
[INFO] SM Model            : false
[INFO] output dir           : ./nodes
[INFO] All completed. Files in ./nodes

# arnohu @ acer-Swift3 in ~/fisco [20:10:54]
$
```

输入以下指令启动所有节点

```
bash nodes/127.0.0.1/start_all.sh
```

成功启动以后，会输出以下页面：



```
arnohu@acer-Swift3:~/fisco
# arnohu @ acer-Swift3 in ~/fisco [20:15:26]
$ bash nodes/127.0.0.1/start_all.sh
try to start node0
node0 start successfully
try to start node1
node1 start successfully
try to start node2
node2 start successfully
try to start node3
node3 start successfully

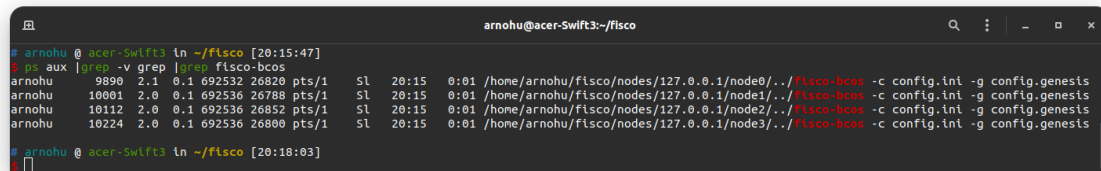
# arnohu @ acer-Swift3 in ~/fisco [20:15:47]
$
```

可以看到四个节点都已经启动完毕了。

可以通过以下命令检查进程是否启动：

```
ps aux |grep -v grep |grep fisco-bcos
```

命令执行结果如下：



```
arnohu@acer-Swift3:~/fisco
# arnohu @ acer-Swift3 in ~/fisco [20:15:47]
$ ps aux |grep -v grep |grep fisco-bcos
arno  9890  2.1  0.1 692532 26820 pts/1    Sl   20:15   0:01 /home/arnohu/fisco/nodes/127.0.0.1/node0/./fisco-bcos -c config.ini -g config.genesis
arno 10001  2.0  0.1 692536 26788 pts/1    Sl   20:15   0:01 /home/arnohu/fisco/nodes/127.0.0.1/node1/./fisco-bcos -c config.ini -g config.genesis
arno 10112  2.0  0.1 692536 26852 pts/1    Sl   20:15   0:01 /home/arnohu/fisco/nodes/127.0.0.1/node2/./fisco-bcos -c config.ini -g config.genesis
arno 10224  2.0  0.1 692536 26800 pts/1    Sl   20:15   0:01 /home/arnohu/fisco/nodes/127.0.0.1/node3/./fisco-bcos -c config.ini -g config.genesis

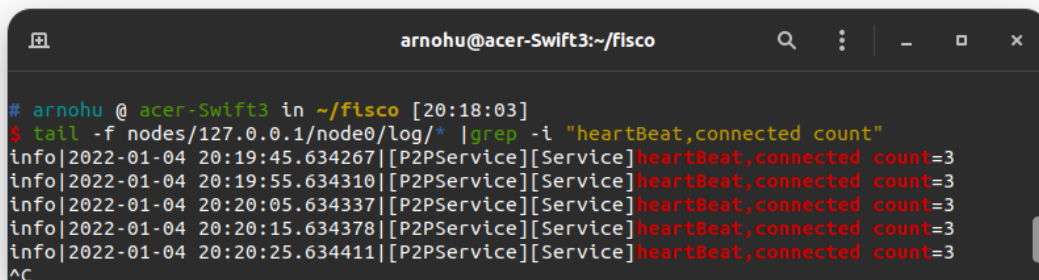
# arnohu @ acer-Swift3 in ~/fisco [20:18:03]
$
```

可以看到我们有 4 个进程正在运行，每个进程对应 1 个节点。

可以通过以下命令查看每个节点的网络链接数目(以 node0 为例)：

```
tail -f nodes/127.0.0.1/node0/log/* |grep -i "heartBeat,connected count"
```

运行结果如下：



```
arnohu@acer-Swift3:~/fisco
# arnohu @ acer-Swift3 in ~/fisco [20:18:03]
$ tail -f nodes/127.0.0.1/node0/log/* |grep -i "heartBeat,connected count"
info|2022-01-04 20:19:45.634267|[P2PService][Service]heartBeat,connected count=3
info|2022-01-04 20:19:55.634310|[P2PService][Service]heartBeat,connected count=3
info|2022-01-04 20:20:05.634337|[P2PService][Service]heartBeat,connected count=3
info|2022-01-04 20:20:15.634378|[P2PService][Service]heartBeat,connected count=3
info|2022-01-04 20:20:25.634411|[P2PService][Service]heartBeat,connected count=3
^C
```

可以看到 `node0` 和其他 3 个节点均有链接，网络连接正常。

2.1.1.2 配置和使用控制台

首先我们要安装控制台的运行依赖(**java 14**)，指令如下所示：

```
# ubuntu系统安装java
sudo apt install -y default-jdk
```

接下来，下载控制台：

```
cd ~/fisco && curl -LO https://github.com/FISCO-BCOS/console/releases/download/v3.0.0-rc1/download_console.sh && bash
download_console.sh
```

下载成功后的终端如图所示：



```
arnohu@acer-Swift3:~/fisco
# arnohu @ acer-Swift3 in ~/fisco [20:25:45]
$ cd ~/fisco && curl -LO https://github.com/FISCO-BCOS/console/releases/download/v3.0.0-rc1/download_console.sh && bash download_console.sh
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed
100 658    100 658    0    0  1107      0 --:--:-- --:--:-- --:--:-- 1105
100 3672   100 3672    0    0  1647      0 0:00:02 0:00:02 --:--:-- 17825
[INFO] Downloading console 3.0.0-rc1 from https://github.com/FISCO-BCOS/console/releases/download/v3.0.0-rc1/console.tar.gz
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed
100 653    100 653    0    0  1200      0 --:--:-- --:--:-- --:--:-- 1200
100 58.4M   100 58.4M    0    0 2600k      0 0:00:23 0:00:23 --:--:-- 2671k
[INFO] Download console successfully
[INFO] unzip console successfully

# arnohu @ acer-Swift3 in ~/fisco [20:49:13]
$
```

接下来，拷贝控制台配置文件：

```
cp -n console/conf/config-example.toml console/conf/config.toml
```

配置控制台证书，脚本在生成节点的同时，生成了 `SDK` 证书，可直接拷贝生成的证书供控制台使用：

```
cp -r nodes/127.0.0.1/sdk/* console/conf
```

接着启动并使用控制台：

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

启动成功后的终端如图所示：


```

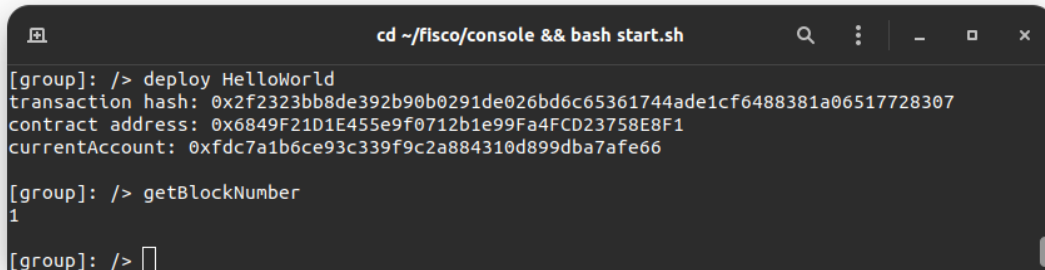
    constructor() public {
        name = "Hello, World!";
    }

    function get() public view returns (string memory) {
        return name;
    }

    function set(string memory n) public {
        name = n;
    }
}

```

之后我们通过 `deploy HelloWorld` 命令部署合约，同时可以用 `getBlockNumber` 查看块高：



