第 2 章 Chapter 2

超级账本初体验

本章先简单介绍一下 Hyperledger Fabric 1.0 的环境搭建，快速地体验一下超级账本的功能。本书所有的内容都是基于 Hyperledger Fabric 1.0 的，在后面章节中我们偶尔也会用到“超级账本”这个词，指的也是超级账本的 Hyperledger Fabric 1.0 项目。

2.1 基础环境安装

Hyperledger Fabric 1.0 依赖 Docker 执行智能合约，需要先安装 Docker 和 Docker Compose 的运行环境。

2.1.1 Docker 的安装和使用

Docker 支持 Linux、Mac、Windows 等多个平台，安装文档参考：https://docs.docker.com/engine/installation。

1. 在 Linux 环境下 Docker 的安装

Ubuntu、Debian、CentOS 等 Linux 系统，可以通过 Docker 官方提供的脚本进行安装：

```
curl -sSL https://get.docker.com | sh
```

然后把用户加入到 docker 组，非 root 用户 USER 可以执行 docker 命令（可能需要重新登录生效）：

```
sudo usermod -aG docker $USER
```

如果是 Ubuntu 或者 Debian 操作系统，修改 Docker 的配置文件 /etc/default/docker，增

加 Docker 的 socket 绑定，运行在 Docker 中的进程才能通过映射的 socket 调用 Docker 的 API 执行镜像编译和创建容器等操作。

```
DOCKER_OPTS="-s=aufs -r=true --api-cors-header='*' -H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock "
```

接着，重启 Docker 服务让配置生效：

```
sudo service docker start
```

CentOS 系统采用 Systemd 进行系统和服务管理，配置文件的修改方法是不一样的。CentOS 系统下 Docker 的配置文件是 /etc/sysconfig/docker，同样要修改 DOCKER_OPTS 选项。还需要修改 /usr/lib/systemd/system/docker.service 文件，在 [Service] 的 ExexStart= 下面增加一行 $DOCKER_OPTS，如下所示：

```
[Service]
Type=notify
NotifyAccess=all
EnvironmentFile=-/etc/sysconfig/docker
EnvironmentFile=-/etc/sysconfig/docker-storage
EnvironmentFile=-/etc/sysconfig/docker-network
Environment=GOTRACEBACK=crash
Environment=DOCKER_HTTP_HOST_COMPAT=1
Environment=PATH=/usr/libexec/docker:/usr/bin:/usr/sbin
ExecStart=/usr/bin/dockerd-current \
        --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current \
        --default-runtime=docker-runc \
        --exec-opt native.cgroupdriver=systemd \
        --userland-proxy-path=/usr/libexec/docker/docker-proxy-current \
        $DOCKER_OPTS \
        $OPTIONS \
        $DOCKER_STORAGE_OPTIONS \
        $DOCKER_NETWORK_OPTIONS \
        $ADD_REGISTRY \
        $BLOCK_REGISTRY \
        $INSECURE_REGISTRY
```

重启服务让配置生效：

```
systemctl daemon-reload
systemctl restart docker.service
```

2. 其他环境下 Docker 的安装

Windows 和 Mac 都提供了安装包，直接下载即可安装：

- Docker for Mac: https://download.docker.com/mac/stable/Docker.dmg
- Docker for Windows: https://download.docker.com/win/stable/InstallDocker.msi

3. Docker 国内镜像仓库

国外的镜像下载较慢，可以设置国内的镜像，阿里云和 DaoCloud 都提供镜像加速的服务，需要登录注册才能使用。

- 阿里云：登录容器 Hub 服务 https://cr.console.aliyun.com 的控制台，左侧的加速器帮助页面会显示为你独立分配的加速地址。
- DaoCloud：在 https://www.daocloud.io 进行注册登录，然后点击加速器，就可以获取加速器的相关配置。

修改 Docker 镜像仓库的办法是在 DOCKER_OPTS 里增加 registry-mirror 参数，比如：

```
DOCKER_OPTS="-s=aufs -r=true --api-cors-header='*' -H tcp://0.0.0.0:2375
-H unix:///var/run/docker.sock
--registry-mirror=http://069f616f.m.daocloud.io"
```

