第5章

“区块链+物联网”应用之区块链网络环境搭建与管理

本章学习目标

(1) 掌握安装和配置虚拟化环境的能力，以便在其中创建和管理 Ubuntu 虚拟机。

(2) 学习如何在 Ubuntu 操作系统上安装系统工具以及编程语言环境。

(3) 理解并实践如何搭建 Hyperledger Fabric 环境。

(4) 学会测试区块链运行环境的方法。

(5) 掌握用户配置与管理的基础，能够设计并实践用户注册和授权机制。

视频讲解

5.1 区块链网络环境搭建

在区块链网络环境搭建的过程中，读者首先需要准备基础的虚拟化环境，并在此基础上安装和配置 Linux 操作系统(如 Ubuntu 22.04)。然后，读者需学习安装必需的系统工具，这些工具对于后续搭建 Hyperledger Fabric 环境至关重要。在成功安装所需工具后，读者将被指导通过一系列的命令来创建 Hyperledger Fabric 的工作环境，获取必要的源码，以及学会如何在本地搭建和测试一个基础的 Fabric 网络。

5.1.1 准备基本环境

在开始进行区块链网络的搭建之前，首先需要准备基本的软件环境，包括虚拟机的安装以及操作系统的配置。

1. 安装 VMware WorkStation 17 Pro 虚拟机

虚拟机软件是一种能够在计算机硬件和操作系统之间提供抽象层的技术，它创建了一个模拟的计算环境，使得在同一台物理计算机上可以同时运行多个操作系统。这种技术极大地提升了计算资源的利用率，也为软件开发和测试提供了极大的便利。目前市场上存在多种虚拟机软件，如 VirtualBox、VMware、Virtual PC 等，每种软件都有其独特的特点和应用领域。本章将以 VMware WorkStation 17 Pro 为例，详细阐述其安装和配置过程，为读者提供一个规范的指导。

1）下载安装包

访问 VMware 官网(www.vmware.com)，下载 VMware WorkStation 17 Pro 版本的安装包，如图 5.1 所示。

图 5.1　VMware 官网下载页面

2）安装软件

启动下载的安装软件，按照提示同意官方的使用协议，选择软件的安装路径，如图 5.2 和图 5.3 所示。

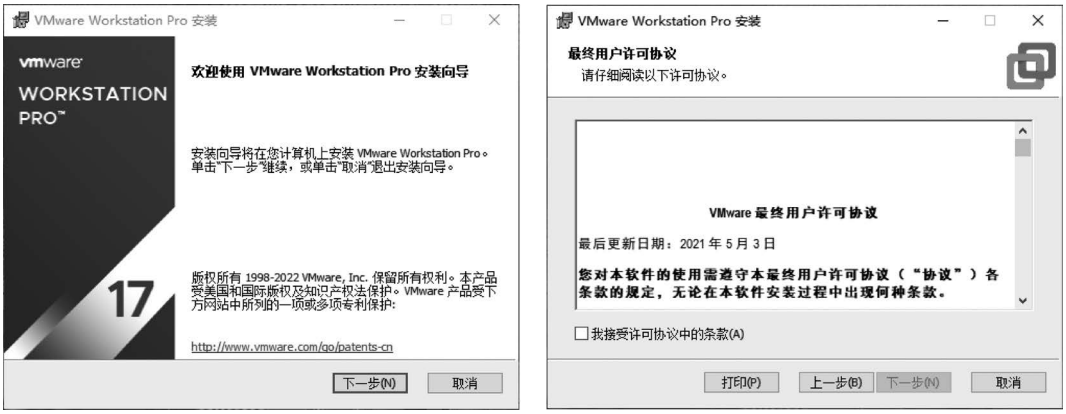

图 5.2　启动安装软件的界面

3）安装过程

安装过程中，可按照默认选项进行，直到安装完成，如图 5.4 所示。

4）软件激活及试用选项

在安装完成后，当用户第一次启动 VMware Workstation 17 Pro 时，系统会提示用户输入激活密钥来激活软件。用户需要在这个步骤中输入购买或获得的激活密钥。如果用户选择不输入激活密钥，他们通常会有一个试用期(例如 30 天)可以使用软件的全部功能。

通过以上步骤，我们已经成功安装了 VMware WorkStation 17 Pro 虚拟机，为后续操作系统的安装提供了环境。启动后的虚拟机软件界面如图 5.5 所示。

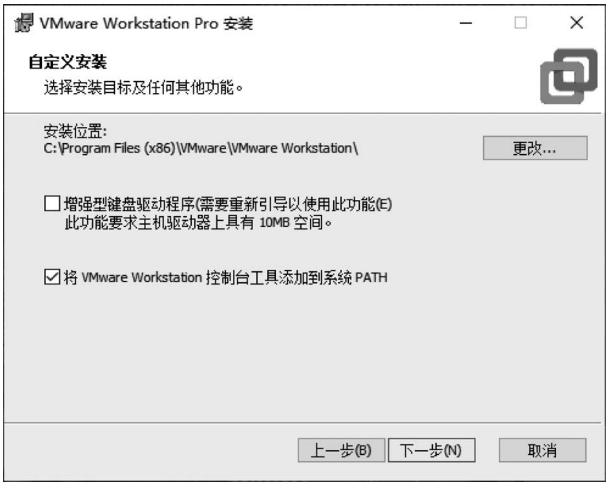

图 5.3　选择虚拟机软件安装路径的界面

图 5.4　虚拟机软件安装完成界面

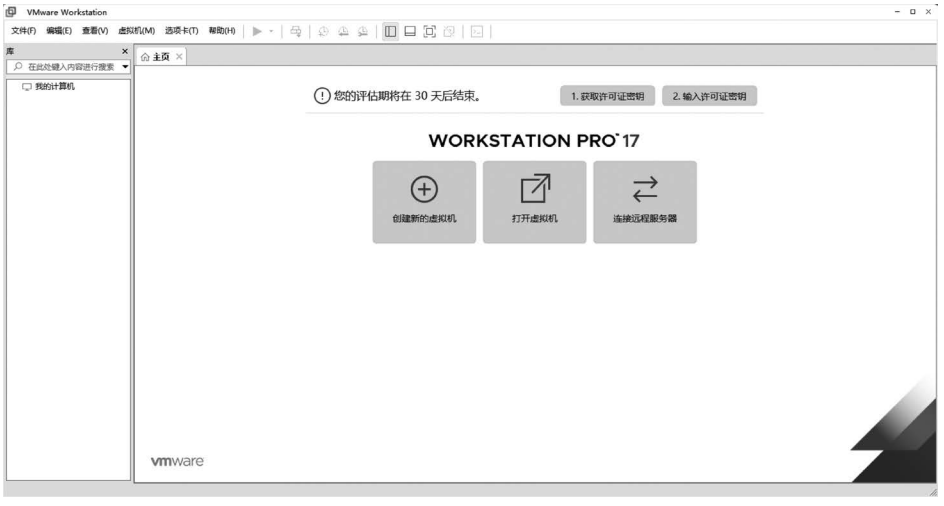

图 5.5　启动后的虚拟机软件界面

2. 安装 Ubuntu 22.04 操作系统

虚拟机安装后，选择安装 Ubuntu 22.04 桌面版操作系统，为区块链网络的搭建提供稳定的运行环境。

1）下载镜像文件

下载 Ubuntu 22.04 桌面版 ISO 镜像文件（https://releases.ubuntu.com/22.04/ubuntu-22.04.1-desktop-amd64.iso），如图 5.6 所示。