```

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

[group]: /> deploy HelloWorld
transaction hash: 0x2f2323bb8de392b90b0291de026bd6c65361744ade1cf6488381a06517728307
contract address: 0x6849F21D1E455e9f0712b1e99Fa4FCD23758E8F1
currentAccount: 0xfdc7a1b6ce93c339f9c2a884310d899dba7afe66

[group]: /> getBlockNumber
1
[group]: /> 

```

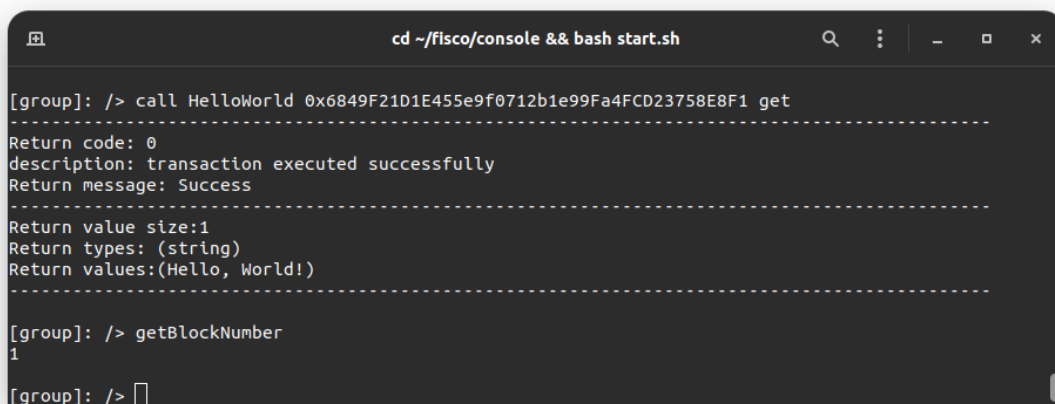
之后我们可以通过 `call` 指令调用 `HelloWorld` 合约，调用的方法如下：

```
call HelloWorld <deploy 指令返回的合约地址> <要调用的接口>
```

比如我们要调用 `get` 函数，就可以这样使用：

```
call HelloWorld 0x6849F21D1E455e9f0712b1e99Fa4FCD23758E8F1 get
```

调用完成后，控制台输出如下：



```

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

[group]: /> call HelloWorld 0x6849F21D1E455e9f0712b1e99Fa4FCD23758E8F1 get
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:1
Return types: (string)
Return values:(Hello, World!)
-----

[group]: /> getBlockNumber
1
[group]: /> 

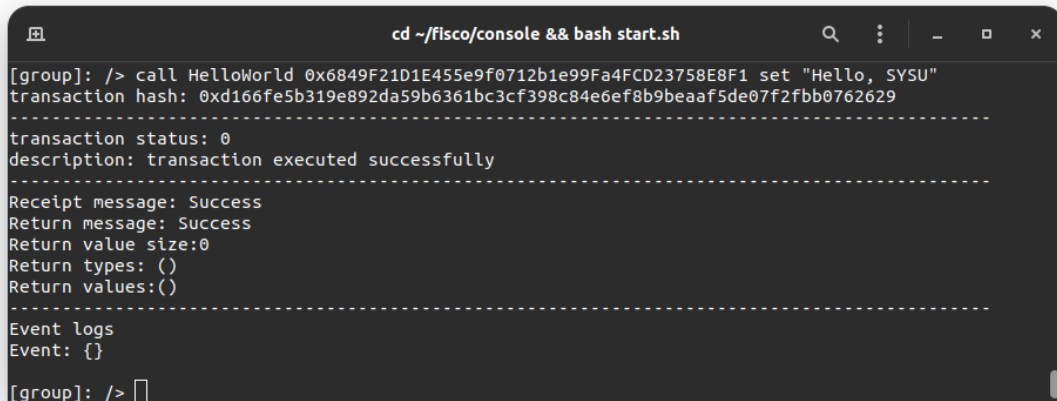
```

可以看到，调用结束后返回 `Hello, World!`，且使用 `getBlockNumber` 指令可以看到块高不变。

同样地，我们可以用以下指令调用 `set` 方法：

```
call HelloWorld 0x6849F21D1E455e9f0712b1e99Fa4FCD23758E8F1 set "Hello, SYSU"
```

执行之后，控制台的输出如下：



```
cd ~/fisco/console && bash start.sh
[group]: /> call HelloWorld 0x6849F21D1E455e9f0712b1e99Fa4FCD23758E8F1 set "Hello, SYSU"
transaction hash: 0xd166fe5b319e892da59b6361bc3cf398c84e6ef8b9beaaf5de07f2fbb0762629
-----
transaction status: 0
description: transaction executed successfully
-----
Receipt message: Success
Return message: Success
Return value size:0
Return types: ()
Return values:()
-----
Event logs
Event: {}
[group]: /> 
```

2.1.2 加入新节点

2.1.2.1 准备扩容所需的文件

通过翻阅官方提供的 `build_chain.sh` 的文档，可以通过 `-C` 的命令行参数来添加节点，官方的文档也可以通过 `bash build_chain.sh -h` 在命令行中翻阅。

扩容节点需要准备以下文件：

- **CA 证书和 CA 私钥**：用于为新扩容的节点颁发节点证书。
- **节点配置文件 `config.ini`**：可从已有的节点目录中拷贝。
- **节点创世块配置文件 `config.genesis`**：可从已有节点目录中拷贝。
- **节点连接配置 `node.json`**：配置所有节点连接的IP和端口信息，可从已有的节点目录中拷贝，并加上新节点的IP和端口。

首先，先创建扩容配置存放目录：

```
# 进入操作目录
cd ~/fisco

# 创建扩容配置存放目录
mkdir config
```

接下来，拷贝根证书和根证书私钥：

```
cp -r nodes/ca config
```

之后，从 `node0` 中拷贝节点配置文件 `config.ini`，创世块配置文件 `config.genesis` 以及节点连接配置文件 `nodes.json`：

```
cp nodes/127.0.0.1/node0/config.ini config/
cp nodes/127.0.0.1/node0/config.genesis config/
cp nodes/127.0.0.1/node0/nodes.json config/nodes.json.tmp
```

之后设置新节点 P2P 和 RPC 监听端口：

```
sed -i 's/listen_port=30300/listen_port=30304/g' config/config.ini
sed -i 's/listen_port=20200/listen_port=20204/g' config/config.ini
```

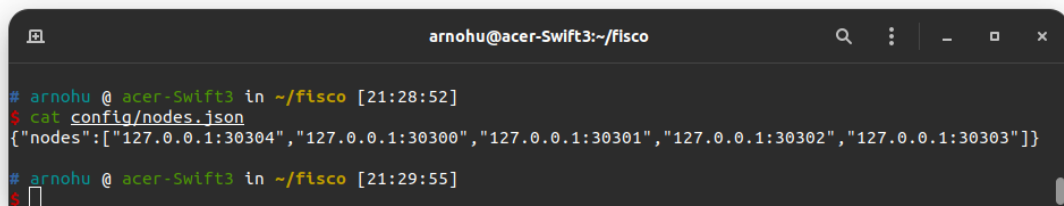
将新节点加入到 `nodes.json` 中：

```
sed -e 's/"nodes":\[/"nodes":\[ "127.0.0.1:30304",/' config/nodes.json.tmp >
config/nodes.json
```

最后，我们确认以下新节点的连接信息：

```
cat config/nodes.json
```

输出的结果如下：

A terminal window titled 'arnohu@acer-Swift3:~/fisco' showing the command 'cat config/nodes.json' and its output. The output is a JSON array of node addresses: ["127.0.0.1:30304", "127.0.0.1:30300", "127.0.0.1:30301", "127.0.0.1:30302", "127.0.0.1:30303"]. The terminal also shows the user's prompt and the time of execution [21:28:52] and [21:29:55].