重启 Docker 服务就可以使用镜像加速了。

Docker 在 Windows 和 Mac 中的版本可以在图形界面添加镜像仓库。

4. Docker 常用命令

Docker 常用命令如表 2-1 所示。

表 2-1 Docker 常用命令

命令	举例	说明
docker images	docker images	查看主机上的镜像文件列表
docker pull	docker pull hyperledger/fabric-peer	从镜像仓库中下载镜像文件
docker tag	docker tag hyperledger/fabric-tools:x86_64-1.0.0 hyperledger/fabric-tools:latest	给镜像文件打标签，x86_64-1.0.0 标记为 latest
docker run	docker run -it --name cli ubuntu /bin/bash	从镜像中启动容器，其中：cli 是容器名称，ubuntu 是镜像名称，-it 是以交互方式启动，/bin/bash 是启动容器时执行的命令
docker logs	docker logs -f cli	查看容器日志
docker ps	docker ps -a	查看主机上的容器，其中：参数 -a 会显示已经停止的容器
docker port	docker port peer0.org1.example.com	查看容器映射的端口
docker rm	docker rm cli	删除容器，其中：cli 为容器名称或者 id
docker --help	docker --help	查看帮助文档

更多的命令请查看帮助文档和在线文档：https://docs.docker.com/engine/reference/commandline/docker。

2.1.2　Docker Compose 的安装和使用

Docker Compose 能够在一个主机上创建出相互隔离的网络，通过命令行管理多个 Docker 容器，快速启动、停止和更新容器。

1. Docker Compose 的安装

Docker 在 Windows 和 Mac 中都已经集成了 Docker Compose 工具，不需要单独安装。在 Linux 系统下有多种安装方法，如下所示：

（1）通过 pip 进行安装

```
sudo apt install python-pip
sudo pip install docker-compose
```

（2）直接下载文件

```
curl -L
https://github.com/docker/compose/releases/download/1.17.1/docker-compose-`uname -s`-
`uname -m` -o /usr/local/bin/docker-composechmod +x /usr/local/bin/docker-compose
```

2. Docker Compose 的配置文件

Compose 采用 YAML 文件定义 Docker 容器之间的依赖，设置环境变量和文件的持久化。我们看一个配置文件 examples/e2e_cli/base/docker-compose-base.yaml 的节选：

```
version: '2'

services:

    orderer.example.com:
        container_name: orderer.example.com
        image: hyperledger/fabric-orderer
        environment:
            - ORDERER_GENERAL_LOGLEVEL=debug
            - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
            - ORDERER_GENERAL_GENESISMETHOD=file

        - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
        - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
        - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
        # enabled TLS
        - ORDERER_GENERAL_TLS_ENABLED=true
        - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
        - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
        - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
        working_dir: /opt/gopath/src/github.com/hyperledger/fabric
    command: orderer
    volumes:

- ../channel-artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
- ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/
msp:/var/hyperledger/orderer/msp
- ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/
tls/:/var/hyperledger/orderer/tls
    ports:
        - 7050:7050
```

```
peer0.org1.example.com:
    container_name: peer0.org1.example.com
    extends:
        file: peer-base.yaml
        service: peer-base
    environment:
        - CORE_PEER_ID=peer0.org1.example.com
        - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
        - CORE_PEER_CHAINCODELISTENADDRESS=peer0.org1.example.com:7052
        - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
        - CORE_PEER_LOCALMSPID=Org1MSP
    volumes:
        - /var/run/:/host/var/run/
- ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.
com/msp:/etc/hyperledger/fabric/msp

- ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.
com/tls:/etc/hyperledger/fabric/tls
    ports:
        - 7051:7051
        - 7052:7052
        - 7053:7053
```

在这个节选的配置文件中，一共定义了 1 个排序服务节点 orderer.example.com 和 1 个 Peer 节点 peer0.org1.example.com。Docker Compose 目前有 3 个版本，这个配置文件采用的 version 2 的语法，配置文件的解释如表 2-2 所示。