图 5.6　Ubuntu 桌面系统镜像下载页面

2）创建虚拟机

启动 VMware WorkStation，创建一个新的虚拟机，并选择 Linux 作为操作系统类型，版本选择 Ubuntu，如图 5.7 所示。

图 5.7　创建虚拟机界面

3）设置安装路径

默认安装路径为 C 盘，可以自定义安装路径，如图 5.8 所示。

图 5.8　设置虚拟机安装位置界面

4）配置虚拟机

建议设置内存大小不小于 4GB，磁盘大小不小于 50GB，以确保系统运行流畅。其他配置可使用默认推荐，如图 5.9 所示。

图 5.9　虚拟机配置界面

5）加载操作系统镜像

在“虚拟机设置”中，加载步骤 1）中下载的 Ubuntu ISO 镜像文件，如图 5.10 所示。

6）启动虚拟机

启动虚拟机，进入操作系统配置界面，按照提示完成系统的配置，如图 5.11 和图 5.12 所示。

图 5.10　加载操作系统镜像文件界面

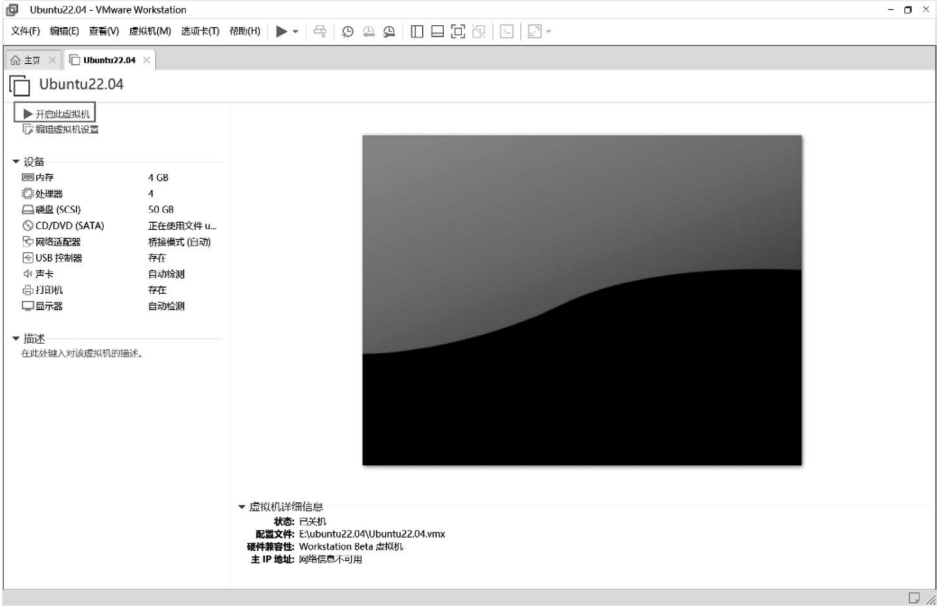

图 5.11　启动虚拟机界面

图 5.12　操作系统配置界面

7）安装操作系统

配置完成后，选择“安装 Ubuntu”进行操作系统的安装。在安装过程中，会联网下载所需文件，如图 5.13 所示。

图 5.13　虚拟机系统安装界面

8）启动 Ubuntu 操作系统

操作系统安装成功后重启系统，即可完成 Ubuntu 操作系统的安装，重启并成功登录后，将看到 Ubuntu 22.04 的系统桌面，如图 5.14 所示。

通过以上步骤，我们已经在虚拟机中成功安装并配置了 Ubuntu 22.04 操作系统，为区块链网络的搭建提供了基本的运行环境。

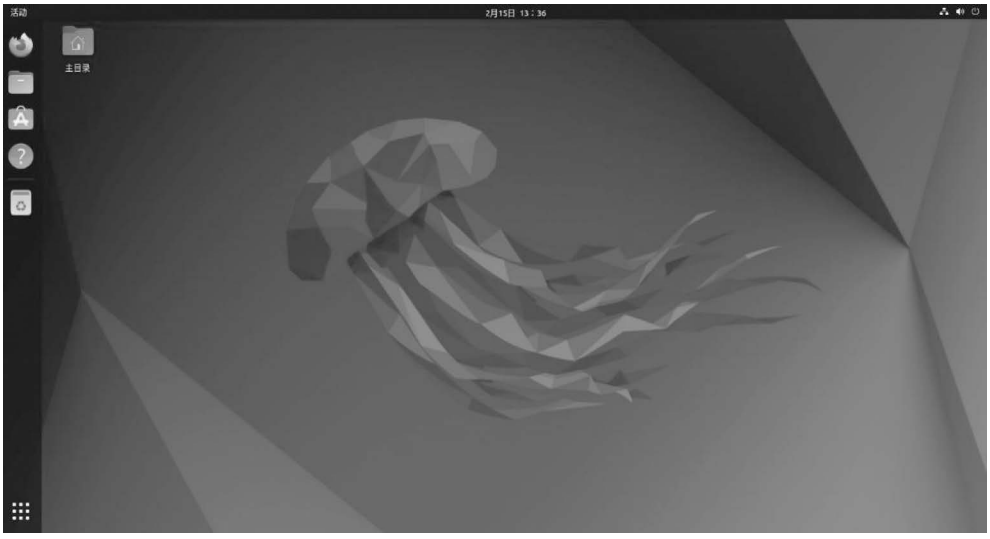

图 5.14　Ubuntu 22.04 系统桌面

5.1.2　系统工具安装

在进行区块链技术研究与开发的过程中，特别是在搭建 Hyperledger Fabric 的开发环境时，一系列系统工具的安装变得尤为重要。这些工具不仅为 Hyperledger Fabric 项目的正确安装提供了基础支持，也确保了其在后续的运行过程中的稳定性与效率。下面将详细介绍这些必要的工具，并解释其在 Hyperledger Fabric 开发环境中的作用。

1. Git

Git 是一种分布式版本控制系统，用于跟踪项目中文件的变化，并协助多个开发者之间的协作。其设计目标是提高效率和可靠性，它允许开发者在各自的工作站上独立进行开发，同时确保代码历史的完整性和一致性。在 Hyperledger Fabric 的开发环境搭建过程中，Git 用于从官方仓库中获取必要的代码和文档。

1）安装 Git

打开系统的终端界面，输入以下命令并执行，以在 Ubuntu 系统中安装 Git：

```
1.  sudo apt install git
```

首次使用 sudo 命令时，需要用户输入系统密码来确认安装操作。

2）对 Git 进行版本验证

安装完成后，输入以下命令并执行，以验证 Git 的安装并查看其版本，如图 5.15 所示。

```
1.  git -- version
```

2. cURL

cURL（Client for URLs）是一种广泛使用的命令行工具，它支持多种协议，包括 HTTP、HTTPS、FTP 和 SCP，用于在网络上传输数据。其功能强大，可用于发出各种网络请求，并查看或下载结果。在 Hyperledger Fabric 的应用场景中，cURL 通常用于与区块链

```
s@s-virtual-machine:~$ sudo apt install git
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
git 已经是最新版 (1:2.34.1-1ubuntu1.10)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 80 个软件包未被升级。
s@s-virtual-machine:~$ git --version
git version 2.34.1
```

