Hyperledger Fabric 开发入门

作者：左川民

准备开始

为了方便讲解介绍，开发入门环节准备了一个 Fabric 网络的例子场景。例子场景包含 2 个组织，每个组织维护 2 个 peer 节点。还有一个 solo 类型的排序服务（ordering service）。网络实体需要的加密资料（x509 证书）已经预先生成并放在恰当的目录和配置文件。开发人员不需要修改这些证书中的任何一个。examples/e2e_cli 目录包含了 docker compose 文件和一些脚本文件，这些脚本可以用来创建和测试例子场景的区块链网络。
本章内容也演示了配置生成工具 configtxgen 的用法，这个工具用于生成网络的配置。在本章结束时，你就会有一个覆盖全部功能的交易网络。

先决条件

完成下列操作安装 Fabric 源代码，然后构建 configtxgen 工具。

- 跟着以下步骤安装开发环境，确保你已经设置了$GOPATH 环境变量。这个过程会加载很多软件依赖到你的本地机器。
- 克隆 Fabric 源代码到本地。

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

- 构建 configtxgen 工具。

如果你在 LINUX 环境运行，打开一个终端窗口，然后在 fabric 目录执行如下命令：

```
cd $GOPATH/src/github.com/hyperledger/fabric
make configtxgen
# if you see an error stating - 'ltdl.h' file not found
sudo apt install libtool libltdl-dev
# then run the make again
make configtxgen
```

如果你运行的是 OSX，先要安装 Xcode 8 或以上版本。然后打开一个终端窗口，从 fabric 目录执行如下命令：

```
# install Homebrew
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
# add gnu-tar
brew install gnu-tar --with-default-names
```

```
# add libtool
brew install libtool
# make the configtxgen
make configtxgen
```

构建成功会有类似如下的信息输出：

```
build/bin/configtxgen
CGO_CFLAGS=" "
GOBIN=/Users/johndoe/work/src/github.com/hyperledger/fabric/build/bin go install
-ldflags "-X
github.com/hyperledger/fabric/common/metadata.Version=1.0.0-snapshot-8d3275f -X
github.com/hyperledger/fabric/common /metadata.BaseVersion=0.3.0 -X
github.com/hyperledger/fabric/common/metadata.BaseDockerLabel=org.hyperledger.f
abric"       github.com/hyperledger/fabric/common/configtx/tool/configtxgen
Binary available as build/bin/configtxgen``
```

可执行文件放在 fabric 源代码目录的 build/bin/目录下。

拉取 fabric 镜像

为了加快开发环境搭建过程，我们用脚本执行所有繁琐工作。脚本生成了配置所需的制品，然后创建了一个本地网络，驱动 chaincode 测试。切换到 examples/e2e_cli 目录：

首先，从 docker hub 拉取 fabric 网络所需要的镜像。

```
# make the script an executable
chmod +x download-dockerimages.sh
# run the script
./download-dockerimages.sh
```

这个过程可能会耗费几分钟时间。
在终端你会看到如下输出：

```
===> List out hyperledger docker images
hyperledger/fabric-ca           latest              35311d8617b4        5 weeks ago         239.7 MB
hyperledger/fabric-ca           x86_64-1.0.0-alpha  35311d8617b4        5 weeks ago         239.7 MB
hyperledger/fabric-couchdb      latest              f3ce31e25872        5 weeks ago         1.515 GB
hyperledger/fabric-couchdb      x86_64-1.0.0-alpha  f3ce31e25872        5 weeks ago         1.515 GB
hyperledger/fabric-kafka        latest              589dad0b93fc        5 weeks ago         1.299 GB
hyperledger/fabric-kafka        x86_64-1.0.0-alpha  589dad0b93fc        5 weeks ago         1.299 GB
hyperledger/fabric-zookeeper    latest              9a51f5be29c1        5 weeks ago         1.314 GB
hyperledger/fabric-zookeeper    x86_64-1.0.0-alpha  9a51f5be29c1        5 weeks ago         1.314 GB
hyperledger/fabric-orderer      latest              5685fd77ab7c        5 weeks ago         181.6 MB
hyperledger/fabric-orderer      x86_64-1.0.0-alpha  5685fd77ab7c        5 weeks ago         181.6 MB
hyperledger/fabric-peer         latest              784c5d41ac1d        5 weeks ago         184.2 MB
hyperledger/fabric-peer         x86_64-1.0.0-alpha  784c5d41ac1d        5 weeks ago         184.2 MB
hyperledger/fabric-javaenv      latest              a08f85d8f0a9        5 weeks ago         1.424 GB
hyperledger/fabric-javaenv      x86_64-1.0.0-alpha  a08f85d8f0a9        5 weeks ago         1.424 GB
hyperledger/fabric-ccenv        latest              91792014b61f        5 weeks ago         1.294 GB
hyperledger/fabric-ccenv        x86_64-1.0.0-alpha  91792014b61f        5 weeks ago         1.294 GB
hyperledger/fabric-baseimage    x86_64-0.3.0        f4751a503f02        12 weeks ago        1.271 GB
hyperledger/fabric-baseos       x86_64-0.3.0        c3a4cf3b3350        12 weeks ago        160.9 MB
```

