

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

```
1 vim /etc/sysconfig/network-scripts/ifcfg-ens33
2 # 在/etc/sysconfig/network-scripts/ifcfg-ens33文件中修改以下6行
3 ONBOOT=yes
4 BOOTPROTO=static
5 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
6 NETMASK="255.255.255.0"
7 GATEWAY="192.168.30.2"
8 DNS1="223.5.5.5"
```

网络和 Internet > 高级网络设置 > 查看其他属性

VMware Network Adapter VMnet8 属性

IP 分配:	手动
IPv4 地址:	192.168.30.1
IPv4 掩码:	255.255.255.0
IPv4 网关:	192.168.30.2

编辑

DNS 服务器分配:	手动
IPv4 DNS 服务器:	223.5.5.5 (未加密) 223.6.6.6 (未加密)

编辑

链接速度(接收/传输):	100/100 (Mbps)
本地连接 IPv6 地址:	fe80::880e:5569:f23:e782%4
IPv4 地址:	192.168.30.1
IPv4 DNS 服务器:	223.5.5.5 (未加密) 223.6.6.6 (未加密)
制造商:	VMware, Inc.
描述:	VMware Virtual Ethernet Adapter for VMnet8
驱动程序版本:	14.0.0.5
物理地址(MAC):	00-50-56-C0-00-08

复制

```
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"
```

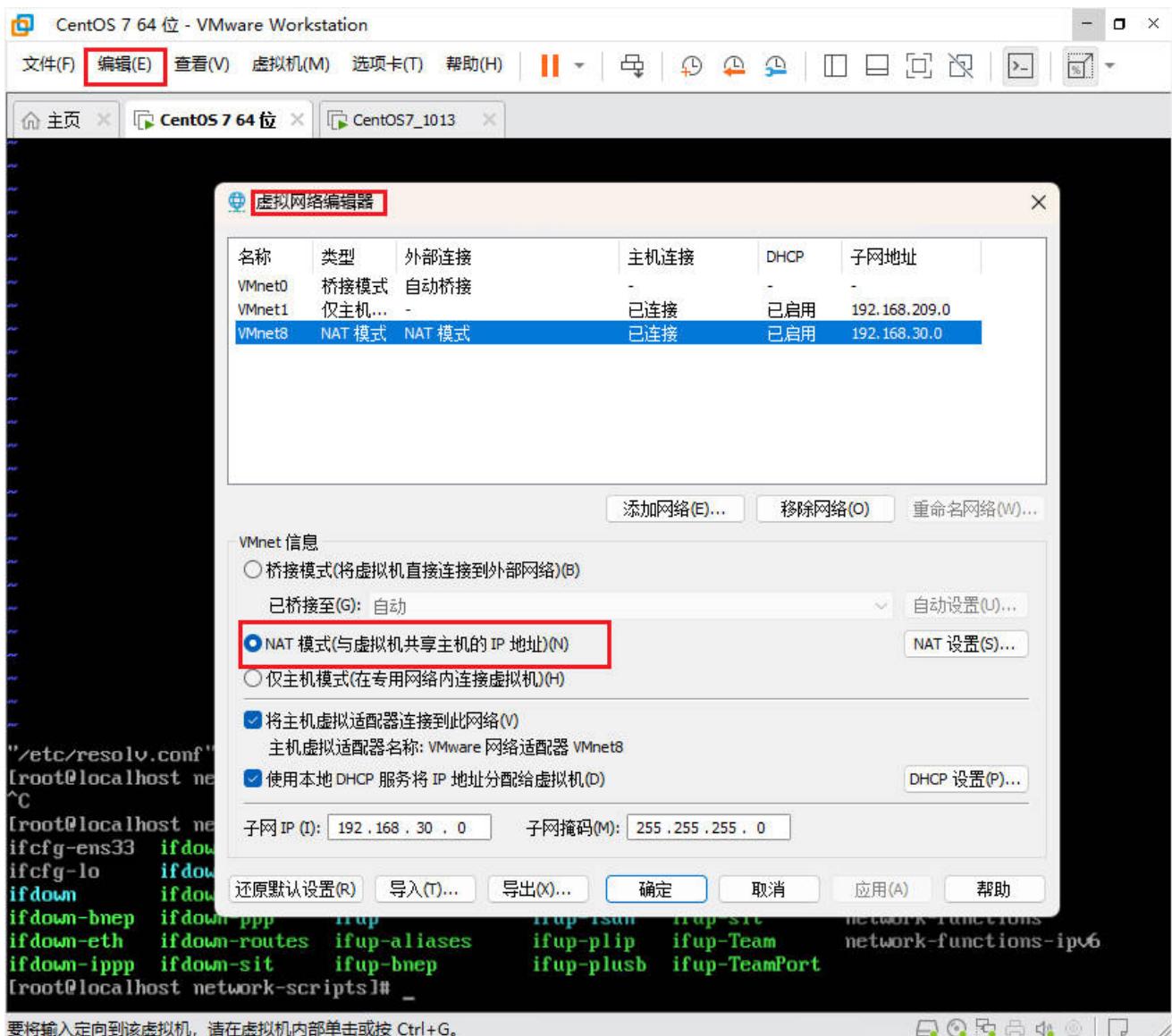
```
[root@localhost ~]# vim /etc/sysconfig/network-scripts/ifcfg-ens33
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft 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: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

```
1 # 查看防火墙状态
2 systemctl status firewalld
3 # 停止防火墙
4 systemctl stop firewalld
5 # 禁用防火墙
6 systemctl disable firewalld
```

2.3 重启网络服务

```
1 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 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

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

3.4 清除缓存更新yum列表

```
1 yum clean all
2 yum makecache
3 yum update
```

3.5 安装ifconfig

```
1 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

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

4.3 下载git源码包

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

4.4 安装git

```
1 # 解压git安装包到指定目录/usr/local/environment/git  
2 tar zxvf git-2.38.1.tar.gz -C /usr/local/environment/git-2.27.0  
3 # 配置安装路径为/usr/local/environment/git  
4 make prefix=/usr/local/environment/git-2.27.0 all  
5 make prefix=/usr/local/environment/git-2.27.0 install  
6 # 查看git版本  
7 git --version
```

```
[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

```
1 # 配置环境变量
2 vim /etc/profile
3 # 在文件末尾添加这两行
4 export GIT_HOME=/usr/local/environment/git-2.27.0
5 export PATH=.:$GIT_HOME/bin:$PATH
6 # 使文件生效
7 source /etc/profile
8 # 查看版本号
9 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

```
1 rpm --import https://mirror.go-repo.io/centos/RPM-GPG-KEY-GO-REPO
2 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

```
1 # 查询golang的可用版本（建议安装1.16版本）
2 yum --showduplicates list golang
3 yum install golang-1.16.1-0.el7
4
5 # 查看go版本
6 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）

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

5.4 配置代理

```
1 go env -w GO111MODULE=on
2 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
G0111MODULE="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_com=bugzilla --enable-bootstrap --enable-shared --enable-threads=posix --enable-check-system-zlib --enable_cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-ild-id --with-linker-hash-style=gnu --enable-languages=c,c++,objc,obj-c++,java,fortran,ada,go,lto --enable-initfini-array --disable-libgcj --with-isl=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-isl-install --with-cloog=/builddir/build/BUILD/gcc-4.8.5-20150702/obj-x86_64-redhat-linux/cloog-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)
Swap : 0 M RSS (0 MB VSZ)
```

6.2 升级gcc

```
1 yum install centos-release-scl
2 yum install devtoolset-7-gcc*
3
4 vim ~/.bash_profile
5 # 在文件~/.bash_profile中添加一行，使得下次打开新窗口也能生效
6 scl enable devtoolset-7 bash
7 # 查看gcc 版本
8 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-threads=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-only --enable-plugin --with-linker-hash-style=gnu --enable-initfini-array --with-default-libstdcxx-abi=c++-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）

```
1 uname -r
```

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

7.2 安装相关依赖

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

7.3 设置yum源

```
1 yum-config-manager --add-repo  
http://download.docker.com/linux/centos/docker-ce.repo (中央仓库)  
2  
3 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

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

7.5 安装docker

```
1 # 选择18.03.1这个版本进行安装  
2 yum -y install docker-ce-3:23.0.1-1.el7  
3 # 查看安装的docker版本  
4 docker -v 或者 docker version  
5  
6 # 卸载docker  
7 yum remove docker-ce docker-ce-cli containerd.io  
8 rm -rf /var/lib/docker  
9 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

```
1 # 启动docker
2 systemctl start docker
3 # 查看docker当前状态
4 systemctl status docker
5 # 设置开机自启
6 systemctl enable docker
```

7.7 配置docker镜像源

```
1 # 查看docker的镜像源
2 docker info
3 # 在目录/etc/docker下, 如果没有daemon.json文件, 则新建daemon.json文件, 文
4 件内容如下
5 {
6     "registry-mirrors": [
7         "https://registry.hub.docker.com",
8         "http://hub-mirror.c.163.com",
9         "https://docker.mirrors.ustc.edu.cn",
10        "https://registry.docker-cn.com"
11    ]
12 }
```

配置前

```
Kernel Version: 3.10.0-1100.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
CPUs: 1
Total Memory: 972.3MiB
Name: localhost.localdomain
ID: J4CS:DWJN:QERZ:GYYW:IGM5:YEAL:NALI:MRQD:M5II:2V40:JHZE:SCY3
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/ 默认的docker镜像源
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
```

配置后

```
Name: localhost.localdomain
ID: J4CS:DWJN:QERZ:GYYW:IGM5:YEAL:NALI:MRQD:M5II:2V40:JHZE:SCY3
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
 127.0.0.0/8
Registry Mirrors:
 https://registry.hub.docker.com/
 http://hub-mirror.c.163.com/
 https://docker.mirrors.ustc.edu.cn/
 https://registry.docker-cn.com/
Live Restore Enabled: false
```