图 5.15　安装并查看 Git 版本信息

网络中的节点进行交互，执行如提交交易、查询账本等操作。

1）安装 cURL

在终端输入以下命令并执行，以安装 cURL：

```
1.  sudo apt install curl
```

2）查看 cURL 的版本信息

安装完成后，输入以下命令并执行，查看 cURL 的版本信息，如图 5.16 所示。

```
1.  curl --version
```

```
s@s-virtual-machine:~$ sudo apt install curl
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
curl 已经是最新版 (7.81.0-1ubuntu1.14)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 80 个软件包未被升级。
s@s-virtual-machine:~$ curl --version
curl 7.81.0 (x86_64-pc-linux-gnu) libcurl/7.81.0 OpenSSL/3.0.2 zlib/1.2.11 brotli/1.0.9 zstd/1.4.8 libidn2/2.3.2 libpsl/0.21.0 (+l
ibidn2/2.3.2) libssh/0.9.6/openssl/zlib nghttp2/1.43.0 librtmp/2.3 OpenLDAP/2.5.15
Release-Date: 2022-01-05
Protocols: dict file ftp ftps gopher gophers http https imap imaps ldap ldaps mqtt pop3 pop3s rtmp rtsp scp sftp smb smbs smtp smt
ps telnet tftp
Features: alt-svc AsynchDNS brotli GSS-API HSTS HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL TLS
-SRP UnixSockets zstd
```

图 5.16　安装并查看 cURL 版本信息

3. Docker 和 Docker-Compose

Docker 是一个开源的应用容器引擎，它允许开发者将应用及其依赖项封装在一个轻量级的、可移植的容器中。这个容器可以在任何安装了 Docker 的系统上运行，提供了一种简单高效的应用分发和部署方式。在 Hyperledger Fabric 的开发环境中，Docker 用于创建一个稳定且隔离的运行环境，极大地简化了网络组件的部署和测试过程。

1）安装 Docker

在终端输入以下命令并执行，以安装 Docker：

```
1.  sudo apt install docker.io
```

2）查看 Docker 的版本信息

安装完成后，输入以下命令并执行，查看 Docker 的版本信息，如图 5.17 所示。确保终端中显示了 Docker 的版本信息，并记下版本号以供未来参考。

```
1.  docker --version
```

Docker-Compose 是一个用于定义和管理多容器 Docker 应用程序的工具。它允许开发者

```
s@s-virtual-machine:~$ sudo apt install docker.io
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
docker.io 已经是最新版 (24.0.5-0ubuntu1~22.04.1)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 80 个软件包未被升级。
s@s-virtual-machine:~$ docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~22.04.1
```

图 5.17　安装并查看 Docker 版本信息

通过 YAML 文件来配置应用服务的所有属性，包括服务、网络和卷等。在 Hyperledger Fabric 的开发环境中，Docker-Compose 起着关键作用，用于编排和管理网络中的各种组件，如 Peer 节点、Orderer 节点和 CA 服务器等，确保这些组件能够以正确的配置和顺序启动并协同工作。

1）安装 Docker-Compose

在终端输入以下命令并执行，以安装 Docker-Compose：

```
sudo apt install docker-compose
```

2）查看 Docker-Compose 的版本信息

```
docker-compose --version
```

安装完成后，输入以下命令并执行，查看 Docker-Compose 的版本信息，如图 5.18 所示。确保终端中显示了 Docker-Compose 的版本信息，并记下版本号以供未来参考。

```
s@s-virtual-machine:~$ sudo apt install docker-compose
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
docker-compose 已经是最新版 (1.29.2-1)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 80 个软件包未被升级。
s@s-virtual-machine:~$ docker-compose --version
docker-compose version 1.29.2, build unknown
```

图 5.18　安装并查看 Docker-Compose 版本信息

4. Go 语言

Go 又称为 Golang，是一种开源的编程语言，由 Google 设计并开发。其设计初衷是提高编程效率，支持并发编程，使程序员能够更方便地编写高效且稳定的代码。在 Hyperledger Fabric 中，大多数组件都是用 Go 语言编写的，这就要求开发者在构建和运行 Hyperledger Fabric 代码时必须在其开发环境中安装 Go。本节将指导读者如何在其系统中安装和配置 Go 语言环境，并确保其能够正确运行。

1）Go 语言的安装

为了保证区块链网络的稳定运行，建议安装 Go 语言的 1.18 版本。

首先，在终端输入以下命令并执行，下载 Go 1.18 的安装包，如图 5.19 所示。

```
sudo wget https://studygolang.com/dl/golang/go1.18.linux-amd64.tar.gz
```

输入以下命令，解压安装包到指定目录(/usr/local)，即完成了 Go 语言的安装。

```
sudo tar -C /usr/local -xzf go1.18.linux-amd64.tar.gz
```

```
s@s-virtual-machine:~$ sudo wget https://studygolang.com/dl/golang/go1.18.linux-amd64.tar.gz
--2023-11-01 20:36:48--  https://studygolang.com/dl/golang/go1.18.linux-amd64.tar.gz
正在解析主机 studygolang.com (studygolang.com)... 47.245.32.231
正在连接 studygolang.com (studygolang.com)|47.245.32.231|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 303 See Other
位置: https://golang.google.cn/dl/go1.18.linux-amd64.tar.gz [跟随至新的 URL]
--2023-11-01 20:36:50--  https://golang.google.cn/dl/go1.18.linux-amd64.tar.gz
正在解析主机 golang.google.cn (golang.google.cn)... 220.181.174.34
正在连接 golang.google.cn (golang.google.cn)|220.181.174.34|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 302 Found
位置: https://dl.google.com/go/go1.18.linux-amd64.tar.gz [跟随至新的 URL]
--2023-11-01 20:36:51--  https://dl.google.com/go/go1.18.linux-amd64.tar.gz
正在解析主机 dl.google.com (dl.google.com)... 180.163.151.33
正在连接 dl.google.com (dl.google.com)|180.163.151.33|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度: 141702072 (135M) [application/x-gzip]
正在保存至: ‘go1.18.linux-amd64.tar.gz’

go1.18.linux-amd64.tar.gz       100%[=====================================================>] 135.14M  2.84MB/s    用时 24s

2023-11-01 20:37:15 (5.63 MB/s) - 已保存 ‘go1.18.linux-amd64.tar.gz’ [141702072/141702072])
```

图 5.19　下载 Go 安装包

2）配置环境变量

在终端输入以下命令，打开 profile 文件进行编辑：

```
1.  sudo gedit/etc/profile
```

在文件的末尾添加以下内容，以配置 Go 环境变量，如图 5.20 所示。

```
1.  export GOROOT = /usr/local/go
2.  export GOPATH =  $ HOME/go
3.  export PATH =  $ PATH: $ GOROOT/bin: $ GOPATH/bin
```