现在运行创建 fabric 区块链网络的脚本：

```
./network_setup.sh up <channel-ID>
```

如果你传递 channel-ID 参数，那么你的通道名将默认为 mychannel。在你的终端窗口，

你会看到各种命令的 chaincode 日志输出。当脚本执行完成，你可以看到如下的信息输出。

```
==================== Query on PEER3 on channel 'mychannel' is successful
====================
==================== All GOOD, End-2-End execution completed ====================
```

这个时候，你的区块链网络已经创建成功并运行起来了，而且已经成功地完成了测试。下边我们继续基于搭建的区块链测试网络探索一下其他高级功能。

环境清理

开发过程中，我们经常需要清除历史数据，从一个干净的环境重新开始测试，免得遗留的数据和环境信息影响调试开发，这个时候我们可以通过以下操作清除历史信息。

查看所有 docker 容器，只显示 ID

```
docker ps -aq
```

停止所有 docker 容器：

```
docker stop $(docker ps -aq)
```

删除所有 docker 容器：

```
docker rm $(docker ps -aq)
```

注意：以上的 docker rm 命令执行前，必须先执行 docker stop 命令，否则会提示错误：

```
Error response from daemon: You cannot remove a running container
```

也可以直接用一个命令完成,参数-f 表示强制删除

```
docker rm -f $(docker ps -aq)
```

接下来在终端执行 docker 镜像命令查看 chaincode 镜像

```
docker images
```

该命令会输出如下类似的信息：

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
dev-peer3-mycc-1.0	latest	017e4a8a8b66	10 days ago	176.2 MB
dev-peer0-mycc-1.0	latest	69b801a8836b	10 days ago	176.2 MB
dev-peer2-mycc-1.0	latest	3cd082780051	10 days ago	176.2 MB
hyperledger/fabric-ca	latest	35311d8617b4	5 weeks ago	239.7 MB
hyperledger/fabric-ca	x86_64-1.0.0-alpha	35311d8617b4	5 weeks ago	239.7 MB
hyperledger/fabric-couchdb	latest	f3ce31e25872	5 weeks ago	1.515 GB
hyperledger/fabric-couchdb	x86_64-1.0.0-alpha	f3ce31e25872	5 weeks ago	1.515 GB
hyperledger/fabric-kafka	latest	589dad0b93fc	5 weeks ago	1.299 GB
hyperledger/fabric-kafka	x86_64-1.0.0-alpha	589dad0b93fc	5 weeks ago	1.299 GB
hyperledger/fabric-zookeeper	latest	9a51f5be29c1	5 weeks ago	1.314 GB
hyperledger/fabric-zookeeper	x86_64-1.0.0-alpha	9a51f5be29c1	5 weeks ago	1.314 GB
hyperledger/fabric-orderer	latest	5685fd77ab7c	5 weeks ago	181.6 MB
hyperledger/fabric-orderer	x86_64-1.0.0-alpha	5685fd77ab7c	5 weeks ago	181.6 MB
hyperledger/fabric-peer	latest	784c5d41ac1d	5 weeks ago	184.2 MB
hyperledger/fabric-peer	x86_64-1.0.0-alpha	784c5d41ac1d	5 weeks ago	184.2 MB
hyperledger/fabric-javaenv	latest	a08f85d8f0a9	5 weeks ago	1.424 GB
hyperledger/fabric-javaenv	x86_64-1.0.0-alpha	a08f85d8f0a9	5 weeks ago	1.424 GB
hyperledger/fabric-ccenv	latest	91792014b61f	5 weeks ago	1.294 GB
hyperledger/fabric-ccenv	x86_64-1.0.0-alpha	91792014b61f	5 weeks ago	1.294 GB
hyperledger/fabric-baseimage	x86_64-0.3.0	f4751a503f02	12 weeks ago	1.271 GB
hyperledger/fabric-baseos	x86_64-0.3.0	c3a4cf3b3350	12 weeks ago	160.9 MB

