问题1

电脑需要联网

1.1 环境版本说明

虚拟机系统：centos7
windows系统：win11
给虚拟机分配的内存大小：4G
给虚拟机分配的外存大小：40G
长安链版本：2.3.2
虚拟机中Golang版本：go version go1.16 linux/amd64
虚拟机中docker版本：23.0.1
远程连接工具下载地址Mobaxterm：https://mobaxterm.mobatek.net/download.html

问题2

2.1 给虚拟机配置静态ip地址

```
vim /etc/sysconfig/network-scripts/ifcfg-ens33
# 在/etc/sysconfig/network-scripts/ifcfg-ens33文件中修改以下6行
ONBOOT=yes
BOOTPROTO=static
IPADDR="192.168.31.128" # 保证ip地址范围和windows电脑的以太网适配器VMnet8的IP地址在同一个网段就行
NETMASK="255.255.255.0"
GATEWAY="192.168.30.2"
DNS1="223.5.5.5"
```


```
TYPE="Ethernet"
PROXY_METHOD="none"
BROWSER_ONLY="no"
#BOOTPROTO="dhcp"
BOOTPROTO="static"
IPADDR=192.168.30.128
NETMASK=255.255.255.0
GATEWAY=192.168.30.2
DNS1=223.5.5.5
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="yes"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
IPV6_ADDR_GEN_MODE="stable-privacy"
NAME="ens33"
UUID="45b6697a-4a9e-4ea2-91ba-c54d6ba2ad4e"
DEVICE="ens33"
ONBOOT="yes"
```

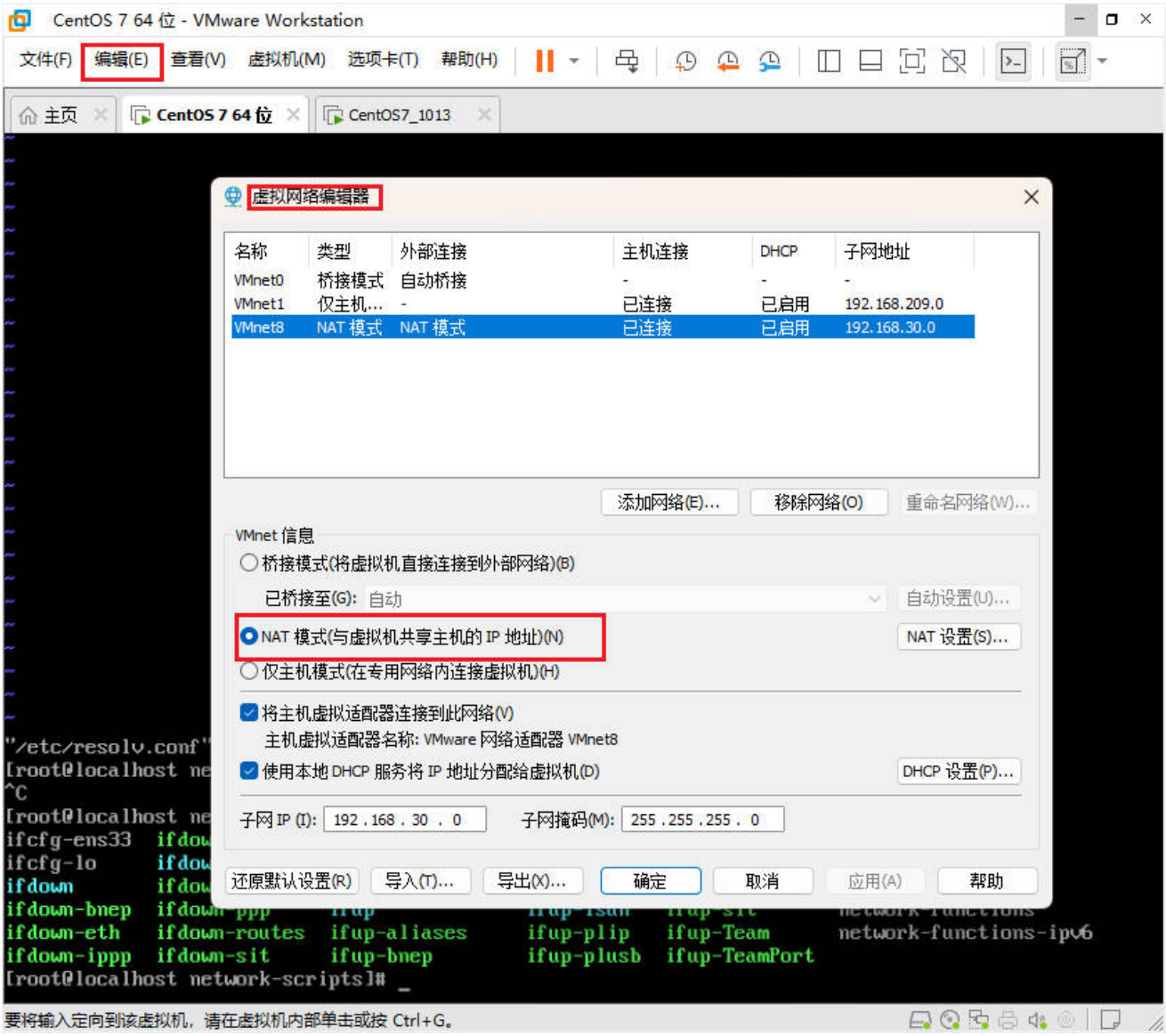
```
[root@localhost ~]# vim /etc/sysconfig/network-scripts/ifcfg-ens33
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:3b:c7:65 brd ff:ff:ff:ff:ff:ff
    inet 192.168.30.128/24 brd 192.168.30.255 scope global ens33
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe3b:c765/64 scope link
       valid_lft forever preferred_lft forever
3: br-8b18e47b803f: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:b3:a0:db:74 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-8b18e47b803f
       valid_lft forever preferred_lft forever
4: br-5973c068abca: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ba:f2:a7:e8 brd ff:ff:ff:ff:ff:ff
```

2.2 关闭防火墙

```
# 查看防火墙状态
systemctl status firewalld
# 停止防火墙
systemctl stop firewalld
# 禁用防火墙
systemctl disable firewalld
```

2.3 重启网络服务

```
systemctl network restart
```



问题3

3.1 ifconfig命令不存在

```
[root@localhost ~]# ifconfig
-bash: ifconfig: command not found
[root@localhost ~]#
```

3.2 运行yum -y install ifconfig命令后报错

```
[root@localhost ~]# yum -y install ifconfig
Loaded plugins: fastestmirror
Determining fastest mirrors
Could not retrieve mirrorlist http://mirrorlist.centos.org/?release=7&arch=x86_64&repo=os&infra=stock error was
14: curl#6 - "Could not resolve host: mirrorlist.centos.org; Unknown error"


 One of the configured repositories failed (Unknown),
 and yum doesn't have enough cached data to continue. At this point the only
 safe thing yum can do is fail. There are a few ways to work "fix" this:

     1. Contact the upstream for the repository and get them to fix the problem.

     2. Reconfigure the baseurl/etc. for the repository, to point to a working
        upstream. This is most often useful if you are using a newer
        distribution release than is supported by the repository (and the
        packages for the previous distribution release still work).

     3. Run the command with the repository temporarily disabled
            yum --disablerepo=<repoid> ...

     4. Disable the repository permanently, so yum won't use it by default. Yum
        will then just ignore the repository until you permanently enable it
        again or use --enablerepo for temporary usage:

            yum-config-manager --disable <repoid>
        or
            subscription-manager repos --disable=<repoid>

     5. Configure the failing repository to be skipped, if it is unavailable.
        Note that yum will try to contact the repo. when it runs most commands,
        so will have to try and fail each time (and thus. yum will be be much
        slower). If it is a very temporary problem though, this is often a nice
        compromise:

            yum-config-manager --save --setopt=<repoid>.skip_if_unavailable=true

Cannot find a valid baseurl for repo: base/7/x86_64
```

3.3 使用yum install wget命令失败

```
# 更新yum源
curl -o /etc/yum.repos.d/CentOS-Base.repo
http://mirrors.aliyun.com/repo/Centos-7.repo
# 安装wget
yum -y install wget
```

3.4 清除缓存更新yum列表

```
yum clean all
yum makecache
yum update
```

3.5 安装ifconfig

```
yum install net-tools
```

问题 4

4.1 使用git报错

```
[root@localhost yum.repos.d]# git
-bash: git: command not found
[root@localhost yum.repos.d]# git version
-bash: git: command not found
[root@localhost yum.repos.d]#
```

4.2 安装相关依赖

```
yum install curl-devel expat-devel gettext-devel openssl-devel
zlib-devel
yum install gcc-c++ perl-ExtUtils-MakeMaker
```

4.3 下载git源码包

```
# 下载链接
https://github.com/git/git/releases/tag/v2.42.0
```

4.4 安装git

```
# 解压git安装包到指定目录/usr/local/environment/git
tar zxvf git-2.38.1.tar.gz -C /usr/local/environment/git-2.27.0
# 配置安装路径为/usr/local/environment/git
make prefix=/usr/local/environment/git-2.27.0 all
make prefix=/usr/local/environment/git-2.27.0 install
# 查看git版本
git --vesion
```

```
[root@localhost environment]# git --version
git version 1.8.3.1
[root@localhost environment]#
```

4.5 注意点

这里可能出现一个问题，如果你之前已经安装过git了，版本比较低，这里展示的可能就是你之前的版本，因为系统默认是使用/usr/bin/git下的git，这时候如果想使用你安装的最新版的git，那么操作如下

```
# 配置环境变量
vim /etc/profile
# 在文件末尾添加这两行
export GIT_HOME=/usr/local/environment/git-2.27.0
export PATH=.:$GIT_HOME/bin:$PATH
# 使文件生效
source /etc/profile
# 查看版本号
git --version
```

```
[root@localhost environment]# vim /etc/profile
[root@localhost environment]# source /etc/profile
[root@localhost environment]# git --version
git version 2.27.0
```

问题5

5.1 使用yum install go命令报错

```
[root@localhost environment]# yum install go
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.aliyun.com
 * extras: mirrors.aliyun.com
 * updates: mirrors.aliyun.com
No package go available.
Error: Nothing to do
[root@localhost environment]# yum install golang
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.aliyun.com
 * extras: mirrors.aliyun.com
 * updates: mirrors.aliyun.com
No package golang available.
Error: Nothing to do
```

5.2 配置go的安装源

```
rpm --import https://mirror.go-repo.io/centos/RPM-GPG-KEY-GO-REPO
curl -s https://mirror.go-repo.io/centos/go-repo.repo | tee
/etc/yum.repos.d/go-repo.repo
```

```
[root@localhost environment]# rpm --import https://mirror.go-repo.io/centos/RPM-GPG-KEY-GO-REPO
[root@localhost environment]# curl -s https://mirror.go-repo.io/centos/go-repo.repo | tee /etc/yum.repos.d/go-repo.repo
[go-repo]
name=go-repo - CentOS
baseurl=https://mirror.go-repo.io/centos/$releasever/$basearch/
enabled=1
gpgcheck=1
gpgkey=https://mirror.go-repo.io/centos/RPM-GPG-KEY-GO-REPO
```

5.3 执行yum install go

```
# 查询golang的可用版本（建议安装1.16版本）
yum --showduplicates list golang
yum install golang-1.16.1-0.el7

# 查看go版本
go version
```

```
[root@localhost environment]# go version
go version go1.21.3 linux/amd64
[root@localhost environment]#
```

5.4 通过go源码包安装（建议通过这种安装方式）

```
wget https://studygolang.com/dl/golang/go1.16.3.linux-amd64.tar.gz
tar zxvf go1.16.3.linux-amd64.tar.gz
vim /etc/profile
# 在文件/etc/profile中添加以下内容
export GOROOT=/usr/local/environment/go
export GOBIN=$GOROOT/bin
export GOPATH=/usr/local/environment/go_project
export PATH=.:$GOBIN:$GOPATH:$PATH
# 使添加的内容生效
source /etc/profile
```

5.4 配置代理

```
go env -w GO111MODULE=on
go env -w GOPROXY=https://goproxy.cn,direct
```

配置前

```
[root@localhost ~]# go env
GO111MODULE=''
GOARCH='amd64'
GOBIN=''
GOCACHE='/root/.cache/go-build'
GOENV='/root/.config/go/env'
GOEXE=''
GOEXPERIMENT=''
GOFLAGS=''
GOHOSTARCH='amd64'
GOHOSTOS='linux'
GOINSECURE=''
GOMODCACHE='/root/go/pkg/mod'
GONOPROXY=''
GONOSUMDB=''
GOOS='linux'
GOPATH='/root/go'
GOPRIVATE=''
GOPROXY=''
GOROOT='/usr/lib/golang'
GOSUMDB=''
GOTMPDIR=''
GOTOOLCHAIN=''
```

配置后

```
[root@localhost scripts]# go env
GO111MODULE="on"
GOARCH="amd64"
GOBIN=""
GOCACHE="/root/.cache/go-build"
GOENV="/root/.config/go/env"
GOEXE=""
GOFLAGS=""
GOHOSTARCH="amd64"
GOHOSTOS="linux"
GOINSECURE=""
GOMODCACHE="/usr/local/environment/gopath/pkg/mod"
GONOPROXY=""
GONOSUMDB=""
GOOS="linux"
GOPATH="/usr/local/environment/gopath"
GOPRIVATE=""
GOPROXY="https://goproxy.cn,direct"
GOROOT="/usr/local/environment/go"
GOSUMDB="sum.golang.org"
GOTMPDIR=""
GOTOOLDIR="/usr/local/environment/go/pkg/tool/linux_amd64"
GOVCS=""
GOVERSION="go1.16"
GCCGO="gccgo"
AR="ar"
```

问题6

6.1 检查gcc版本

```
1 gcc --version
```

```
[root@localhost environment]# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/4.8.5/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-
bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-shared --enable-threads=posix --enable-check
with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --en
ild-id --with-linker-hash-style=gnu --enable-languages=c,c++,objc,obj-c++,java,fortran,ada,go,lto --e
-enable-initfini-array --disable-libgcj --with-isl=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_6
/isl-install --with-cloog=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/cloog-inst
nu-indirect-function --with-tune=generic --with-arch_32=x86-64 --build=x86_64-redhat-linux
Thread model: posix
gcc version 4.8.5 20150623 (Red Hat 4.8.5-44) (GCC)
[root@localhost environment]# yum install centos-release-scl
Loaded plugins: fastestmirror
Existing lock /var/run/yum.pid: another copy is running as pid 82090.
Another app is currently holding the yum lock; waiting for it to exit...
  The other application is: yum
    Memory :  99 M RSS (499 MB VSZ)
```

6.2 升级gcc

```
yum install centos-release-scl
yum install devtoolset-7-gcc*

vim ~/.bash_profile
# 在文件~/.bash_profile中添加一行，使得下次打开新窗口也能生效
scl enable devtoolset-7 bash
# 查看gcc 版本
gcc -v
```

```
[root@localhost ~]# gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/opt/rh/devtoolset-7/root/usr/libexec/gcc/x86_64-redhat-linux/7/lto-wrapper
Target: x86_64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,lto --prefix=/opt/rh
/devtoolset-7/root/usr --mandir=/opt/rh/devtoolset-7/root/usr/share/man --infodir=/opt/rh/devtoolset-7
/root/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-thread
s=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable
-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-onl
y --enable-plugin --with-linker-hash-style=gnu --enable-initfini-array --with-default-libstdcxx-abi=gc
c4-compatible --with-isl=/builddir/build/BUILD/gcc-7.3.1-20180303/obj-x86_64-redhat-linux/isl-install
--enable-libmpx --enable-gnu-indirect-function --with-tune=generic --with-arch_32=i686 --build=x86_64-
redhat-linux
Thread model: posix
gcc version 7.3.1 20180303 (Red Hat 7.3.1-5) (GCC)
```

问题 7

7.1 查看当前系统的内核版本（建议3.10以上）

```
uname -r
```

```
[root@localhost ~]# uname -r
3.10.0-1160.el7.x86_64
```

7.2 安装相关依赖

```
yum install -y yum-utils device-mapper-persistent-data lvm2
```

7.3 设置yum源

```
yum-config-manager --add-repo
http://download.docker.com/linux/centos/docker-ce.repo（中央仓库）

yum-config-manager --add-repo http://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo（阿里仓库）
```