图 5.20　配置 Go 环境变量

保存并关闭编辑器，输入以下命令，使环境变量配置立即生效：

```
1.  source  /etc/profile
```

3）验证 Go 语言的安装

在终端中输入以下命令，查看 Go 语言版本。如果成功输出了 Go 语言的版本信息，则

表明 Go 语言已经成功安装，如图 5.21 所示。

```
go version
```

```
s@s-virtual-machine:~$ go version
go version go1.18 linux/amd64
```

图 5.21　查看 Go 语言版本

5. Node.js 和 NPM

Node.js 是一个开源、跨平台的 JavaScript 运行时环境，它允许服务器端和网络应用的开发，而 NPM(Node Package Manager)是 Node.js 的包管理工具，用于管理项目中的依赖关系。在 Hyperledger Fabric 的开发过程中，Node.js 和 NPM 通常用于开发链码(Chaincode，运行在区块链网络中的智能合约)和客户端应用程序。

Fabric 官方提供了 Node.js 智能合约示例，NPM 是随同 Node.js 一起安装的包管理工具，用于解决 Node.js 代码部署。

1) 安装 Node.js 和 NPM

在终端中输入并执行如下命令，进行 Node.js 和 NPM 的安装，如图 5.22 所示。

```
sudo apt-get install npm
```

```
s@s-virtual-machine:~$ sudo apt-get install npm
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
npm 已经是最新版 (8.5.1~ds-1)。
升级了 0 个软件包，新安装了 0 个软件包，要卸载 0 个软件包，有 80 个软件包未被升级。
```

图 5.22　Node.js 与 NPM 的安装

2) 验证 Node.js 和 NPM 的安装

在终端中输入并执行如下命令，查看 Node.js 与 NPM 的版本信息，如图 5.23 所示。

```
npm version
```

```
s@s-virtual-machine:~$ npm version
{
  npm: '8.5.1',
  node: '12.22.9',
  v8: '7.8.279.23-node.56',
  uv: '1.43.0',
  zlib: '1.2.11',
  brotli: '1.0.9',
  ares: '1.18.1',
  modules: '72',
  nghttp2: '1.43.0',
  napi: '8',
  llhttp: '2.1.4',
  http_parser: '2.9.4',
  openssl: '1.1.1m',
  cldr: '40.0',
  icu: '70.1',
  tz: '2023c',
  unicode: '14.0'
}
```

图 5.23　查看 Node.js 与 NPM 的版本信息

5.1.3　搭建 Hyperledger Fabric 环境

1) 创建工作目录

打开系统的终端界面，输入并执行如下命令，创建 Fabric 所在目录并进入该目录。

```
mkdir -p go/src/github.com/Hyperledger
cd go/src/github.com/Hyperledger
```

2) 获取 Hyperledger Fabric 源码

在终端中输入并执行如下命令，从 GitHub 上下载 Fabric 源码。

```
git clone https://github.com/hyperledger/fabric.git
```

3) 切换至指定版本

在终端中输入并执行如下命令，查看所有可用的标签并切换到指定版本(本示例中为

2.2.0)。

```
1.  cd fabric && git tag -l
2.  git checkout v2.2.0
```

4) 拉取 Fabric 镜像

在终端中输入并执行如下命令,执行脚本拉取所需的 Docker 镜像。

```
1.  cd scripts && sudo ./bootstrap.sh
```

5) 查看镜像列表

镜像拉取完成后,在终端中输入并执行如下命令,查看拉取的镜像列表,如图 5.24 所示。

```
1.  sudo docker images
```

```
s@s-virtual-machine:~$ docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
busybox                       latest    a416a98b71e2   3 months ago    4.26MB
hyperledger/fabric-tools      2.2       5eb2356665e7   3 years ago     519MB
hyperledger/fabric-tools      2.2.0     5eb2356665e7   3 years ago     519MB
hyperledger/fabric-tools      latest    5eb2356665e7   3 years ago     519MB
hyperledger/fabric-peer       2.2       760f304a3282   3 years ago     54.9MB
hyperledger/fabric-peer       2.2.0     760f304a3282   3 years ago     54.9MB
hyperledger/fabric-peer       latest    760f304a3282   3 years ago     54.9MB
hyperledger/fabric-orderer    2.2       5fb8e97da88d   3 years ago     38.4MB
hyperledger/fabric-orderer    2.2.0     5fb8e97da88d   3 years ago     38.4MB
hyperledger/fabric-orderer    latest    5fb8e97da88d   3 years ago     38.4MB
hyperledger/fabric-ccenv      2.2       aac435a5d3f1   3 years ago     586MB
hyperledger/fabric-ccenv      2.2.0     aac435a5d3f1   3 years ago     586MB
hyperledger/fabric-ccenv      latest    aac435a5d3f1   3 years ago     586MB
hyperledger/fabric-baseos     2.2       aa2bdf8013af   3 years ago     6.85MB
hyperledger/fabric-baseos     2.2.0     aa2bdf8013af   3 years ago     6.85MB
hyperledger/fabric-baseos     latest    aa2bdf8013af   3 years ago     6.85MB
hyperledger/fabric-nodeenv    2.2       ab88fe4d29dd   3 years ago     293MB
hyperledger/fabric-nodeenv    2.2.0     ab88fe4d29dd   3 years ago     293MB
hyperledger/fabric-nodeenv    latest    ab88fe4d29dd   3 years ago     293MB
hyperledger/fabric-javaenv    2.2       56c30f316b23   3 years ago     504MB
hyperledger/fabric-javaenv    2.2.0     56c30f316b23   3 years ago     504MB
hyperledger/fabric-javaenv    latest    56c30f316b23   3 years ago     504MB
hyperledger/fabric-ca         1.4       743a758fae29   3 years ago     154MB
hyperledger/fabric-ca         1.4.7     743a758fae29   3 years ago     154MB
hyperledger/fabric-ca         latest    743a758fae29   3 years ago     154MB
```

图 5.24 显示拉取的镜像列表

执行完以上步骤后,Fabric 运行环境的安装基本就完成了。为了验证 Fabric 网络是否成功搭建,接下来将运行 Fabric 自带的测试网络。

5.1.4 测试区块链运行环境

区块链运行环境安装完成后,可以通过运行 Fabric 自带的测试网络,进行区块链运行环境的测试,以确保区块链系统正常运行。

1）启动测试网络

在终端中输入以下命令，以启动 Fabric 的网络测试：

```
cd ./fabric-samples/test-network
./network.sh up
```