```
arnohu@acer-Swift3:~/fisco [21:28:52]
$ cat config/nodes.json
{"nodes":["127.0.0.1:30304","127.0.0.1:30300","127.0.0.1:30301","127.0.0.1:30302","127.0.0.1:30303"]}
arnohu@acer-Swift3:~/fisco [21:29:55]
```

可以看到新节点已经加入到 `nodes.json` 中了。

2.1.2.2 扩容新节点

我们可以使用 `build_chain.sh` 脚本来扩容新节点，如下所示：

```
# 调用build_chain.sh扩容节点，新节点扩容到nodes/127.0.0.1/node4目录
# -c：指定扩容配置config.ini， config.genesis和nodes.json路径
# -d：指定CA证书和私钥的路径
# -o：指定扩容节点配置所在目录
bash build_chain.sh -C expand -c config -d config/ca -o nodes/127.0.0.1/node4
```

扩容完毕后，终端输出如下：


```
arnohu@acer-Swift3:~/fisco
[INFO] Generate nodes/127.0.0.1/node4/conf cert successful!
[INFO] generate_node_cert success...
[INFO] generate_node_account ...
[INFO] generate_node_account success...
[INFO] copy configurations ...
[INFO] copy configurations success...
=====
[INFO] fisco-bcos Path      : bin/fisco-bcos
[INFO] sdk dir               : nodes/127.0.0.1/sdk
[INFO] SM Model              : false
[INFO] output dir            : nodes/127.0.0.1/node4
[INFO] All completed. Files in nodes/127.0.0.1/node4

# arnohu @ acer-Swift3 in ~/fisco [21:33:22]
$
```

2.1.2.3 启动扩容节点

```
bash nodes/127.0.0.1/node4/start.sh
```

使用以下命令查看 `node4` 节点是否启动：

```
ps aux |grep -v grep |grep fisco-bcos
```

输出的结果如下：

```
arnohu@acer-Swift3:~/fisco
# arnohu @ acer-Swift3 in ~/fisco [21:34:21]
$ ps aux |grep -v grep |grep fisco-bcos
arno  9890  1.9  0.2 705848 34240 pts/1    Sl   20:15   1:34 /home/arnohu/fisco/nodes/127.0.0.1/node0/./fisco-bcos -c config.ini -g config.genesis
arno 10001  1.9  0.2 705852 33800 pts/1    Sl   20:15   1:34 /home/arnohu/fisco/nodes/127.0.0.1/node1/./fisco-bcos -c config.ini -g config.genesis
arno 10112  1.9  0.2 704828 33364 pts/1    Sl   20:15   1:34 /home/arnohu/fisco/nodes/127.0.0.1/node2/./fisco-bcos -c config.ini -g config.genesis
arno 10224  1.9  0.2 705852 33352 pts/1    Sl   20:15   1:33 /home/arnohu/fisco/nodes/127.0.0.1/node3/./fisco-bcos -c config.ini -g config.genesis
arno 19618 1.6  0.1 691504 20276 pts/1    Sl   21:34   0:00 /home/arnohu/fisco/nodes/127.0.0.1/node4/./fisco-bcos -c config.ini -g config.genesis

# arnohu @ acer-Swift3 in ~/fisco [21:34:54]
$
```

可以看到 `node4` 已经启动了。

接下来我们要获取节点 ID，以方便我们将其加入为观察者节点或共识节点：

```
cat nodes/127.0.0.1/node4/conf/node.nodeid
```

其输出如下：

```
arnohu@acer-Swift3:~/fisco
# arnohu @ acer-Swift3 in ~/fisco [21:34:54]
$ cat nodes/127.0.0.1/node4/conf/node.nodeid
4ca8e94b1760b56ec3ddfbe858b9bbd3518b101b231131a889c4ccc21d809f9ee184a69f8c180eb51428a6a6905d86e6a24510ae0c97fc5806e00e60140ed

# arnohu @ acer-Swift3 in ~/fisco [21:39:45]
$
```

可以得到节点 ID 如下：

```
4ca8e94b176b0b56ec3ddfbe6858b9bbd3518b101b231131a889c4ccc21d809f9ee184a69f8c180eb51428a6fa6905d86e6a24510ae0c97fc5806e00e60140ed
```

接下来启动控制台，将 `node4` 设置为观察者节点：

```
# 启动控制台
cd ~/fisco/console && bash start.sh
```

使用 `addObserver` 将节点设置为观察者节点，并用 `getObserverList` 确认节点是否成功加入观察节点：



```
cd ~/fisco/console && bash start.sh
Welcome to FISCO BCOS console(3.0.0-rc1)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.

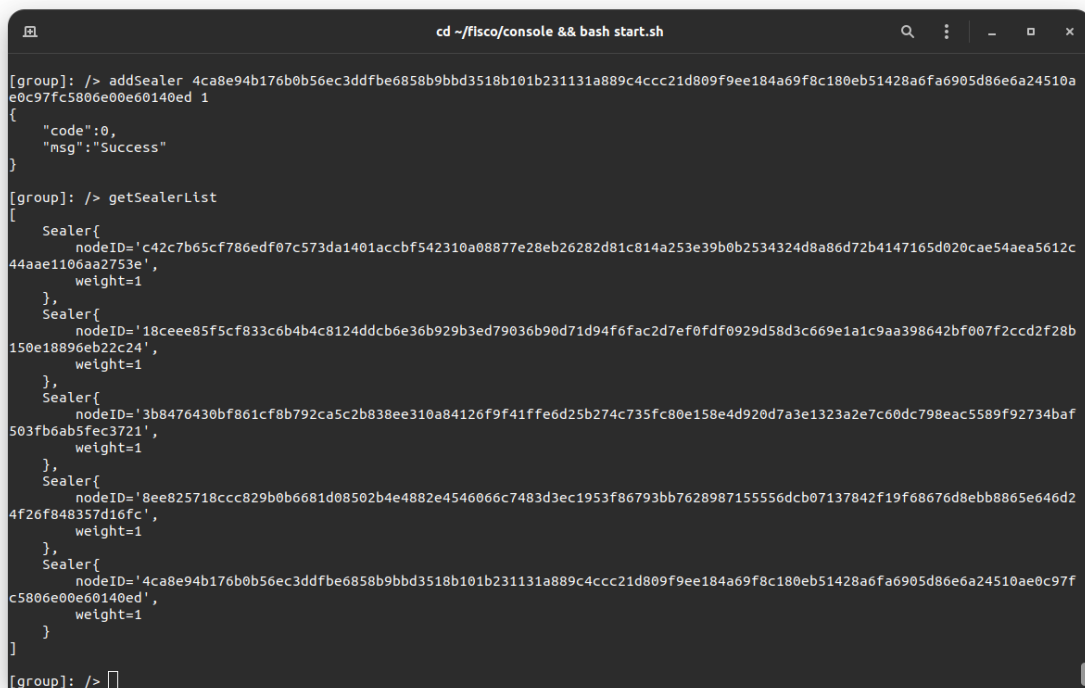
=====
[group]: /> addObserver 4ca8e94b176b0b56ec3ddfbe6858b9bbd3518b101b231131a889c4ccc21d809f9ee184a69f8c180eb51428a6fa6905d86e6a24510ae0c97fc5806e00e60140ed
{
  "code":0,
  "msg":"Success"
}

[group]: /> getObserver
Undefined command: "getObserver". Try "help".