7.4 查看可选择的docker版本

```
yum list docker-ce --showduplicates | sort -r
```

7.5 安装docker

```
# 选择18.03.1这个版本进行安装
yum -y install docker-ce-3:23.0.1-1.el7
# 查看安装的docker版本
docker -v 或者 docker version

# 卸载docker
yum remove docker-ce docker-ce-cli containerd.io
rm -rf /var/lib/docker
rm -rf /var/lib/containerd
```

```
[root@localhost ~]# docker -v
Docker version 23.0.1, build a5ee5b1
[root@localhost ~]# docker version
Client: Docker Engine - Community
 Version:           23.0.1
 API version:       1.42
 Go version:        go1.19.5
 Git commit:        a5ee5b1
 Built:             Thu Feb  9 19:51:00 2023
 OS/Arch:           linux/amd64
 Context:           default

Server: Docker Engine - Community
 Engine:
  Version:          23.0.1
  API version:      1.42 (minimum version 1.12)
  Go version:       go1.19.5
  Git commit:       bc3805a
  Built:            Thu Feb  9 19:48:42 2023
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.18
  GitCommit:        2456e983eb9e37e47538f59ea18f2043c9a73640
 runc:
  Version:          1.1.4
  GitCommit:        v1.1.4-0-g5fd4c4d
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

7.6 启动docker

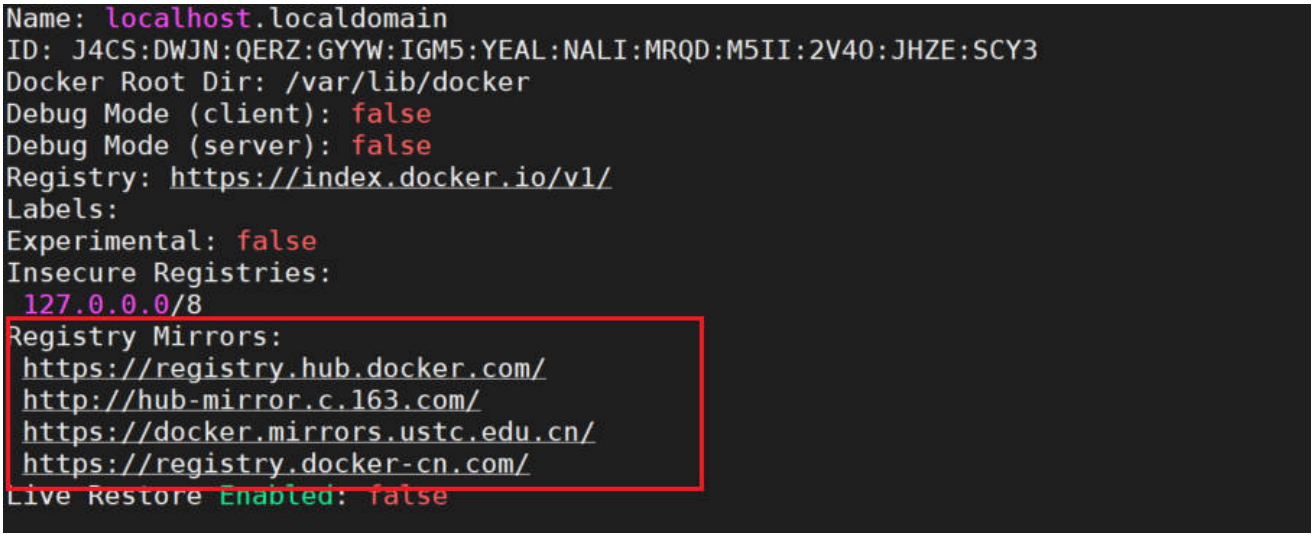
```
# 启动docker
systemctl start docker
# 查看docker当前状态
systemctl status docker
# 设置开机自启
systemctl enable docker
```

7.7 配置docker镜像源

```
# 查看docker的镜像源
docker info
# 在目录/etc/docker下，如果没有daemon.json文件，则新建daemon.json文件，文件内容如下
{
        "registry-mirrors": ["https://registry.hub.docker.com",
                             "http://hub-mirror.c.163.com",
                             "https://docker.mirrors.ustc.edu.cn",
                             "https://registry.docker-cn.com"]
}
```

配置前

配置后



7.8 重载守护进程，重启docker

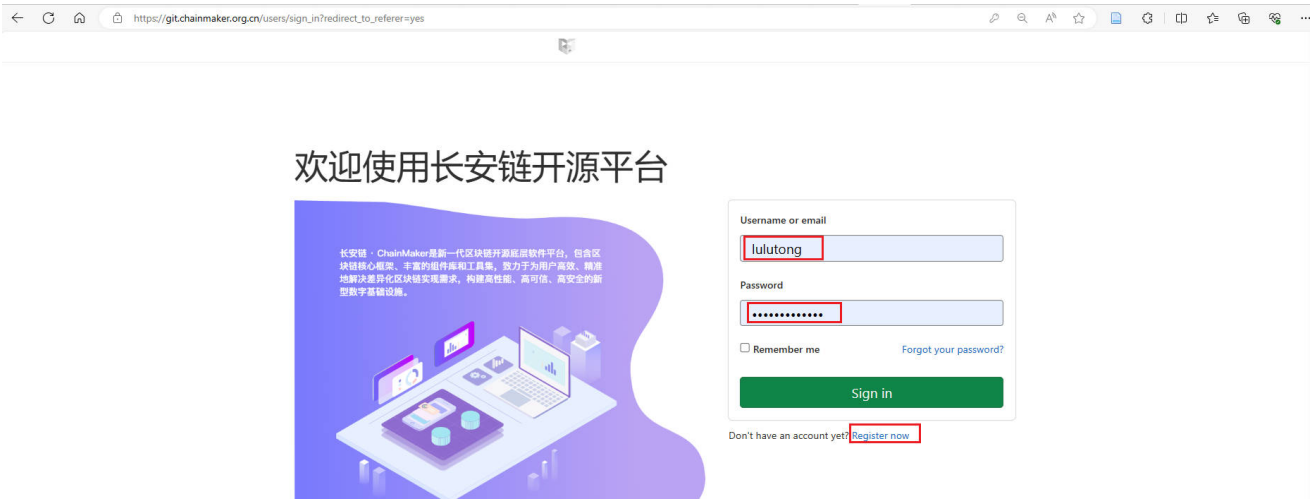
```
systemctl daemon-reload
systemctl stop docker
systemctl start docker
```

问题8

8.1 使用命令行部署长安链

（参考的官方文档链接https://docs.chainmaker.org.cn/v2.3.2/html/quickstart/%E9%80%9A%E8%BF%87%E5%91%BD%E4%BB%A4%E8%A1%8C%E4%BD%93%E9%AA%8C%E9%93%BE.html）

8.1.1 账号注册



8.1.2 源码下载

```
# 下载chainmaker-go源码到本地
git clone -b v2.3.2 --depth=1 https://git.chainmaker.org.cn/chainmaker/chainmaker-go.git
# 下载证书生成工具源码到本地
git clone -b v2.3.0  --depth=1 https://git.chainmaker.org.cn/chainmaker/chainmaker-cryptogen.git
```

```
[root@localhost home]# ls
chainv2.2.1  chainv2.3  experiment
[root@localhost home]# mkdir chainv2.3.2
[root@localhost home]# ls
chainv2.2.1  chainv2.3  chainv2.3.2  experiment
[root@localhost home]# cd chainv2.3.2/
[root@localhost chainv2.3.2]# ls
[root@localhost chainv2.3.2]# git clone -b v2.3.2 --depth=1 https://git.chainmaker.org.cn/chainmaker/chainmaker-go.git
Cloning into 'chainmaker-go'...
Username for 'https://git.chainmaker.org.cn': lulutong
Password for 'https://lulutong@git.chainmaker.org.cn':
remote: Enumerating objects: 1632, done.
remote: Counting objects: 100% (1632/1632), done.
remote: Compressing objects: 100% (1258/1258), done.
remote: Total 1632 (delta 380), reused 1101 (delta 213), pack-reused 0
Receiving objects: 100% (1632/1632), 19.36 MiB | 435.00 KiB/s, done.
Resolving deltas: 100% (380/380), done.
Note: switching to '0231a39d414bfd3af223759e4d4e9730090e479f'.
```

需要提前在长安链官网注册账户

```
[root@localhost chainv2.3.2]# ls
chainmaker-go
[root@localhost chainv2.3.2]# git clone -b v2.3.0  --depth=1 https://git.chainmaker.org.cn/chainmaker/chainmaker-cryptogen.git
Cloning into 'chainmaker-cryptogen'...
remote: Enumerating objects: 37, done.
remote: Counting objects: 100% (37/37), done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 37 (delta 9), reused 32 (delta 7), pack-reused 0
Unpacking objects: 100% (37/37), 51.35 KiB | 122.00 KiB/s, done.
[root@localhost chainv2.3.2]# ls
chainmaker-cryptogen  chainmaker-go
```

8.1.3 源码编译

```
# 编译证书生成工具(这个步骤需要等待的时间较长)
cd chainmaker-cryptogen
make
```

```
[root@localhost chainv2.3.2]# cd chainmaker-cryptogen/
[root@localhost chainmaker-cryptogen]# make
chainmaker-cryptogen/
chainmaker-cryptogen/bin/
chainmaker-cryptogen/bin/chainmaker-cryptogen
chainmaker-cryptogen/config/
chainmaker-cryptogen/config/crypto_config_template.yml
[root@localhost chainmaker-cryptogen]# ls
bin  config  LICENSE  Makefile  NOTICE  README.md  release  src
```

8.1.4 配置文件生成

```
# 将编译好的chainmaker-cryptogen，软连接到chainmaker-go/tools目录
## 进入工具目录
cd chainmaker-go/tools
## 软连接chainmaker-cryptogen到tools目录下
ln -s ../../chainmaker-cryptogen/ .

# 进入chainmaker-go/scripts目录，执行prepare.sh脚本生成单链4节点集群配置，存于路径chainmaker-go/build中
## 进入脚本目录
```

```
cd ../scripts
## 生成单链4节点集群的证书和配置
./prepare.sh 4 1
# 查看生成好的节点证书和配置
yum install tree
tree -L 3 ../build/
```

```
[root@localhost chainv2.3.2]# ls
chainmaker-cryptogen  chainmaker-go
[root@localhost chainv2.3.2]# cd chainmaker-go/tools
[root@localhost tools]# ln -s ../../chainmaker-cryptogen/ .
[root@localhost tools]# cd ../scripts
[root@localhost scripts]# ./prepare.sh 4 1
begin check params...
param P2P_PORT 11301
param RPC_PORT 12301
param VM_GO_RUNTIME_PORT 32351
param VM_GO_ENGINE_PORT 22351
input consensus type (1-TBFT(default),3-MAXBFT,4-RAFT):
param CONSENSUS_TYPE 1
input log level (DEBUG|INFO(default)|WARN|ERROR):
param LOG_LEVEL INFO
enable vm go (YES|NO(default))YES
vm go transport protocol (uds|tcp(default))
input vm go log level (DEBUG|INFO(default)|WARN|ERROR):
param VM_GO_LOG_LEVEL INFO
param VM_GO_TRANSPORT_PROTOCOL tcp
param ENABLE_VM_GO true

config node total 4
begin generate node1 config...
begin node1 chain1 cert config...
begin node1 trust config...
begin generate node2 config...
```

```
[root@localhost scripts]# ls
bin                     cluster_quick_stop.sh  prepare_pk.sh    range_cluster_quick_start.sh
build_release.sh        docker                 prepare_pwk.sh   service
cluster_quick_start.sh  gomod_update.sh        prepare.sh       test
[root@localhost scripts]# tree -L 3 ../build/
../build/
├── config
│   ├── node1
│   │   ├── certs
│   │   ├── chainconfig
│   │   ├── chainmaker.yml
│   │   └── log.yml
│   ├── node2
│   │   ├── certs
│   │   ├── chainconfig
│   │   ├── chainmaker.yml
│   │   └── log.yml
│   ├── node3
│   │   ├── certs
│   │   ├── chainconfig
│   │   ├── chainmaker.yml
│   │   └── log.yml
│   └── node4
│       ├── certs
│       ├── chainconfig
│       ├── chainmaker.yml
│       └── log.yml
```

8.1.5 编译及安装包制作

```
# 生成证书（prepare.sh脚本）后执行build_release.sh脚本，将编译chainmaker-go模块，并打包生成安装，存于路径chainmaker-go/build/release中(这个步骤需要等待的时间较长，而且大概率会报错)
./build_release.sh
tree ../build/release/
```

```
[root@localhost scripts]# ./build_release.sh
build chainmaker /home/chainv2.3.2/chainmaker-go...
build for linux or mac
warning: ignoring symlink /home/chainv2.3.2/chainmaker-go/tools/chainmaker-cryptogen
tar zcf crypto-config...
tar: Removing leading `../' from member names
tar zcf chainmaker-v2.3.2-wx-org1.chainmaker.org...
tar zcf chainmaker-v2.3.2-wx-org2.chainmaker.org...
tar zcf chainmaker-v2.3.2-wx-org3.chainmaker.org...
tar zcf chainmaker-v2.3.2-wx-org4.chainmaker.org...
wait tar...
[root@localhost scripts]# tree ../build/release/
../build/release/
├── chainmaker-v2.3.2-wx-org1.chainmaker.org-20231017111031-x86_64.tar.gz
├── chainmaker-v2.3.2-wx-org2.chainmaker.org-20231017111031-x86_64.tar.gz
├── chainmaker-v2.3.2-wx-org3.chainmaker.org-20231017111031-x86_64.tar.gz
├── chainmaker-v2.3.2-wx-org4.chainmaker.org-20231017111031-x86_64.tar.gz
└── crypto-config-20231017111031.tar.gz

0 directories, 5 files
```

8.1.6 启动节点集群

```
# 执行cluster_quick_start.sh脚本，会解压各个安装包，调用bin目录中的start.sh脚本，启动chainmaker节点
./cluster_quick_start.sh normal
# 启动成功后，将*.tar.gz备份，以免下次启动再次解压缩时文件被覆盖
mkdir -p ../build/bak
mv ../build/release/*.tar.gz ../build/bak
# 若需要关闭集群，使用脚本：
./cluster_quick_stop.sh
```

```
[root@localhost scripts]# ./cluster_quick_start.sh normal
===> Unzip chainmaker installation package
chainmaker-v2.3.2-wx-org1.chainmaker.org/
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/chainmaker
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/start.sh
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/stop.sh
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/restart.sh
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/version.sh
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/docker-vm-standalone-start.sh
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/docker-vm-standalone-stop.sh
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/chainmaker.service
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/init.sh
chainmaker-v2.3.2-wx-org1.chainmaker.org/bin/run.sh
chainmaker-v2.3.2-wx-org1.chainmaker.org/lib/
chainmaker-v2.3.2-wx-org1.chainmaker.org/lib/libwasmer.so
chainmaker-v2.3.2-wx-org1.chainmaker.org/lib/wxdec
chainmaker-v2.3.2-wx-org1.chainmaker.org/config/
chainmaker-v2.3.2-wx-org1.chainmaker.org/config/wx-org1.chainmaker.org/
chainmaker-v2.3.2-wx-org1.chainmaker.org/config/wx-org1.chainmaker.org/certs/
chainmaker-v2.3.2-wx-org1.chainmaker.org/config/wx-org1.chainmaker.org/certs/ca/
```

```
===> Starting chainmaker cluster
START ==>  /home/chainv2.3.2/chainmaker-go/build/release/chainmaker-v2.3.2-wx-org1.chainmaker.org
start docker vm service container succeed: VM-GO-wx-org1.chainmaker.org-name-20231017111104
chainmaker is starting, pls check log...
START ==>  /home/chainv2.3.2/chainmaker-go/build/release/chainmaker-v2.3.2-wx-org2.chainmaker.org
start docker vm service container succeed: VM-GO-wx-org2.chainmaker.org-name-20231017111105
chainmaker is starting, pls check log...
START ==>  /home/chainv2.3.2/chainmaker-go/build/release/chainmaker-v2.3.2-wx-org3.chainmaker.org
start docker vm service container succeed: VM-GO-wx-org3.chainmaker.org-name-20231017111106
chainmaker is starting, pls check log...
START ==>  /home/chainv2.3.2/chainmaker-go/build/release/chainmaker-v2.3.2-wx-org4.chainmaker.org
start docker vm service container succeed: VM-GO-wx-org4.chainmaker.org-name-20231017111107
chainmaker is starting, pls check log...
[root@localhost scripts]# mkdir -p ../build/bak
[root@localhost scripts]# mv ../build/release/*.tar.gz ../build/bak
```

8.1.7 查看节点启动使用正常