7.8 重载守护进程，重启docker

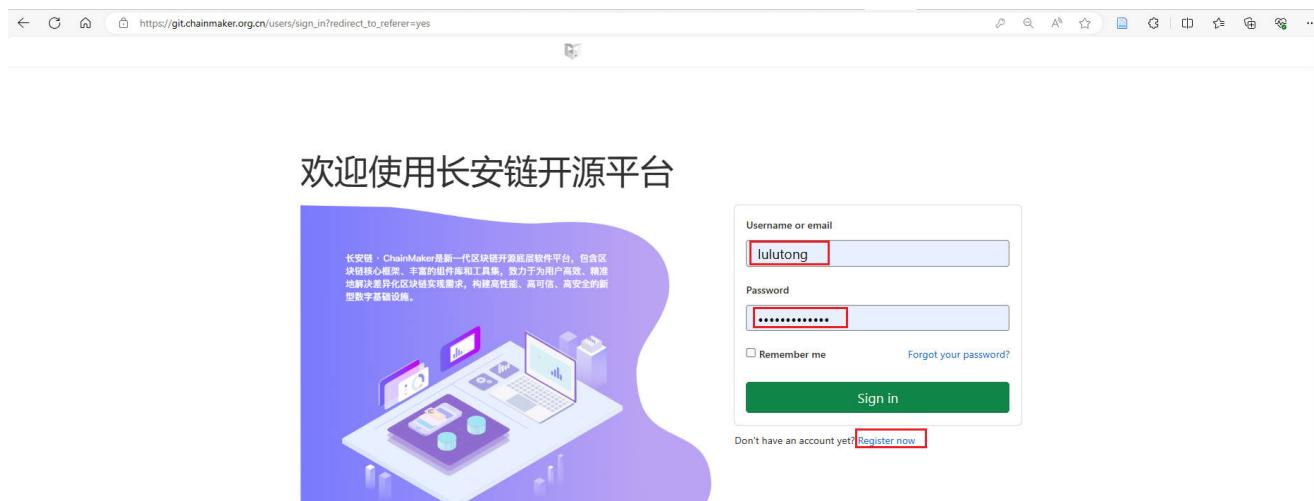
```
1 systemctl daemon-reload  
2 systemctl stop docker  
3 systemctl start docker
```

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

```
1 # 下载chainmaker-go源码到本地  
2 git clone -b v2.3.2 --depth=1  
  https://git.chainmaker.org.cn/chainmaker/chainmaker-go.git  
3 # 下载证书生成工具源码到本地  
4 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 需要提前在长安链官网注册账户
Password for 'https://lulutong@git.chainmaker.org.cn': [REDACTED]
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'.
```

```
[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

```
1 # 编译证书生成工具(这个步骤需要等待的时间较长)
2 cd chainmaker-cryptogen
3 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

```
1 # 将编译好的chainmaker-cryptogen, 软连接到chainmaker-go/tools目录
2 ## 进入工具目录
3 cd chainmaker-go/tools
4 ## 软连接chainmaker-cryptogen到tools目录下
5 ln -s ../../chainmaker-cryptogen/ .
6
7 # 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
8 ## 进入脚本目录
```

```
9 cd ../scripts
10 ## 生成单链4节点集群的证书和配置
11 ./prepare.sh 4 1
12 # 查看生成好的节点证书和配置
13 yum install tree
14 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_pkw.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
    ├── node2
    │   ├── certs
    │   ├── chainconfig
    │   └── chainmaker.yml
    ├── node3
    │   ├── certs
    │   ├── chainconfig
    │   └── chainmaker.yml
    └── node4
        ├── certs
        ├── chainconfig
        └── chainmaker.yml
```

8.1.5 编译及安装包制作

```
1 # 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)
2 ./build_release.sh
3 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

```
1 # 执行cluster_quick_start.sh脚本，会解压各个安装包，调用bin目录中的
2   start.sh脚本，启动chainmaker节点
3 ./cluster_quick_start.sh normal
4 # 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
5 mkdir -p ..../build/bak
6 mv ..../build/release/*.tar.gz ..../build/bak
7 # 若需要关闭集群，使用脚本：
8 ./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/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

```
1 # 查看节点启动使用正常
2
3 ## 查看进程是否存在
4 ps -ef |grep chainmaker | grep -v grep
5 ## 查看端口是否监听
6 netstat -lptn | grep 1230
```

```

7 ## 检查节点是否有ERROR日志
8 cat ./build/release/*/bin/panic.log
9 cat ./build/release/*/log/system.log # 这个日志文件的内容较多
10 cat ./build/release/*/log/system.log |grep "ERROR\|put
block\|all necessary"
11 ==如果看到all necessary peers connected则表示节点已经准备就绪。 ==
12
13 # 关闭某个进程占用的端口
14 netstat -anp | grep 8080
15 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/chainma
ker.yml name-20231017111104
root 3773 1 1 11:26 pts/0 00:00:18 ./chainmaker start -c ../config/wx-org2.chainmaker.org/chainma
ker.yml name-20231017111105
root 4952 1 1 11:26 pts/0 00:00:17 ./chainmaker start -c ../config/wx-org3.chainmaker.org/chainma
ker.yml name-20231017111106
root 6164 1 1 11:26 pts/0 00:00:18 ./chainmaker start -c ../config/wx-org4.chainmaker.org/chainma
ker.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
[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: LibP2
P
2023-10-17 11:25:58.841 [INFO] [Blockchain] blockchain/chainmaker_server.go:112 load net tls key file pa
th: /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 [QmW5scT
FvfYpqzu8YpYqGocEbpXvfU4N4P1anNmFREsMef](1/27/PROPOSE) send state to [QmcgjQUNGlu6SmTMSAChvDhjCguryEPzEK7Cu3jXUz
NJxG](1/27/PROPOSE)
2023-10-17 11:46:26.585 [INFO] [Consensus] @chain1 v2@v2.3.3/tbft_status_broadcaster.go:194 [QmW5scT
FvfYpqzu8YpYqGocEbpXvfU4N4P1anNmFREsMef](1/27/PROPOSE) send state to [QmNa5xTUyZhEjZpwjch1qJtq3QzfY7dEdFKTcG3LwU
3LmU](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]: p
ut 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 nec
essary peers connected.
2023-10-17 11:26:03.484 [INFO] [Storage] @chain1 v2@v2.3.4/blockstore_impl.go:304 chain[chain1]: p
ut 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 nec
essary peers connected.
2023-10-17 11:26:07.536 [INFO] [Storage] @chain1 v2@v2.3.4/blockstore_impl.go:304 chain[chain1]: p
ut 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 nec
essary peers connected.
2023-10-17 11:26:11.416 [INFO] [Storage] @chain1 v2@v2.3.4/blockstore_impl.go:304 chain[chain1]: p
ut 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 nec
essary 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

```
1 # 编译cmc
2 cd /home/chainv2.3.2/chainmaker-go/tools/cmc
3 go build
4 # 配置测试数据
5 cp -rf ../../build/crypto-config ../../tools/cmc/testdata/ # 使用
chainmaker-cryptogen生成的测试链的证书
6 # 查看help
7 ./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
[root@localhost cmc]#
```

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

```

1 ./cmc client contract user create \
2 --contract-name=fact_wasmer_1 \
3 --runtime-type=WASMER \
4 --byte-code-path=./testdata/claim-wasm-demo/rust-fact-2.0.0.wasm \
\
5 --version=1.0 \
6 --sdk-conf-path=./testdata/sdk_config.yml \
7 --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 \
8 --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 \
9 --sync-result=true \
10 --params={}

```

```

[root@localhost cmc]# ./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={}
{
  "contract_result": {
    "gas_used": 11607,
    "result": {
      "address": "dc1d1ebeblebe795d780ea59f971e8674f1bad7f",
      "creator": {
        "member_id": "client1.sign.wx-org1.chainmaker.org",
        "member_info": "TGRQqa0Jjdf7dGnf0oIhxoGwxvENYtd8HN0EHhIlb0=",
        "member_type": 1,
        "org_id": "wx-org1.chainmaker.org",
        "role": "CLIENT",
        "uid": "8cf30b716cldfcf024fe79b5ea839f9c3fe1dfe877cbfdd02d19c5e6ef51eb88"
      },
      "name": "fact_wasmer_1",
      "runtime_type": 2,
      "version": "1.0"
    }
  },
  "tx_block_height": 2,
  "tx_id": "178ec9f1fef44d04ca56f10b8ba17ec1663f8e0484c7454aa628bec6f7475426",
  "tx_timestamp": 1697516150
}

```

调用wasm类型的示例合约fact_wasmer_1

```

1 ./cmc client contract user invoke \
2 --contract-name=fact_wasmer_1 \
3 --method=save \
4 --sdk-conf-path=./testdata/sdk_config.yml \
5 --params="\
6   {"file_name": "name007", "file_hash": "ab3456df5799b87c77e7f88", "time": "6543234"}" \
6 --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": "178eca209c9dc46dca91c130fa8bc01de12cafdee234abaae480e04eba25afd"
      }
    ],
    "gas_used": 238549
  },
  "tx_block_height": 3,
  "tx_id": "178eca209c9dc46dca91c130fa8bc01de12cafdee234abaae480e04eba25afd",
  "tx_timestamp": 1697516350
}

```

查询合约fact_wasmer_1中的方法find_by_file_hash