如果一切正常，Fabric 测试网络将正确启动，如图 5.25 所示。

```
Creating network "fabric_test" with the default driver
Creating volume "docker_orderer.example.com" with default driver
Creating volume "docker_peer0.org1.example.com" with default driver
Creating volume "docker_peer0.org2.example.com" with default driver
Creating peer0.org1.example.com ... done
Creating orderer.example.com    ... done
Creating peer0.org2.example.com ... done
Creating cli                    ... done
CONTAINER ID   IMAGE                               COMMAND             CREATED                  STATUS                  PORTS
                          NAMES
23e99ee68fc2   hyperledger/fabric-tools:latest     "/bin/bash"         Less than a second ago   Up Less than a second
                          cli
c7f854b2920a   hyperledger/fabric-peer:latest      "peer node start"   1 second ago             Up Less than a second   0.0.0.0:9051->9051/tcp, :::9051->9051/tcp, 7051/tcp, 0.0.0.0:19051->
19051/tcp, :::19051->19051/tcp   peer0.org2.example.com
21f9a8d70df1   hyperledger/fabric-orderer:latest   "orderer"           1 second ago             Up Less than a second   0.0.0.0:7050->7050/tcp, :::7050->7050/tcp, 0.0.0.0:17050->17050/tcp,
 :::17050->17050/tcp             orderer.example.com
bfa50b598013   hyperledger/fabric-peer:latest      "peer node start"   1 second ago             Up Less than a second   0.0.0.0:7051->7051/tcp, :::7051->7051/tcp, 0.0.0.0:17051->17051/tcp,
 :::17051->17051/tcp             peer0.org1.example.com
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ docker ps -a
CONTAINER ID   IMAGE                               COMMAND             CREATED          STATUS          PORTS
                NAMES
23e99ee68fc2   hyperledger/fabric-tools:latest     "/bin/bash"         9 seconds ago    Up 8 seconds
                cli
c7f854b2920a   hyperledger/fabric-peer:latest      "peer node start"   10 seconds ago   Up 8 seconds    0.0.0.0:9051->9051/tcp, :::9051->9051/tcp, 7051/tcp, 0.0.0.0:19051->19051/tcp, :::190
51->19051/tcp   peer0.org2.example.com
21f9a8d70df1   hyperledger/fabric-orderer:latest   "orderer"           10 seconds ago   Up 8 seconds    0.0.0.0:7050->7050/tcp, :::7050->7050/tcp, 0.0.0.0:17050->17050/tcp, :::17050->17050/
tcp             orderer.example.com
bfa50b598013   hyperledger/fabric-peer:latest      "peer node start"   10 seconds ago   Up 8 seconds    0.0.0.0:7051->7051/tcp, :::7051->7051/tcp, 0.0.0.0:17051->17051/tcp, :::17051->17051/
tcp             peer0.org1.example.com
```

图 5.25　Fabric 测试网络正确启动的结果

2）节点加入通道

在终端中输入以下命令，将已创建的 peer 节点加入通道：

```
./network.sh createChannel
```

节点正确加入通道后，结果如图 5.26 所示。

```
+ configtxlator proto_decode --input config_block.pb --type common.Block
+ jq '.data.data[0].payload.data.config'
+ jq '.channel_group.groups.Application.groups.Org2MSP.values += {"AnchorPeers":{"mod_policy": "Admins","value":{"anchor_peers": [{"host": "peer0.org2.example.com","port": 9051}]},"version
": "0"}}' Org2MSPconfig.json
Generating anchor peer update transaction for Org2 on channel mychannel
+ configtxlator proto_encode --input Org2MSPconfig.json --type common.Config
+ configtxlator proto_encode --input Org2MSPmodified_config.json --type common.Config
+ configtxlator compute_update --channel_id mychannel --original original_config.pb --updated modified_config.pb
+ configtxlator proto_decode --input config_update.pb --type common.ConfigUpdate
+ jq .
++ cat config_update.json
+ echo '{"payload":{"header":{"channel_header":{"channel_id":"mychannel", "type":2}},"data":{"config_update":{' '"channel_id":' '"mychannel",' '"isolated_data":' '{},' '"read_set":' '{' '"
groups":' '{' '"Application":' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"",' '"policy":' null, '"vers
ion":' '"0"' '},' '"Endorsement":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Wr
iters":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"' '}' '},' '"version":
' '"0"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"0"' '},'
 '"write_set":' '{' '"groups":' '{' '"Application":' '{' '"groups":' '{' '"Org2MSP":' '{' '"groups":' '{},' '"mod_policy":' '"Admins",' '"policies":' '{' '"Admins":' '{' '"mod_policy":' '"
",' '"policy":' null, '"version":' '"0"' '},' '"Endorsement":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '},' '"Readers":' '{' '"mod_policy":' '"",' '"policy":' null,
'"version":' '"0"' '},' '"Writers":' '{' '"mod_policy":' '"",' '"policy":' null, '"version":' '"0"' '}' '},' '"values":' '{' '"AnchorPeers":' '{' '"mod_policy":' '"Admins",' '"value":' '{'
 '"anchor_peers":' '[' '{' '"host":' '"peer0.org2.example.com",' '"port":' 9051 '}' ']' '},' '"version":' '"0"' '},' '"MSP":' '{' '"mod_policy":' '"",' '"value":' null, '"version":' '"0"'
'}' '},' '"version":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"version":' '"1"' '}' '},' '"mod_policy":' '"",' '"policies":' '{},' '"values":' '{},' '"v
ersion":' '"0"' '}' '}}}}'
+ configtxlator proto_encode --input config_update_in_envelope.json --type common.Envelope
2023-11-01 12:45:16.661 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-11-01 12:45:16.668 UTC [channelCmd] update -> INFO 002 Successfully submitted channel update
Anchor peer set for org 'Org2MSP' on channel 'mychannel'
Channel 'mychannel' joined
```

图 5.26　节点正确加入通道后的结果

3）安装和部署智能合约

在终端中输入以下命令，将示例智能合约部署到创建的 Fabric 区块链网络中：

```
./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go
```

智能合约正确部署后，结果如图 5.27 所示。

```
+ peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile /home/s/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples
/test-network/organizations/ordererOrganizations/example.com/orders/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem --channelID mychannel --name basic --peerAddresses local
host:7051 --tlsRootCertFiles /home/s/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example
.com/tls/ca.crt --peerAddresses localhost:9051 --tlsRootCertFiles /home/s/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network/organizations/peerOrganizations/org2
.example.com/peers/peer0.org2.example.com/tls/ca.crt --version 1.0 --sequence 1
+ res=0
2023-11-01 20:46:37.114 CST [chaincodeCmd] ClientWait -> INFO 001 txid [4043df840fbc357a8cc93bcb31477aadddc6d5f3fe2d5cd822c770f6566bb5b0] committed with status (VALID) at localhost:9051
2023-11-01 20:46:37.143 CST [chaincodeCmd] ClientWait -> INFO 002 txid [4043df840fbc357a8cc93bcb31477aadddc6d5f3fe2d5cd822c770f6566bb5b0] committed with status (VALID) at localhost:7051
Chaincode definition committed on channel 'mychannel'
Using organization 1
Querying chaincode definition on peer0.org1 on channel 'mychannel'...
Attempting to Query committed status on peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org1 on channel 'mychannel'
Using organization 2
Querying chaincode definition on peer0.org2 on channel 'mychannel'...
Attempting to Query committed status on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name basic
+ res=0
Committed chaincode definition for chaincode 'basic' on channel 'mychannel':
Version: 1.0, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org2 on channel 'mychannel'
Chaincode initialization is not required
```

图 5.27　智能合约正确部署的结果

4）初始化 Fabric 账本

在与 Fabric 测试网络交互前，需要在 test-network 目录下执行以下命令配置环境变量：

```
export PATH=${PWD}/../bin:$PATH
export FABRIC_CFG_PATH=$PWD/../config/
export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"
export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051
```

然后，在终端中输入以下命令，初始化 Fabric 账本：

```
peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'
```