[group]: /> getObserverList
[
  4ca8e94b176b0b56ec3ddfbe6858b9bbd3518b101b231131a889c4ccc21d809f9ee184a69f8c180eb51428a6fa6905d86e6a24510ae0c97fc5806e00e60140ed
]

[group]: />
```

同样地，可以通过 `addSealer` 将扩容节点设置为共识节点：



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

[group]: /> addSealer 4ca8e94b176b0b56ec3ddfbe6858b9bbd3518b101b231131a889c4ccc21d809f9ee184a69f8c180eb51428a6fa6905d86e6a24510ae0c97fc5806e00e60140ed 1
{
  "code":0,
  "msg":"Success"
}

[group]: /> getSealerList
[
  Sealer{
    nodeId='c42c7b65cf786edf07c573da1401accbf542310a08877e28eb26282d81c814a253e39b0b2534324d8a86d72b4147165d020cae54aea5612c44aae1106aa2753e',
    weight=1
  },
  Sealer{
    nodeId='18ceee85f5cf833c6b4b4c8124ddcb6e36b929b3ed79036b90d71d94f6fac2d7ef0fdf0929d58d3c669e1a1c9aa398642bf007f2ccd2f28b150e18896eb22c24',
    weight=1
  },
  Sealer{
    nodeId='3b8476430bf861cf8b792ca5c2b838ee310a84126f9f41ffe6d25b274c735fc80e158e4d920d7a3e1323a2e7c60dc798eac5589f92734baf503fb6ab5fec3721',
    weight=1
  },
  Sealer{
    nodeId='8ee825718ccc829b0b6681d08502b4e4882e4546066c7483d3ec1953f86793bb7628987155556dcb07137842f19f68676d8ebb8865e646d24f26f848357d16fc',
    weight=1
  },
  Sealer{
    nodeId='4ca8e94b176b0b56ec3ddfbe6858b9bbd3518b101b231131a889c4ccc21d809f9ee184a69f8c180eb51428a6fa6905d86e6a24510ae0c97fc5806e00e60140ed',
    weight=1
  }
]

[group]: />
```

至此，`node4` 节点加入完毕。

2.2.3 编写智能合约

我们尝试自己编写一个智能合约并部署到我们的联盟私有链上，我们模仿 `soliditylang` 官方提供的简单的加密货币的智能合约进行合约的编写，如下所示：

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 < 0.7.0;

contract Coin {
    /* address 是一种 160 位的变量类型，它不允许任何的算数操作，
     * 因此 address 适合存储合约的地址或者外部账户公钥的哈希值。
     * "public" 保留字能够让该属性被其他的合约访问，它会自动产生函数
     * 让其他合约能够直接访问该属性，在这里自动产生的函数是 minter()
     */
    address public minter;

    /* 下列 mapping 将 address 映射为 uint
     * 可以将 mapping 看成是哈希表，该哈希表会被虚拟初始化，
     * 使得每个可能的键值都存在，并映射到一个零值。
     * 需要注意的是，既不能获取所有键值，也不能获取所有值
     * public 产生的函数是 balances(address _account)
     */
    mapping (address => uint) public balances;

    /* event 允许客户对我们声明的特定合同改变做出反应
     * 可以理解为，event 会向其他客户发送一个信号，告诉用户有合约发生了改变
     * 所以客户只需要监听对应的网络接口，就知道有什么合约过来了
     */
    event Sent(address from, address to, uint amount);

    /* 构造函数只会在初始化中被调用
     * msg 是一种特殊的全局变量，msg.sender 始终是当前函数调用的来源地址
     */
    constructor() public {
        minter = msg.sender;
    }

    /* mint 函数能够发送一定数量的新的货币给某个地址
     * require 函数要求只有合约的创建人能够发放货币
     */
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    /* send 函数能够让客户之间交易货币
     * require 函数要求只有合约的创建人能够发放货币
     */
    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

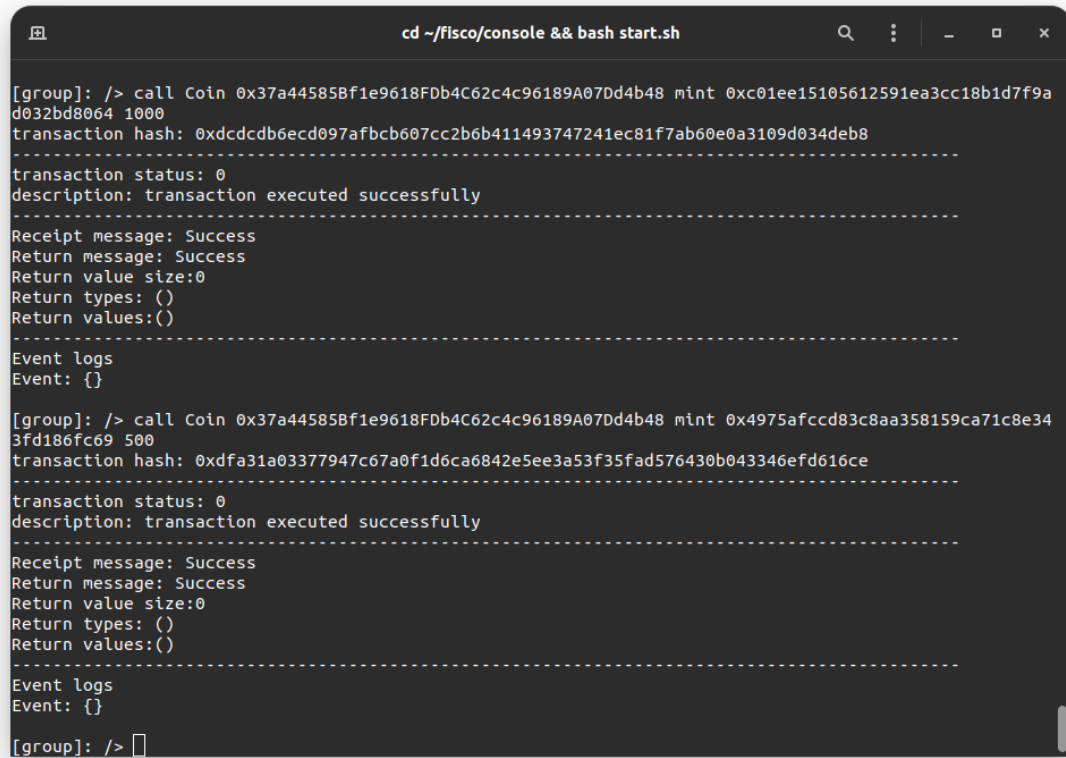
我们将该合约复制到我们私有链的合约目录下：


```
account1: 0xc01ee15105612591ea3cc18b1d7f9ad032bd8064
account2: 0x4975afccd83c8aa358159ca71c8e343fd186fc69
```