用如下命令删除这些镜像

```
docker rmi <IMAGE ID> <IMAGE ID> <IMAGE ID>
```

删除镜像后，还要切换到/fabric/examples/e2e_cli/crypto/orderer 目录,删除 orderer.block 文件和 channel.tx 文件。

配置交易生成器（Configuration Transaction Generator）

Configtxgen 工具用于创建两个制品:orderer 驱动区块和通道配置交易。Orderer 驱动区块是排序服务的创世区块，通道交易文件在通道创建的时间被广播到 orderer。

文件 configtx.yaml 包含了例子网络的定义，呈现了例子网络组建的拓扑结构，两个组织机构成员（Org0 和 Org1），每个管理和维护两个 peer。同时，每个组织也指明了每个网络实体的加密证书材料存储的文件系统位置。目录 crypto 包含了每个实体的 admin 证书，ca 证书，签名证书和私钥。目录名分别为：

admincerts cacerts keystore signcerts

为了使用方便，脚本文件 generateCfgTrx.sh 整理了运行 configtxgen 的处理过程。这个脚本执行后，会生成下列两个文件：orderer.block 和 channel.tx。文件位于 /fabric/examples/e2e_cli/crypto/orderer 目录。

运行脚本 generateCfgTrx.sh

确保你当前位于 e2e_cli 目录：

cd $GOPATH/src/github.com/hyperledger/fabric/examples/e2e_cli

generateCfgTrx.sh 脚本接收一个通道 ID 参数（channel-ID）。如果没有提供这个参数，生成的通道会取默认的通道名 mychannel。换句话说，参数 channel-ID 是可选的。

注意通道名不支持下划线，这里不会报错，但是后边启动网络会出错，这是一个大坑。

命令如下：

./generateCfgTrx.sh <channel-ID>

运行脚本后，你的终端会输出类似如下的信息：

```
2017/04/21 14:47:16 Generating new channel configtx
2017/04/21 14:47:16 Creating no-op MSP instance
2017/04/21 14:47:16 Obtaining default signing identity
2017/04/21 14:47:16 Creating no-op signing identity instance
2017/04/21 14:47:16 Serialinzing identity
2017/04/21 14:47:16 signing message
```

```
2017/04/21 14:47:16 signing message

2017/04/21 14:47:16 Writing new channel tx
```

正如前面所述，这个脚本执行后，会生成 2 个文件 orderer.block 和 channel.tx。文件位于/fabric/examples/e2e_cli/crypto/orderer 目录。

Orderer.block 文件是排序服务的创世区块，channel.tx 文件包含了新的通道的配置信息。正如先前描述的，这两个文件都是根据 configtx.yaml 文件的配置内容生成，并且包含像加密证书资料和网络节点信息的数据。

注意：你也可以手动输入 generateCfgTrx.sh 脚本中的命令执行来达到同样的效果。如果你确实打算这么做，你必须替换掉 fabric 源码目录的 configtx.yaml 文件，切换到 common/configtx/tool 目录,用 e2e_cli 目录下的 configtx.yaml 文件替换掉当前目录的 configtx.yaml 文件。然后返回到 fabric 目录执行命令，你应该在 fabric 目录运行这些命令而不是 e2e_cli 目录。