账本正确初始化后，结果如图 5.28 所示。

```
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ export CORE_PEER_TLS_ENABLED=true
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ export CORE_PEER_LOCALMSPID="Org1MSP"
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.
com/peers/peer0.org1.example.com/tls/ca.crt
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/
users/Admin@org1.example.com/msp
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ export CORE_PEER_ADDRESS=localhost:7051
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.co
m --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localh
ost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/orga
nizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'
2023-11-01 20:51:24.442 CST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
```

图 5.28　账本正确初始化的结果

5）调用智能合约

在终端中输入以下命令，实现调用已部署的智能合约并查看调用结果：

```
1.  peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'
2.  peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.
    com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/
    orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic
    --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/
    peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --
    peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/
    peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"
    function":"TransferAsset","Args":["asset6","Christopher"]}'
```

智能合约正确调用后，运行结果如图 5.29 所示。

```
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'
[{"ID":"asset1","color":"blue","size":5,"owner":"Tomoko","appraisedValue":300},{"ID":"asset2","color":"red","size":5,"owner":"Brad","appraisedValue":400},{"ID":"asset3","color":"green","si
ze":10,"owner":"Jin Soo","appraisedValue":500},{"ID":"asset4","color":"yellow","size":10,"owner":"Max","appraisedValue":600},{"ID":"asset5","color":"black","size":15,"owner":"Adriana","app
raisedValue":700},{"ID":"asset6","color":"white","size":15,"owner":"Michel","appraisedValue":800}]
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.co
m --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n basic --peerAddresses localh
ost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/orga
nizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"TransferAsset","Args":["asset6","Christopher"]}'
2023-11-01 20:52:35.369 CST [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ peer chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'
[{"ID":"asset1","color":"blue","size":5,"owner":"Tomoko","appraisedValue":300},{"ID":"asset2","color":"red","size":5,"owner":"Brad","appraisedValue":400},{"ID":"asset3","color":"green","si
ze":10,"owner":"Jin Soo","appraisedValue":500},{"ID":"asset4","color":"yellow","size":10,"owner":"Max","appraisedValue":600},{"ID":"asset5","color":"black","size":15,"owner":"Adriana","app
raisedValue":700},{"ID":"asset6","color":"white","size":15,"owner":"Christopher","appraisedValue":800}]
```

图 5.29　智能合约调用运行结果

6）关闭测试网络

如果测试完成，可以通过以下命令关闭测试网络：

```
1.  ./network.sh down
```

网络正确关闭后，结果如图 5.30 所示。

```
s@s-virtual-machine:~/桌面/go/src/github.com/Hyperledger/fabric/scripts/fabric-samples/test-network$ ./network.sh down
Stopping network
Stopping cli                    ... done
Stopping peer0.org2.example.com ... done
Stopping orderer.example.com    ... done
Stopping peer0.org1.example.com ... done
Removing cli                    ... done
Removing peer0.org2.example.com ... done
Removing orderer.example.com    ... done
Removing peer0.org1.example.com ... done
Removing network fabric_test
Removing volume docker_orderer.example.com
Removing volume docker_peer0.org1.example.com
Removing volume docker_peer0.org2.example.com
Removing network fabric_test
WARNING: Network fabric_test not found.
Removing volume docker_peer0.org3.example.com
WARNING: Volume docker_peer0.org3.example.com not found.
No containers available for deletion
Untagged: dev-peer0.org2.example.com-basic_1.0-3cfcf67978d6b3f7c5e0375660c995b21db19c4330946079afc3925ad7306881-6a8fb30d4a694b406bf4f255c7e84f22faeb3190e416cfb1d742ae192a406805:latest
Deleted: sha256:a12ba87346326108875d1feefcfca316d267ceafb5fd02f7934a9e134743bce3
Deleted: sha256:c0fd9c2b448de8195610035655b3033254a41f118192c70e135a663cfb037f0d
Deleted: sha256:a441ebfe1fa4885ddda3ad6f4c4aaaca1582cd2243c4cf2aca74a0820b8914a9
Deleted: sha256:d736bae0a69d51004debf50c7d7678be53203c83a5ee6cc9de57e3c92456ed79
Untagged: dev-peer0.org1.example.com-basic_1.0-3cfcf67978d6b3f7c5e0375660c995b21db19c4330946079afc3925ad7306881-c6c446ce0bcf4e0229bc2660312ba359cb00358628cf199bfc1ae60f85b825a7:latest
Deleted: sha256:1d3b5cc467b95fed435a40cd67aa4e0a586eb72284fab5adb9052aae73eb23b4
Deleted: sha256:2372862df7f3ac3edfe35e0715ce96559c14b0e70d8f28a8f4e267703e4e3fad
Deleted: sha256:3219762aa5188f3b459ecb92ba9032d41c205183bd84fe005c9b34f26939848e
Deleted: sha256:a76d034e51ecc01d5de422912bce7f5a264203ffe579ca60f5ca690a125c273f
```

图 5.30　网络关闭运行结果

5.2　用户配置与管理

视频讲解

用户配置与管理是“区块链＋物联网”应用的关键组成部分，涉及用户身份的认证与授

权。本节将介绍如何设计和实现一个用户注册系统，该系统能够识别并验证用户身份，同时管理用户的访问权限。我们将提供具体的代码实例，以指导读者实现用户注册信息的加密签名、验证。此外，本节还将讨论用户授权的机制，包括为已注册的用户分配和管理权限。

5.2.1 用户注册

在“区块链＋物联网”应用中，用户需要在使用应用之前进行注册，以获取用户账号。用户账号在应用中扮演着身份标识的角色，具有用户身份识别和验证的功能。以下是有关用户注册的详细说明。

1. 设计用户注册信息

用户注册信息是在用户注册时所需提供的数据。这些信息用于验证用户身份、管理用户权限以及记录用户操作。如表 5.1 所示，典型的用户注册信息包括身份证号(ID)和用户角色(Role)等字段。身份证号用于唯一标识用户，而用户角色通常分为管理员和普通用户，管理员具有更高的权限，可以为普通用户授权执行特定操作。具体注册信息需要根据应用的实际需求设置。

表 5.1 用户注册信息

字　段	含　义
UserName	用户名
Password	密码
Name	姓名
ID	身份证号
Phone	手机号码
Email	电子邮箱
Role	用户角色

2. 实践用户注册

在本节中，我们将详细讲解用户注册智能合约的代码示例，合约中主要包括用户注册信息的签名、验证、上链、获取以及其他相关辅助代码。

1) 用户注册信息签名

首先，构造用户注册前的待签名数据结构，将其序列化为 JSON 格式字符串，并使用 SHA-256 算法计算哈希摘要，最后使用用户注册私钥对哈希摘要进行签名并保存签名结果。

```
1.  func UserInfoSign(user User) error {
2.   // 构造用户注册信息
3.   userForSign : = User{
4.    UserName: user.UserName,
5.    Password: user.Password,
6.    Name:     user.Name,
7.    ID:       user.ID,
8.    Phone:    user.Phone,
9.    Email:    user.Email,
10.   Role:     user.Role,
```

```
    }
    // 序列化用户信息
    userForSignJSON, err := json.Marshal(userForSign)
    if err != nil {
     return err
    }

    // 计算哈希摘要
    hashText := sha256.New()
    hashText.Write(userForSignJSON)

    // 加载私钥, LoadpriKey()的代码详见 4.1.3
    if err = LoadpriKey(); err != nil {
     return err
    }

    // 签名数据
    r, s, err := ecdsa.Sign(randSign.Reader, PriKey, hashText.Sum(nil))
    if err != nil {
     log.Println(err)
    }
    userSign["r"], userSign["s"] = r, s
    return nil
  }
```