```
# 查看节点启动使用正常

## 查看进程是否存在
ps -ef |grep chainmaker | grep -v grep
## 查看端口是否监听
netstat -lptn | grep 1230
```

```
## 检查节点是否有ERROR日志
cat ../build/release/*/bin/panic.log
cat ../build/release/*/log/system.log # 这个日志文件的内容较多
cat ../build/release/*/log/system.log |grep "ERROR\|put block\|all necessary"
==如果看到all necessary peers connected则表示节点已经准备就绪。==

# 关闭某个进程占用的端口
netstat -anp | grep 8080
kill -9 进程号(比如23487)
```

```
[root@localhost scripts]# ps -ef|grep chainmaker | grep -v grep
root       2566      1  2 11:25 pts/0    00:00:26 ./chainmaker start -c ../config/wx-org1.chainmaker.org/chainmaker.yml name-20231017111104
root       3773      1  1 11:26 pts/0    00:00:18 ./chainmaker start -c ../config/wx-org2.chainmaker.org/chainmaker.yml name-20231017111105
root       4952      1  1 11:26 pts/0    00:00:17 ./chainmaker start -c ../config/wx-org3.chainmaker.org/chainmaker.yml name-20231017111106
root       6164      1  1 11:26 pts/0    00:00:18 ./chainmaker start -c ../config/wx-org4.chainmaker.org/chainmaker.yml name-20231017111107
root      70363  70343  0 08:25 ?        00:00:32 ./chainmaker-explorer.bin -config ../configs/
[root@localhost scripts]# netstat -lptn | grep 1230
tcp6       0      0 :::12301                :::*                    LISTEN      2566/./chainmaker
tcp6       0      0 :::12302                :::*                    LISTEN      3773/./chainmaker
tcp6       0      0 :::12303                :::*                    LISTEN      4952/./chainmaker
tcp6       0      0 :::12304                :::*                    LISTEN      6164/./chainmaker
[root@localhost scripts]# cat ../build/release/*/bin/panic.log
using crypto CryptoEngine = tjfoc
using crypto CryptoEngine = tjfoc
using crypto CryptoEngine = tjfoc
using crypto CryptoEngine = tjfoc
using crypto CryptoEngine = tjfoc
using crypto CryptoEngine = tjfoc
using crypto CryptoEngine = tjfoc
using crypto CryptoEngine = tjfoc
[root@localhost scripts]# cat ../build/release/*/log/system.log
2023-10-17 11:25:58.840 [INFO]  [Blockchain]    blockchain/chainmaker_server.go:90      load net provider: LibP2P
2023-10-17 11:25:58.841 [INFO]  [Blockchain]    blockchain/chainmaker_server.go:112     load net tls key file path: /home/chainv2.3.2/chainmaker-go/build/release/chainmaker-v2.3.2-wx-org1.chainmaker.org/config/wx-org1.chainm
```

```
2023-10-17 11:46:26.585 [INFO]  [Consensus] @chain1     v2@v2.3.3/tbft_status_broadcaster.go:194        [QmW5scTFvfYPqzu8YpYqGocEbpXvfU4N4P1anNmFREsMef](1/27/PROPOSE) send state to [QmcgjQUngLu6SmTMSAchvDhjCguryEPzEK7Cu3jXUzNJxG](1/27/PROPOSE)
2023-10-17 11:46:26.585 [INFO]  [Consensus] @chain1     v2@v2.3.3/tbft_status_broadcaster.go:194        [QmW5scTFvfYPqzu8YpYqGocEbpXvfU4N4P1anNmFREsMef](1/27/PROPOSE) send state to [QmNa5xTUyZhEjZpwjch1qJtq3QzfY7dEdFKTcG3LwU3LmU](1/27/PROPOSE)
[root@localhost scripts]# cat ../build/release/*/log/system.log |grep "ERROR\|put block\|all necessary"
2023-10-17 11:25:59.428 [INFO]  [Storage] @chain1       v2@v2.3.4/blockstore_impl.go:304        chain[chain1]: put block[0] hash[e6b5483e9666c087eed2a085bfdd1059b043f08f21d4cdd4ec6ea10b9b3dc8bd] (txs:1 bytes:19445),
2023-10-17 11:26:14.470 [INFO]  [Net]   libp2pnet/libp2p_connection_supervisor.go:122   [ConnSupervisor] all necessary peers connected.
2023-10-17 11:26:03.484 [INFO]  [Storage] @chain1       v2@v2.3.4/blockstore_impl.go:304        chain[chain1]: put block[0] hash[e6b5483e9666c087eed2a085bfdd1059b043f08f21d4cdd4ec6ea10b9b3dc8bd] (txs:1 bytes:19445),
2023-10-17 11:26:13.524 [INFO]  [Net]   libp2pnet/libp2p_connection_supervisor.go:122   [ConnSupervisor] all necessary peers connected.
2023-10-17 11:26:07.536 [INFO]  [Storage] @chain1       v2@v2.3.4/blockstore_impl.go:304        chain[chain1]: put block[0] hash[e6b5483e9666c087eed2a085bfdd1059b043f08f21d4cdd4ec6ea10b9b3dc8bd] (txs:1 bytes:19445),
2023-10-17 11:26:12.598 [INFO]  [Net]   libp2pnet/libp2p_connection_supervisor.go:122   [ConnSupervisor] all necessary peers connected.
2023-10-17 11:26:11.416 [INFO]  [Storage] @chain1       v2@v2.3.4/blockstore_impl.go:304        chain[chain1]: put block[0] hash[e6b5483e9666c087eed2a085bfdd1059b043f08f21d4cdd4ec6ea10b9b3dc8bd] (txs:1 bytes:19445),
2023-10-17 11:26:16.475 [INFO]  [Net]   libp2pnet/libp2p_connection_supervisor.go:122   [ConnSupervisor] all necessary peers connected.
[root@localhost scripts]#
```

8.2 使用CMC命令行工具部署、调用合约

为了验证所搭建的链功能是否正常，可以通过`cmc`命令行工具来进行验证

8.2.1 编译及配置

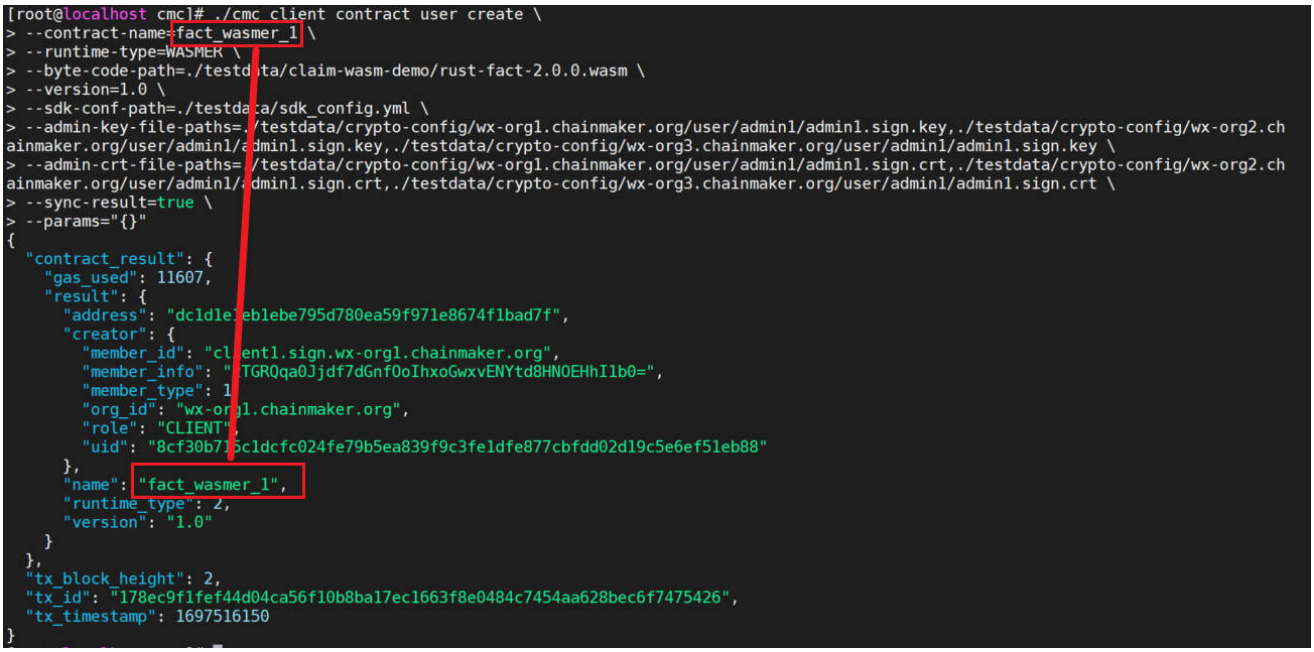
```
# 编译cmc
cd /home/chainv2.3.2/chainmaker-go/tools/cmc
go build
# 配置测试数据
cp -rf ../../build/crypto-config ../../tools/cmc/testdata/ # 使用chainmaker-cryptogen生成的测试链的证书
# 查看help
./cmc --help
```

```
[root@localhost chainv2.3.2]# cd chainmaker-go/
[root@localhost chainmaker-go]# cd tools/cmc
[root@localhost cmc]# pwd
/home/chainv2.3.2/chainmaker-go/tools/cmc
[root@localhost cmc]# go build
[root@localhost cmc]# cp -rf ../../build/crypto-config ../../tools/cmc/testdata/
[root@localhost cmc]#
bash: cd: ../../chain
[root@localhost cmc]# ./cmc --help
Command line interface for interacting with ChainMaker daemon.
For detailed logs, please see ./sdk.log
```

8.2.2 部署示例合约1（wasm类型的合约）

创建wasm类型的示例合约fact_wasmer_1

```
./cmc client contract user create \
--contract-name=fact_wasmer_1 \
--runtime-type=WASMER \
--byte-code-path=./testdata/claim-wasm-demo/rust-fact-2.0.0.wasm \
--version=1.0 \
--sdk-conf-path=./testdata/sdk_config.yml \
--admin-key-file-paths=./testdata/crypto-config/wx-org1.chainmaker.org/user/admin1/admin1.sign.key,./testdata/crypto-config/wx-org2.chainmaker.org/user/admin1/admin1.sign.key,./testdata/crypto-config/wx-org3.chainmaker.org/user/admin1/admin1.sign.key \
--admin-crt-file-paths=./testdata/crypto-config/wx-org1.chainmaker.org/user/admin1/admin1.sign.crt,./testdata/crypto-config/wx-org2.chainmaker.org/user/admin1/admin1.sign.crt,./testdata/crypto-config/wx-org3.chainmaker.org/user/admin1/admin1.sign.crt \
--sync-result=true \
--params="{}"
```



调用wasm类型的示例合约fact_wasmer_1

```
./cmc client contract user invoke \
--contract-name=fact_wasmer_1 \
--method=save \
--sdk-conf-path=./testdata/sdk_config.yml \
--params="
{\"file_name\":\"name007\",\"file_hash\":\"ab3456df5799b87c77e7f88
\",\"time\":\"6543234\"}" \
--sync-result=true
```

```
[root@localhost cmc]# ./cmc client contract user invoke \
> --contract-name=fact_wasmer_1 \
> --method=save \
> --sdk-conf-path=./testdata/sdk_config.yml \
> --params="{\"file_name\":\"name007\",\"file_hash\":\"ab3456df5799b87c77e7f88\",\"time\":\"6543234\"}" \
> --sync-result=true
{
  "contract_result": {
    "contract_event": [
      {
        "contract_name": "fact_wasmer_1",
        "contract_version": "1.0",
        "event_data": [
          "ab3456df5799b87c77e7f88",
          "name007",
          "6543234"
        ],
        "topic": "topic_vx",
        "tx_id": "178eca209c9dc46dca91c130fa8bc01de12cafdcee234abaae480e04eba25afd"
      }
    ],
    "gas_used": 238549
  },
  "tx_block_height": 3,
  "tx_id": "178eca209c9dc46dca91c130fa8bc01de12cafdcee234abaae480e04eba25afd",
  "tx_timestamp": 1697516350
}
```

查询合约fact_wasmer_1中的方法find_by_file_hash

```
./cmc client contract user get \
--contract-name=fact_wasmer_1 \
--method=find_by_file_hash \
--sdk-conf-path=./testdata/sdk_config.yml \
--params="{\"file_hash\":\"ab3456df5799b87c77e7f88\"}"
```

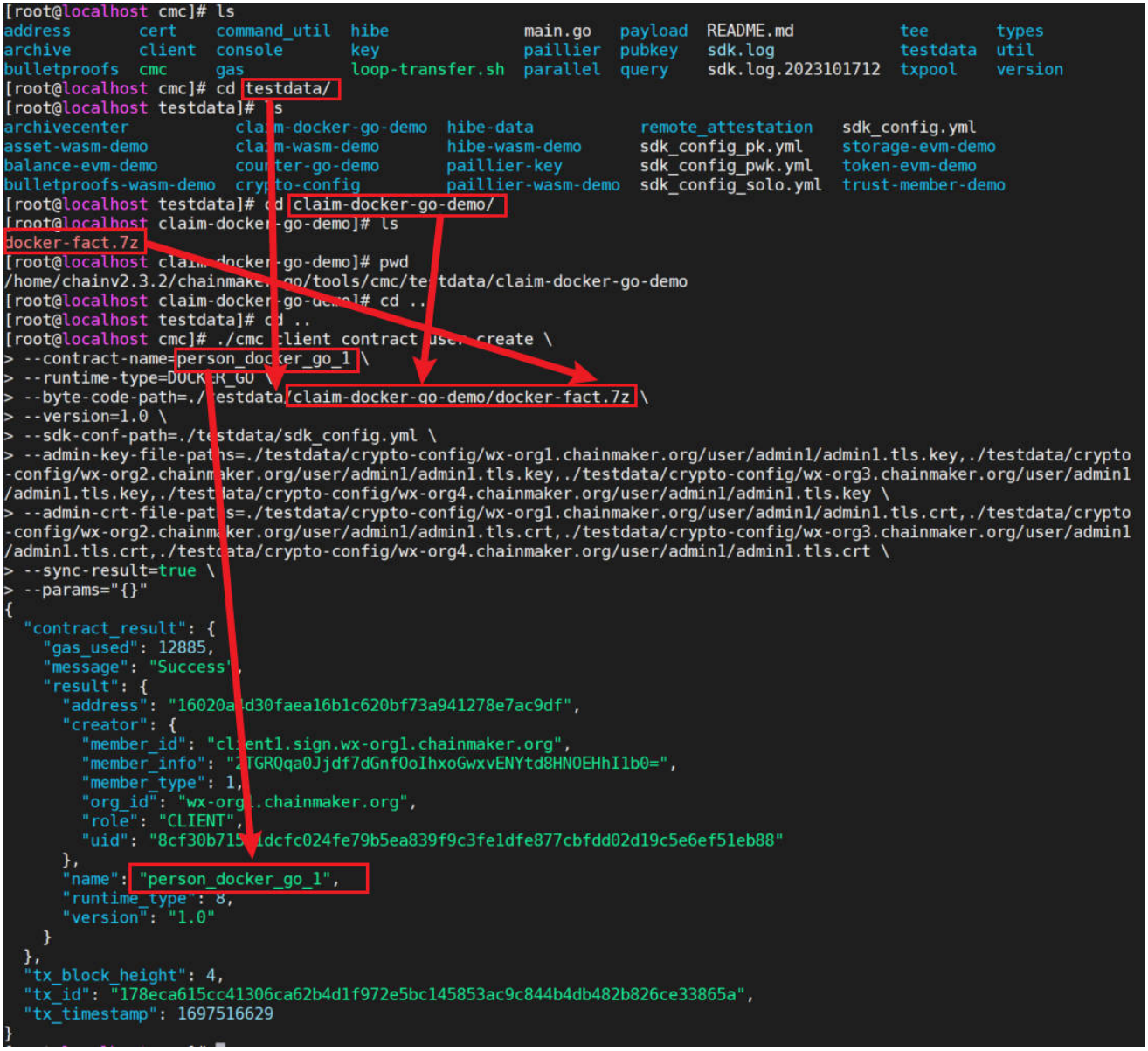
```
[root@localhost cmc]# ./cmc client contract user get \
> --contract-name=fact_wasmer_1 \
> --method=find_by_file_hash \
> --sdk-conf-path=./testdata/sdk_config.yml \
> --params="{\"file_hash\":\"ab3456df5799b87c77e7f88\"}"
{
  "contract_result": {
    "gas_used": 369115,
    "result": "eyJmaWxlX2hhc2giOiJhYjM0NTZkZjU3OTliODdjNzdlN2Y4OCIsImZpbGVfbmFtZSI6Im5hbWUwMDciLCJ0aW1lIjo2NTQzMjM0fQ=="
  },
  "message": "SUCCESS",
  "tx_id": "178eca3662a5d0d8cad9aceb09b5aba861a065e306e744e9854338183edd9e88"
}
```

8.2.3 部署示例合约2（docker-go类型的合约）

参考官方文档链接（https://docs.chainmaker.org.cn/instructions/%E4%BD%BF%E7%94%A8Golang%E8%BF%9B%E8%A1%8C%E6%99%BA%E8%83%BD%E5%90%88%E7%BA%A6%E5%BC%80%E5%8F%91.html）

创建docker-go类型的示例合约person_docker_go_1