表 2-2　配置文件的解释

选项	举例	说明
version	version: '2'	采用 version 2 的语法
services		定义服务列表
orderer.example.com	根据服务名称自定义	自定义的服务名称，需要保持唯一
container_name	container_name: orderer.example.com	容器名称
image	image: hyperledger/fabric-orderer	容器使用的镜像文件
environment	- CORE_PEER_LOCALMSPID=Org1MSP	传递给容器的环境变量
working_dir	working_dir: /opt/gopath/src/github.com/hyperledger/fabric	容器启动的工作目录
command	command: orderer	容器启动命令
volumes	- /var/run/:/host/var/run/	宿主机和容器之间的目录映射
ports	- 7050:7050	宿主机和容器之间的端口映射
extends	file: common.yml	服务扩展，基于 common.yml 文件
extends	service: peer-base	服务扩展，基础服务是 peer-base

更多不同版本的配置文件说明请参考在线帮助文档：https://docs.docker.com/compose/compose-file。

3. Docker Compose 的常用命令

Docker Compose 的常用命令如表 2-3 所示。

表 2-3 Docker Compose 的常用命令

命令	举例	说明
docker-compose up	docker-compose -f docker-compose-cli.yaml up -d	根据配置文件 docker-compose-cli.yaml 启动容器，其中：-f 指定配置文件的名称，-d 设置以后台方式运行
docker-compose down	docker-compose -f docker-compose-cli.yaml down	停止配置文件 docker-compose-cli.yaml 的容器
docker-compose pull	docker-compose -f docker-compose-cli.yaml pull	批量下载所需的镜像文件

更多的命令查看帮助文档和在线文档：https://docs.docker.com/compose/reference。

2.1.3 下载超级账本源代码

超级账本的源代码都托管在 https://gerrit.hyperledger.org/r/#/admin/projects/ 下面，并在 https://github.com/hyperledger 上提供只读代码。最好的方式是直接通过 git 下载：

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

也可以打包下载文件后解压：https://github.com/hyperledger/fabric/archive/release.zip。

2.2 超级账本部署调用

先最小化地体验一下超级账本的环境，更详细的部署流程参考第 11 章。

2.2.1 下载 Docker 镜像文件

超级账本源码 scripts 目录下有多个下载镜像的脚本，我们可以修改权限以后直接运行：

```
# 进入 fabric/scripts 目录
chmod +x bootstrap-1.0.0.sh
# MacOS 系统执行如下命令（不下载二进制文件）
sed -i '' 's/curl/#curl/g' bootstrap-1.0.0.sh
# 其他系统执行如下命令（不下载二进制文件）
sed -i 's/curl/#curl/g' bootstrap-1.0.0.sh
# 直接下载 Docker 镜像文件
./bootstrap-1.0.0.sh
```

根据网络情况，可能需要等待一段时间。下面是下载的 Docker 镜像文件：

```
localhost:dive-into-fabric clarity$ docker images
REPOSITORY                      TAG                 IMAGE ID            SIZE
hyperledger/fabric-tools        latest              0403fd1c72c7        1.32GB
hyperledger/fabric-tools        x86_64-1.0.0        0403fd1c72c7        1.32GB
hyperledger/fabric-couchdb      latest              2fbdbf3ab945        1.48GB
```

```
hyperledger/fabric-couchdb      x86_64-1.0.0        2fbdbf3ab945        1.48GB
hyperledger/fabric-kafka        latest              dbd3f94de4b5        1.3GB
hyperledger/fabric-kafka        x86_64-1.0.0        dbd3f94de4b5        1.3GB
hyperledger/fabric-zookeeper    latest              e545dbf1c6af        1.31GB
hyperledger/fabric-zookeeper    x86_64-1.0.0        e545dbf1c6af        1.31GB
hyperledger/fabric-orderer      latest              e317ca5638ba        179MB
hyperledger/fabric-orderer      x86_64-1.0.0        e317ca5638ba        179MB
hyperledger/fabric-peer         latest              6830dcd7b9b5        182MB
hyperledger/fabric-peer         x86_64-1.0.0        6830dcd7b9b5        182MB
hyperledger/fabric-javaenv      latest              8948126f0935        1.42GB
hyperledger/fabric-javaenv      x86_64-1.0.0        8948126f0935        1.42GB
hyperledger/fabric-ccenv        latest              7182c260a5ca        1.29GB
hyperledger/fabric-ccenv        x86_64-1.0.0        7182c260a5ca        1.29GB
hyperledger/fabric-ca           latest              a15c59ecda5b        238MB
hyperledger/fabric-ca           x86_64-1.0.0        a15c59ecda5b        238MB
```