```

1 ./cmc client contract user get \
2 --contract-name=fact_wasmer_1 \
3 --method=find_by_file_hash \
4 --sdk-conf-path=./testdata/sdk_config.yml \
5 --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": "eyJmaWxlXhhc2giOiJhYjM0NTZkZjU3OTliODdjNzdlN2Y40CIiImZpbGVfbmFtZSI6Im5hbWUwMDciLCJ0aWliIjo2NTQzMjM0fQ=="
  },
  "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

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

```

[root@localhost cmc]# ls
address      cert   command_util  hibe          main.go  payload  README.md    tee        types
archive      client  console     key           paillier  pubkey  sdk.log      testdata  util
bulletproofs  cmc    gas         loop-transfer.sh parallel query   sdk.log.2023101712 txpool   version
[root@localhost cmc]# cd testdata/
[root@localhost testdata]# ls
archivecenter      claim-docker-go-demo  hibe-data          remote_attestation  sdk_config.yml
asset-wasm-demo   claim-wasm-demo       hibe-wasm-demo    sdk_config_pk.yml  storage-evm-demo
balance-evm-demo   counter-go-demo     paillier-key     sdk_config_pkw.yml token-evm-demo
bulletproofs-wasm-demo crypto-to-config paillier-wasm-demo sdk_config_solo.yml trust-member-demo
[root@localhost testdata]# cd claim-docker-go-demo/
[root@localhost claim-docker-go-demo]# ls
docker-fact.7z
[root@localhost claim-docker-go-demo]# pwd
/home/chaininv2.3.2/chainmaker-go/tools/cmc/testdata/claim-docker-go-demo
[root@localhost claim-docker-go-demo]# cd ..
[root@localhost testdata]# cd ..
[root@localhost cmc]# ./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={}
{
  "contract_result": {
    "gas_used": 12885,
    "message": "Success",
    "result": {
      "address": "16020add30faea16b1c620bf73a941278e7ac9df",
      "creator": {
        "member_id": "client1.sign.wx-org1.chainmaker.org",
        "member_info": "2TGRQqa0Jjdf7dGnf0oIhx0GwxvENYtd8HN0EHHI1b0",
        "member_type": 1,
        "org_id": "wx-org1.chainmaker.org",
        "role": "CLIENT",
        "uid": "8cf30b7154dcfc024fe79b5ea839f9c3fe1dfe877cbfdd02d19c5e6ef51eb88"
      },
      "name": "person_docker_go_1",
      "runtime_type": 8,
      "version": "1.0"
    }
  },
  "tx_block_height": 4,
  "tx_id": "178eca615cc41306ca62b4d1f972e5bc145853ac9c844b4db482b826ce33865a",
  "tx_timestamp": 1697516629
}

```

调用docker-go类型的示例合约 person_docker_go_1

```

1  ./cmc client contract user invoke \
2  --contract-name=person_docker_go_1 \
3  --method=save \
4  --sdk-conf-path=./testdata/sdk_config.yml \
5  --params="
6    {"file_name": "name007", "file_hash": "ab3456df5799b87c77e7f88",
7     "time": "6543234"} \
8  --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": "bmFtZTAwN2FiMzQ1NmRmNTc50WI4N2M3N2U3Zjg4"
  },
  "tx_block_height": 5,
  "tx_id": "178eca9a697aa352ca65f3e71a48e69e7dfdc0ff727343229ed64d409753b950",
  "tx_timestamp": 1697516874
}
```

查询合约person_docker_go_1中的方法`findByFileHash`

```
1 ./cmc client contract user get \
2 --contract-name=person_docker_go_1 \
3 --method=findByFileHash \
4 --sdk-conf-path=./testdata/sdk_config.yml \
5 --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", 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
    "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": "eyJGaWxlSGFzaCI6ImFiMzQ1NmRmNTc50WI4N2M3N2U3Zjg4IiwiRmlsZU5hbWUiOiJuYWI1MDA3IiwidGltZSI6NjU0MzIzNH0="
  },
  "message": "SUCCESS",
  "tx_id": "178ecad26afdf5d87cae20d304730be108067418250b948b0a84177d7aaebbd5"
```

问题9

9.1 编写Golang合约

```
1 package main
2
3 // 合约结构体，合约名称需要写入main()方法当中
4 type FactContract struct {
5 }
6
7 // 合约必须实现下面两个方法：
8 // InitContract() protogo.Response
9 // UpgradeContract() protogo.Response
10 // InvokeContract(method string) protogo.Response
11
12 // 用于合约的部署
13 // @return:      合约返回结果，包括Success和Error
14 func (f *FactContract) InitContract() protogo.Response {
15     return sdk.Success([]byte("Init contract success"))
16 }
17
18 // 用于合约的升级
19 // @return:      合约返回结果，包括Success和Error
20 func (f *FactContract) UpgradeContract() protogo.Response {
21     return sdk.Success([]byte("Upgrade contract success"))
22 }
23
24 // 用于合约的调用
25 // @param method: 交易请求调用的方法
26 // @return:      合约返回结果，包括Success和Error
27 func (f *FactContract) InvokeContract(method string)
28     protogo.Response {
29     switch method {
30     case "save":
31         return f.save()
32     case "findByFileHash":
33     default:
34         return sdk.Error("invalid method")
35     }
36 }
37
38 // sdk代码中，有且仅有一个main()方法
39 func main() {
40     // 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
```

```
41 // 其中，TestContract为用户实现合约的具体名称
42     err := sandbox.Start(new(FactContract))
43     if err != nil {
44         log.Fatal(err)
45     }
46 }
```

9.2 编写编译合约的脚本

```
1 # build.sh
2 #!/bin/bash
3
4 contractName=$1
5 if [[ ! -n $contractName ]]; then
6     echo "contractName is empty. use as: ./build.sh contractName"
7     exit 1
8 fi
9
10 go build -ldflags="-s -w" -o $contractName
11
12 7z a $contractName $contractName
13 rm -f $contractName
```