```
./cmc client contract user create \
--contract-name=person_docker_go_1 \
--runtime-type=DOCKER_GO \
--byte-code-path=./testdata/claim-docker-go-demo/docker-fact.7z \
--version=1.0 \
--sdk-conf-path=./testdata/sdk_config.yml \
--admin-key-file-paths=./testdata/crypto-config/wx-org1.chainmaker.org/user/admin1/admin1.tls.key,./testdata/crypto-config/wx-org2.chainmaker.org/user/admin1/admin1.tls.key,./testdata/crypto-config/wx-org3.chainmaker.org/user/admin1/admin1.tls.key,./testdata/crypto-config/wx-org4.chainmaker.org/user/admin1/admin1.tls.key \
--admin-crt-file-paths=./testdata/crypto-config/wx-org1.chainmaker.org/user/admin1/admin1.tls.crt,./testdata/crypto-config/wx-org2.chainmaker.org/user/admin1/admin1.tls.crt,./testdata/crypto-config/wx-org3.chainmaker.org/user/admin1/admin1.tls.crt,./testdata/crypto-config/wx-org4.chainmaker.org/user/admin1/admin1.tls.crt \
--sync-result=true \
--params="{}"
```



调用docker-go类型的示例合约person_docker_go_1

```
./cmc client contract user invoke \
--contract-name=person_docker_go_1 \
--method=save \
--sdk-conf-path=./testdata/sdk_config.yml \
--params="
{\"file_name\":\"name007\",\"file_hash\":\"ab3456df5799b87c77e7f88\",\"time\":\"6543234\"}" \
--sync-result=true
```

```
[root@localhost cmc]# ./cmc client contract user invoke \
> --contract-name=person_docker_go_1 \
> --method=save \
> --sdk-conf-path=./testdata/sdk_config.yml \
> --params="{\"file_name\":\"name007\",\"file_hash\":\"ab3456df5799b87c77e7f88\",\"time\":\"6543234\"}" \
> --sync-result=true
{
  "contract_result": {
    "contract_event": [
      {
        "contract_name": "person_docker_go_1",
        "event_data": [
          "ab3456df5799b87c77e7f88",
          "name007"
        ],
        "topic": "topic_vx",
        "tx_id": "178eca9a697aa352ca65f3e71a48e69e7dfdc0ff727343229ed64d409753b950"
      }
    ],
    "gas_used": 134,
    "message": "Success",
    "result": "bmFtZTAwN2FiMzQ1NmRmNTc5OWI4N2M3N2U3Zjg4"
  },
  "tx_block_height": 5,
  "tx_id": "178eca9a697aa352ca65f3e71a48e69e7dfdc0ff727343229ed64d409753b950",
  "tx_timestamp": 1697516874
}
```

查询合约person_docker_go_1中的方法findByFileHash

```
./cmc client contract user get \
--contract-name=person_docker_go_1 \
--method=findByFileHash  \
--sdk-conf-path=./testdata/sdk_config.yml \
--params="{\"file_hash\":\"ab3456df5799b87c77e7f88\"}"
```

```
[root@localhost cmc]# ./cmc client contract user get \
> --contract-name=person_docker_go_1 \
> --method=find_by_file_hash \
> --sdk-conf-path=./testdata/sdk_config.yml \
> --params="{\"file_hash\":\"ab3456df5799b87c77e7f88\"}"
{
  "code": 4,
  "contract_result": {
    "code": 1,
    "gas_used": 114,
    "message": "Fail",
    "result": "aW52YWxpZCBtZXRob2Q="
  },
  "message": "txStatusCode:4, resultCode:1, contractName[person_docker_go_1] method[find_by_file_hash] txType[QUERY_CONTRACT], Fail",
  "tx_id": "178ecab286a34678cac93d35ef56fabe556deb30096146e0b65b9d017bf1139f"
}
[root@localhost cmc]# ./cmc client contract user get \
> --contract-name=person_docker_go_1 \
> --method=findByFileHash  \
> --sdk-conf-path=./testdata/sdk_config.yml \
> --params="{\"file_hash\":\"ab3456df5799b87c77e7f88\"}"
{
  "contract_result": {
    "gas_used": 126,
    "message": "Success",
    "result": "eyJGaWxlSGFzaCI6ImFiMzQ1NmRmNTc5OWI4N2M3N2U3Zjg4IiwiRmlsZU5hbWUiOiJuYW1lMDA3IiwidGltZSI6NjU0MzIzNH0="
  },
  "message": "SUCCESS",
  "tx_id": "178ecad26afd5d87cae20d304730be108067418250b948b0a84177d7aaebbdf5"
}
```

由于合约中定义的方法名称是findByFileHash,而不是find_by_file_hash,因此导致报错

问题9

9.1 编写Golang合约

```go
package main

// 合约结构体，合约名称需要写入main()方法当中
type FactContract struct {
}

// 合约必须实现下面两个方法：
// InitContract() protogo.Response
// UpgradeContract() protogo.Response
// InvokeContract(method string) protogo.Response

// 用于合约的部署
// @return:     合约返回结果，包括Success和Error
func (f *FactContract) InitContract() protogo.Response {
return sdk.Success([]byte("Init contract success"))
}

// 用于合约的升级
// @return:     合约返回结果，包括Success和Error
func (f *FactContract) UpgradeContract() protogo.Response {
return sdk.Success([]byte("Upgrade contract success"))
}

// 用于合约的调用
// @param method: 交易请求调用的方法
// @return:     合约返回结果，包括Success和Error
func (f *FactContract) InvokeContract(method string)
protogo.Response {
    switch method {
    case "save":
        return f.save()
    case "findByFileHash":
        return f.findByFileHash()
    default:
        return sdk.Error("invalid method")
    }
}

// sdk代码中，有且仅有一个main()方法
func main() {
   // main()方法中，下面的代码为必须代码，不建议修改main()方法当中的代码
```

```
    // 其中，TestContract为用户实现合约的具体名称
     err := sandbox.Start(new(FactContract))
     if err != nil {
         log.Fatal(err)
     }
}
```

9.2 编写编译合约的脚本

```
# build.sh
#!/bin/bash

contractName=$1
if  [[ ! -n $contractName ]] ;then
    echo "contractName is empty. use as: ./build.sh contractName"
    exit 1
fi

go build -ldflags="-s -w" -o $contractName

7z a $contractName $contractName
rm -f $contractName
```

9.3 安装7z

```
yum -y install epel-release
yum -y install p7zip p7zip-plugins
```

9.4 编译合约

```
# 给脚本文件添加执行的权限
chmod +x build.sh
./build 合约名
```

问题10

10.1 Java SDK使用

参考官方文档链接（https://docs.chainmaker.org.cn/v2.3.2/html/sdk/JavaSDK%E4%BD%BF%E7%94%A8%E8%AF%B4%E6%98%8E.html）

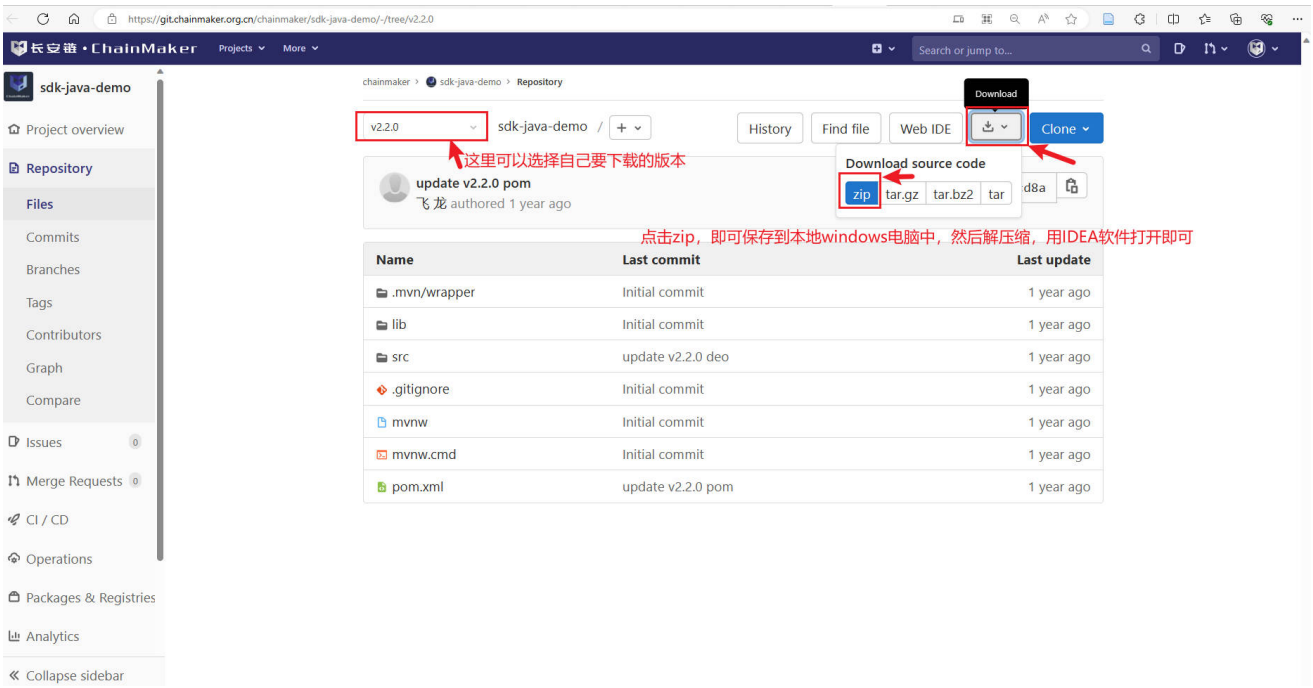
10.1.1 环境依赖(jdk安装在本地windows)

```
jdk8，jdk11，jdk17
# jdk镜像下载地址：https://mirrors.yangxingzhen.com/jdk/
# 若已安装，请通过命令查看版本：
java -version
```

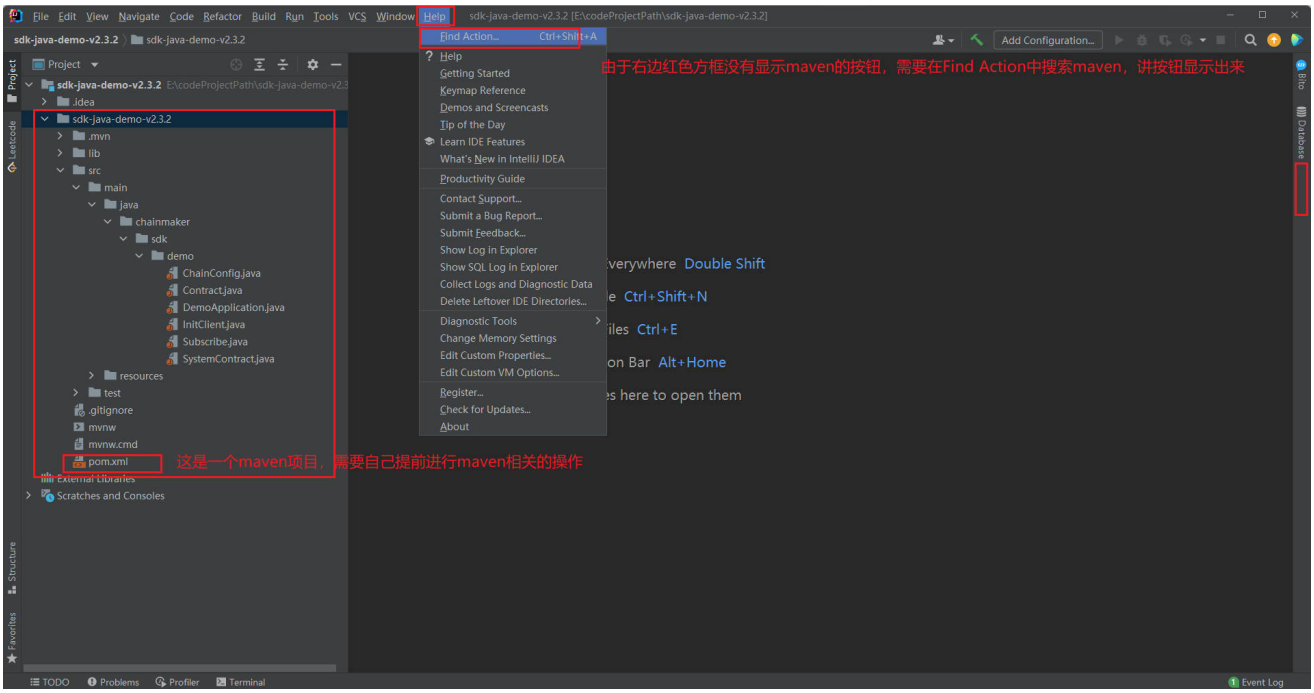
```
E:\>java -version
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