REPOSITORY 代表的是镜像的仓库名称，每个仓库下面都有打了不同 TAG 的标签名称，代表不同的版本。通常最少有两个标签，一个是 latest；另外一个的命名规则是“主机 CPU 类型 – 超级账本主版本号 –snapshot– 代码库版本号”，其中主机 CPU 类型为 x86_64，说明是 Intel 的 64 位 CPU，超级账本的主版本为 1.0.0，snapshot 是固定名称，代码库版本号为 58cde93，它是 git 代码库最近一次提交版本号的前 7 位。snapshot 和代码库版本号只有通过本地编译的时候才会出现。每次 make docker 的时候都会检查是否有文件改动，如果有变化的文件，则会重新构建，生成新的镜像再标记成 latest。镜像文件详细的解释请参考第 11 章的相关内容。

2.2.2　部署超级账本网络

运行超级账本需要设置较多的初始化配置，我们先绕开初始化过程，用 fabric-samples 工程中已经生成的配置文件来体验部署安装的过程：

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

进入 basic-network 目录，利用 docker-compose 启动容器：

```
cd fabric-samples/basic-network
docker-compose -f docker-compose.yml up -d
```

查看已经启动的容器（输出进行了删减）：

```
localhost:basic-network clarity$ docker ps
CONTAINER ID    IMAGE                                       NAMES
efddfbf4fc0a    hyperledger/fabric-peer:x86_64-1.0.0        peer0.org1.example.com
606d13c1e7a2    hyperledger/fabric-couchdb:x86_64-1.0.0     couchdb
d8c870db8634    hyperledger/fabric-ca:x86_64-1.0.0          ca.example.com
c6f25a5e6fd6    hyperledger/fabric-tools:x86_64-1.0.0       cli
a5f6331c5bc5    hyperledger/fabric-orderer:x86_64-1.0.0     orderer.example.com
```

切换到管理员用户再创建通道和加入通道：

```
# 切换环境到管理员用户的 MSP，进入 Peer 节点容器 peer0.org1.example.com
docker exec -it -e "CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@
org1.example.com/msp" peer0.org1.example.com bash
# 创建通道
peer channel create -o orderer.example.com:7050 -c mychannel -f /etc/hyperledger/
configtx/channel.tx
# 加入通道
peer channel join -b mychannel.block
# 退出 Peer 节点容器 peer0.org1.example.com
exit
    # 退出 Peer 节点容器 peer0.org1.example.com，进入 cli 容器安装链码和实例化：
    # 进入 cli 容器
docker exec -it cli /bin/bash
# 给 Peer 节点 peer0.org1.example.com 安装链码
peer chaincode install -n mycc -v v0 -p github.com/chaincode_example02
# 实例化链码
peer chaincode instantiate -o orderer.example.com:7050 -C mychannel -n mycc -v
v0 -c '{"Args":["init","a","100","b","200"]}'
```

2.2.3 链码调用和查询

链码实例化以后，可以查询初始值，同样是在 cli 容器里执行下面的操作：

```
peer chaincode query -C mychannel -n mycc -v v0 -c '{"Args":["query","a"]}'
```

查询结果显示为 Query Result: 100，详细信息如下：

```
2017-08-09 14:47:05.853 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing
local MSP
2017-08-09 14:47:05.853 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining
default signing identity
2017-08-09 14:47:05.853 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003
Using default escc
2017-08-09 14:47:05.854 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004
Using default vscc
2017-08-09 14:47:05.854 UTC [msp/identity] Sign -> DEBU 005 Sign: plaintext: 0A9
1070A6708031A0C08A9A694D00510...6D7963631A0A0A0571756572790A0161
2017-08-09 14:47:05.854 UTC [msp/identity] Sign -> DEBU 006 Sign: digest: E18FC9
7C13D550C5E3349AAD49523A6D7C71B4E51C219CD9A8799DEF54FFFE66
Query Result: 100
2017-08-09 14:47:05.886 UTC [main] main -> INFO 007 Exiting.....
```