启动 Fabric 网络

使用我们之前从 docker hub 上拉取的镜像文件，用 docker-compose 文件启动区块链网络。如果你还没有拉取这些 fabric 镜像文件，根据前面“拉取 fabric 镜像”章节介绍的方法先获取那些镜像。docker-compose 文件嵌入了一个脚本文件 script.sh，这个脚本把 peers 加入到通道，然后发送读写请求给 peers。这样，你没有提交命令就可以看到交易流程。如果你不想利用脚本，想要手动执行交易，直接跳到“手动执行交易”章节。

确保你依然在 e2e_cli 目录，通过如下命令启动网络：

```
CHANNEL_NAME=<channel-id> docker-compose up -d
```

如果你创建了唯一的通道名，那么确保传入那个参数，否则传入默认的 mychannel 字符串，比如：

```
CHANNEL_NAME=zuochannel docker-compose up -d
```

执行命令后，输出日志如下：

```
Creating orderer0
Creating peer0
Creating peer1
Creating peer2
Creating peer3
Creating cli
```

大约等 30 秒，后台有一些交易发送给了 peers。执行 docker ps 命令可以查看这些活动的容器，你可以看到类似如下的输出:

```
[username@hostname e2e_cli]$ docker ps
CONTAINER ID        IMAGE                      COMMAND                  CREATED              STATUS
146c8c526a55        dev-peer3-mycc-1.0         "chaincode -peer.addr"   About a minute ago   Up About a minute
0fcf7395ad25        dev-peer0-mycc-1.0         "chaincode -peer.addr"   About a minute ago   Up About a minute
decb8d22b06a        dev-peer2-mycc-1.0         "chaincode -peer.addr"   About a minute ago   Up About a minute
7e0b9dba133b        hyperledger/fabric-peer    "peer node start --pe"   About a minute ago   Up About a minute
1a359f770422        hyperledger/fabric-peer    "peer node start --pe"   About a minute ago   Up About a minute
1eb6172132dc        hyperledger/fabric-peer    "peer node start --pe"   About a minute ago   Up About a minute
08b9cc93b274        hyperledger/fabric-peer    "peer node start --pe"   About a minute ago   Up About a minute
f3581c66b844        hyperledger/fabric-orderer "orderer"                About a minute ago   Up About a minute
```

docker-compose 背后的执行过程

- Script.sh 脚本存放在 CLI 容器内部。脚本驱动创建通道命令 createChannel 执行，通道名是用户输入的通道参数名。

- 创建通道命令 createChannel 会读取先前创建的 channel.tx 文件。createChannel 命令执行后会生成一个该通道对应区块链的创世区块文件 channelname.block，默认是 mychannel.block。该文件存储在文件系统中。

- joinChannel 命令将创世区块 mychannel.block 信息传递给 4 个 peer。

- 现在我们已经有一个包含 4 个 peer 的通道，还有 2 个组织。Peer0 和 peer1 属于组织 Org0，peer2 和 peer3 属于组织 Org1。这些组织信息定义在 configtx.yaml 中。

- Chaincode 的例子 chaincode_example0 安装在 peer0 和 peer2。

- 然后 Chaincode 在 peer2 上实例化。实例化 chaincode 指的是启动容器，然后初始化与链代码相关的 key/value 键值对。这个例子初始值是[“a”,”100” “b”,”200”]。实例化的结果是名字为 dev-peer2-mycc-1.0 的容器启动。注意这个容器上特定于 peer2 的。

- 实例化命令也可以传递背书策略参数。本例中背书策略为`-P "OR ('Org0MSP.member','Org1MSP.member')"，意思是任何交易必须由绑定到 Org0 或者 Org1 的 peer 背书。

- 针对 a 值的查询请求被发布到 peer0。Chaincode 先前是安装到 peer0 上，所以这会启动另外一个容器 dev-peer0-mycc-1.0，然后返回查询结果。由于没有 write 操作发生，所以“a”的值依然是“100”。