2）用户注册信息验证

首先，构造用户注册前的待签名数据结构，使用 SHA-256 算法计算序列化后的用户注册信息 JSON 字符串的哈希摘要，然后使用注册用户的公钥验证用户数字签名。如果验证通过，则返回 true，否则返回 false。

```
func VerifyUserSign(user User) bool {
 // 构造用户注册信息
 userInfo := User{
  UserName: user.UserName,
  Password: user.Password,
  Name:     user.Name,
  ID:       user.ID,
  Phone:    user.Phone,
  Email:    user.Email,
  Role:     user.Role,
 }
 // 序列化用户信息
 userInfoJSON, err := json.Marshal(userInfo)
 if err != nil {
  return false
 }

 // 计算哈希摘要
 hashText := sha256.New()
 hashText.Write(userInfoJSON)

```

```
    // 使用公钥验证签名
    if !ecdsa.Verify(userInfo.PubKey, hashText.Sum(nil), userInfo.Sign["r"], userInfo.
    Sign["s"]) {
     return false
    }
    return true
   }
```

3）用户注册信息上链

首先，获取用户上传的注册信息，构建用户注册信息结构体，进行用户注册信息签名并将其保存在用户注册信息结构体中。最后，将用户注册信息结构体序列化后上链存证。

```
   func (s *SmartContract) UserRegister(ctx contractapi.TransactionContextInterface,
   userName string, password string, name string, id string, phone string, email string, role
   string) error {
    // 加载公钥 PubKey,LoadpubKey()的代码详见 4.1.3 节
    if err := LoadpubKey(); err != nil {
     return err
    }
    // 构建用户注册信息结构体
    user := User{
     UserName: userName,
     Password: password,
     Name:     name,
     ID:       id,
     Phone:    phone,
     Email:    email,
     Role:     role,
     PubKey:   PubKey,
    }
    // 用户信息签名
    if err := UserInfoSign(user); err != nil {
     return err
    }

    // 将用户信息序列化
    user.Sign = userSign
    userJSON, err := json.Marshal(user)
    if err != nil {
     return err
    }
    // 用户注册信息上链
    if err != ctx.GetStub().PutState(id, userJSON); err != nil {
     return err
    }
   return nil
   }
```

4）用户注册信息获取

首先，使用用户 ID 从链上获取用户注册信息，验证用户注册信息的数字签名是否有效。如果验证通过，则返回序列化后的用户注册信息 JSON 对象，否则返回错误信息。

```
func (s *SmartContract) GetUser(ctx contractapi.TransactionContextInterface, id
string) (*User, error) {
  userJSON, err := ctx.GetStub().GetState(id)
  if err != nil {
   return nil, err
  }
  if userJSON == nil {
   return nil, fmt.Errorf("未找到 ID 为 %s 的用户", id)
  }

  var user User
  if err = json.Unmarshal(userJSON, &user); err != nil {
   return nil, err
  }
  // 验证用户签名
  if !VerifyUserSign(user) {
   return nil, err
  }
  return &user, fmt.Errorf("用户签名验证失败")
 }
```

5）相关辅助代码

除了上述主要功能代码，用户注册智能合约还包括通用的 ECDSA 数字签名功能代码以及用户注册信息结构体等相关辅助代码。

```
var userSign map[string]*big.Int
type User struct {
 UserName string                 `json:"UserName"`
 Password string                 `json:"Password"`
 Name     string                 `json:"Name"`
 ID       string                 `json:"ID"`
 Phone    string                 `json:"Phone"`
 Email    string                 `json:"Email"`
 Role     string                 `json:"Role"`
 Sign     map[string]*big.Int    `json:"Sign"`
 PubKey   *ecdsa.PublicKey       `json:"PubKey"`
}
```

5.2.2　用户授权

用户授权是在用户注册通过后的步骤，它用于控制用户对数据和操作的访问。在“区块链＋物联网”应用中，用户初始角色通常为普通用户，其权限较低，需要管理员进行授权才能执行一些操作，如设备注册、设备更改等。

1. 设计用户授权信息

用户注册后的默认角色为普通用户，权限较低，无法执行设备注册、设备删除等操作，此时需要管理员为其授予相应权限。如表 5.2 所示，典型的用户授权信息主要包括授权类型(Type)、授权开始时间(StartTime)和授权结束时间(EndTime)等内容。

表 5.2 用户授权信息

字 段	含 义
UserName	用户名
StartTime	授权开始时间
EndTime	授权结束时间
Type	授权类型

在用户授权操作中，这些信息用于验证和控制用户的权限，以确保数据和操作的安全性，还可以利用这些信息对授权进行追溯和查证，便于监管。表 5.2 中关键字段说明如下：

（1）**授权类型**：用户可以对设备执行的操作，包括设备注册、设备更改和设备删除三种。在用户操作设备时，用于用户的权限验证。

（2）**授权开始时间**：用户获得授权的开始时间，在该时间之后用户可以执行已授权操作，使用中国标准时间格式（CST），如“2023-01-30 11:24:15”。

（3）**授权结束时间**：用户授权的到期时间，在该时间之后用户无法执行授权过期操作，使用中国标准时间格式（CST），表示用户权限的到期时间，如“2023-12-31 11:24:15”。

2. 实践用户授权

用户授权智能合约与用户注册智能合约类似，包括用户授权信息的签名、验证、上链、获取以及其他相关辅助代码。

1）用户授权信息签名和验证

用户由管理员进行授权，因此在签名时需要使用管理员私钥，而在验证时需要使用用户注册智能合约中的 GetUser()方法获取管理员的公钥来验证数字签名。

```
// 用户授权信息签名
func UserAuthSign(userAuth UserAuth) error {
 userAuthForSign := UserAuth{
  UserName:   userAuth.UserName,
  StartTime:  userAuth.StartTime,
  EndTime:    userAuth.EndTime,
  Type:       userAuth.Type,
 }

 userAuthForSignJSON, err := json.Marshal(userAuthForSign)
 if err != nil {
  return err
 }
 hashText := sha256.New()
 hashText.Write(userAuthForSignJSON)
 // 加载管理员私钥并签名
 if err = LoadpriKey(); err != nil {
  return err
 }
 r, s, err := ecdsa.Sign(strings.NewReader(randSign), PriKey, hashText.Sum(nil))
 if err != nil {
  log.Println(err)
 }
```

```
    userAuthSign["r"], userAuthSign["s"] = r, s
    return nil
}
// 用户授权信息验证
func VerifyUserAuthSign(userAuth UserAuth) bool {
    userAuthForSign := UserAuth{
        UserName: userAuth.UserName,
        StartTime: userAuth.StartTime,
        EndTime: userAuth.EndTime,
        Type: userAuth.Type,
    }

    userInfoJSON, err := json.Marshal(userAuthForSign)
    if err != nil {
        return false
    }
    hashText := sha256.New()
    hashText.Write(userInfoJSON)
    // 使用管理员公钥验证数字签名
    if !ecdsa.Verify(userAdmin.PubKey, hashText.Sum(nil), userAuth.Sign["r"], userAuth.Sign["s"]) {
        return false
    }
    return true
}
```