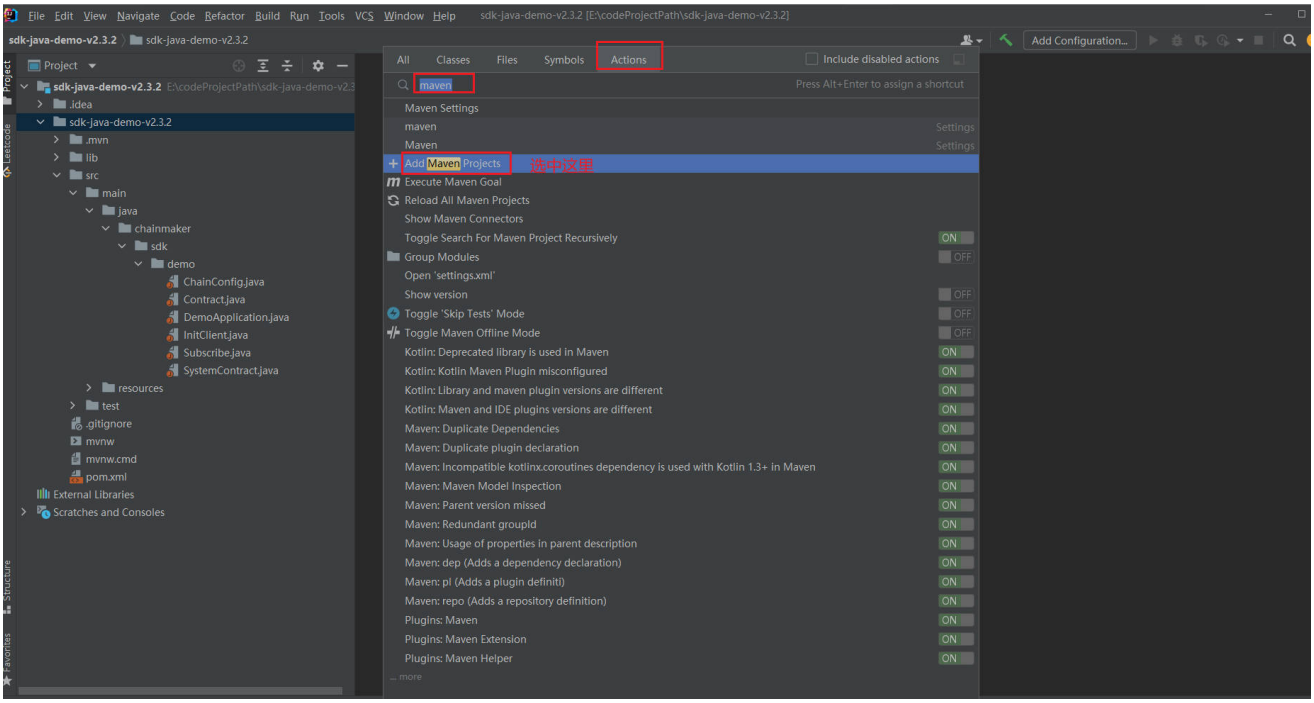
10.1.2 应用demo下载到本地windows电脑

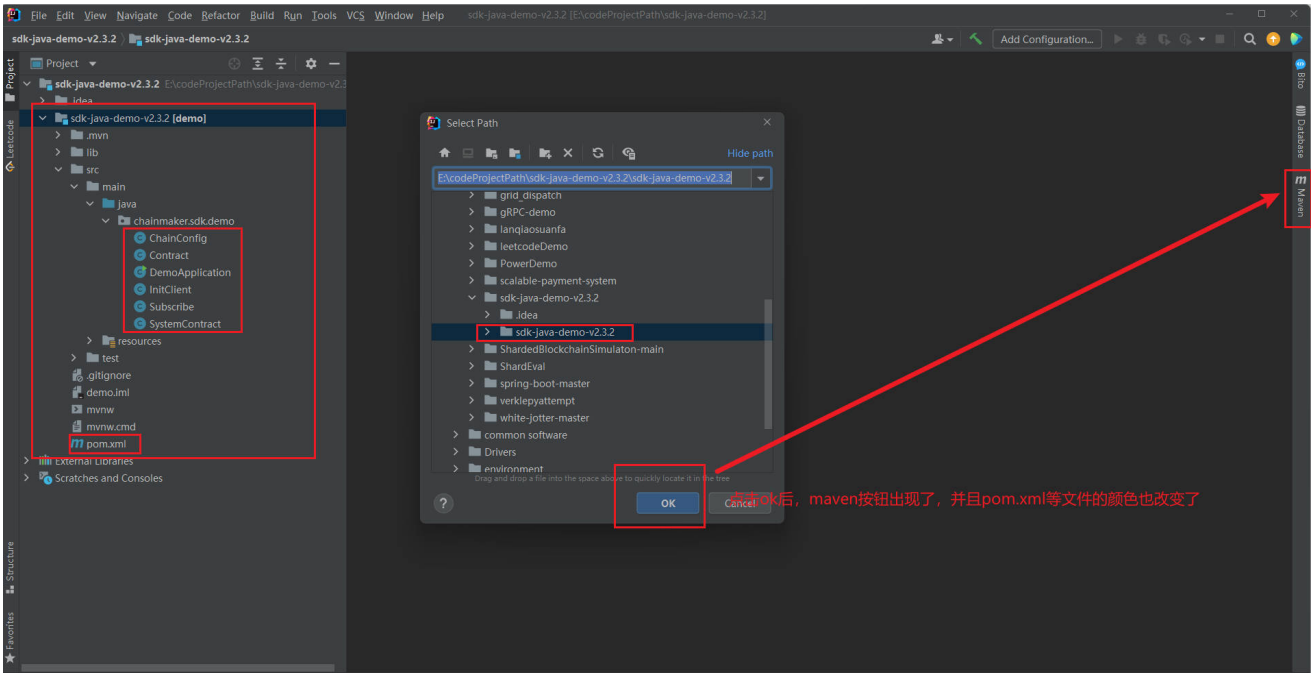
```
# 下载地址
https://git.chainmaker.org.cn/chainmaker/sdk-java-demo/-/tree/v2.2.0
```

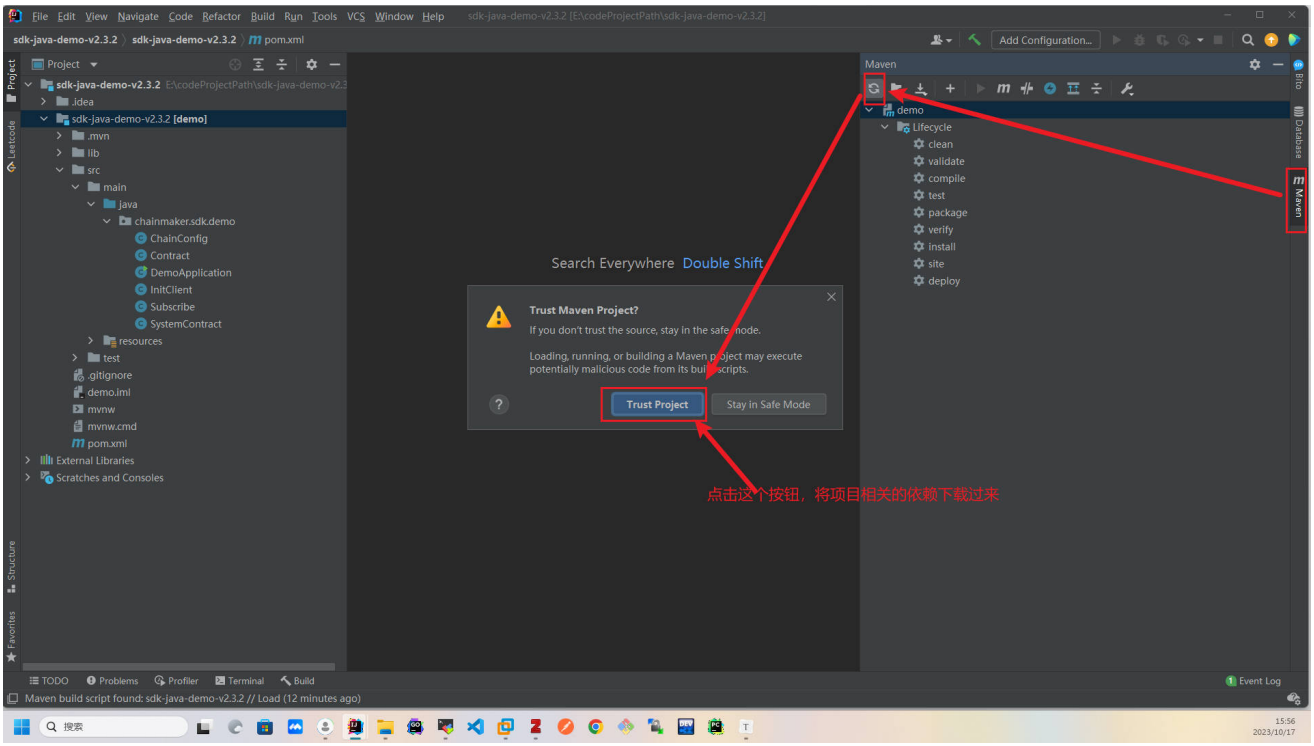


10.1.3 解压缩应用demo，并用IDEA软件打开

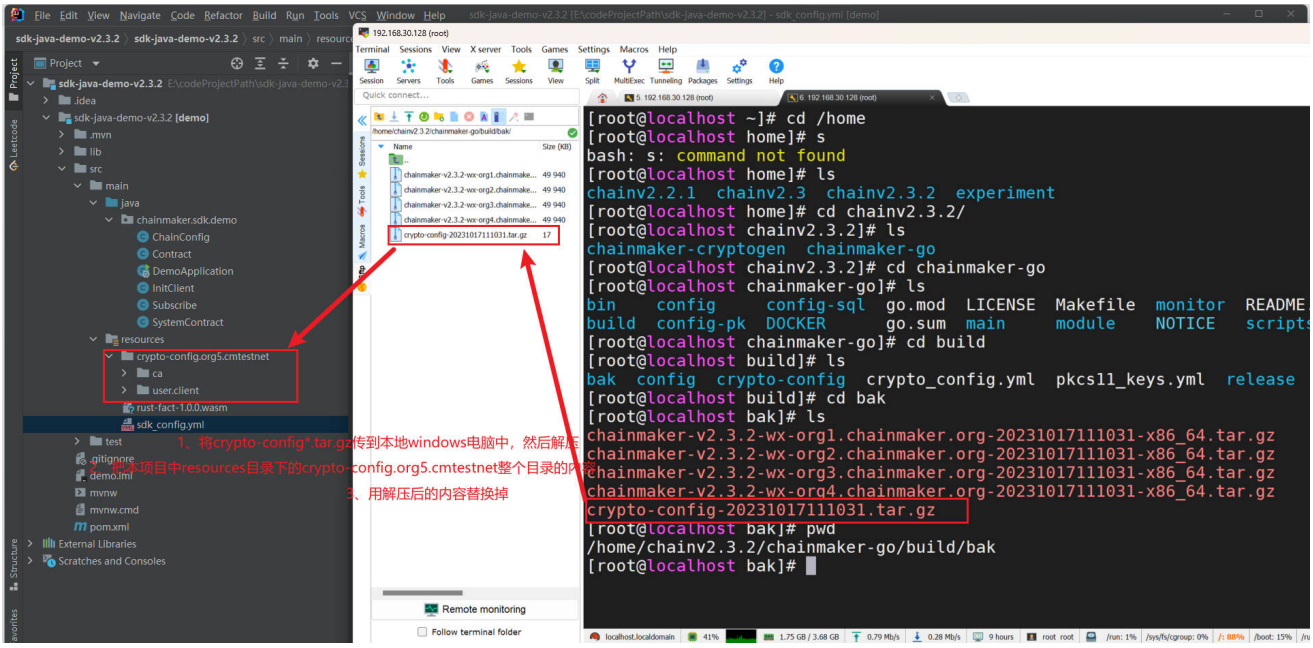


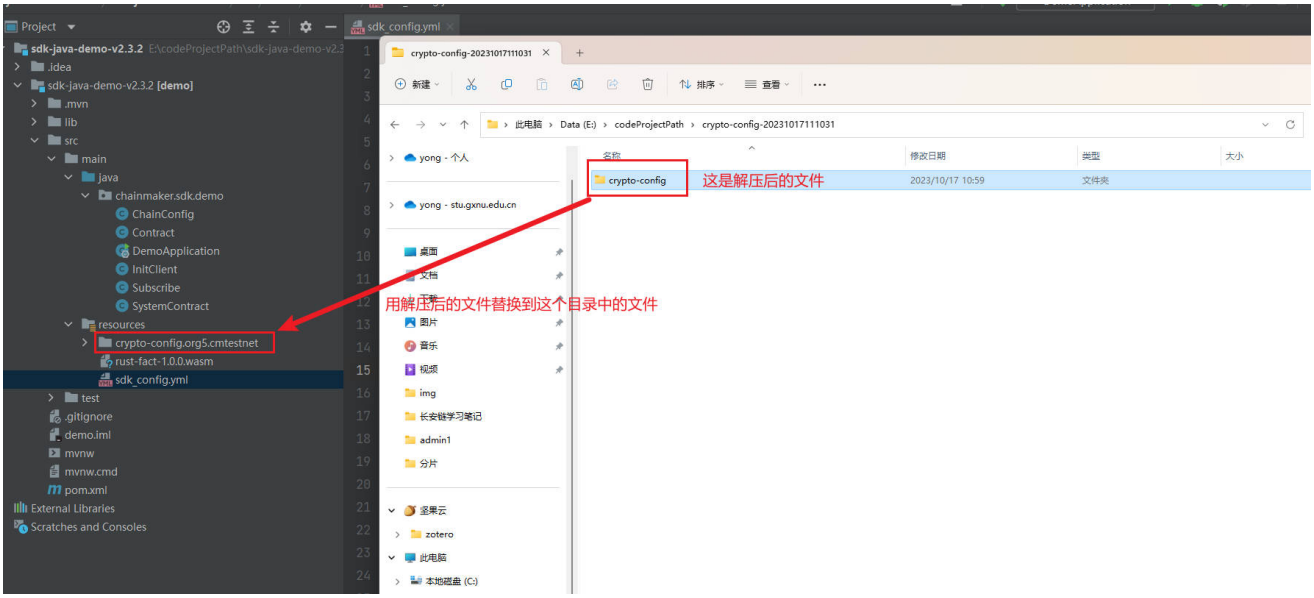


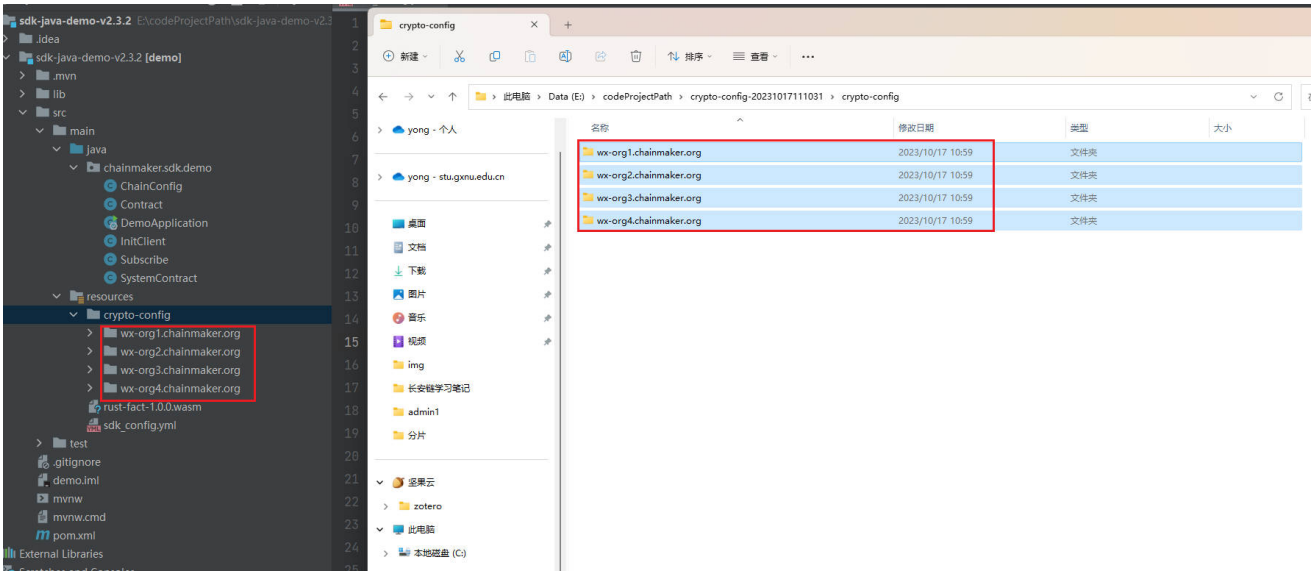




10.1.4 替换掉目录resources/crypto-config.org5.cmtestnet中的文件


1、将crypto-config*.tar.gz传到本地windows电脑中，然后解压
2、把本项目中resources目录下的crypto-config.org5.cmtestnet整个目录的内容
3、用解压后的内容替换掉
[root@localhost ~]# cd /home
[root@localhost home]# s
bash: s: command not found
[root@localhost home]# ls
chainv2.2.1 chainv2.3 chainv2.3.2 experiment
[root@localhost home]# cd chainv2.3.2/
[root@localhost chainv2.3.2]# ls
chainmaker-cryptogen chainmaker-go
[root@localhost chainv2.3.2]# cd chainmaker-go
[root@localhost chainmaker-go]# ls
bin config config-sql go.mod LICENSE Makefile monitor README
build config-pk DOCKER go.sum main module NOTICE scripts
[root@localhost chainmaker-go]# cd build
[root@localhost build]# ls
bak config crypto-config crypto_config.yml pkcs11_keys.yml release
[root@localhost build]# cd bak
[root@localhost bak]# ls
chainmaker-v2.3.2-wx-org1.chainmaker.org-20231017111031-x86_64.tar.gz
chainmaker-v2.3.2-wx-org2.chainmaker.org-20231017111031-x86_64.tar.gz
chainmaker-v2.3.2-wx-org3.chainmaker.org-20231017111031-x86_64.tar.gz
chainmaker-v2.3.2-wx-org4.chainmaker.org-20231017111031-x86_64.tar.gz
crypto-config-20231017111031.tar.gz
[root@localhost bak]# pwd
/home/chainv2.3.2/chainmaker-go/build/bak
[root@localhost bak]#
Remote monitoring
Follow terminal folder