9.3 安装7z

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

9.4 编译合约

```
1 # 给脚本文件添加执行的权限
2 chmod +x build.sh
3 ./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>)

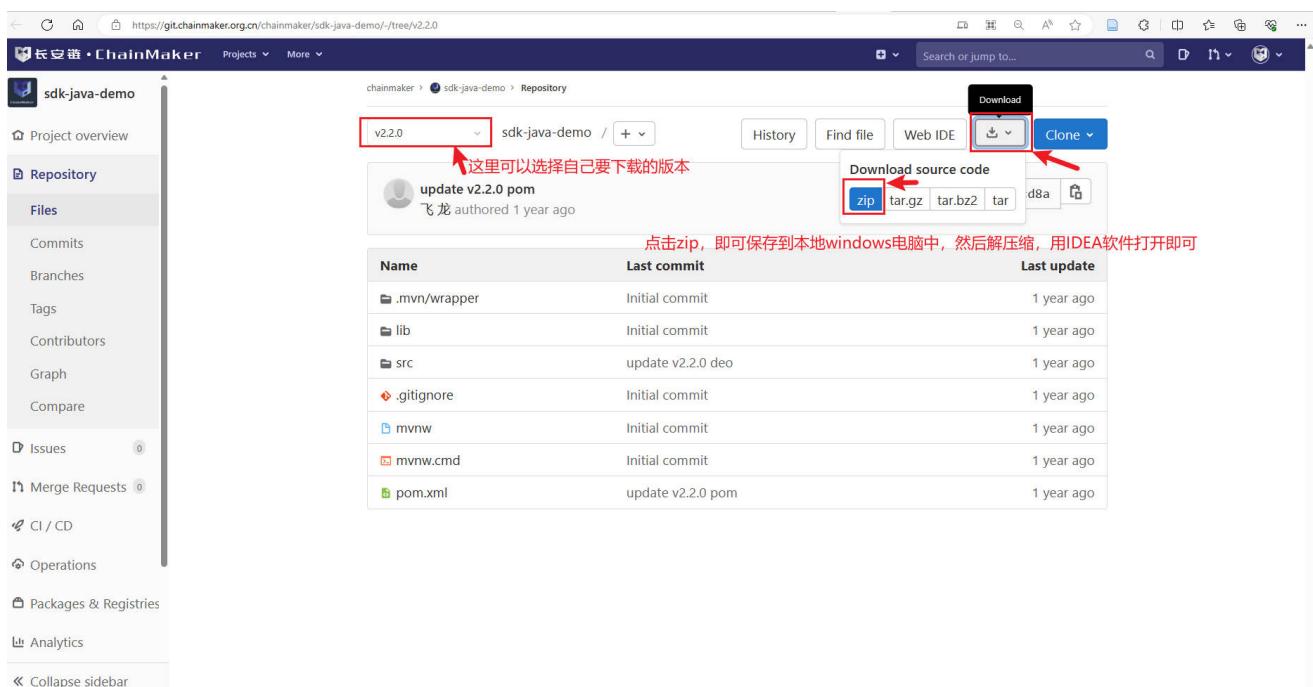
10.1.1 环境依赖(jdk安装在本地windows)

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

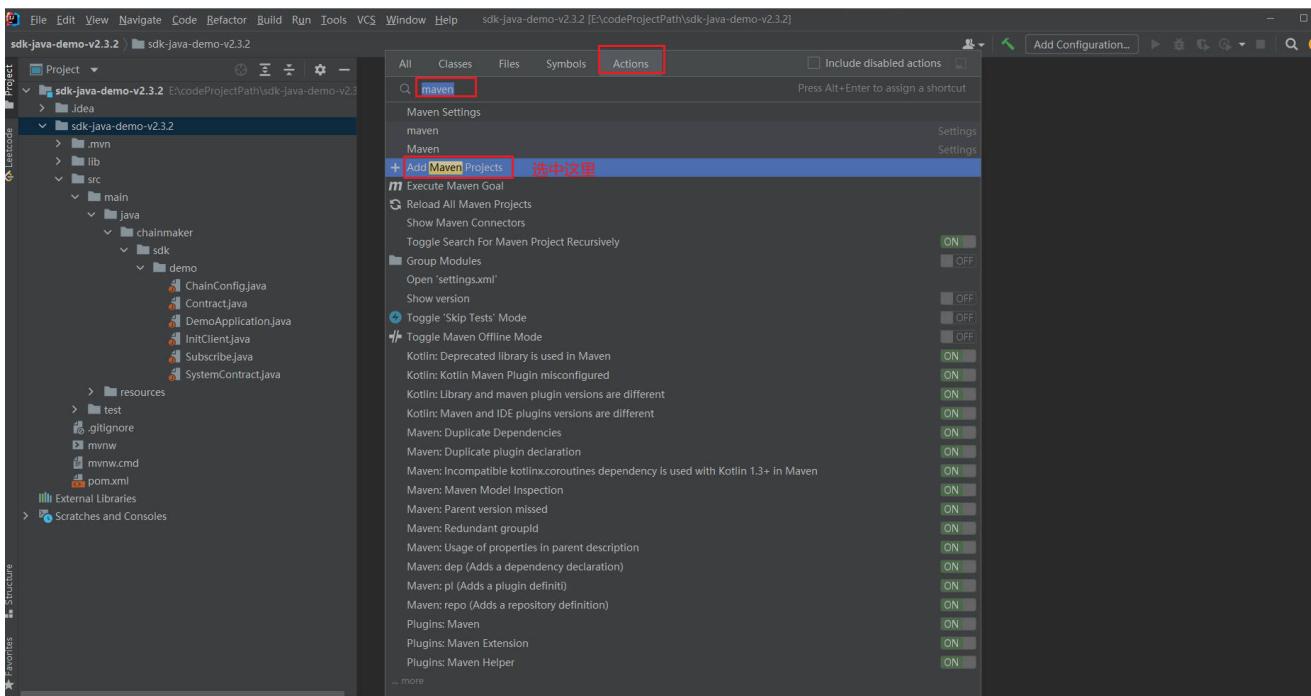
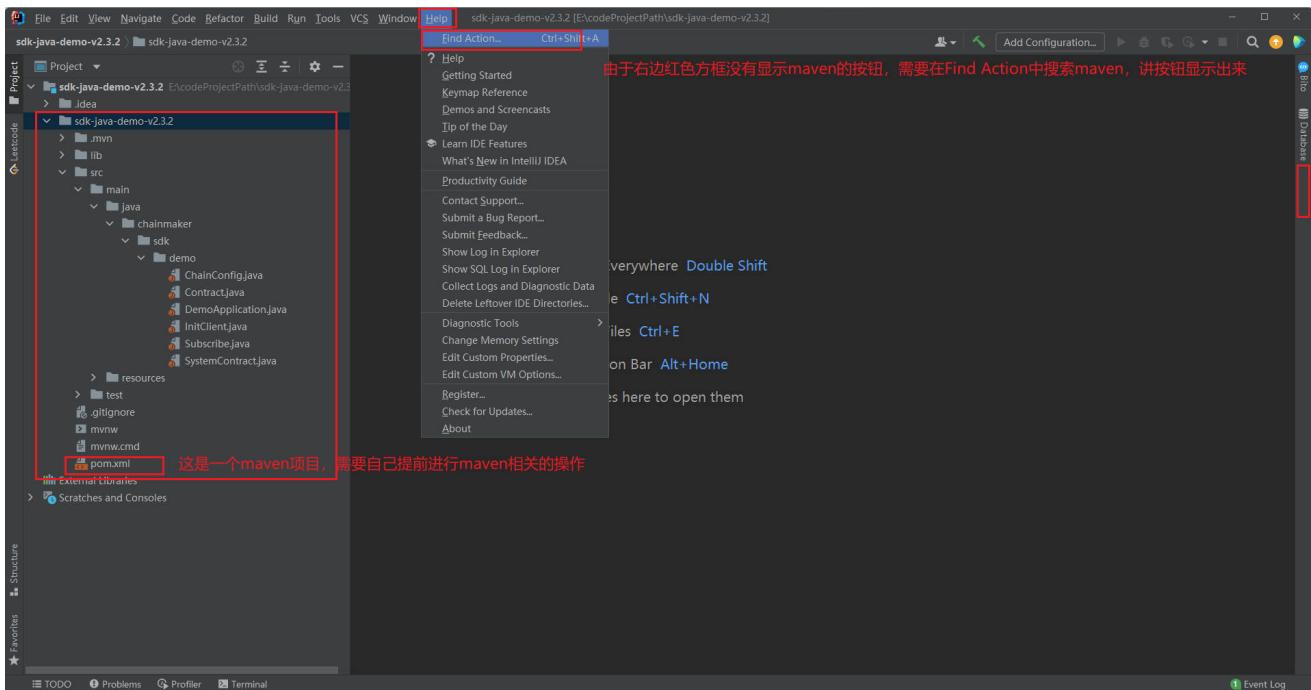
```
E:\>java -version
java version "1.8.0_131"
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

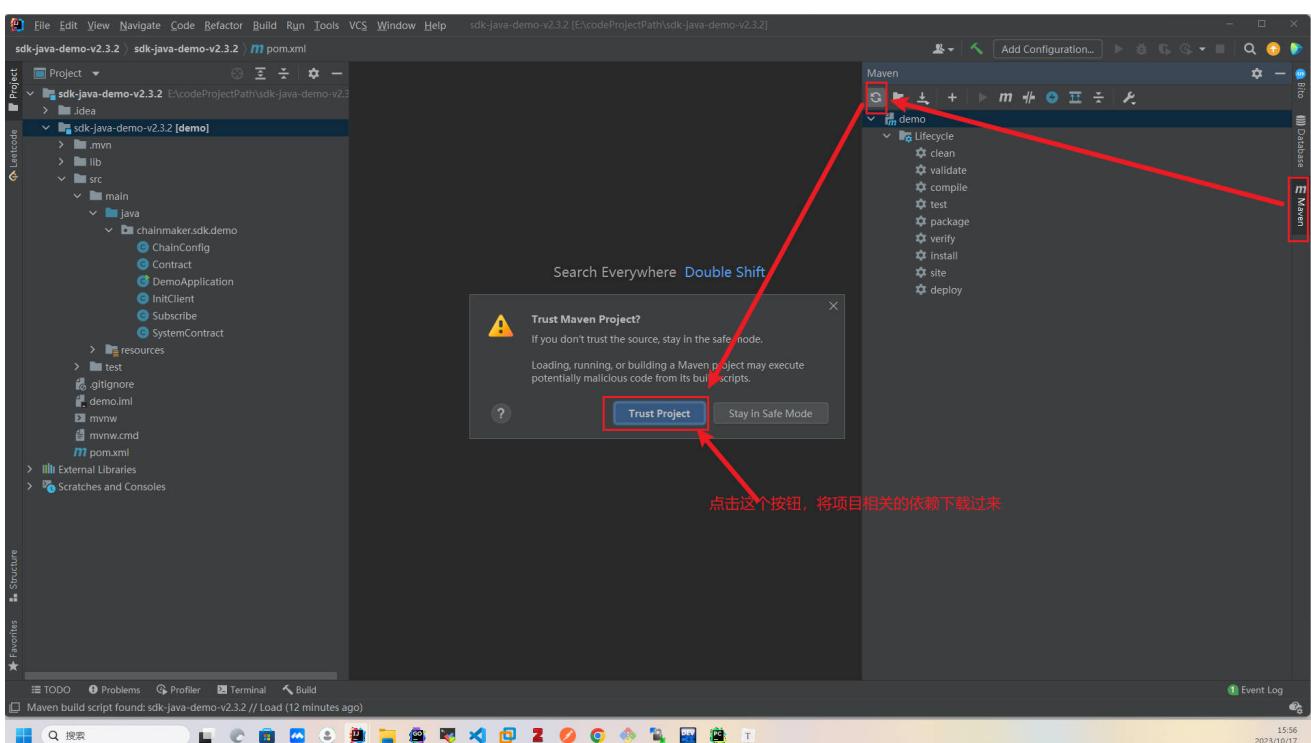
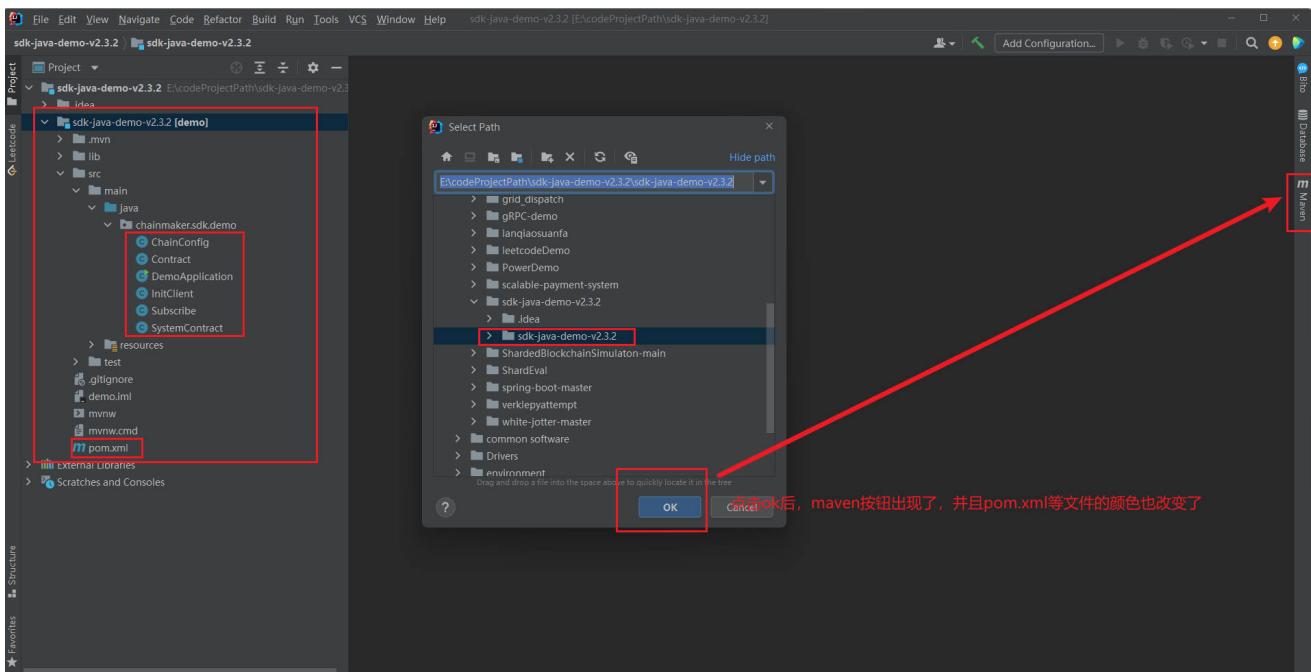
10.1.2 应用demo下载到本地windows电脑

```
1 # 下载地址
2 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整个目录的文件替换掉
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]#
```