调用链码，从“a”转移 10 到“b”：

```
peer chaincode invoke -C mychannel -n mycc -v v0 -c '{"Args":["invoke","a",
"b","10"]}'
```

显示调用成功的结果：

```
2017-08-09 14:49:46.018 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 0cb
```

```
Chaincode invoke successful. result: status:200
```

再次查询“a”和“b”的值：

```
peer chaincode query -C mychannel -n mycc -v v0 -c '{"Args":["query","a"]}'
peer chaincode query -C mychannel -n mycc -v v0 -c '{"Args":["query","b"]}'
```

查询结果显示“a”的值为 Query Result: 90，“b”的值为 Query Result: 210。

2.2.4　常见错误

1. 请求调用者权限不足

调用的时候设置了错误的 MSP，比如需要管理员才能执行创建通道的操作，但是设置了普通的成员 MSP，会出现 Error: Got unexpected status: BAD_REQUEST 的错误：

```
2017-08-09 14:49:04.652 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing
local MSP
2017-08-09 14:49:04.652 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining
default signing identity
2017-08-09 14:49:04.654 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and
orderer connections initialized
2017-08-09 14:49:04.656 UTC [msp] GetLocalMSP -> DEBU 004 Returning existing
local MSP
2017-08-09 14:49:04.656 UTC [msp] GetDefaultSigningIdentity -> DEBU 005 Obtaining
default signing identity
2017-08-09 14:49:04.657 UTC [msp] GetLocalMSP -> DEBU 006 Returning existing
local MSP
2017-08-09 14:49:04.657 UTC [msp] GetDefaultSigningIdentity -> DEBU 007 Obtaining
default signing identity
2017-08-09 14:49:04.657 UTC [msp/identity] Sign -> DEBU 008 Sign: plaintext: 0A8
8060A074F7267314D535012FC052D...53616D706C65436F6E736F727469756D
2017-08-09 14:49:04.657 UTC [msp/identity] Sign -> DEBU 009 Sign: digest: F77320
AE89B131CE75A858A4A450CF0F35301DA62FE1DE465CAEF4439F6FC520
2017-08-09 14:49:04.657 UTC [msp] GetLocalMSP -> DEBU 00a Returning existing
local MSP
2017-08-09 14:49:04.657 UTC [msp] GetDefaultSigningIdentity -> DEBU 00b Obtaining
default signing identity
2017-08-09 14:49:04.657 UTC [msp] GetLocalMSP -> DEBU 00c Returning existing
local MSP
2017-08-09 14:49:04.657 UTC [msp] GetDefaultSigningIdentity -> DEBU 00d Obtaining
default signing identity
2017-08-09 14:49:04.657 UTC [msp/identity] Sign -> DEBU 00e Sign: plaintext: 0AB
F060A1508021A0608E0D591D00522...A38A58EED7B94AC4CB800B86F0A5EF03
2017-08-09 14:49:04.658 UTC [msp/identity] Sign -> DEBU 00f Sign: digest: 475A33
426FA36D50F090AAF7C3AAAB2BF34339191BA74D4A60BF13460B241329
Error: Got unexpected status: BAD_REQUEST
Usage:
    peer channel create [flags]
```

```
Flags:
    -c, --channelID string    In case of a newChain command, the channel ID to
    create.
    -f, --file string          Configuration transaction file generated by a tool
    such as configtxgen for submitting to orderer
    -t, --timeout int        Channel creation timeout (default 5)

Global Flags:
        --cafile string               Path to file containing PEM-encoded trusted
        certificate(s) for the ordering endpoint
        --logging-level string       Default logging level and overrides, see
        core.yaml for full syntax
    -o, --orderer string            Ordering service endpoint
        --test.coverprofile string   Done (default "coverage.cov")
        --tls                Use TLS when communicating with the orderer endpoint
    -v, --version            Display current version of fabric peer server
```