接下来，我们给第一个账户发放 1000 个虚拟货币，给第二个账户发放 500 个虚拟货币，如下所示：

```
call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 mint
0xc01ee15105612591ea3cc18b1d7f9ad032bd8064 1000

call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 mint
0x4975afccd83c8aa358159ca71c8e343fd186fc69 500
```



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

[group]: /> call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 mint 0xc01ee15105612591ea3cc18b1d7f9a
d032bd8064 1000
transaction hash: 0xdcdddb6ecd097afbc607cc2b6b411493747241ec81f7ab60e0a3109d034deb8
-----
transaction status: 0
description: transaction executed successfully
-----
Receipt message: Success
Return message: Success
Return value size:0
Return types: ()
Return values:()
-----
Event logs
Event: {}

[group]: /> call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 mint 0x4975afccd83c8aa358159ca71c8e34
3fd186fc69 500
transaction hash: 0xdfa31a03377947c67a0f1d6ca6842e5ee3a53f35fad576430b043346efd616ce
-----
transaction status: 0
description: transaction executed successfully
-----
Receipt message: Success
Return message: Success
Return value size:0
Return types: ()
Return values:()
-----
Event logs
Event: {}

[group]: /> 
```

接下来，我们查询目前 3 个账户的货币情况（2 个新账户加上 1 个造币账户）：

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

[group]: /> call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 balances 0xfdc7a1b6ce93c339f9c2a88431
0d899dba7afe66
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:1
Return types: (uint256)
Return values:(0) ← 造币账户中没有任何货币
-----

[group]: /> call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 balances 0x4975afccd83c8aa358159ca71c
8e343fd186fc69
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:1
Return types: (uint256)
Return values:(500) ← 账户2有500个货币
-----

[group]: /> call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 balances 0xc01ee15105612591ea3cc18b1d
7f9ad032bd8064
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:1
Return types: (uint256)
Return values:(1000) ← 账户1有1000个货币
-----

[group]: /> 
```

可以看到我们货币发放成功了。

接下来，我们尝试让账户 2 转 500 个货币给账户 1，首先我们要切换账户为账户 2：

```
loadAccount 0x4975afccd83c8aa358159ca71c8e343fd186fc69
```

接下来，让账户 2 转账 500 个货币给账户 1：

```
call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 send
0xc01ee15105612591ea3cc18b1d7f9ad032bd8064 500
```

查看一下两个账户的余额：

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

[group]: /> call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 balances 0xc01ee15105612591ea3cc18b1d7f9ad032bd8064
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:1
Return types: (uint256)
Return values:(1500) ← 账户 1 目前有 1500 枚货币
-----

[group]: /> call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 balances 0x4975afccd83c8aa358159ca71c8e343fd186fc69
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:1
Return types: (uint256)
Return values:(0) ← 账户 2 目前没有货币
-----

[group]: />
```

可以看到账户 1 的余额变成了 1500，而账户 2 的余额变成了 0

我们再尝试让账户 2 转 500 给账户 1，结果如下：

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

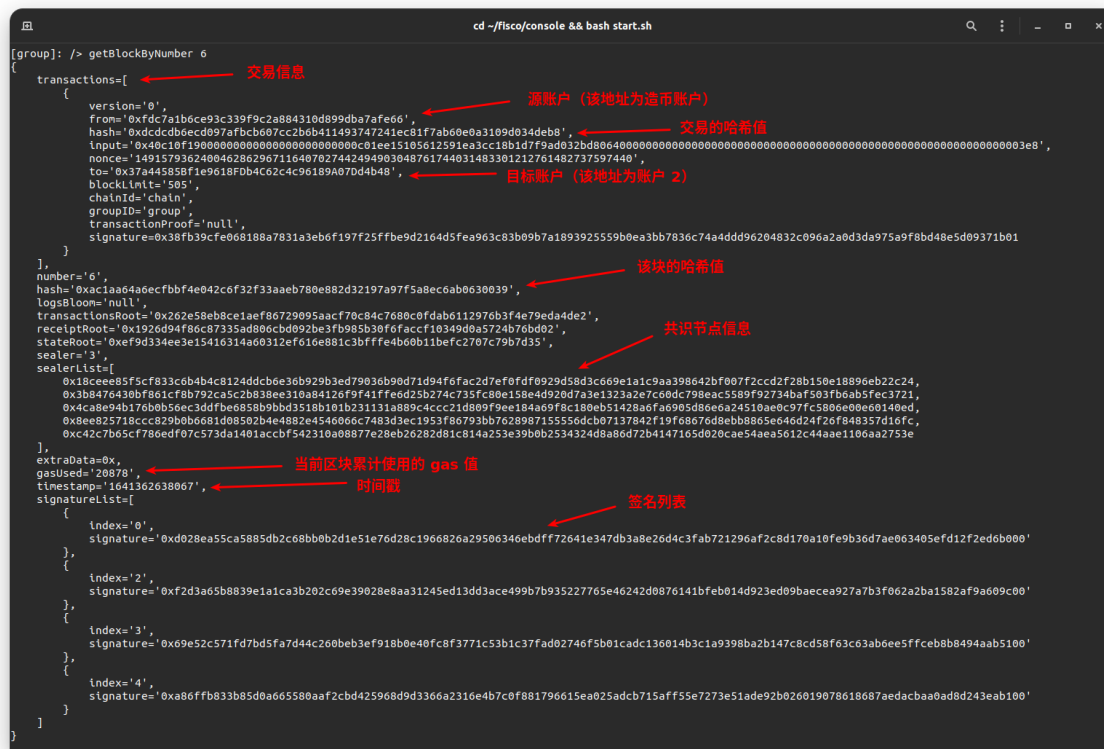
[group]: /> call Coin 0x37a44585Bf1e9618FDb4C62c4c96189A07Dd4b48 send 0xc01ee15105612591ea3cc18b1d7f9ad032bd8064 500
transaction hash: 0xcf8a5155b80c7829a96f5df5ed43bd6f2dbe5bd1012c02899cbd42094a9c5521
-----
transaction status: 16
-----
Receipt message: Insufficient balance.
Return message: Insufficient balance.
-----

[group]: />
```

可以看到此时，系统提示我们账户余额不足。

2.2.5 查看区块并对字段进行解释

使用 `getBlockByNumber` 即可通过块的高度来查看块的信息，我们查看第六个块的信息，如下所示：



2.2 项目设计说明

在本节中，我将着重说明我是如何围绕着**项目背景**来进行智能合约的设计的。

首先，我实现的功能有以下四种：

- 核心企业向下游企业签发应收账款。
- 下游企业之间进行应收账款的转让。
- 下游企业根据自身的应收账款向银行申请融资。
- 核心企业结清下游企业的应收账款。

除了上述以上四个功能以外，我们还需要能够保存各个公司的数据，我是通过使用 `fisco` 自带的 `KVTable` 实现的，该 `KVTable` 一共就有两种数据，分别为：