用解压后的文件替换到这个目录中的文件

名称	修改日期	类型	大小
wx-org1.chainmaker.org	2023/10/17 10:59	文件夹	
wx-org2.chainmaker.org	2023/10/17 10:59	文件夹	
wx-org3.chainmaker.org	2023/10/17 10:59	文件夹	
wx-org4.chainmaker.org	2023/10/17 10:59	文件夹	

10.1.5 修改resources目录下的sdk_config.yml文件

修改前的sdk_config.yml文件

```
1 chain_client:
2   # 链ID
3   chain_id: "chainmaker_testnet_chain"
4   # 组织ID
5   org_id: "org5.cmtestnet"
6   # 客户端用户私钥路径
7   user_key_file_path: "src/main/resources/crypto-
8     config/org5.cmtestnet/user/client/ytf002.tls.key"
9   # 客户端用户证书路径
10  user_crt_file_path: "src/main/resources/crypto-
11    config/org5.cmtestnet/user/client/ytf002.tls.crt"
12  # 客户端用户交易签名私钥路径(若未设置, 将使用user_key_file_path)
13  user_sign_key_file_path: "src/main/resources/crypto-
14    config/org5.cmtestnet/user/client/ytf002.sign.key"
15  # 客户端用户交易签名证书路径(若未设置, 将使用user_crt_file_path)
16  user_sign_crt_file_path: "src/main/resources/crypto-
17    config/org5.cmtestnet/user/client/ytf002.sign.crt"
18
19  retry_limit: 10
20  # 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
21  <=0则使用默认值 500
22  retry_interval: 500
23  # 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
24  # 否已添加此别名, 如果没有则自动上链此证书别名,
25  # 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。
26  # alias: mycert5
27
28 nodes:
29   - # 节点地址, 格式为: IP:端口:连接数
30     node_addr: "certnode1.chainmaker.org.cn:13301"
31     # 节点连接数
32     conn_cnt: 10
33     # RPC连接是否启用双向TLS认证
34     enable_tls: true
35     # 信任证书池路径
36     trust_root_paths:
37       - "src/main/resources/crypto-
38         config/org5.cmtestnet/ca/org1.cmtestnet"
39     # TLS hostname
40     tls_host_name: "consensus1.tls.org1.cmtestnet"
```

```
33      - # 节点地址, 格式为: IP:端口:连接数
34          node_addr: "certnode2.chainmaker.org.cn:13302"
35          # 节点连接数
36          conn_cnt: 10
37          # RPC连接是否启用双向TLS认证
38          enable_tls: true
39          # 信任证书池路径
40          trust_root_paths:
41              - "src/main/resources/crypto-
config/org5.cmtestnet/ca/org2.cmtestnet"
42          # TLS hostname
43          tls_host_name: "consensus1.tls.org2.cmtestnet"
44      - # 节点地址, 格式为: IP:端口:连接数
45          node_addr: "certnode3.chainmaker.org.cn:13303"
46          # 节点连接数
47          conn_cnt: 10
48          # RPC连接是否启用双向TLS认证
49          enable_tls: true
50          # 信任证书池路径
51          trust_root_paths:
52              - "src/main/resources/crypto-
config/org5.cmtestnet/ca/org3.cmtestnet"
53          # TLS hostname
54          tls_host_name: "consensus1.tls.org3.cmtestnet"
55      - # 节点地址, 格式为: IP:端口:连接数
56          node_addr: "certnode4.chainmaker.org.cn:13303"
57          # 节点连接数
58          conn_cnt: 10
59          # RPC连接是否启用双向TLS认证
60          enable_tls: true
61          # 信任证书池路径
62          trust_root_paths:
63              - "src/main/resources/crypto-
config/org5.cmtestnet/ca/org4.cmtestnet"
64          # TLS hostname
65          tls_host_name: "consensus1.tls.org4.cmtestnet"
66      archive:
67          # mysql archivecenter 归档中心 数据归档链外存储相关配置
68          type: "mysql"
69          dest: "root:123456:localhost:3306"
70          secret_key: xxx
71          # 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
72          # 如果是false,则不访问归档中心,只访问链数据
```

```
73 archive_center_query_first: false
74 # 归档中心
75 archive_center_config:
76   chain_genesis_hash:
77     a23a0c6fd402010efbc31741b2a868c9e6558471fda1e8d1f16673ef683e06db
78     archive_center_http_url: http://127.0.0.1:13119
79     request_second_limit: 10
80     rpc_address: 127.0.0.1:13120
81     tls_enable: false
82     tls:
83       server_name: archiveserver1.tls.wx-org.chainmaker.org
84       priv_key_file:
85         src/test/resources/archivecenter/archiveclient1.tls.key
86       cert_file:
87         src/test/resources/archivecenter/archiveclient1.tls.crt
88       trust_ca_list:
89         - src/test/resources/archivecenter/ca
90     max_send_msg_size: 200
91     max_recv_msg_size: 200
92
93
94   rpc_client:
95     # grpc客户端最大接受容量(MB)
96     max_receive_message_size: 16
97
98   pkcs11:
99     enabled: false # pkcs11 is not used by default
100
101   # 交易结果是否订阅获取
102   enable_tx_result_dispatcher: true
103
104   ##连接池配置
105   connPool:
106     # 最大连接数
107     maxTotal: 100
108     # 最少空闲连接
109     minIdle: 1
110     #最大空闲连接
111     maxIdle: 20
112     #连接空闲最小保活时间, 默认即为30分钟(18000000), 单位: ms
113     minEvictableIdleTime: 350000
114     #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
115     timeBetweenEvictionRuns: 60000
```

```
112      #没有空闲连接时，获取连接是否阻塞
113      blockWhenExhausted: true
114      #当没有空闲连接时，获取连接阻塞等待时间，单位：ms
115      maxWaitMillis: 3000
116
117
```