2. 传递错误的通道名称

比如通道名称是 mychannel，传递参数的时候写成了错误的 myc：

```
peer chaincode query -C myc -n mycc -v v0 -c '{"Args":["query","a"]}'
```

会出现如下错误：

```
2017-08-09 14:40:36.703 UTC [msp] GetLocalMSP -> DEBU 001 Returning existing
local MSP
2017-08-09 14:40:36.703 UTC [msp] GetDefaultSigningIdentity -> DEBU 002 Obtaining
default signing identity
2017-08-09 14:40:36.706 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003
Using default escc
2017-08-09 14:40:36.706 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004
Using default vscc
2017-08-09 14:40:36.707 UTC [msp/identity] Sign -> DEBU 005 Sign: plaintext: 0A9
1070A6708031A0C08A4A394D00510...30300A000A04657363630A0476736363
2017-08-09 14:40:36.707 UTC [msp/identity] Sign -> DEBU 006 Sign: digest: F085E1
89A0765713209DD8802DDF57B054EBCD294305A500F56BC34CB3D2E577
Error: Error endorsing chaincode: rpc error: code = Unknown desc = chaincode
error (status: 500, message: chaincode exists mycc)
Usage:
    peer chaincode instantiate [flags]

Flags:
    -C, --channelID string   The channel on which this command should be executed
    (default "testchainid")
    -c, --ctor string       Constructor message for the chaincode in JSON format
    (default "{}")
    -E, --escc string    The name of the endorsement system chaincode to be used
    for this chaincode
    -l, --lang string     Language the chaincode is written in (default "golang")
    -n, --name string     Name of the chaincode
```

```
      -P, --policy string       The endorsement policy associated to this chaincode
      -v, --version string      Version of the chaincode specified in install/
      instantiate/upgrade commands
      -V, --vscc string         The name of the verification system chaincode to be
      used for this chaincode

Global Flags:
          --cafile string              Path to file containing PEM-encoded trusted
          certificate(s) for the ordering endpoint
          --logging-level string     Default logging level and overrides, see core.
          yaml for full syntax
      -o, --orderer string              Ordering service endpoint
          --test.coverprofile string   Done (default "coverage.cov")
          --tls                 Use TLS when communicating with the orderer endpoint
```

2.3　节点的配置参数传递规则

在 docker-compose.yml 文件中，我们可以看到有 ORDERER_GENERAL_LEDGERTYPE=ram 的设置，这是传递给节点的参数。给节点传递参数的方法有多种方式：**环境变量**、**配置文件**、**动态环境变量**、**默认值**。

程序在启动的时候会读取配置文件和环境变量的值，分别保存到不同变量缓存起来，在程序需要获取某个变量值的时候，不同传递方法的参数读取流程图如图 2-1 所示。

从图 2-1 中可以看到，如果配置了自动从环境变量获取参数的值，那么每次都实时地从环境变量中获取，否则依次读取程序启动时从环境变量、配置文件中读取后缓存到内存中的值，优先获取到的值作为返回值。如果都没有获取到，则返回空，交给程序进行处理，程序可能会以默认值运行，也可能会报错停止运行，这跟业务逻辑有关系。所以，只有在设置了自动从环境变量中获取参数的情况下，才能在运行时通过修改环境变量改变参数的值。

每个环境变量的名称都有一个前缀，每个模块都是单独设置的，比如 ORDERER_GENERAL_LEDGERTYPE 的前缀是 ORDERER，通常每个模块的前缀是不一样的。比如这里的 ORDERER 代表的是排序服务节点，Peer 节点的变量名称前缀是 CORE。环境变量名称是以“_”作为分隔符的，代表一种层级，是和配置文件一一对应的，比如 ORDERER_GENERAL_LEDGERTYPE 对应 orderer.yaml 配置文件的 General.LedgerType。环境变量和配置文件的变量名称都是不区分大小写的，内部会统一转换成小写的变量名称进行处理。

配置文件路径优先读取环境变量设置的路径，排序服务节点和 Peer 节点都是相同的环境变量。如果没有设置环境变量，则默认是应用程序所在的目录，然后环境变量 GOPATH 路径对应到模块代码工程下的目录，比如排序服务节点的目录是 $GOPATH/src/github.com/hyperledger/fabric/orderer，Peer 节点的目录是 $GOPATH/src/github.com/hyperledger/fabric/peer。