- `company`：公司名
- `asset_receivable`：应收账款

接下来，我们着重说明上述四种函数是如何在智能合约中实现的。

2.2.1 公司注册

由于我们是通过 `KVTable` 进行数据存储的，故我们需要通过该公司在表中进行注册，并存储它现有的应收账款，具体实现的代码如下：

```
function register(string memory company, uint256 assetReceivable) public
returns (int256) {
    int256 retCode          = 0;
    bool  ret                = true;
    uint256 tempAssetReceivable = 0;

    // 查询该公司名是否在表中
    (ret, tempAssetReceivable) = select(company);
```



```

        if (ret == false) {
            // 此公司不在表中, 需要将此公司插入表中
            string memory assetReceivableStr = uint2str(assetReceivable);
            KVField memory kv1 = KVField("asset_receivable",
assetReceivableStr);
            KVField[] memory KVFields = new KVField[](1);
            KVFields[0] = kv1;
            Entry memory entry = Entry(KVFields);

            // 表项已经创建好了, 现在将该表项插入表中
            int256 count = tf.set("asset_r", company, entry);

            // 判断插入是否成功
            if (count == 1) {
                retCode = 0;
            }
            else {
                retCode = -2;
            }
        }
        else {
            retCode = -1;
        }

        emit RegisterEvent(retCode, company, assetReceivable);

        return retCode;
    }

```

代码比较长, 我们简单的分几步描述一下代码的逻辑:

1. 当公司要注册的时候, 我们要根据输入的公司名去查找该公司是否已经在表中了。
2. 若该公司不在表中, 则使用 `KVTable` 的接口将该公司插入到表里。
3. 若该公司在表中, 则返回错误码 `-1`。
4. 若该公司插入到表失败, 则返回错误码 `-2`。

2.2.2 签发账款

首先, 我们既然要签发账款, 那么我们就需要明确, **签发的对象必须是核心公司**, 其次, 由于 `solidity` 特殊的性质, 它的整数会发生溢出, 所以我们还必须**确保签发的金额不会发生溢出**, 就此我们展示一下我们的代码实现, 并在后面简述一下我们的代码逻辑:

```

function send(string memory fromCompany, string memory toCompany, uint256
assetReceivableTo) public returns (int256) {
    int256 retCode = 0;
    bool ret1 = true;
    bool ret2 = true;
    uint256 tempAssetReceivable1 = 0;
    uint256 tempAssetReceivable2 = 0;

    // 查询签发公司是否在表中
    (ret1, tempAssetReceivable1) = select(fromCompany);
    // 查询目标公司是否在表中
    (ret2, tempAssetReceivable2) = select(toCompany);

```

```

        if (keccak256(bytes(fromCompany)) != keccak256(bytes(coreCompanyName))) {
            // 签发公司不是核心公司
            retCode = -1;
        }
        else if (ret1 == false) {
            // 签发公司不在表中
            retCode = -2;
        }
        else if (ret2 == false) {
            // 目标公司不在表中
            retCode = -3;
        }
        else if (tempAssetReceivable2 + assetReceivableTo < tempAssetReceivable2)
    {
        // 签发的金额发生了溢出
        retCode = -4;
    }
    else {
        // 此时对表进行修改
        string memory companyNewAssetReceivable =
uint2str(tempAssetReceivable2 + assetReceivableTo);
        KVField memory kv1 =
KVField("asset_receivable", companyNewAssetReceivable);
        KVField[] memory KVFields = new KVField[](1);
        KVFields[0] = kv1;
        Entry memory entry = Entry(KVFields);

        // 数据项设置完毕，将其同步到表中
        int256 count = tf.set("asset_r", toCompany, entry);

        // 判断同步是否成功
        if (count != 1) {
            retCode = -5;
        }
        else {
            retCode = 0;
        }
    }

    emit SendEvent(retCode, fromCompany, toCompany, assetReceivableTo);

    return retCode;
}

```

代码的运行逻辑如下：

1. 首先对表进行查询。
2. 如果签发公司不是核心公司，则返回错误码 -1
3. 如果签发公司不在表中，则返回错误码 -2
4. 如果目标公司不在表中，则返回错误码 -3
5. 如果签发金额发生了溢出，则返回错误码 -4
6. 如果上述步骤都没有发生错误，则对表进行修改，使得接收公司的应收账款金额得到增加。
7. 如果对表的修改操作发生了错误，那么就返回错误码 -5
8. 如果上述步骤都没有问题，则返回成功码 0

2.2.3 转让账款

转让账款可能会比较复杂一点，它不仅要考虑金额溢出的问题，还要考虑金额不足的问题，他的代码如下：

```
function transfer(string memory fromCompany, string memory toCompany, uint256
assetReceivableTo) public returns (int256) {
    int256 retCode          = 0;
    bool    ret1             = true;
    bool    ret2             = true;
    uint256 tempAssetReceivable1 = 0;
    uint256 tempAssetReceivable2 = 0;

    // 查询签发公司是否在表中
    (ret1, tempAssetReceivable1) = select(fromCompany);
    // 查询目标公司是否在表中
    (ret2, tempAssetReceivable2) = select(toCompany);

    if (keccak256(bytes(fromCompany)) == keccak256(bytes(coreCompanyName))) {
        // 转让的源公司为核心公司
        retCode = -1;
    }
    else if (ret1 == false) {
        // 转让的源公司不存在
        retCode = -2;
    }
    else if (ret2 == false) {
        // 转让的目标公司不存在
        retCode = -3;
    }
    else if (tempAssetReceivable1 < assetReceivableTo) {
        // 转让的金额不足
        retCode = -4;
    }
    else if (tempAssetReceivable2 + assetReceivableTo < tempAssetReceivable2)
    {
        // 转让的金额溢出
        retCode = -5;
    }
    else {
        // 创建数据项
        string memory tempAssetReceivableStr =
uint2str(tempAssetReceivable1 - assetReceivableTo);
        KVField memory kv                    = KVField("asset_receivable",
tempAssetReceivableStr);
        KVField[] memory KVFields           = new KVField[](1);
        KVFields[0]                          = kv;
        Entry memory entry                  = Entry(KVFields);

        // 更新转让的源公司的应收账款
        int256 count = tf.set("asset_r", fromCompany, entry);
        if (count != 1) {
            // 对表的修改失败
            retCode = -6;
        }
        else {
            // 创建另一个数据项
```

```

        tempAssetReceivableStr = uint2str(tempAssetReceivable2 +
assetReceivableTo);
        kv                        = KVField("asset_receivable",
tempAssetReceivableStr);
        KVFields                = new KVField[](1);
        KVFields[0]             = kv;
        entry                   = Entry(KVFields);

        // 更新转让的目标公司的应收账款
        count = tf.set("asset_r", toCompany, entry);
        if (count != 1) {
            // 对表的修改失败
            retCode = -6;
        }
        else {
            retCode = 0;
        }
    }
}

emit TransferEvent(retCode, fromCompany, toCompany, assetReceivableTo);

return retCode;
}

```

它的运行逻辑如下：

1. 查询源公司和目标公司。
2. 如果源公司为核心公司，则返回错误码 -1，核心公司不能进行转让欠款的操作。
3. 如果源公司不存在，则返回错误码 -2
4. 如果目标公司不存在，则返回错误码 -3
5. 如果转让的金额不足，则返回错误码 -4
6. 如果转让的金额溢出，则返回错误码 -5
7. 如果上述步骤都没有发生错误，那么就对表进行修改，使得源公司的应收账款减少，而目标公司的应收账款增加。
8. 如果对表的操作发生了错误，则返回错误码 -6
9. 如果上述操作都没有问题，则返回成功码 0

2.2.4 银行融资

银行融资就比较简单了，只需要判断该公司的应收账款是否小于它所需要贷款的金额，若小于等于，则予以贷款，若大于，则不予以贷款。

其代码如下所示：

```

function financing(string memory fromCompany, uint256 assetReceivableTo)
public returns (int256) {
    int256 retCode          = 0;
    bool    ret              = true;
    uint256 tempAssetReceivable = 0;

    // 对表进行查询
    (ret, tempAssetReceivable) = select(fromCompany);

    // 判断该公司是否存在