- 从 a 转移 10 给 b 的调用发送给了 peer0。

- Chaincode 安装到 peer3 上。

- 对 a 的查询发送到 peer3。这个启动了名为 dev-peer3-mycc-1.0 的第三个 chaincode 容器。查询结果 90 被返回，这反应了先前的交易是正确的。

这表明了什么？

如果要对账本进行读写操作，Chaincode 必须安装在执行读写的 peer 上。此外，只有当在 peer 上针对 chaincode 执行 read/write 操作时，这个 peer 上才会启动该 chaincode 容器。（比如，查询“a”的值）。交易导致容器启动。此外，同一通道中的所有 peer（包括那些没有 install chaincode 的 peer，就像上例中的 PEER3）维护账本的精确拷贝，账本包含了区块链，区块链上以区块的形式存储了不可变的，有序的记录。还包含一个维护当前 fabric 状态的状态数据库。最终，在 peer 上 install chaincode 之后就可以直接使用该 peer 上的 chaincode 了（就像上例中的 PEER3），因为之前已经 instantiate 过了 （译注：即同一 channel 中的 chaincode 只需一次 instantiate）。

查看交易

查看 docker cli 容器的日志

```
docker logs -f cli
```

类似的输出如下：

```
2017-04-21 07:29:47.000 UTC [logging] InitFromViper -> DEBU 001 Setting default
logging level to DEBUG for command 'chaincode'
2017-04-21 07:29:47.004 UTC [msp] GetLocalMSP -> DEBU 002 Returning existing local
MSP
2017-04-21 07:29:47.004 UTC [msp] GetDefaultSigningIdentity -> DEBU 003 Obtaining
default signing identity
2017-04-21 07:29:47.004 UTC [msp] Sign -> DEBU 004 Sign: plaintext:
0A91050A5B0803220B796F7572636861...6D7963631A0A0A0571756572790A0161
2017-04-21 07:29:47.004 UTC [msp] Sign -> DEBU 005 Sign: digest:
97B886DD5EFDEBC46F12B38E2C5EA5C0E4AEE17CADAF04CF596809FAC04C6CA9
Query Result: 90
2017-04-21 07:29:47.892 UTC [main] main -> INFO 006 Exiting.....
===================== Query on PEER3 on channel 'yourchannel' is successful
=====================
===================== All GOOD, End-2-End execution completed =====================
```

也可以实时查看日志，这个过程需要两个终端，首先删除 docker 容器：

```
docker stop -f $(docker ps -aq)
docker rm -f $(docker ps -aq)
```

在第一个终端启动 docker-compose 脚本

```
# add the appropriate CHANNEL_NAME parm
CHANNEL_NAME=<channel-id> docker-compose up -d
```

在第二个终端执行脚本：

```
docker logs -f cli
```

这个命令会输出由脚本 script.sh 产生的交易的活动日志。

查看链代码日志

对每个 chaincode 容器单独查看 log，输出：

```
$ docker logs dev-peer2-mycc-1.0
04:30:45.947 [BCCSP_FACTORY] DEBU : Initialize BCCSP [SW]
ex02 Init
Aval = 100, Bval = 200

$ docker logs dev-peer0-mycc-1.0
04:31:10.569 [BCCSP_FACTORY] DEBU : Initialize BCCSP [SW]
ex02 Invoke
Query Response:{"Name":"a","Amount":"100"}
ex02 Invoke
Aval = 90, Bval = 210

$ docker logs dev-peer3-mycc-1.0
04:31:30.420 [BCCSP_FACTORY] DEBU : Initialize BCCSP [SW]
ex02 Invoke
Query Response:{"Name":"a","Amount":"90"}
```

手动执行交易

用下列命令删除当前运行的容器

```
docker rm -f $(docker ps -aq)
```

如果不加-f 参数就必须先执行命令：docker stop $(docker ps -aq)
在终端执行 docker images 命令查看 chaincode 镜像命令，输出结果类似如下：

```
chuan@docker-test01-vm ~]$
chuan@docker-test01-vm ~]$ docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
dev-peer3-mycc-1.0          latest              017e4a8a8b66        13 days ago         176.2 MB
dev-peer0-mycc-1.0          latest              69b801a8836b        13 days ago         176.2 MB
dev-peer2-mycc-1.0          latest              3cd082780051        13 days ago         176.2 MB
```