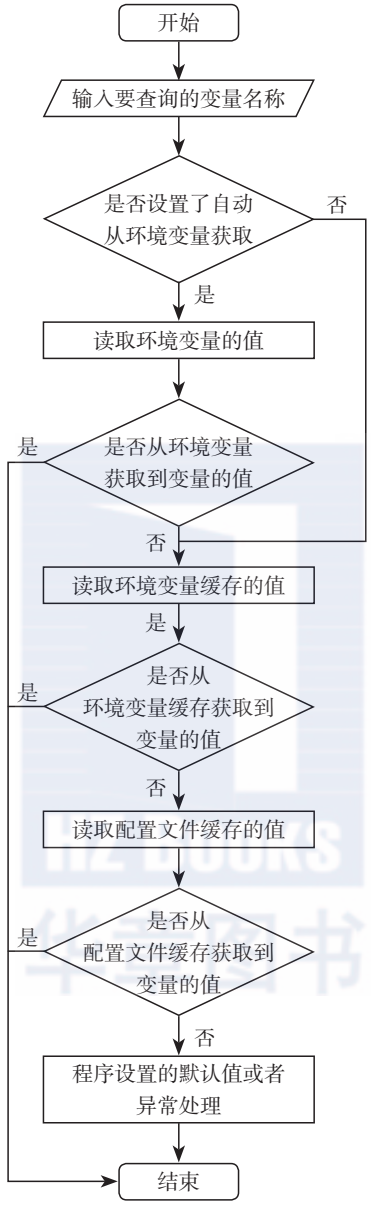

图 2-1 节点配置参数的传递规则

一般情况下，配置文件的名称设置成模块前缀的小写，比如排序服务节点的配置文件名称是 orderer，Peer 节点的配置文件名称是 core。

配置文件名称的后缀支持：json、toml、yaml、properties、props、prop，它们分别对应 JSON 文件、TOML 文件、YAML 文件和 Properties 文件，程序设置配置文件的时候如果不指定后缀，则按支持的后缀顺序在配置文件路径下进行搜索，找到第一个匹配的文件作

为最终的配置文件。文件路径和文件后缀按照广度优先的搜索顺序，即在同一路径下匹配完所有文件后缀再进入下一个文件路径。假设 Peer 节点没有设置配置文件路径，$GOPATH 的路径是 /opt/gopath，则有两个目录下的文件如下所示：

```
vagrant@hyperledger-devenv:v0.2.2-58cde93: /opt/gopath/bin$ tree .
.
├── core.yaml
└── peer
vagrant@hyperledger-devenv:v0.2.2-58cde93: /opt/gopath/bin$ tree /opt/gopath/
src/github.com/hyperledger/fabric/peer
.
├── core.json
└── peer
```

搜索路径顺序是：

```
./core.json
./core.toml
./core.yaml
./core.properties
./core.props
./core.prop
/opt/gopath/src/github.com/hyperledger/fabric/peer/core.json
/opt/gopath/src/github.com/hyperledger/fabric/peer/core.toml
/opt/gopath/src/github.com/hyperledger/fabric/peer/core.yaml
/opt/gopath/src/github.com/hyperledger/fabric/peer/core.properties
/opt/gopath/src/github.com/hyperledger/fabric/peer/core.props
/opt/gopath/src/github.com/hyperledger/fabric/peer/core.prop
```

所以最终获取到的配置文件是：./core.yaml，而不是 /opt/gopath/src/github.com/hyperledger/fabric/peer/core.json。

2.4　本章小结

本章零基础地介绍了如何快速体验超级账本搭建的区块链网络，我们先绕过了比较复杂的初始化配置，用官方提供的 fabric-samples 提供的配置和链码示例，展示了如何调用和查询链码，对 Hyperledger Fabric 实现的功能有一个初步的认识。这部分更为详细的初始化配置和网络部署等内容将在第 11 章会详细介绍。后面的章节我们会带你一起领略一下区块链的奥秘。