```

```

    if (ret == false) {
        // 该公司不存在
        retCode = -1;
    }
    else if (tempAssetReceivable < assetReceivableTo) {
        // 该公司的应收账款不足以申请该数目的贷款金额
        retCode = -2;
    }
    else {
        retCode = 0;
    }

    emit FinancingEvent(retCode, fromCompany, assetReceivableTo);

    return retCode;
}

```

其运行逻辑如下：

1. 查询贷款的公司
2. 如果该公司不存在，则返回错误码 -1
3. 如果其贷款的金额大于其应收账款，则返回错误码 -2
4. 若上述步骤都没有问题，则返回 0

2.2.5 结清账款

结清账款主要是核心公司向下游公司付清欠款，使其应收账款得到清空，其代码如下：

```

function settle(string memory fromCompany, string memory toCompany) public
returns (int256) {
    int256 retCode = 0;
    bool ret1 = true;
    bool ret2 = true;
    uint256 tempAssetReceivable1 = 0;
    uint256 tempAssetReceivable2 = 0;

    // 查询源公司是否在表中
    (ret1, tempAssetReceivable1) = select(fromCompany);
    // 查询目标公司是否在表中
    (ret2, tempAssetReceivable2) = select(toCompany);

    if (keccak256(bytes(fromCompany)) != keccak256(bytes(coreCompanyName))) {
        // 结清账款的源公司不是核心公司
        retCode = -1;
    }
    else if (ret1 == false) {
        // 结清账款的源公司不存在
        retCode = -2;
    }
    else if (ret2 == false) {
        // 结清账款的目标公司不存在
        retCode = -3;
    }
    else {
        // 创建数据项
        string memory tempAssetReceivable2Str = uint2str(0);
    }
}

```

```

        KVField memory kv =
KVField("asset_receivable", tempAssetReceivable2Str);
        KVField[] memory KVFields = new KVField[](1);
        KVFields[0] = kv;
        Entry memory entry = Entry(KVFields);

        // 更新目标公司的应收账款
        int256 count = tf.set("asset_r", toCompany, entry);

        if (count != 1) {
            // 对表的修改失败
            retCode = -4;
        }
        else {
            retCode = 0;
        }
    }

    emit SettleEvent(retCode, fromCompany, toCompany);

    return retCode;
}

```

其运行逻辑如下：

1. 查询源公司和目标公司
2. 如果源公司不是核心公司，则返回错误码 -1
3. 如果源公司不存在，则返回错误码 -2
4. 如果目标公司不存在，则返回错误码 -3
5. 如果上述步骤都没问题，则对目标公司的应收账款清零。
6. 如果上一步对表进行的清空操作中发生了错误，那么就返回错误码 -4
7. 如果上述几个步骤都没有问题，则返回错误码 0

2.3 Java 应用实现

在写好了智能合约以后，接下来，就是要通过 java 来实现我们的程序了。

第一步，我们需要把智能合约转换为 java 代码，如下所示：

```
bash contract2java.sh solidity -p org.fisco.bcos.supplyFinance.contract
```

接下来就是通过 Java 的 IDE 来编写客户端的代码，我们以其中一个例子来描述，其他的与此类似：

```

public void transferAsset(String fromCompanyAccount, String toCompanyAccount,
    BigInteger amount) {
    try {
        String contractAddress = loadCompanyAddr();
        SupplyChainFinance company = SupplyChainFinance.load(contractAddress,
client, cryptoKeyPair);
        TransactionReceipt receipt = company.transfer(fromCompanyAccount,
toCompanyAccount, amount);
        Tuple1<BigInteger> transferOutput = company.getTransferOutput(receipt);
        if (receipt.getStatus() == 0) {
            if (Objects.equals(transferOutput.getValue1(),
BigInteger.valueOf(0))) {

```

```

        System.out.printf(
            " transfer success => from_company: %s, to_company: %s,
amount: %s \n",
            fromCompanyAccount, toCompanyAccount, amount);
    }
    else if (Objects.equals(transferOutput.getValue1(),
BigInteger.valueOf(-1))) {
        System.out.printf(" transfer error, the from_company is core
company \n");
    }
    else if (Objects.equals(transferOutput.getValue1(),
BigInteger.valueOf(-2))) {
        System.out.printf(" transfer error, the from_company is not exist
\n");
    }
    else if (Objects.equals(transferOutput.getValue1(),
BigInteger.valueOf(-3))) {
        System.out.printf(" transfer error, the to_company is not exist
\n");
    }
    else if (Objects.equals(transferOutput.getValue1(),
BigInteger.valueOf(-4))) {
        System.out.printf(" transfer error, the amount is not enough
\n");
    }
    else if (Objects.equals(transferOutput.getValue1(),
BigInteger.valueOf(-5))) {
        System.out.printf(" transfer error, the amount is overflow \n");
    }
    else if (Objects.equals(transferOutput.getValue1(),
BigInteger.valueOf(-6))) {
        System.out.printf(" send failed, there're some error while
operating the table \n");
    }
    else {
        System.out.printf(" unknown error \n");
    }
} else {
    System.out.println(" receipt status is error, maybe transaction not
exec. status is: " + receipt.getStatus());
}
} catch (Exception e) {
    logger.error(" transferAsset exception, error message is {}",
e.getMessage());
    System.out.printf(" transfer asset account failed, error message is
%s\n", e.getMessage());
}
}
}

```

可以看到，其实实现起来非常简单，就是通过调用 `solidity` 转换后的函数接口来实现客户端的逻辑，比如我们要实现应收账款的转让，就调用 `transfer` 接口即可，之后用一个 `Tuple1<BigInteger>` 类型的变量来接受函数调用的返回值，根据返回值，将不同的结果展示在终端上。

3 功能测试

我将展示如何将我的智能合约部署到链上，并完成相关的调用：

首先，开启本地节点：

```
bash nodes/127.0.0.1/start_all.sh
```

其次，打开控制台：

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

将我们合约部署到链上：



```
node4 start successfully
# arnohu @ acer-Swift3 in ~/.fisco [17:36:40]
$ cd ~/fisco/console && bash start.sh
=====
Welcome to FISCO BCOS console(3.0.0-rc1)!
Type 'help' or 'h' for help. Type 'quit' or 'q' to quit console.

  $$$$$$\\$$$$$\\ $$$$$$\\ $$$$$$\\ $$$$$$\\ $$$$$$\\ $$$$$$\\ $$$$$$\\ $$$$$$\\ $$$$$$\\
  $$____| $$ | $$____\\| $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ |
  $$$\\ | $$ | $$$\\ \\| $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ |
  $$$$ | $$ | $$$$$$\\ $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ | $$ |
  | $$ | $$ | $$$\\ | $$ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ |
  | $$ | $$ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ |
  \\$$ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ | $$$\\ |

=====
[group]: /> deploy SupplyChainFinance
transaction hash: 0x5addf4b1edd4e7b4ff9d67877372ab93aa39d8366276342f2c9bed2fa7d8f1a1
contract address: 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D
currentAccount: 0x4975afccd83c8aa358159ca71c8e343fd186fc69
[group]: /> 
```