修改后的sdk_config.yml文件(用# xxx标记修改处)

```
1 chain_client:
2     # 链ID
3     chain_id: "chain1" # xxx
4     # 组织ID
5     org_id: "wx-org1.chainmaker.org" # xxx
6     # 客户端用户私钥路径
7     user_key_file_path: "src/main/resources/crypto-config/wx-
org1.chainmaker.org/user/client1/client1.tls.key" # xxx
8     # 客户端用户证书路径
9     user_crt_file_path: "src/main/resources/crypto-config/wx-
org1.chainmaker.org/user/client1/client1.tls.crt" # xxx
10    # 客户端用户交易签名私钥路径(若未设置，将使用user_key_file_path)
11    user_sign_key_file_path: "src/main/resources/crypto-config/wx-
org1.chainmaker.org/user/client1/client1.sign.key" # xxx
12    # 客户端用户交易签名证书路径(若未设置，将使用user_crt_file_path)
13    user_sign_crt_file_path: "src/main/resources/crypto-config/wx-
org1.chainmaker.org/user/client1/client1.sign.crt" # xxx
14
15    retry_limit: 10
16    # 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
17    <=0则使用默认值 500
18    retry_interval: 500
19    # 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
20    # 否已添加此别名，如果没有则自动上链此证书别名。
21    alias: mycert5
22    nodes:
23        - # 节点地址，格式为：IP:端口:连接数
24          node_addr: "192.168.30.128:12301" # xxx
25          # 节点连接数
26          conn_cnt: 10
27          # RPC连接是否启用双向TLS认证
28          enable_tls: true
```

```
28      # 信任证书池路径
29      trust_root_paths:
30          - "src/main/resources/crypto-config/wx-
31            org1.chainmaker.org/ca" # xxx
32          # TLS hostname
33          tls_host_name: "chainmaker.org" # xxx
34          - # 节点地址, 格式为: IP:端口:连接数
35          node_addr: "192.168.30.128:12302" # xxx
36          # 节点连接数
37          conn_cnt: 10
38          # RPC连接是否启用双向TLS认证
39          enable_tls: true
40          # 信任证书池路径
41          trust_root_paths:
42              - "src/main/resources/crypto-config/wx-
43                org1.chainmaker.org/ca" # xxx
44              # TLS hostname
45              tls_host_name: "chainmaker.org" # xxx
46              - # 节点地址, 格式为: IP:端口:连接数
47              node_addr: "192.168.30.128:12303" # xxx
48              # 节点连接数
49              conn_cnt: 10
50              # RPC连接是否启用双向TLS认证
51              enable_tls: true
52              # 信任证书池路径
53              trust_root_paths:
54                  - "src/main/resources/crypto-config/wx-
55                    org1.chainmaker.org/ca" # xxx
56                  # TLS hostname
57                  tls_host_name: "chainmaker.org" # xxx
58                  - # 节点地址, 格式为: IP:端口:连接数
59                  node_addr: "192.168.30.128:12304" # xxx
60                  # 节点连接数
61                  conn_cnt: 10
62                  # RPC连接是否启用双向TLS认证
63                  enable_tls: true
64                  # 信任证书池路径
65                  trust_root_paths:
66                      - "src/main/resources/crypto-config/wx-
67                        org1.chainmaker.org/ca" # xxx
68                      # TLS hostname
69                      tls_host_name: "chainmaker.org" # xxx
70                      archive:
```

```
67      # mysql archivecenter 归档中心 数据归档链外存储相关配置
68      type: "mysql"
69      dest: "root:123456:localhost:3306"
70      secret_key: xxx
71      # 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
72      # 如果是false,则不访问归档中心,只访问链数据
73      # archive_center_query_first: false # xxx
74      # # 归档中心
75      # archive_center_config:
76      #   chain_genesis_hash:
77      #     a23a0c6fd402010efbc31741b2a868c9e6558471fda1e8d1f16673ef683e06db
78      #   archive_center_http_url: http://192.168.30.128:13119
79      #   request_second_limit: 10
80      #   rpc_address: 192.168.30.128:13120
81      #   tls_enable: false
82      #   tls:
83      #     server_name: archiveserver1.tls.wx-org.chainmaker.org
84      #     priv_key_file:
85      #       src/test/resources/archivecenter/archiveclient1.tls.key
86      #     cert_file:
87      #       src/test/resources/archivecenter/archiveclient1.tls.crt
88      #     trust_ca_list:
89      #       - src/test/resources/archivecenter/ca
90      #   max_send_msg_size: 200
91      #   max_recv_msg_size: 200 # xxx
92
93
94      rpc_client:
95          # grpc客户端最大接受容量(MB)
96          max_receive_message_size: 16
97
98          pkcs11:
99              enabled: false # pkcs11 is not used by default
100
101         # 交易结果是否订阅获取
102         enable_tx_result_dispatcher: true
103
104         ##连接池配置
105         connPool:
106             # 最大连接数
107             maxTotal: 100
108             # 最少空闲连接
109             minIdle: 1
110             #最大空闲连接
```

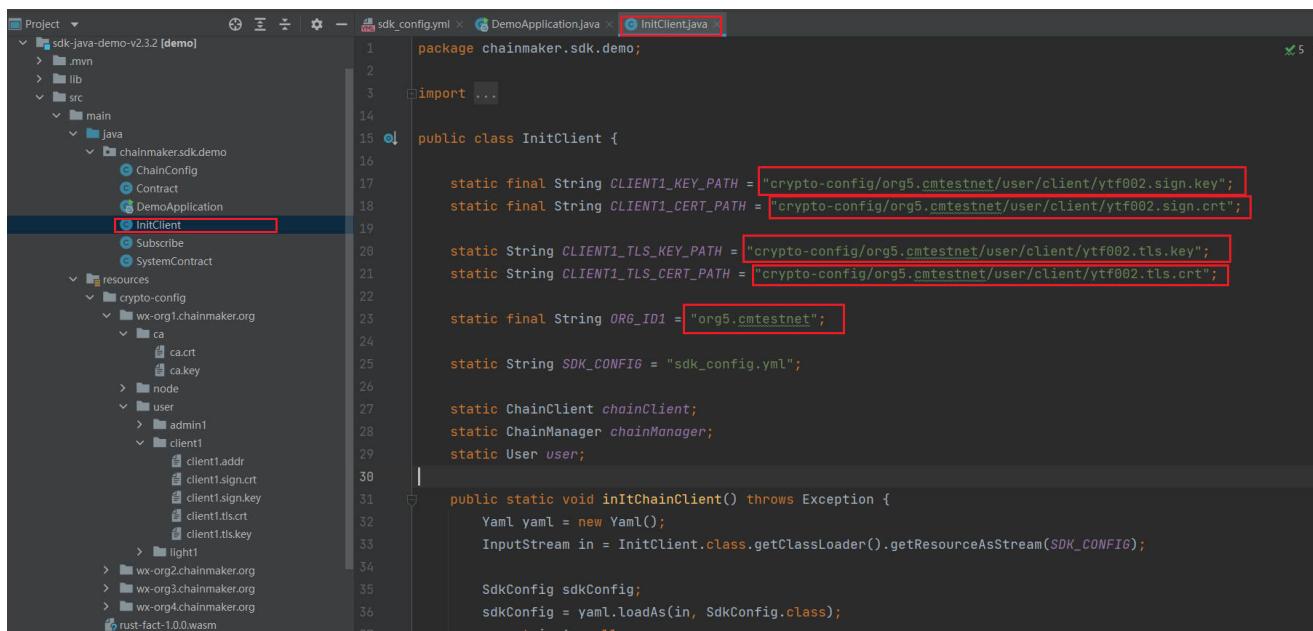
```

107     maxIdle: 20
108     #连接空闲最小保活时间， 默认即为30分钟(18000000)， 单位: ms
109     minEvictableIdleTime: 350000
110     #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
111     timeBetweenEvictionRuns: 60000
112     #没有空闲连接时，获取连接是否阻塞
113     blockWhenExhausted: true
114     #当没有空闲连接时，获取连接阻塞等待时间，单位: ms
115     maxWaitMillis: 3000
116

```

10.1.6 修改initClient.java文件

要修改的部分如下



```

package chainmaker.sdk.demo;
import ...
public class InitClient {
    static final String CLIENT1_KEY_PATH = "crypto-config/org5.cmtestnet/user/client/ytfo02.sign.key";
    static final String CLIENT1_CERT_PATH = "crypto-config/org5.cmtestnet/user/client/ytfo02.sign.crt";
    static String CLIENT1_TLS_KEY_PATH = "crypto-config/org5.cmtestnet/user/client/ytfo02.tls.key";
    static String CLIENT1_TLS_CERT_PATH = "crypto-config/org5.cmtestnet/user/client/ytfo02.tls.crt";
    static final String ORG_ID1 = "org5.cmtestnet";
    static String SDK_CONFIG = "sdk_config.yml";
    static ChainClient chainClient;
    static ChainManager chainManager;
    static User user;
    public static void initChainClient() throws Exception {
        Yaml yaml = new Yaml();
        InputStream in = InitClient.class.getClassLoader().getResourceAsStream(SDK_CONFIG);
        SdkConfig sdkConfig;
        sdkConfig = yaml.loadAs(in, SdkConfig.class);
        assert in != null;
    }
}

```

修改后的内容如下

```

1 static final String CLIENT1_KEY_PATH = "crypto-config/wx-
org1.chainmaker.org/user/client1/client1.sign.key";
2     static final String CLIENT1_CERT_PATH = "crypto-config/wx-
org1.chainmaker.org/user/client1/client1.sign.crt";
3
4     static String CLIENT1_TLS_KEY_PATH = "crypto-config/wx-
org1.chainmaker.org/user/client1/client1.tls.key";
5     static String CLIENT1_TLS_CERT_PATH = "crypto-config/wx-
org1.chainmaker.org/user/client1/client1.tls.crt";
6
7     static final String ORG_ID1 = "wx-org1.chainmaker.org";

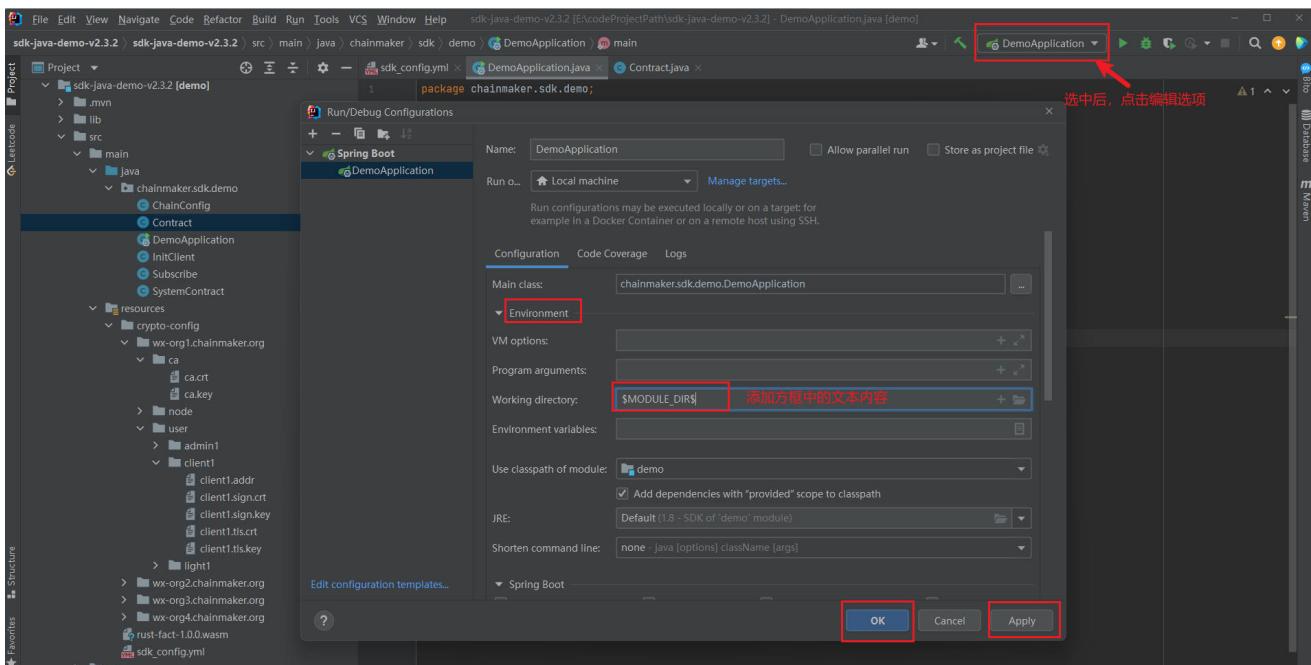
```

The screenshot shows an IDE interface with a project structure on the left and code editor on the right.