这是解压后的文件
用解压后的文件替换到这个目录中的文件



10.1.5 修改resources目录下的sdk_config.yml文件

修改前的sdk_config.yml文件

```yaml
chain_client:
  # 链ID
  chain_id: "chainmaker_testnet_chain"
  # 组织ID
  org_id: "org5.cmtestnet"
  # 客户端用户私钥路径
  user_key_file_path: "src/main/resources/crypto-config/org5.cmtestnet/user/client/ytf002.tls.key"
  # 客户端用户证书路径
  user_crt_file_path: "src/main/resources/crypto-config/org5.cmtestnet/user/client/ytf002.tls.crt"
  # 客户端用户交易签名私钥路径(若未设置，将使用user_key_file_path)
  user_sign_key_file_path: "src/main/resources/crypto-config/org5.cmtestnet/user/client/ytf002.sign.key"
  # 客户端用户交易签名证书路径(若未设置，将使用user_crt_file_path)
  user_sign_crt_file_path: "src/main/resources/crypto-config/org5.cmtestnet/user/client/ytf002.sign.crt"

  retry_limit: 10
  # 同步交易结果模式下，每次轮训交易结果时的等待时间，单位：ms 删除此项或设为<=0则使用默认值 500
  retry_interval: 500
  # 当前签名证书的别名。当设置此配置项时，chain client 对象将自动检查链上是否已添加此别名，如果没有则自动上链此证书别名，
  # 并且后续所有交易都会使用别名，别名可降低交易体大小。若为空则不启用。
  #  alias: mycert5
  nodes:
    - # 节点地址，格式为：IP:端口:连接数
      node_addr: "certnode1.chainmaker.org.cn:13301"
      # 节点连接数
      conn_cnt: 10
      # RPC连接是否启用双向TLS认证
      enable_tls: true
      # 信任证书池路径
      trust_root_paths:
        - "src/main/resources/crypto-config/org5.cmtestnet/ca/org1.cmtestnet"
      # TLS hostname
      tls_host_name: "consensus1.tls.org1.cmtestnet"
```

```
    - # 节点地址，格式为：IP:端口:连接数
      node_addr: "certnode2.chainmaker.org.cn:13302"
      # 节点连接数
      conn_cnt: 10
      # RPC连接是否启用双向TLS认证
      enable_tls: true
      # 信任证书池路径
      trust_root_paths:
        - "src/main/resources/crypto-config/org5.cmtestnet/ca/org2.cmtestnet"
      # TLS hostname
      tls_host_name: "consensus1.tls.org2.cmtestnet"
    - # 节点地址，格式为：IP:端口:连接数
      node_addr: "certnode3.chainmaker.org.cn:13303"
      # 节点连接数
      conn_cnt: 10
      # RPC连接是否启用双向TLS认证
      enable_tls: true
      # 信任证书池路径
      trust_root_paths:
        - "src/main/resources/crypto-config/org5.cmtestnet/ca/org3.cmtestnet"
      # TLS hostname
      tls_host_name: "consensus1.tls.org3.cmtestnet"
    - # 节点地址，格式为：IP:端口:连接数
      node_addr: "certnode4.chainmaker.org.cn:13303"
      # 节点连接数
      conn_cnt: 10
      # RPC连接是否启用双向TLS认证
      enable_tls: true
      # 信任证书池路径
      trust_root_paths:
        - "src/main/resources/crypto-config/org5.cmtestnet/ca/org4.cmtestnet"
      # TLS hostname
      tls_host_name: "consensus1.tls.org4.cmtestnet"
  archive:
    # mysql archivecenter 归档中心 数据归档链外存储相关配置
    type: "mysql"
    dest: "root:123456:localhost:3306"
    secret_key: xxx
  # 如果为true且归档中心配置打开,那么查询数据优先从归档中心查询
  # 如果是false,则不访问归档中心，只访问链数据
```

```yaml
  archive_center_query_first: false
  # 归档中心
  archive_center_config:
    chain_genesis_hash: a23a0c6fd402010efbc31741b2a868c9e6558471fda1e8d1f16673ef683e06db
    archive_center_http_url: http://127.0.0.1:13119
    request_second_limit: 10
    rpc_address: 127.0.0.1:13120
    tls_enable: false
    tls:
      server_name: archiveserver1.tls.wx-org.chainmaker.org
      priv_key_file: src/test/resources/archivecenter/archiveclient1.tls.key
      cert_file: src/test/resources/archivecenter/archiveclient1.tls.crt
      trust_ca_list:
        - src/test/resources/archivecenter/ca
    max_send_msg_size: 200
    max_recv_msg_size: 200

  rpc_client:
    # grpc客户端最大接受容量(MB)
    max_receive_message_size: 16

  pkcs11:
    enabled: false # pkcs11 is not used by default

  # 交易结果是否订阅获取
  enable_tx_result_dispatcher: true

  ##连接池配置
  connPool:
    # 最大连接数
    maxTotal: 100
    # 最少空闲连接
    minIdle: 1
    #最大空闲连接
    maxIdle: 20
    #连接空闲最小保活时间，默认即为30分钟(18000000)，单位：ms
    minEvictableIdleTime: 350000
    #回收空闲线程的执行周期，单位毫秒。默认值60000ms（60s） ，-1 表示不启用线程回收资源，单位：ms
    timeBetweenEvictionRuns: 60000
```

```
    #没有空闲连接时，获取连接是否阻塞
    blockWhenExhausted: true
    #当没有空闲连接时，获取连接阻塞等待时间，单位：ms
    maxWaitMillis: 3000
```

修改后的sdk_config.yml文件(用# xxx标记修改处)

```
chain_client:
  # 链ID
  chain_id: "chain1" # xxx
  # 组织ID
  org_id: "wx-org1.chainmaker.org" # xxx
  # 客户端用户私钥路径
  user_key_file_path: "src/main/resources/crypto-config/wx-org1.chainmaker.org/user/client1/client1.tls.key" # xxx
  # 客户端用户证书路径
  user_crt_file_path: "src/main/resources/crypto-config/wx-org1.chainmaker.org/user/client1/client1.tls.crt" # xxx
  # 客户端用户交易签名私钥路径(若未设置，将使用user_key_file_path)
  user_sign_key_file_path: "src/main/resources/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.key" # xxx
  # 客户端用户交易签名证书路径(若未设置，将使用user_crt_file_path)
  user_sign_crt_file_path: "src/main/resources/crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.crt" # xxx

  retry_limit: 10
  # 同步交易结果模式下，每次轮训交易结果时的等待时间，单位：ms 删除此项或设为<=0则使用默认值 500
  retry_interval: 500
  # 当前签名证书的别名。当设置此配置项时，chain client 对象将自动检查链上是否已添加此别名，如果没有则自动上链此证书别名，
  # 并且后续所有交易都会使用别名，别名可降低交易体大小。若为空则不启用。
  #  alias: mycert5
  nodes:
    - # 节点地址，格式为：IP:端口:连接数
      node_addr: "192.168.30.128:12301" # xxx
      # 节点连接数
      conn_cnt: 10
      # RPC连接是否启用双向TLS认证
      enable_tls: true
```

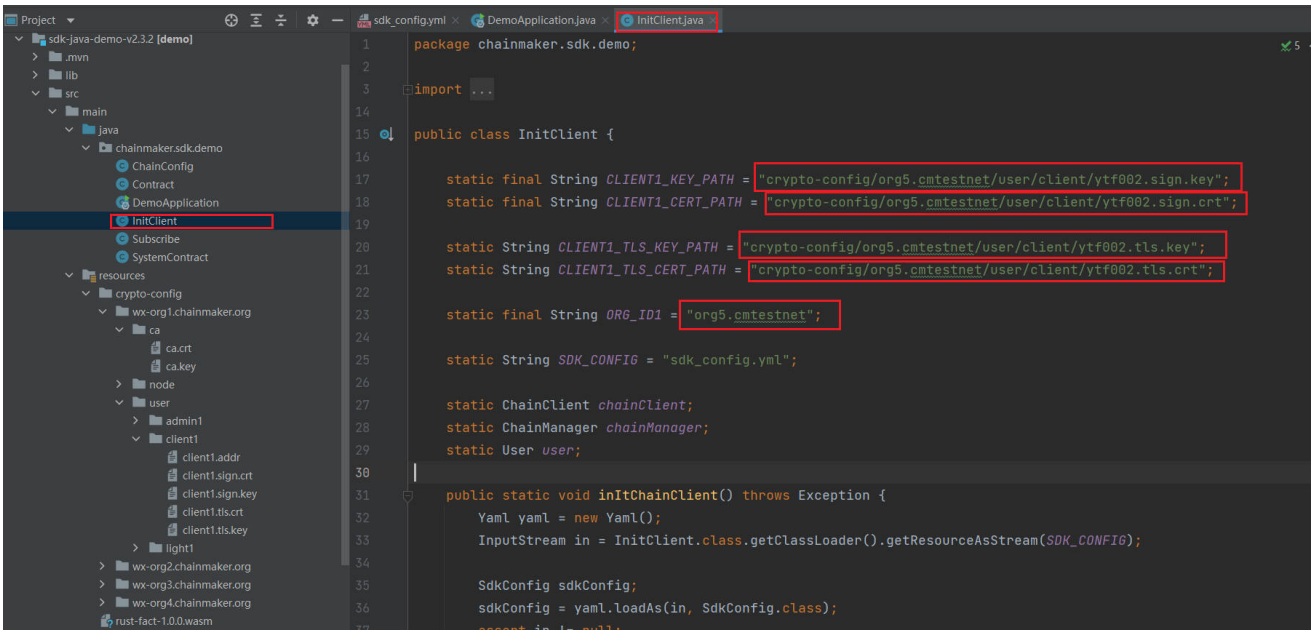
```
      # 信任证书池路径
      trust_root_paths:
        - "src/main/resources/crypto-config/wx-org1.chainmaker.org/ca" # xxx
      # TLS hostname
      tls_host_name: "chainmaker.org" # xxx
    - # 节点地址，格式为：IP:端口:连接数
      node_addr: "192.168.30.128:12302" # xxx
      # 节点连接数
      conn_cnt: 10
      # RPC连接是否启用双向TLS认证
      enable_tls: true
      # 信任证书池路径
      trust_root_paths:
        - "src/main/resources/crypto-config/wx-org1.chainmaker.org/ca" # xxx
      # TLS hostname
      tls_host_name: "chainmaker.org" # xxx
    - # 节点地址，格式为：IP:端口:连接数
      node_addr: "192.168.30.128:12303" # xxx
      # 节点连接数
      conn_cnt: 10
      # RPC连接是否启用双向TLS认证
      enable_tls: true
      # 信任证书池路径
      trust_root_paths:
        - "src/main/resources/crypto-config/wx-org1.chainmaker.org/ca" # xxx
      # TLS hostname
      tls_host_name: "chainmaker.org" # xxx
    - # 节点地址，格式为：IP:端口:连接数
      node_addr: "192.168.30.128:12304" # xxx
      # 节点连接数
      conn_cnt: 10
      # RPC连接是否启用双向TLS认证
      enable_tls: true
      # 信任证书池路径
      trust_root_paths:
        - "src/main/resources/crypto-config/wx-org1.chainmaker.org/ca" # xxx
      # TLS hostname
      tls_host_name: "chainmaker.org" # xxx
  archive:
```

```
    # mysql archivecenter 归档中心 数据归档链外存储相关配置
    type: "mysql"
    dest: "root:123456:localhost:3306"
    secret_key: xxx
  # 如果为true且归档中心配置打开,那么查询数据优先从归档中心查询
  # 如果是false,则不访问归档中心，只访问链数据
#  archive_center_query_first: false # xxx
#  # 归档中心
#  archive_center_config:
#    chain_genesis_hash:
a23a0c6fd402010efbc31741b2a868c9e6558471fda1e8d1f16673ef683e06db
#    archive_center_http_url: http://192.168.30.128:13119
#    request_second_limit: 10
#    rpc_address: 192.168.30.128:13120
#    tls_enable: false
#    tls:
#      server_name: archiveserver1.tls.wx-org.chainmaker.org
#      priv_key_file:
src/test/resources/archivecenter/archiveclient1.tls.key
#      cert_file:
src/test/resources/archivecenter/archiveclient1.tls.crt
#      trust_ca_list:
#        - src/test/resources/archivecenter/ca
#    max_send_msg_size: 200
#    max_recv_msg_size: 200 # xxx

  rpc_client:
    # grpc客户端最大接受容量(MB)
    max_receive_message_size: 16

  pkcs11:
    enabled: false # pkcs11 is not used by default

  # 交易结果是否订阅获取
  enable_tx_result_dispatcher: true

  ##连接池配置
  connPool:
    # 最大连接数
    maxTotal: 100
    # 最少空闲连接
    minIdle: 1
    #最大空闲连接
```

```
      maxIdle: 20
      #连接空闲最小保活时间，默认即为30分钟(18000000)，单位：ms
      minEvictableIdleTime: 350000
      #回收空闲线程的执行周期，单位毫秒。默认值60000ms（60s），-1 表示不启用线程回收资源，单位：ms
      timeBetweenEvictionRuns: 60000
      #没有空闲连接时，获取连接是否阻塞
      blockWhenExhausted: true
      #当没有空闲连接时，获取连接阻塞等待时间，单位：ms
      maxWaitMillis: 3000