可以看到我们的合约地址为：

```
0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D
```

注册汽车公司(核心企业)，轮胎公司和轮毂公司，由于之前已经注册过那些公司了，现在如果重复注册的话会显示该公司已存在，故我们不再演示注册的部分。

核心公司向轮胎公司签发 1000 万的应收账款：


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

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D send carCompany tyreCompany 1000
transaction hash: 0x2e962ea964e3fc5a4d0aee78c533499b67efc1b7c608f4f57284d6dad75d32d8
-----
transaction status: 0
description: transaction executed successfully
-----
Receipt message: Success
Return message: Success
Return value size:1
Return types: (int256)
Return values:(0) ← 签发成功
-----
Event logs
Event: {"SendEvent":[[0]]}

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D select tyreCompany
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:2
Return types: (bool, uint256)
Return values:(true, 1000) ← 轮胎公司的应收账款为 1000 万
-----

[group]: /> 
```

轮胎公司转让应收欠款 250 万给轮毂公司：

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

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D transfer tyreCompany wheelCompany 250
transaction hash: 0x48a2657e41ef3b068d6d55d635874ee139cc373abbd5765d7247e1541ec49623
-----
transaction status: 0
description: transaction executed successfully
-----
Receipt message: Success
Return message: Success
Return value size:1
Return types: (int256)
Return values:(0) ← 转让成功
-----
Event logs
Event: {"TransferEvent":[[0]]}

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D select tyreCompany
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:2
Return types: (bool, uint256)
Return values:(true, 750) ← 轮胎公司应收账款为 750 万
-----

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D select wheelCompany
-----
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:2
Return types: (bool, uint256)
Return values:(true, 250) ← 轮毂公司应收账款为 250 万
-----

[group]: /> 
```

轮胎公司申请 1000 万的贷款，轮毂公司申请 250 万的贷款：

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

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D financing tyreCompany 1000
transaction hash: 0xb3d05fc6c1b5457eddd161d6f90129bc9073719c0f596cd57edd265c0728bbca
-----
transaction status: 0
description: transaction executed successfully
-----
Receipt message: Success
Return message: Success
Return value size:1
Return types: (int256)
Return values:(-2) ← 轮胎公司申请失败，因为它只有 750 万的应收账款，不能申请 1000 万的贷款
-----
Event logs
Event: {"FinancingEvent":[[-2]]}

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D financing wheelCompany 250
transaction hash: 0xa093f9bf782ca21621bba224a70338a41170d721c8b8c59419ec77e0f7485ada
-----
transaction status: 0
description: transaction executed successfully
-----
Receipt message: Success
Return message: Success
Return value size:1
Return types: (int256)
Return values:(0) ← 轮胎公司申请贷款成功
-----
Event logs
Event: {"FinancingEvent":[[0]]}

[group]: />
```

车企结清轮胎企业的欠款：

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

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D settle carCompany tyreCompany
transaction hash: 0xd40b2cfae4f41541530320a0d2d9f30c7ee973c631cc04d01137d4c10f4eee41
-----
transaction status: 0
description: transaction executed successfully
-----
Receipt message: Success
Return message: Success
Return value size:1
Return types: (int256)
Return values:(0) ← 结清成功
-----
Event logs
Event: {"SettleEvent":[[0]]}

[group]: /> call SupplyChainFinance 0x0dFEED1D9bC1BF2208FED6C3C00eE7aACf668F8D select tyreCompany
Return code: 0
description: transaction executed successfully
Return message: Success
-----
Return value size:2
Return types: (bool, uint256)
Return values:(true, 0) ← 轮胎公司不再有应收账款
-----
[group]: />
```

至此，我们的核心功能已经展示完毕，可以看到我们智能合约的编写是正确的。

4 页面展示

接下来，我们展示一下我们的软件的最后的版本，由于本人的个人能力有限，故我们的平台是基于终端实现的：

我们先将平台部署在区块链上：

```
bash finance_run.sh deploy
```

结果如下所示：

```
arnohu@acer-Swift3:~/fisco/supplyFinance-app-1.0/dist
# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:40:18]
$ bash finance_run.sh deploy
deploy Asset success, contract address is 0x2F4de204eDe2876817dAdC543F264c6B237B0110

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:41:10]
$
```

可以看到，我们部署成功了。

接下来我们让核心公司给轮胎公司发放 1000 万的应收账款：

```
arnohu@acer-Swift3:~/fisco/supplyFinance-app-1.0/dist
# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:42:53]
$ bash finance_run.sh send carCompany tyreCompany 1000
send success => from_company: carCompany, to_company: tyreCompany, amout: 1000

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:42:59]
$ bash finance_run.sh query tyreCompany
company tyreCompany, asset receivable: 1000

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:43:06]
$
```

然后我们让轮胎公司给轮毂公司转 250 万的应收账款：

```
arnohu@acer-Swift3:~/fisco/supplyFinance-app-1.0/dist
# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:43:06]
$ bash finance_run.sh transfer tyreCompany wheelCompany 250
transfer success => from_company: tyreCompany, to_company: wheelCompany, amount: 250

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:44:44]
$ bash finance_run.sh query tyreCompany
company tyreCompany, asset receivable: 750

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:44:52]
$ bash finance_run.sh query wheelCompany
company wheelCompany, asset receivable: 250

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:44:59]
$
```

之后让轮胎公司申请 1000 万的贷款，让轮毂公司申请 250 万的贷款：

```
arnohu@acer-Swift3:~/fisco/supplyFinance-app-1.0/dist
# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:53:05]
$ bash finance_run.sh financing tyreCompany 1000
financing error, the amount is bigger than asset receivable

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:53:20]
$ bash finance_run.sh financing wheelCompany 250
financing success => company wheelCompany get 250 from bank

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:53:32]
$
```

可以看到，轮胎公司申请失败了，但是轮毂公司申请成功了。

接下来，我们让车企结算轮胎公司的账款和轮毂公司的账款：

```
arnohu@acer-Swift3:~/fisco/supplyFinance-app-1.0/dist
# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:46:42]
$ bash finance_run.sh settle carCompany tyreCompany
settle success => company tyreCompany haven't asset receivable now

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:48:00]
$ bash finance_run.sh settle carCompany wheelCompany
settle success => company wheelCompany haven't asset receivable now

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:48:06]
$ bash finance_run.sh query tyreCompany
company tyreCompany, asset receivable: 0

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:48:13]
$ bash finance_run.sh query wheelCompany
company wheelCompany, asset receivable: 0

# arnohu @ acer-Swift3 in ~/fisco/supplyFinance-app-1.0/dist [18:48:22]
$
```

可以看到，两家下游公司的账款都被结算清楚了。

5 心得

本次大作业非常有意思，让我初步接触了智能合约，稍微了解了智能合约的编写。

遗憾的是，由于我前期的其他课程的作业量太大，导致我没能好好尽早地开始本次大作业的制作，只能在两天内草草的结束本次作业，使得本次作业制品看起来比较简陋。再加上本人实在是能力有限，做出来的成果只能勉强完成基本内容。

但是，我仍然在这个过程中学习到了很多知识，不光是智能合约的编写，还有我对区块链的理解，大大地提高了我对区块链的兴趣，使我产生了强烈的兴趣继续钻研下去。

一开始我比较疑惑，由于本人的学识有限，在本课程开始之前，我都一直认为比特币是骗局，以太坊是儿戏，但当自己真正地去了解了，才明白其背后的区块链的原理有多么伟大，特别是这次大作业的应用场景，让我了解了传统交易模式的弊端，也让我大概明白了新的区块链的交易模式的便捷之处，让我加深了对区块链的理解。

也很感谢这门课程，真的让我学习到了很多新的知识，让我在未来的人生道路有了更多的选择。