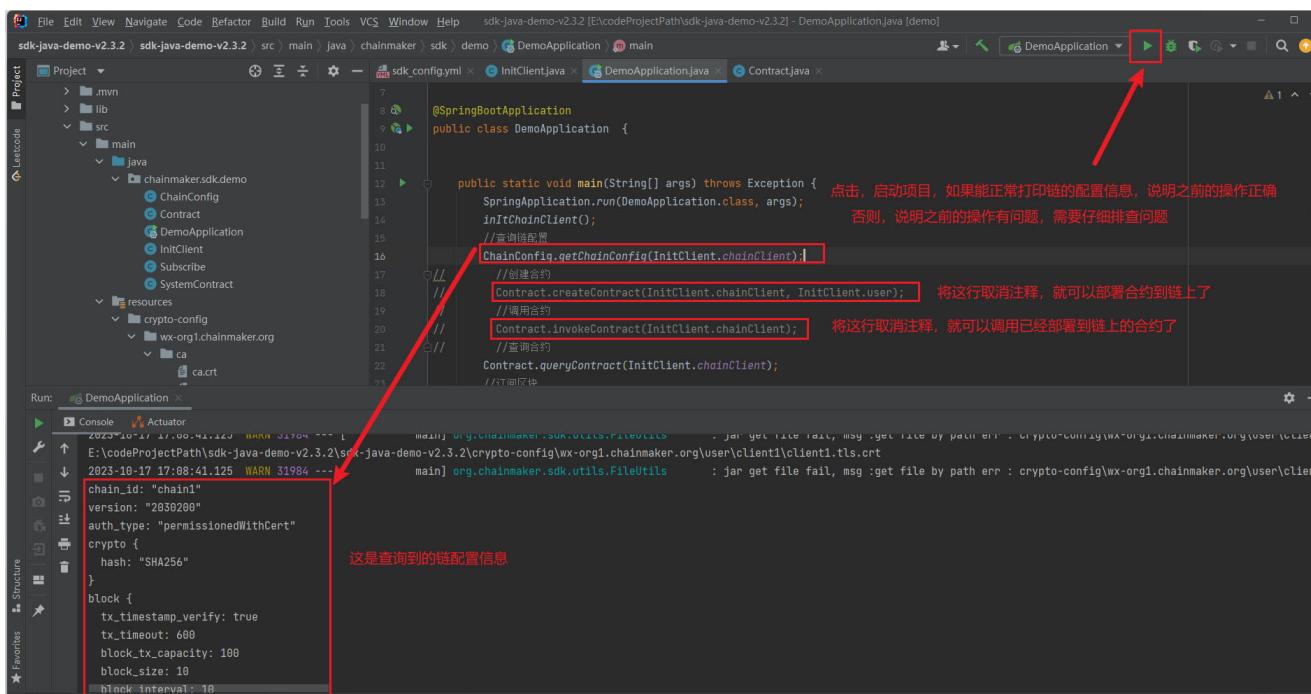
Project Structure:

- SDK Java demo v2.3.2
- .idea
- .mvn
- lib
- src
 - main
 - java
 - chainmaker.sdk.demo
 - ChainConfig
 - Contract
 - DemoApplication
 - InitClient** (highlighted with a red box)
 - Subscribe
 - SystemContract
 - resources
 - crypto-config
 - wx-org1.chainmaker.org
 - ca
 - ca.crt
 - ca.key
 - node
 - user
 - admin1
 - client1
 - client1.addr
 - client1.sign.crt
 - client1.sign.key
 - client1.tls.crt
 - client1.tls.key** (highlighted with a red box)

10.1.7 编辑配置



10.1.8 启动应用demo



The screenshot shows the IntelliJ IDEA interface with the code editor open to `DemoApplication.java`. The code contains several annotations and imports. A red box highlights the line `Contract.queryContract(InitClient.chainClient);` in the main method. The console output below shows the application's log, with a red box highlighting the error message:

```
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: "178ed9ecdf58ed0ca95f32b2ed7c5d3a839287d10c141d3a51f53b1dec6204f"
```

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

10.1.9 部署示例合约

The screenshot shows the IntelliJ IDEA interface with the code editor open to `DemoApplication.java`. The code includes annotations like `@SpringBootApplication` and `@Service`. A red box highlights the line `Contract.createContract(InitClient.chainClient, InitClient.user);` in the main method. Another red box highlights the line `Contract.invokeContract(InitClient.chainClient);` just below it. A note in the code states: "将之前的两行注释取消，其他地方不需要修改".

```
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( mills: 1000 * 10 );
    }
}
```

```
public static void createContract(ChainClient chainClient, User user) {
    ResultOuterClass.TxResponse responseInfo = null;
    try {
        byte[] byteCode = FileUtils.getResourceFileBytes(CONTRACT_FILE_PATH);

        // 1. create payload
        Request.Payload payload = chainClient.createContractCreatePayload(CONTRACT_NAME, version: "1", byteCode);
        // 2. create payloads with endorsement
        Request.EndorsementEntry[] endorsementEntries = SdkUtils
            .getEndorsers(payload, new User[]{user});

        // 3. send request
        responseInfo = chainClient.sendContractManageRequest(payload, endorsementEntries, rpcCallTimeout: 10000);
    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("创建合约结果:");
    System.out.println(responseInfo);
}

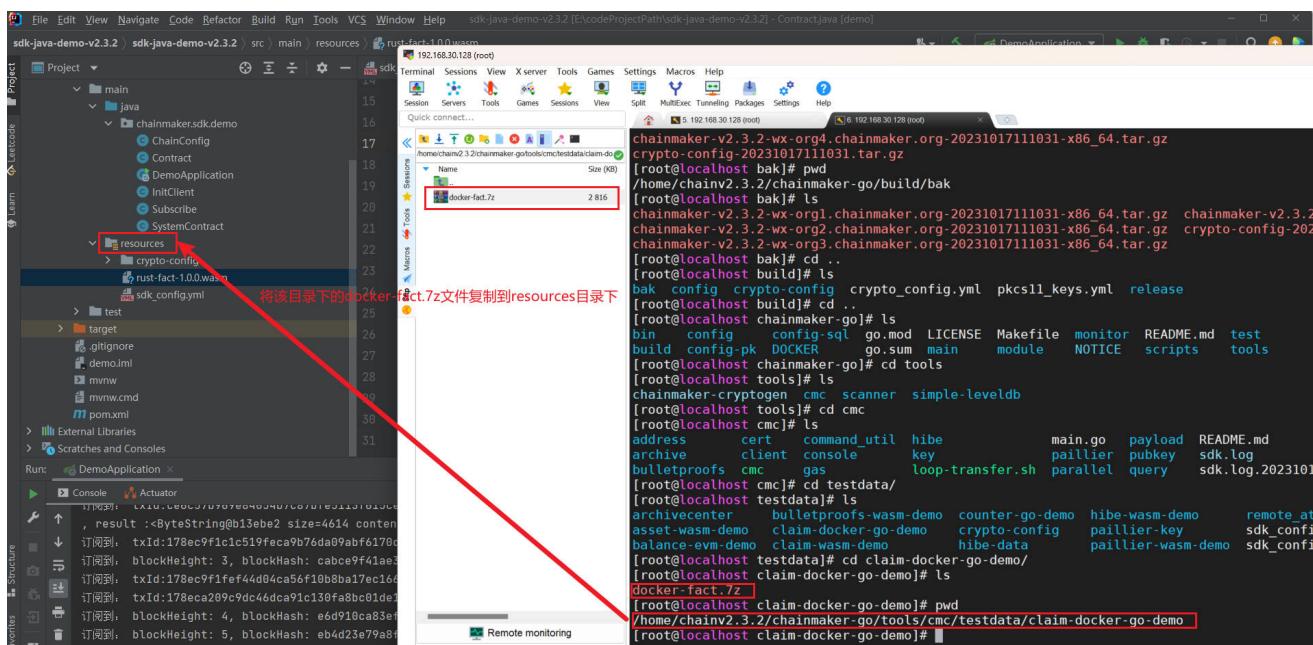
public static void invokeContract(ChainClient chainClient) {
    ResultOuterClass.TxResponse responseInfo = null;
    try {
        responseInfo = chainClient.invokeContract(CONTRACT_NAME, INVOKE_CONTRACT_METHOD,
            txId: null, params: null, rpcCallTimeout: 10000, syncResultTimeout: 10000);
    } 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 {
        responseInfo = chainClient.queryContract(CONTRACT_NAME, QUERY_CONTRACT_METHOD,
            txId: null, params: null, rpcCallTimeout: 10000);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

现在已经有合约部署到链上了，因此，没有报之前的错误了

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

```
1 public class Contract {  
2  
3     private static final String QUERY_CONTRACT_METHOD = "query";  
4     private static final String INVOKE_CONTRACT_METHOD =  
"increase";  
5     private static final String CONTRACT_NAME =  
"counter_sdk_java_demo";
```