确保你有配置文件制品（orderer.block 和 channel.tx，文件位于 /fabric/examples/e2e_cli/crypto/orderer 目录），如果没有通过下列脚本生成：

```
./generateCfgTrx.sh <channel-ID>
```

我这里用的是通道名是 zuochannel,命令如下：

```
./generateCfgTrx.sh zuochannel
```

修改 docker-compose 文件

打开 docker-compose.yaml 文件，添加注释到 script.sh 文件前面。导航到 cli 镜像前面找到如下的行，在前面添加#

```
working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
# command: /bin/bash -c './scripts/script.sh ${CHANNEL_NAME}'
```

保存文件，然后重新启动网络：

```
# make sure you are in the e2e_cli directory where you docker-compose script resides
# add the appropriate CHANNEL_NAME parm
CHANNEL_NAME=<channel-id> docker-compose up -d
```

这里使用前面生成配置文件一样的通道名：

```
CHANNEL_NAME=zuochannel docker-compose up -d
```

命令语法

参考 script.sh 中的创建和加入通道命令，文件位于 scripts 目录：
接下来这些 cli 命令运行在 peer0 上，你需要设置 4 个全局环境变量的值。当你在其他 peers 和 orderer 执行这些命令时，请确保覆盖了相应的值。

```
# 针对 PEER0 的环境变量设置
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer0/localMspConfig
CORE_PEER_ADDRESS=peer0:7051
CORE_PEER_LOCALMSPID="Org0MSP"
CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer0/localMspConfig/cacerts/peerOrg0.pem
```

每个 peer 的环境变量都定义在提供的 docker-compose.yaml 文件里。

创建通道

执行下列命令进入 cli 容器，

```
docker exec -it cli bash
```

成功执行命令后，你会看到如下信息：

```
root@525ddd3cce53:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

你看到的结果除了@后的容器 ID（525ddd3cce53）跟这里不同，其他都会一样。

运行如下创建通道命令，通过-c 指定通道名，-f 指定通道配置交易文件，这里的我们是 crypto/orderer/channel.tx，你也可以你自己的不同名字的配置交易文件。

```
# 文件 channel.tx 和 orderer.block 挂载到 cli 容器的目录：crypto/orderer
peer channel create -o orderer0:7050 -c zuochannel -f crypto/orderer/channel.tx --tls $CORE_PEER_TLS_ENABLED --cafile $GOPATH/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem
```

因为 channel create 命令是针对 orderer 容器运行，因此我们需要覆盖先前设置的环境变量，完整的命令如下：

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig CORE_PEER_LOCALMSPID="OrdererMSP" peer channel create -o orderer0:7050 -c zuochannel -f crypto/orderer/channel.tx --tls $CORE_PEER_TLS_ENABLED --cafile $GOPATH/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem
```

注意：下面的其他命令依然在 cli 容器中执行，而且要记住命令里每个 peer 对应的环境变量。

加入通道

通过命令 channel join 加入特定的 peer 到通道中，命令如下：

```
# 默认地，会加入 peer0 到通道中
peer channel join -b zuochannel.block
```

完整的命令如下：

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer0/localMspConfig CORE_PEER_ADDRESS=peer0:7051 CORE_PEER_LOCALMSPID="Org0MSP" CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer0/localMspConfig/cacerts/peerOrg0.pem peer channel join -b zuochannel.block
```

成功执行命令后，可以通过命令 docker exec -it peer0 bash 进入到 peer0 容器中去验证是否成功加入了通道。切换到如下目录