```

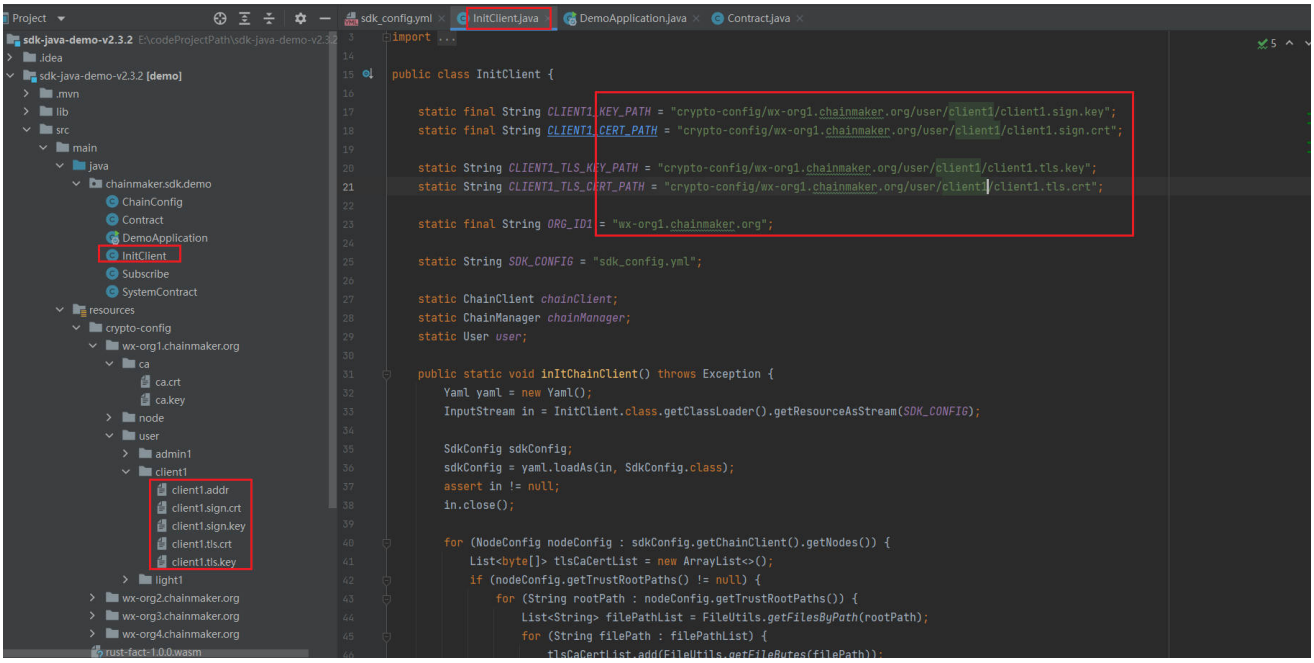
10.1.6 修改initClient.java文件

要修改的部分如下

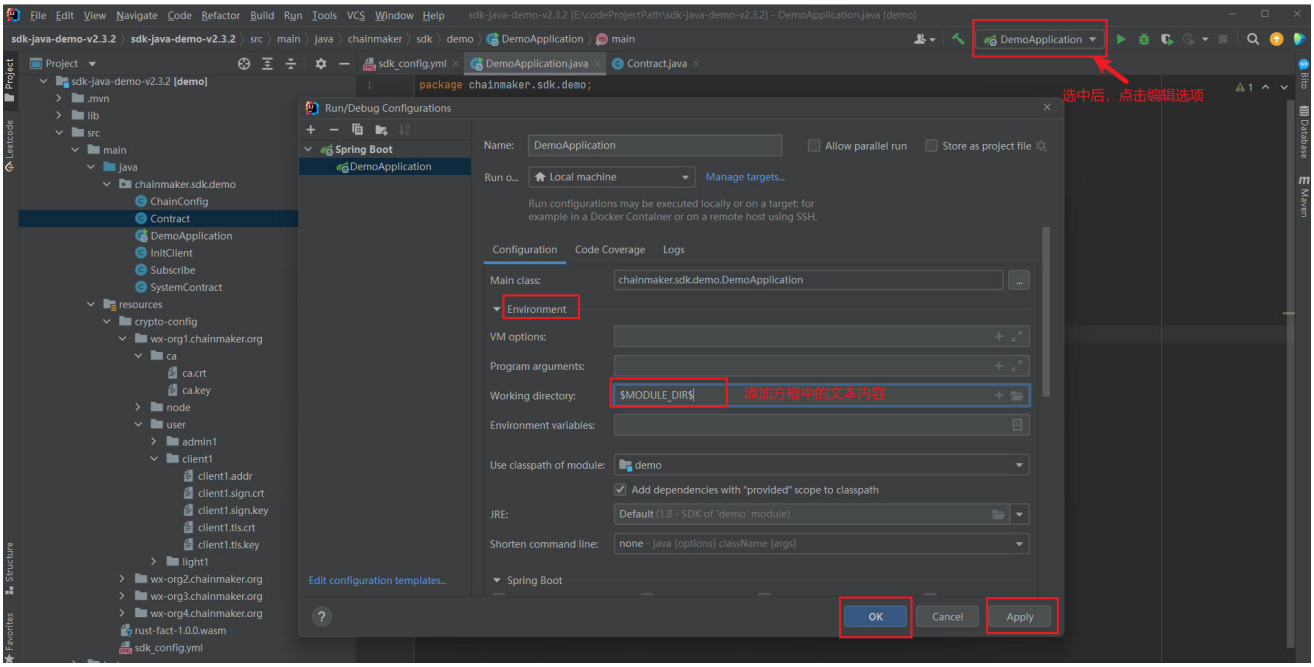


修改后的内容如下

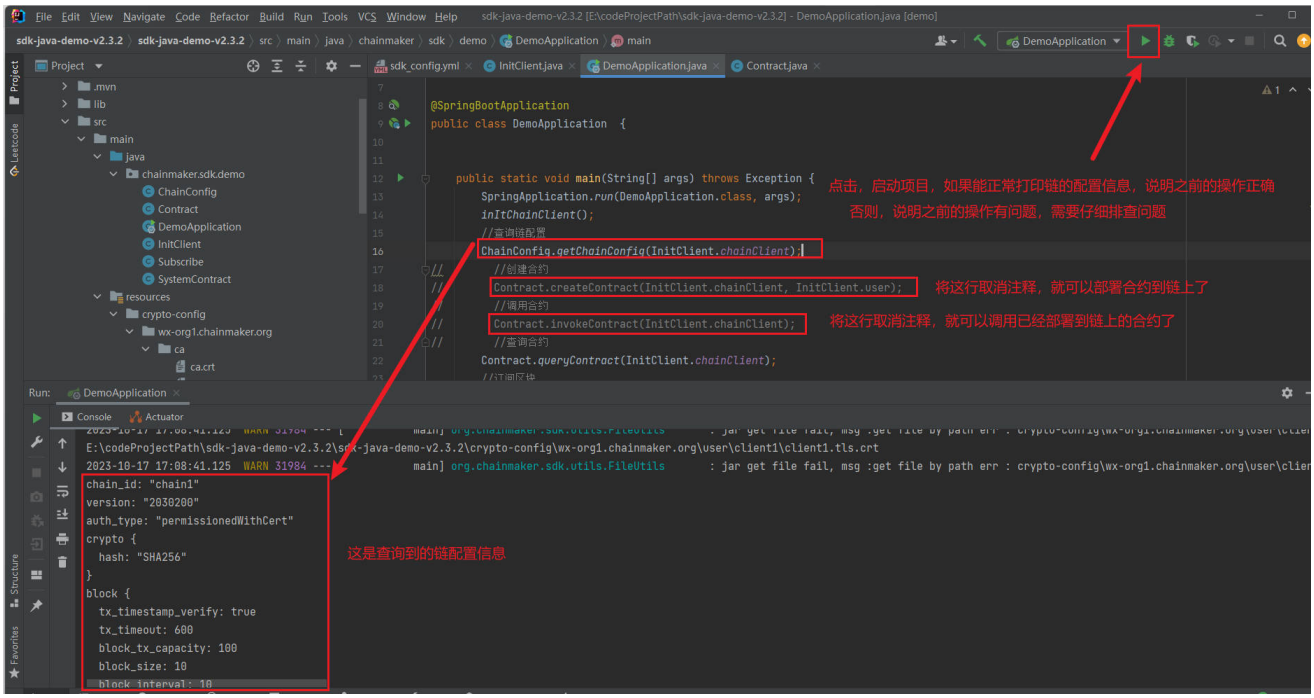
```
static final String CLIENT1_KEY_PATH = "crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.key";
    static final String CLIENT1_CERT_PATH = "crypto-config/wx-org1.chainmaker.org/user/client1/client1.sign.crt";

    static String CLIENT1_TLS_KEY_PATH = "crypto-config/wx-org1.chainmaker.org/user/client1/client1.tls.key";
    static String CLIENT1_TLS_CERT_PATH = "crypto-config/wx-org1.chainmaker.org/user/client1/client1.tls.crt";

    static final String ORG_ID1 = "wx-org1.chainmaker.org";
```

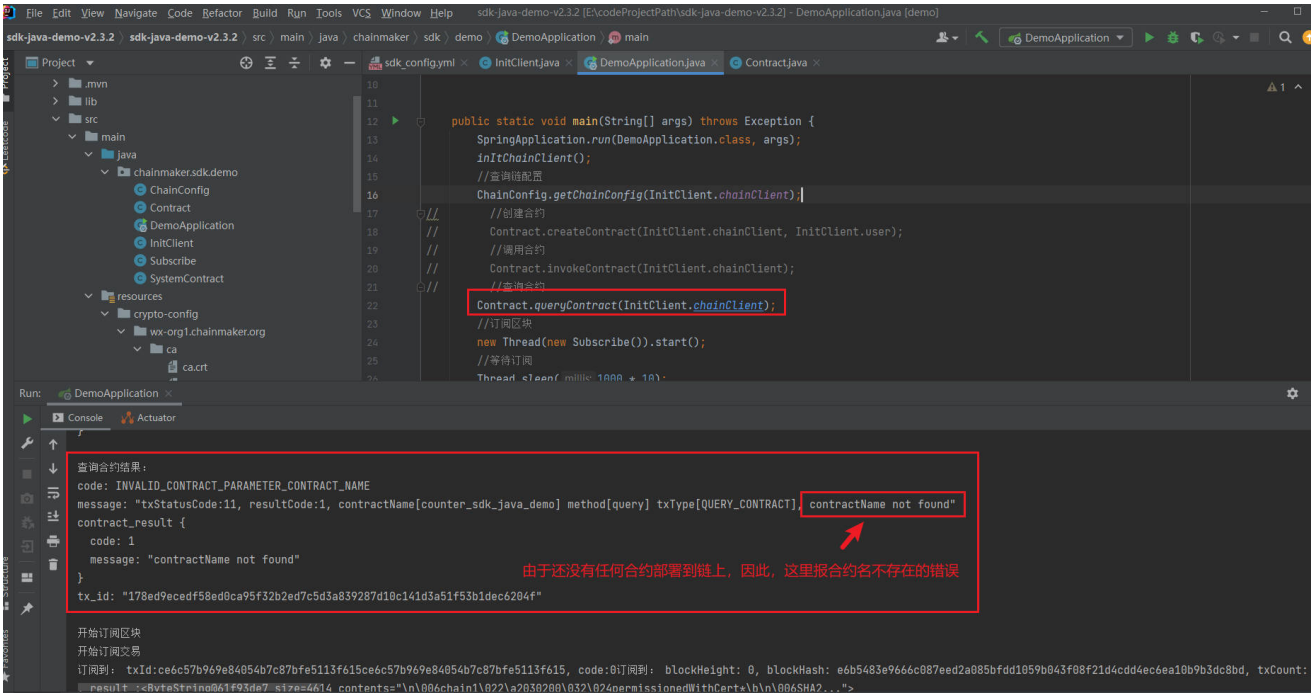


10.1.7 编辑配置

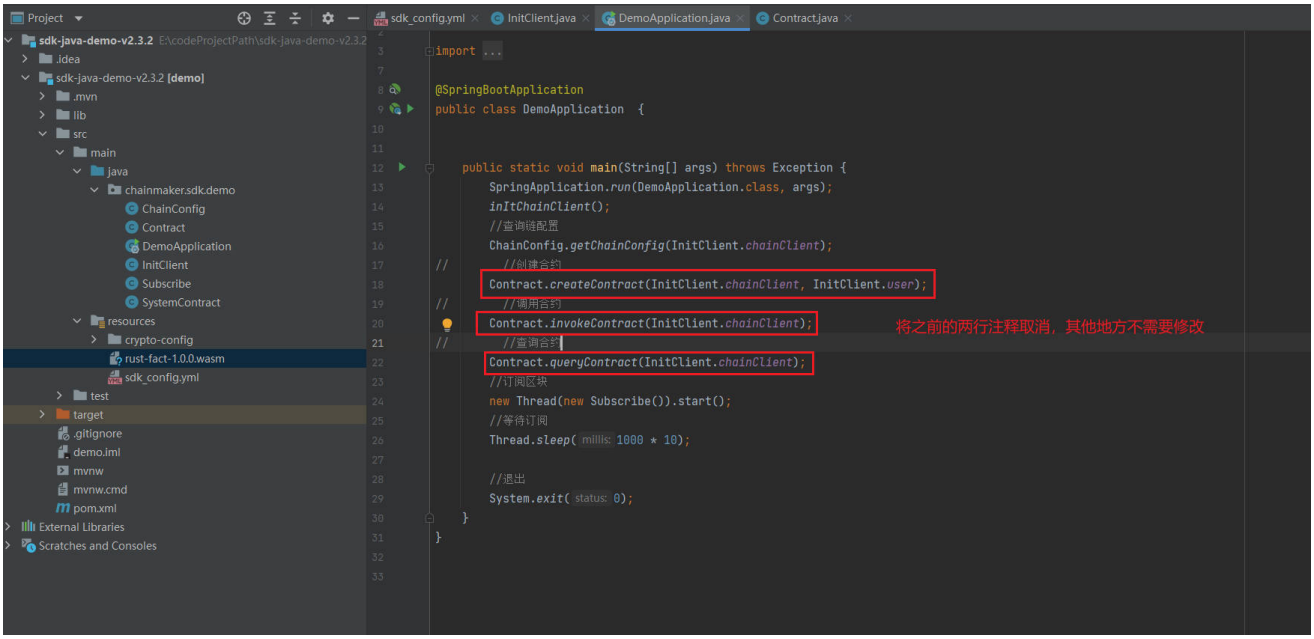


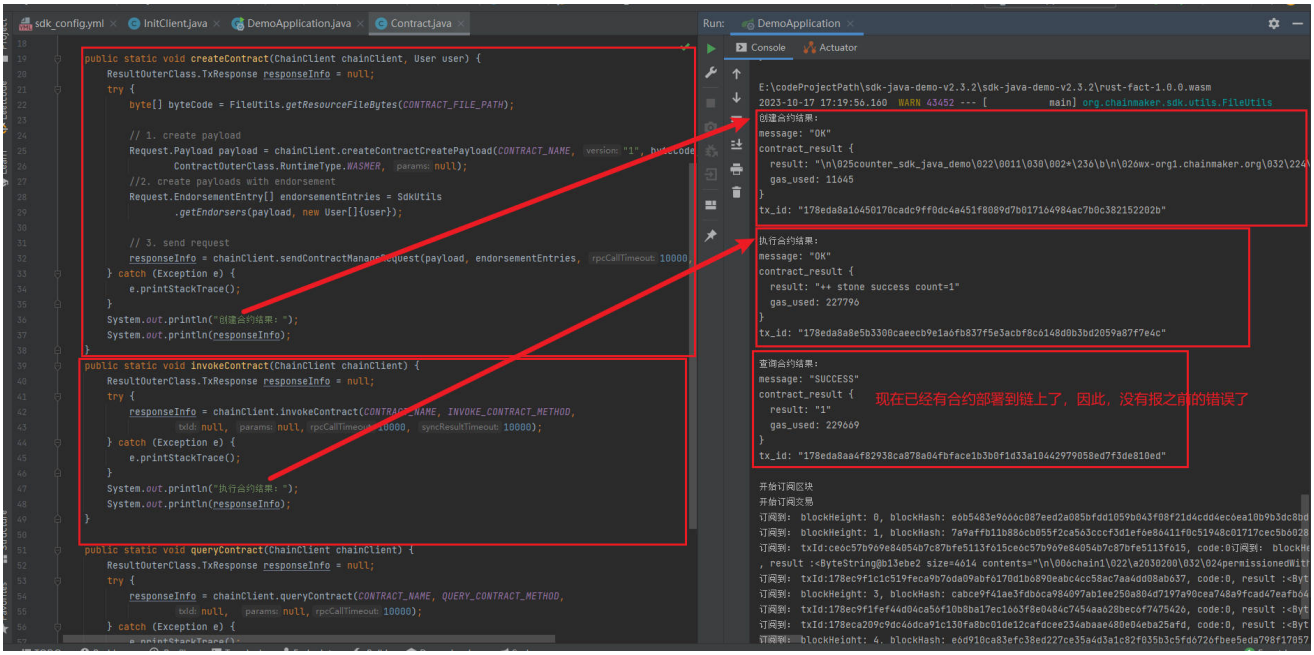
10.1.8 启动应用demo




Project
sdk_config.yml
InitClient.java
DemoApplication.java
Contract.java
public static void main(String[] args) throws Exception {
SpringApplication.run(DemoApplication.class, args);
inItChainClient();
//查询链配置
ChainConfig.getChainConfig(InitClient.chainClient);
//创建合约
Contract.createContract(InitClient.chainClient, InitClient.user);
//调用合约
Contract.invokeContract(InitClient.chainClient);
//查询合约
Contract.queryContract(InitClient.chainClient);
//订阅区块
new Thread(new Subscribe()).start();
//等待订阅
Run: DemoApplication
Console
Actuator
查询合约结果：
code: INVALID_CONTRACT_PARAMETER_CONTRACT_NAME
message: "txStatusCode:11, resultCode:1, contractName[counter_sdk_java_demo] method[query] txType[QUERY_CONTRACT], contractName not found"
contract_result {
code: 1
message: "contractName not found"
}
tx_id: "178ed9ecedf58ed0ca95f32b2ed7c5d3a839287d10c141d3a51f53b1dec6204f"
由于还没有任何合约部署到链上，因此，这里报合约名不存在的错误
开始订阅区块
开始订阅交易
订阅到： txId:ce6c57b969e84054b7c87bfe5113f615ce6c57b969e84054b7c87bfe5113f615, code:0订阅到： blockHeight: 0, blockHash: e6b5483e9666c087eed2a085bfdd1059b043f08f21d4cdd4ec6ea10b9b3dc8bd, txCount:

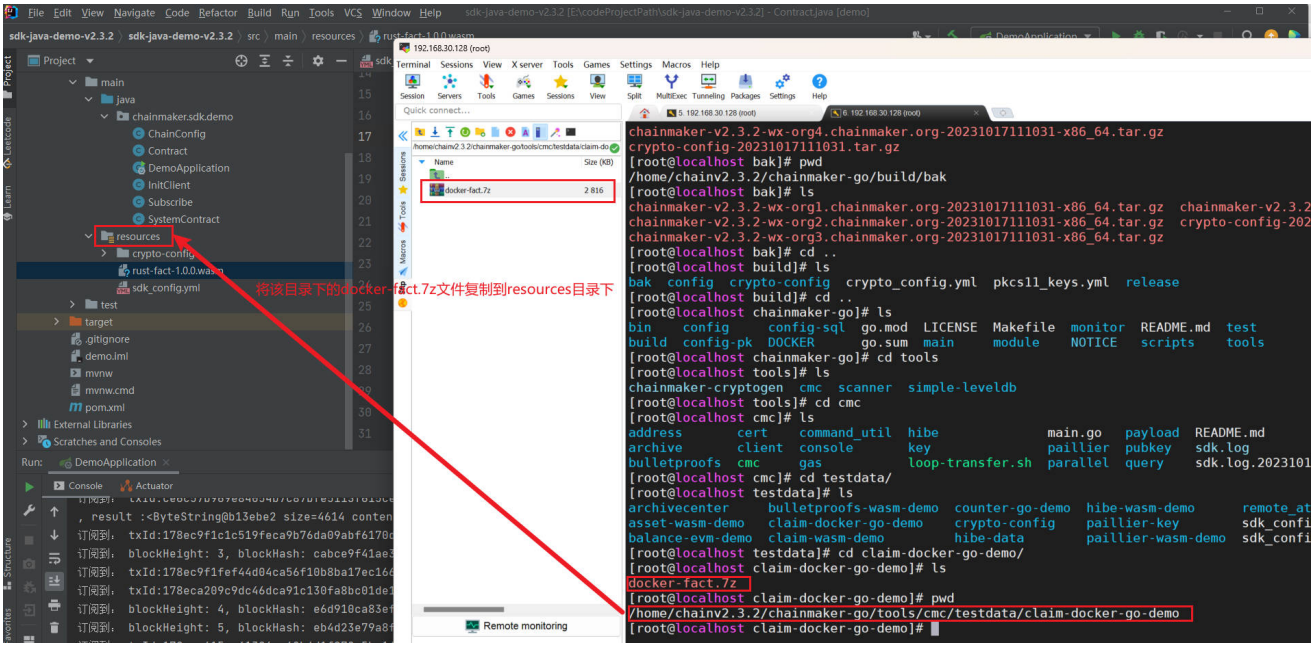
10.1.9 部署示例合约


Project
sdk_config.yml
InitClient.java
DemoApplication.java
Contract.java
import ...
@SpringBootApplication
public class DemoApplication {
public static void main(String[] args) throws Exception {
SpringApplication.run(DemoApplication.class, args);
inItChainClient();
//查询链配置
ChainConfig.getChainConfig(InitClient.chainClient);
//创建合约
Contract.createContract(InitClient.chainClient, InitClient.user);
//调用合约
Contract.invokeContract(InitClient.chainClient);
//查询合约
Contract.queryContract(InitClient.chainClient);
//订阅区块
new Thread(new Subscribe()).start();
//等待订阅
Thread.sleep(millis: 1000 * 10);
//退出
System.exit(status: 0);
将之前的两行注释取消，其他地方不需要修改