2）用户授权信息上链和获取

用户授权信息上链和获取的代码与用户注册智能合约相似。

```
// 用户授权信息上链
func (s *SmartContract) UserAuthorise(ctx contractapi.TransactionContextInterface, userName string, startTime string, endTime string, type_ string) error {
    userAuth := UserAuth{
        UserName: userName,
        StartTime:startTime,
        EndTime:  endTime,
        Type:     type_,
        PubKey:   PubKey,
    }

    if err := UserAuthSign(userAuth); err != nil {
        return err
    }
    userAuth.Sign = userAuthSign
    userAuthJSON, err := json.Marshal(userAuth)
    if err != nil {
        return err
    }
    return ctx.GetStub().PutState(userName, userAuthJSON)
}
// 用户授权信息获取
func (s *SmartContract) GetUserAuth(ctx contractapi.TransactionContextInterface, userName string) (*UserAuth, error) {
```

```
    userAuthJSON, err := ctx.GetStub().GetState(userName)
    if err != nil {
     return nil, err
    }
    if userAuthJSON == nil {
     return nil, err
    }

    var userAuth UserAuth
    if err = json.Unmarshal(userAuthJSON, &userAuth); err != nil {
     return nil, err
    }
    // 获取管理员注册信息
    userAdmin, _ = s.GetUser(ctx, "【admin ID】")
    if !VerifyUserAuthSign(userAuth) {
     return nil, err
    }
    return &userAuth, nil
  }
```

3）相关辅助代码

用户授权智能合约需要与用户注册智能合约一起使用，除了主要功能代码外，还包括用户授权信息结构体等相关辅助代码。

```
var userAuthSign map[string] * big.Int
var userAdmin * User

type UserAuth struct {
 UserName   string                `json:"UserName"`
 StartTime  string                `json:"StartTime"`
 EndTime    string                `json:"EndTime"`
 Type       string                `json:"Type"`
 Sign       map[string] * big.Int `json:"Sign"`
 PubKey      * ecdsa.PublicKey    `json:"PubKey"`
}
```

通过这些实践代码，用户可以进行注册和授权的操作，确保应用能够识别和验证用户身份，并进行权限管理，以满足“区块链＋物联网”应用的需求。这些代码将帮助读者更深入地理解用户配置与管理的关键概念。

5.3 本章小结

本章详细讨论了在“区块链＋物联网”应用场景中搭建和管理 Fabric 区块链网络的步骤和方法。从准备虚拟化环境和操作系统安装，到具体的区块链环境配置，本章提供了全面的指导。同时，本章强调了用户注册和授权流程的重要性，为读者提供了实现这些流程的具体代码，旨在确保网络的有效运行和数据的安全性。读者应当通过实际操作，不仅理解理论知识，而且掌握实践技能，以确保能够在“区块链＋物联网”应用开发中有效地应用本章内容。

习题 5

一、单项选择题

1. 区块链网络搭建的第一步是安装(　　)。
   A. VMware WorkStation 17 Pro 虚拟机
   B. Ubuntu 22.04 操作系统
   C. Docker 和 Docker-Compose
   D. Git
2. 下列哪个不是系统工具安装中的一项?(　　)
   A. Git　　B. cURL　　C. Docker　　D. Adobe Reader
3. 下列哪种工具用于版本控制和协作开发?(　　)
   A. cURL　　B. Go　　C. Git　　D. Docker
4. 下列哪个不是 Hyperledger Fabric 环境搭建的步骤?(　　)
   A. 创建工作目录　　B. 获取 Hyperledger Fabric 源码
   C. 安装 Photoshop 软件　　D. 拉取 Fabric 镜像
5. 用户注册和授权对“区块链+物联网”应用的作用是(　　)。
   A. 加强网络连接　　B. 提高虚拟机性能
   C. 增加网络带宽　　D. 管理用户权限和记录操作
6. 用户注册智能合约中,使用哪种算法计算哈希摘要?(　　)
   A. RSA　　B. SHA-256　　C. MD5　　D. AES
7. 节点加入通道的主要目的是(　　)。
   A. 安装和部署智能合约　　B. 初始化 Fabric 账本
   C. 启动测试网络　　D. 扩展网络的参与者
8. 为什么在区块链网络搭建过程中需要安装 Docker 和 Docker-Compose?(　　)
   A. 用于创建虚拟机　　B. 用于容器化部署区块链组件
   C. 用于操作系统的管理　　D. 用于注册用户
9. 用户注册信息中通常不包含哪个字段?(　　)
   A. 身份证号(ID)　　B. 用户角色(Role)
   C. 银行账户信息　　D. 用户名
10. 关闭测试网络的主要目的是(　　)。
    A. 增加网络的可用性　　B. 提高系统性能
    C. 进行维护和管理　　D. 初始化区块链账本
11. 在“区块链+物联网”应用中,为什么用户注册是关键?(　　)
    A. 用于细粒度的用户授权　　B. 用于创建智能合约
    C. 用于加强网络连接　　D. 用于设置区块链节点
12. 用户授权的作用是(　　)。
    A. 识别和验证用户身份　　B. 加强网络连接
    C. 部署智能合约　　D. 控制用户对数据和操作的访问权限

13. 在“区块链＋物联网”应用中，通常情况下，用户在初始角色下具有什么权限？（　　）

A. 最高权限　　B. 普通用户权限　　C. 管理员权限　　D. 设备注册权限

14. 用户授权的有效性通过什么来验证？（　　）

A. 用户的姓名　　B. 用户的生日　　C. 数字签名　　D. 用户密码

15. 在用户授权信息中，Type 字段代表什么？（　　）

A. 用户的密码强度　　B. 授权类型

C. 用户的年龄　　D. 用户注册的时间

二、简答题

1. 请描述区块链网络搭建的基本步骤。
2. 为什么在区块链网络搭建过程中需要安装 Docker 和 Docker-Compose？
3. 简述 Hyperledger Fabric 搭建环境中“创建工作目录”的目的。
4. 节点加入通道的目的是什么？它在区块链网络中有何作用？
5. 描述 Hyperledger Fabric 中“节点加入通道”的基本步骤。
6. 如何关闭测试网络以便维护和管理区块链环境？
7. 为什么用户注册和授权对“区块链＋物联网”应用至关重要？
8. 解释为什么在用户注册时需要对信息进行哈希处理？
9. 怎样通过 Hyperledger Fabric SDK 调用智能合约的功能？
10. 描述用户注册智能合约中“用户注册信息签名”的作用。

三、编程实践

1. 编写一个简单的伪代码，展示在 Hyperledger Fabric 中如何注册一个新用户。

2. 请根据本章已有的内容，设计一个权限审核与审计智能合约，定期审查用户的权限，更新用户的权限，确保授权策略保持最新并与应用的需求一致。请使用 Go 语言完成该编程任务，示例代码中需要有适当的注释。