```
6     private static final String CONTRACT_FILE_PATH = "rust-fact-
7     1.0.0.wasm";
8
9     public static void createContract(ChainClient chainClient,
10    User user) {
11         ResultOuterClass.TxResponse responseInfo = null;
12         try {
13             byte[] bytecode =
14             Fileutils.getResourceFileBytes(CONTRACT_FILE_PATH);
15
16             // 1. create payload
17             Request.Payload payload =
18             chainClient.createContractCreatePayload(CONTRACT_NAME, "1",
19             bytecode,
20                 ContractOuterClass.RuntimeType.WASMER, null);
21             //2. create payloads with endorsement
22             Request.EndorsementEntry[] endorsementEntries =
23             Sdkutils
24                 .getEndorsers(payload, new User[]{user});
25
26             // 3. send request
27             responseInfo =
28             chainClient.sendContractManageRequest(payload,
29             endorsementEntries, 10000, 10000);
30         } catch (Exception e) {
31             e.printStackTrace();
32         }
33         System.out.println("创建合约结果: ");
34         System.out.println(responseInfo);
35     }
36
37     public static void invokeContract(ChainClient chainClient) {
38         ResultOuterClass.TxResponse responseInfo = null;
39         try {
40             responseInfo =
41             chainClient.invokeContract(CONTRACT_NAME, INVOKE_CONTRACT_METHOD,
42                 null, null, 10000, 10000);
43         } catch (Exception e) {
44             e.printStackTrace();
45         }
46         System.out.println("执行合约结果: ");
47         System.out.println(responseInfo);
48     }
49 }
```

```
40     public static void queryContract(ChainClient chainClient) {
41         ResultOuterClass.TxResponse responseInfo = null;
42         try {
43             responseInfo =
44                 chainClient.queryContract(CONTRACT_NAME, QUERY_CONTRACT_METHOD,
45                                         null, null, 10000);
46         } catch (Exception e) {
47             e.printStackTrace();
48         }
49         System.out.println("查询合约结果: ");
50         System.out.println(responseInfo);
51     }
52 }
```

修改后的Contract.java文件

```
1 package chainmaker.sdk.demo;
2
3 import org.chainmaker.pb.common.ContractOuterClass;
4 import org.chainmaker.pb.common.Request;
5 import org.chainmaker.pb.common.ResultOuterClass;
6 import org.chainmaker.sdk.ChainClient;
7 import org.chainmaker.sdk.User;
8 import org.chainmaker.sdk.utils.Fileutils;
9 import org.chainmaker.sdk.utils.Sdkutils;
10
11 import java.nio.charset.StandardCharsets;
12 import java.util.HashMap;
13 import java.util.Map;
14
15
16 public class Contract {
17
18     private static final String QUERY_CONTRACT_METHOD =
19         "findByFileHash";
20     private static final String INVOKE_CONTRACT_METHOD = "save";
21     private static final String CONTRACT_NAME =
22         "fact_docker_go_2";
23     private static final String CONTRACT_FILE_PATH = "docker-
fact.7z";
24 }
```

```
23     public static void createContract(ChainClient chainClient,
24         User user) {
25         ResultOuterClass.TxResponse responseInfo = null;
26         try {
27             byte[] byteCode =
28                 Fileutils.getResourceFileBytes(CONTRACT_FILE_PATH);
29
30             // 1. create payload
31             Request.Payload payload =
32                 chainClient.createContractCreatePayload(CONTRACT_NAME, "1",
33                 byteCode,
34                     ContractOuterClass.RuntimeType.DOCKER_GO,
35                 null);
36
37             //2. create payloads with endorsement
38             Request.EndorsementEntry[] endorsementEntries =
39                 SdkUtils
40                     .getEndorsers(payload, new User[]{user});
41
42             // 3. send request
43             responseInfo =
44                 chainClient.sendContractManageRequest(payload,
45                 endorsementEntries, 10000, 10000);
46             } catch (Exception e) {
47                 e.printStackTrace();
48             }
49             System.out.println("创建合约结果: ");
50             System.out.println(responseInfo);
51         }
52
53         public static void invokeContract(ChainClient chainClient) {
54             ResultOuterClass.TxResponse responseInfo = null;
55             try {
56                 Map<String, byte[]> params = new HashMap<>();
57                 params.put("method",
58                     "save".getBytes(StandardCharsets.UTF_8));
59                 params.put("file_name",
60                     "name007".getBytes(StandardCharsets.UTF_8));
61                 params.put("file_hash",
62                     "ab3456df5799b87c77e7f88".getBytes(StandardCharsets.UTF_8));
63                 params.put("time",
64                     "6543234".getBytes(StandardCharsets.UTF_8));
65
66                 responseInfo =
67                 chainClient.invokeContract(CONTRACT_NAME, INVOKE_CONTRACT_METHOD,
```

```
53                     null, params,10000, 10000);
54     } catch (Exception e) {
55         e.printStackTrace();
56     }
57     System.out.println("执行合约结果: ");
58     System.out.println(responseInfo);
59 }
60
61     public static void queryContract(ChainClient chainClient) {
62         ResultOuterClass.TxResponse responseInfo = null;
63         try {
64             Map<String, byte[]> params = new HashMap<>();
65             params.put("method",
66 "findByFileHash".getBytes(StandardCharsets.UTF_8));
67             params.put("file_hash",
68 "ab3456df5799b87c77e7f88".getBytes(StandardCharsets.UTF_8));
69             responseInfo =
70             chainClient.queryContract(CONTRACT_NAME, QUERY_CONTRACT_METHOD,
71                     null, params,10000);
72         } catch (Exception e) {
73             e.printStackTrace();
74         }
75     }
76 }
```

修改后的createContract方法

这是合约中定义的两个方法
可以自定义, 符合变量的命名规则即可
保证这两者的名称相同即可

可以自定义

```
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";  
  
    public static void createContract(ChainClient chainClient, User user) {  
        ResultOuterClass.TxResponse responseInfo = null;  
        try {  
            byte[] byteCode = FileUtils.getResourceFileBytes(CONTRACT_FILE_PATH);  
  
            // 1. create payload  
            Request.Payload payload = chainClient.createContractCreatePayload(CONTRACT_NAME, version: "1", byteCode,  
                ContractOuterClass.RuntimeType.DOCKER_GO, params: null);  
            //2. create payloads with endorsement  
            Request.EndorsementEntry[] endorsementEntries = SdkUtils  
                .getEndorsers(payload, new User[]{user});  
  
            // 3. send request  
            responseInfo = chainClient.sendContractManageRequest(payload, endorsementEntries, rpcCallTimeout: 10000);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        System.out.println("创建合约结果: ");  
        System.out.println(responseInfo);  
    }  
  
    public static void invokeContract(ChainClient chainClient) {  
        System.out.println("调用合约结果: ");  
        System.out.println(responseInfo);  
    }  
}
```

修改后的invokeContract方法

```
System.out.println(responseInfo);

public static void invokeContract(ChainClient chainClient) {
    ResultOuterClass.TxResponse responseInfo = null;
    try {
        添加一个params参数
        Map<String, byte[]> params = new HashMap<>();
        params.put("method", "save".getBytes(StandardCharsets.UTF_8));
        //这个method参数必须放进来
        params.put("file_name", "name007".getBytes(StandardCharsets.UTF_8));
        params.put("file_hash", "ab3456df5799b87c77e7f88".getBytes(StandardCharsets.UTF_8));
        params.put("time", "6543234".getBytes(StandardCharsets.UTF_8));
    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("执行合约结果: ");
    System.out.println(responseInfo);
}

下面三个参数是合约中定义的方法中包含的参数
responseInfo = chainClient.invokeContract(CONTRACT_NAME, INVOKE_CONTRACT_METHOD,
    txId: null, params, rpcCallTimeout: 10000, syncResultTimeout: 10000);
}
将null修改成新添加的参数
```

长安链学习笔记md - Typescript

文件(D) 编辑(E) 除薄(R) 样式(S) 检索(W) 主题(T) 帮助(H)

带有Docker-Go类型的示例合约 person_docker_go_1

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

```

59     }
60
61     public static void queryContract(ChainClient chainClient) {
62         ResultOuterClass.TxResponse responseInfo = null;
63         try {
64             Map<String, byte[]> params = new HashMap<>();
65             params.put("method", "findByFileHash".getBytes(StandardCharsets.UTF_8));
66             params.put("file_hash", "ab3456df5799b87c77e7f88".getBytes(StandardCharsets.UTF_8));
67             responseInfo = chainClient.queryContract(CONTRACT_NAME, QUERY_CONTRACT_METHOD,
68                 null, params, pcCallTimeout: 10000);
69         } catch (Exception e) {
70             e.printStackTrace();
71         }
72         System.out.println("查询合约结果: ");
73         System.out.println(responseInfo);
74     }
75 }

```

长安链学习笔记.md - Typora
文件(D) 编辑(B) 高亮(B) 格式(O) 视图(V) 主题(T) 帮助(H)

查询合约person_docker_go_1中的方法findByFileHash

```

1 ./cmc client contract user get \
2   --contract-name=person_docker_go_1 \
3   --method=findByFileHash \
4   --sdk-conf-path=./testdata/sdk_config.yml \
5   --params='{"file_hash": "ab3456df5799b87c77e7f88"}'

```

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\-----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: "178ee38d95e77d84ca99bf4f4c59e9b588ad42c6d94c49c98f7e888c99b9b305"
        contract_name: "fact_docker_go_2"
        event_data: "ab3456df5799b87c77e7f88"
        event_data: "name007"
    }
}
tx_id: "178ee38d95e77d84ca99bf4f4c59e9b588ad42c6d94c49c98f7e888c99b9b305"

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"

```