```
/var/hyperledger/production/ledgersData/chains/chains
```

会发现生成了以通道名命名的目录，目录下生成一个名字为 blockfile_000000 的文件。
通过更改环境变量，可以根据需要加入其它 peer 到通道中。
比如，加入 peer2 到通道中的命令如下：

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer2/localMspConfig CORE_PEER_ADDRESS=peer2:7051 CORE_PEER_LOCALMSPID="Org1MSP" CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer2/localMspConfig/cacerts/peerOrg1.pem peer channel join -b zuochannel.block
```

加入 peer3 到通道中的命令如下：

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer3/localMspConfig CORE_PEER_ADDRESS=peer3:7051 CORE_PEER_LOCALMSPID="Org1MSP" CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/cryp
```

```
to/peer/peer3/localMspConfig/cacerts/peerOrg1.pem peer channel join -b
zuochannel.block
```

在远程 peer 上安装 chaincode

安装 fabric 提供的例子 chaincode 到 peer2 上：

```
# 注意执行这个命令前先设置对应的环境变量
peer chaincode install -n mycc -v 1.0 -p
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

这里我们安装 chaincode 到 peer2 上，因此完整的命令为：

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/p
eer/peer2/localMspConfig CORE_PEER_ADDRESS=peer2:7051
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/cryp
to/peer/peer2/localMspConfig/cacerts/peerOrg1.pem peer chaincode install -n mycc
-v 1.0 -p github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02
```

在 peer2 容器的/var/hyperledger/production/chaincodes 目录下发现生成了一个 mycc.1.0 文件，这证明了 chaincode mycc 安装成功。而在其它同一通道的 peer（如 peer0,peer3,注意，我们并没有把 peer1 加入 zuochannel 通道）对应的容器里不会有此文件。

实例化 chaincode 并定义背书策略

在特定 peer 上实例化 chaincode 会启动一个 chaincode 容器，这个 chaincode 容器专门为这个特定的 peer 工作，然后为 chaincode 设置背书策略。此例中定义的背书策略是只要来自 Org0 或 Org1 的任何一个 peer 背书即可。命令如下：

```
# 执行命令前记得设置环境变量。
# 注意传递正确的-C 参数，也就是通道名，我们例子中使用的是 zuochannel.
peer chaincode instantiate -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile
$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/ca
certs/ordererOrg0.pem -C zuochannel -n mycc -v 1.0 -p
github.com/hyperledger/fabric/examples/chaincode/go/chaincode_example02 -c
'{"Args":["init","a", "100", "b","200"]}' -P "OR
('Org0MSP.member','Org1MSP.member')"
```

注意这里的的-P 参数的写法，这里的背书策略是："OR ('Org0MSP.member','Org1MSP.member')"，详细的背书策略可以通过如下 URL 了解细节：http://hyperledger-fabric.readthedocs.io/en/latest/endorsement-policies.html

执行该命令前，先执行如下的环境变量设置命令，我们在 peer2 上实例化 chaincode。

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer2/localMspConfig CORE_PEER_ADDRESS=peer2:7051
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer2/localMspConfig/cacerts/peerOrg1.pem
```

调用 chaincode

执行 invoke 命令前记住设置全局环境变量

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer2/localMspConfig CORE_PEER_ADDRESS=peer2:7051
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer2/localMspConfig/cacerts/peerOrg1.pem
```

我们在peer2上执行命令,所以环境变量设置为peer2,执行invoke命令前,先执行query命令：

先执行查询命令：

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

查询结果显示：

```
Query Result: 100
```

然后执行 invoke 命令：

```
peer chaincode invoke -o orderer0:7050  --tls $CORE_PEER_TLS_ENABLED --cafile $GOPATH/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem  -C zuochannel -n mycc -c '{"Args":["invoke","a","b","10"]}'
```

注意，通道名不能错，如果这里写成 -C mychannel,必须跟实际创建通道的名字一致。通道名错误，会报如下错误：

```
Error: Error endorsing invoke: rpc error: code = 2 desc = Failed to deserialize creator identity, err MSP Org1MSP is unknown
```

再次执行查询命令（查询 a 的值）：

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

查询结果显示：

```
Query Result: 90
```

再次执行查询命令（查询 b 的值）：

```
peer chaincode query -C zuochannel -n mycc -c '{"Args":["query","b"]}'
```

查询结果显示：