10.1.10 部署docker-go类型的合约

将docker-fact.7z文件复制到resources目录下



修改前的Contract.java文件

```java
public class Contract {

    private static final String QUERY_CONTRACT_METHOD = "query";
    private static final String INVOKE_CONTRACT_METHOD =
"increase";
    private static final String CONTRACT_NAME =
"counter_sdk_java_demo";
```

```
6       private static final String CONTRACT_FILE_PATH = "rust-fact-
    1.0.0.wasm";
7
8       public static void createContract(ChainClient chainClient,
    User user) {
9           ResultOuterClass.TxResponse responseInfo = null;
10          try {
11              byte[] byteCode =
    FileUtils.getResourceFileBytes(CONTRACT_FILE_PATH);
12
13              // 1. create payload
14              Request.Payload payload =
    chainClient.createContractCreatePayload(CONTRACT_NAME, "1",
    byteCode,
15                      ContractOuterClass.RuntimeType.WASMER, null);
16              //2. create payloads with endorsement
17              Request.EndorsementEntry[] endorsementEntries =
    SdkUtils
18                      .getEndorsers(payload, new User[]{user});
19
20              // 3. send request
21              responseInfo =
    chainClient.sendContractManageRequest(payload,
    endorsementEntries, 10000, 10000);
22          } catch (Exception e) {
23              e.printStackTrace();
24          }
25          System.out.println("创建合约结果：");
26          System.out.println(responseInfo);
27      }
28      public static void invokeContract(ChainClient chainClient) {
29          ResultOuterClass.TxResponse responseInfo = null;
30          try {
31              responseInfo =
    chainClient.invokeContract(CONTRACT_NAME, INVOKE_CONTRACT_METHOD,
32                      null, null,10000, 10000);
33          } catch (Exception e) {
34              e.printStackTrace();
35          }
36          System.out.println("执行合约结果：");
37          System.out.println(responseInfo);
38      }
39
```

```
    public static void queryContract(ChainClient chainClient) {
        ResultOuterClass.TxResponse responseInfo = null;
        try {
            responseInfo =
chainClient.queryContract(CONTRACT_NAME, QUERY_CONTRACT_METHOD,
                    null,  null,10000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("查询合约结果：");
        System.out.println(responseInfo);
    }
}
```

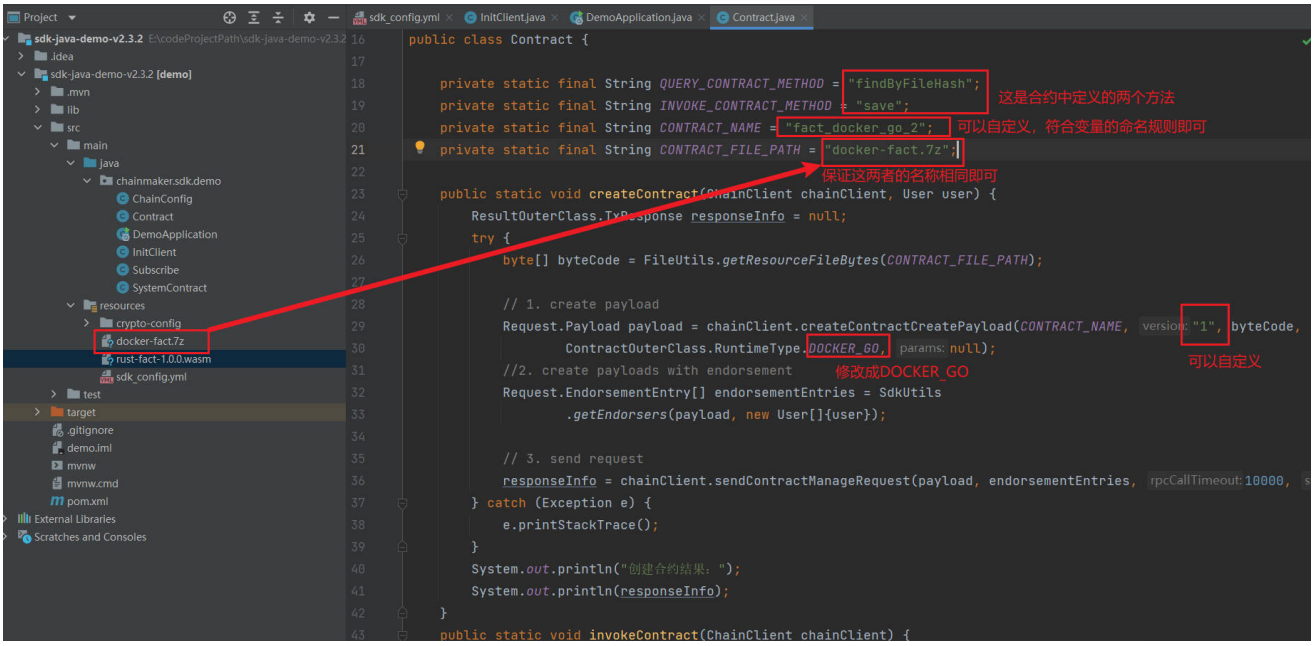
修改后的Contract.java文件

```
package chainmaker.sdk.demo;

import org.chainmaker.pb.common.ContractOuterClass;
import org.chainmaker.pb.common.Request;
import org.chainmaker.pb.common.ResultOuterClass;
import org.chainmaker.sdk.ChainClient;
import org.chainmaker.sdk.User;
import org.chainmaker.sdk.utils.FileUtils;
import org.chainmaker.sdk.utils.SdkUtils;

import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Map;


public class Contract {

    private static final String QUERY_CONTRACT_METHOD =
"findByFileHash";
    private static final String INVOKE_CONTRACT_METHOD = "save";
    private static final String CONTRACT_NAME =
"fact_docker_go_2";
    private static final String CONTRACT_FILE_PATH = "docker-
fact.7z";

```

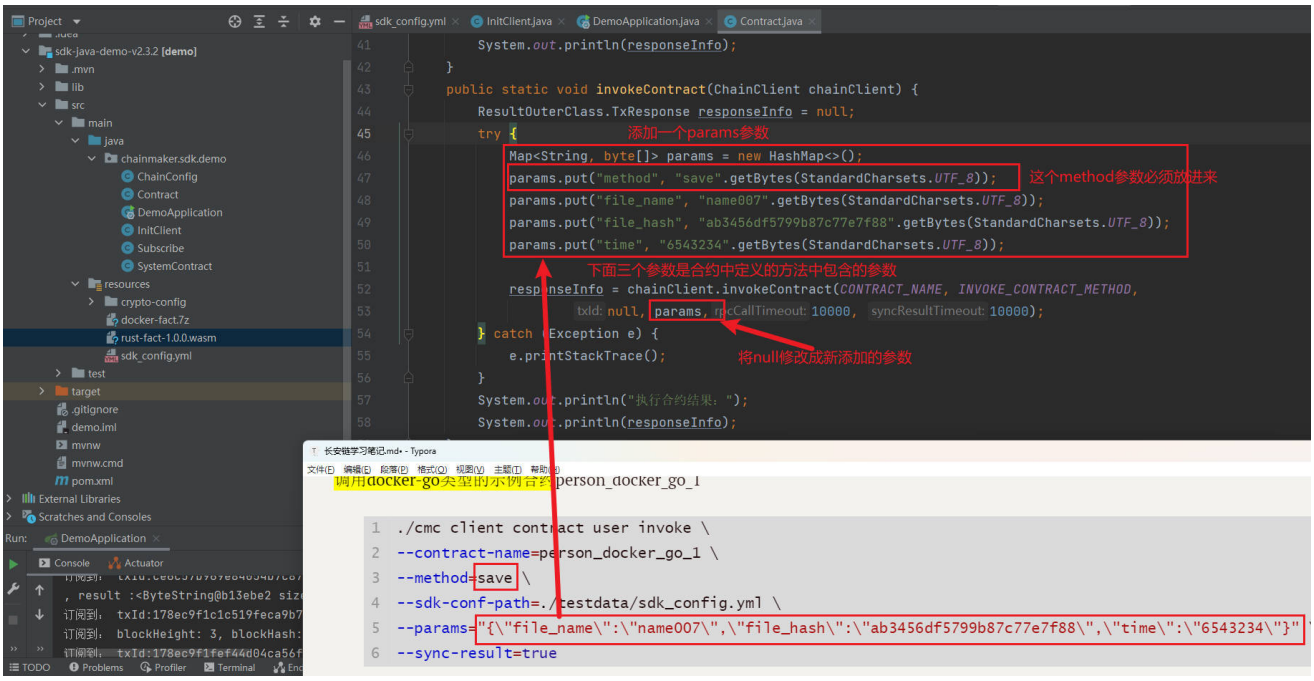
```
23      public static void createContract(ChainClient chainClient,
    User user) {
24          ResultOuterClass.TxResponse responseInfo = null;
25          try {
26              byte[] byteCode =
    FileUtils.getResourceFileBytes(CONTRACT_FILE_PATH);
27
28              // 1. create payload
29              Request.Payload payload =
    chainClient.createContractCreatePayload(CONTRACT_NAME, "1",
    byteCode,
30                      ContractOuterClass.RuntimeType.DOCKER_GO,
    null);
31              //2. create payloads with endorsement
32              Request.EndorsementEntry[] endorsementEntries =
    SdkUtils
33                      .getEndorsers(payload, new User[]{user});
34
35              // 3. send request
36              responseInfo =
    chainClient.sendContractManageRequest(payload,
    endorsementEntries, 10000, 10000);
37          } catch (Exception e) {
38              e.printStackTrace();
39          }
40          System.out.println("创建合约结果：");
41          System.out.println(responseInfo);
42      }
43      public static void invokeContract(ChainClient chainClient) {
44          ResultOuterClass.TxResponse responseInfo = null;
45          try {
46              Map<String, byte[]> params = new HashMap<>();
47              params.put("method",
    "save".getBytes(StandardCharsets.UTF_8));
48              params.put("file_name",
    "name007".getBytes(StandardCharsets.UTF_8));
49              params.put("file_hash",
    "ab3456df5799b87c77e7f88".getBytes(StandardCharsets.UTF_8));
50              params.put("time",
    "6543234".getBytes(StandardCharsets.UTF_8));
51
52              responseInfo =
    chainClient.invokeContract(CONTRACT_NAME, INVOKE_CONTRACT_METHOD,
```

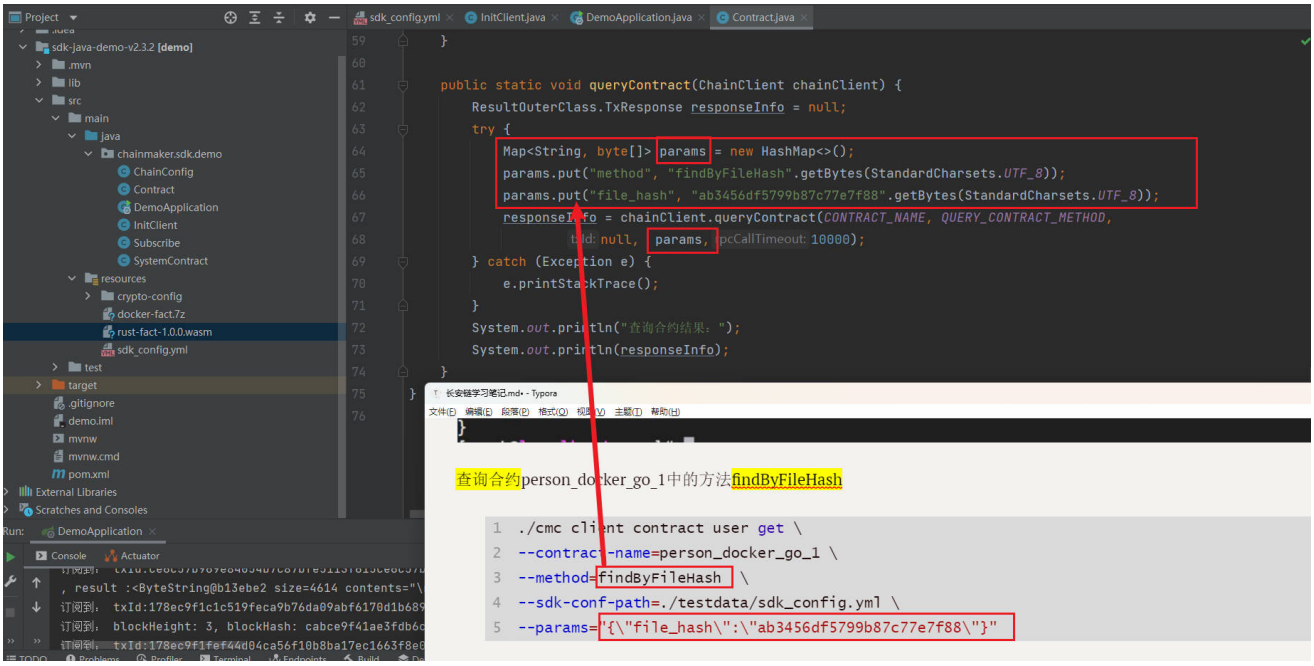
```
                    null, params,10000, 10000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("执行合约结果：");
        System.out.println(responseInfo);
    }

    public static void queryContract(ChainClient chainClient) {
        ResultOuterClass.TxResponse responseInfo = null;
        try {
            Map<String, byte[]> params = new HashMap<>();
            params.put("method",
"findByFileHash".getBytes(StandardCharsets.UTF_8));
            params.put("file_hash",
"ab3456df5799b87c77e7f88".getBytes(StandardCharsets.UTF_8));
            responseInfo =
chainClient.queryContract(CONTRACT_NAME, QUERY_CONTRACT_METHOD,
                    null,  params,10000);
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("查询合约结果：");
        System.out.println(responseInfo);
    }
}

```

修改后的createContract方法



修改后的invokeContract方法





启动，进行合约的部署、调用、查询操作

```
E:\codeProjectPath\sdk-java-demo-v2.3.2\sdk-java-demo-v2.3.2\docker-fact.7z
2023-10-17 20:05:06.810  WARN 70912 --- [           main] org.chainmaker.sdk.utils.FileUtils       : jar get file fail, msg :get file
创建合约结果:
message: "OK"
contract_result {
  result: "\n\020fact_docker_go_2\022\0011\030\b*\236\b\n\026wx-org1.chainmaker.org\032\224\a-----BEGIN CERTIFICATE-----\nMIICdzCCAh
  message: "Success"
  gas_used: 12885
}
tx_id: "178ee38d95e77d84ca99bf4f4c59e9b588ad42c6d94c49c98f7e888c99b9b305"

执行合约结果:
message: "OK"
contract_result {
  result: "name007ab3456df5799b87c77e7f88"
  message: "Success"
  gas_used: 135
  contract_event {
    topic: "topic_vx"
    tx_id: "178ee38f49414800ca5960932198f06b74ac3b6bcb41465f84b567ada4d3d0df"
    contract_name: "fact_docker_go_2"
    event_data: "ab3456df5799b87c77e7f88"
    event_data: "name007"
  }
}
tx_id: "178ee38f49414800ca5960932198f06b74ac3b6bcb41465f84b567ada4d3d0df"

查询合约结果:
message: "SUCCESS"
contract_result {
  result: "{\"FileHash\":\"ab3456df5799b87c77e7f88\",\"FileName\":\"name007\",\"time\":6543234}"
  message: "Success"
  gas_used: 128
}
tx_id: "178ee38fa1868048cac1781c392093db854b41da636e4a5dad57eaa24a5c99f4"
```