```
Query Result: 210
```

如果你细心，就会发现，我们实例化 chaincode 的时候 b 的值是 200，现在是 210,再次验证了我们的交易成功执行了。

使用 CouchDB

可以将 stateDB 默认的 goleveldb 替换成 CouchDB。对于 CouchDB，chaincode 各功能依然可用，但将 chaincode 数据以 JSON 方式存储的话就可使用 CouchDB 的复杂查询的功能。

为了使用 CouchDB，需要下边两步启动 CouchDB 容器并将之与 peer 容器关联。

- 用如下命令构建构建 CouchDB 镜像，确保你当前处于 fabric 目录下
  make couchdb
  这个过程大约需要持续半个小时左右，具体取决于你的网速。couchdb 原来是 c++写的，后来居然改用 erlang 写了，可见 erlang 越来越流行了。这个过程会下载 couchdb 的源代码，然后编译。

- 打开 fabric/examples/e2e_cli/docker-compose.yaml 文件，取消跟 CouchDB 容器相关的注释。这样 chaincode_example02 就可以在 CouchDB 下运行了。

注意：如果你实现了 fabric-couchdb 容器端口到主机端口的映射，请确保你意识到可能存在的安全问题。开发环境中通过端口映射可以通过 CouchDB 的 web 接口实现数据库的可视化访问（Fauxton）。生产环境中一般不会做端口映射，以限制 CouchDB 的外部访问。

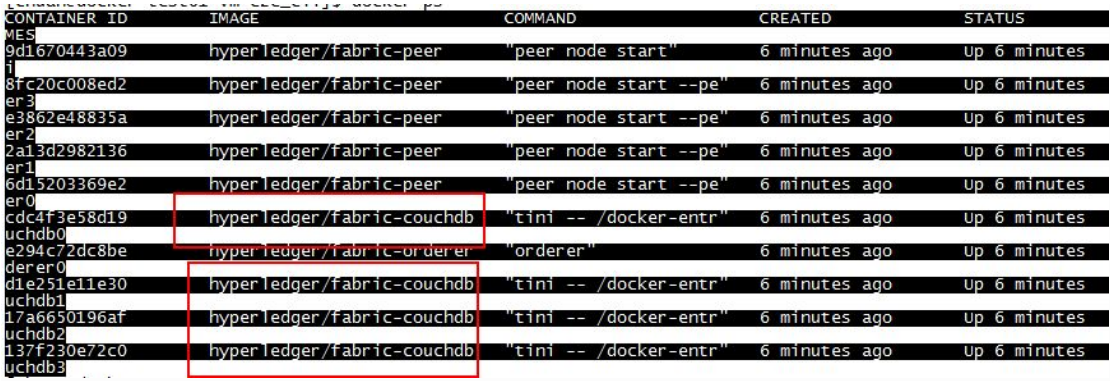
为了使用 CouchDB 的复杂查询功能，chaincode 数据一定要以 JSON 格式存储（例如 fabric/examples/chaincode/go/marbles02）

为了演示使用 CouchDB 的整个过程，我们先用 docker rm -f $(docker ps -aq)删除所有历史记录，删除镜像后，还要切换到/fabric/examples/e2e_cli/crypto/orderer 目录,删除 orderer.block 文件和 channel.tx 文件。

执行./generateCfgTrx.sh goodchannel，生成配置文件，然后执行 CHANNEL_NAME=goodchannel docker-compose up -d，启动区块链网络。命令执行完成会输出如下信息：

```
Creating couchdb2
Creating couchdb3
Creating couchdb1
Creating orderer0
Creating couchdb0
Creating peer0
Creating peer1
Creating peer2
Creating peer3
Creating cli
```

使用 docker ps 可以发现，相比没有使用 couchdb 前，多了 4 个 couchdb 容器：



```
CONTAINER ID        IMAGE                               COMMAND                  CREATED             STATUS
MES
9d1670443a09        hyperledger/fabric-peer             "peer node start"        6 minutes ago       Up 6 minutes
i
8fc20c008ed2        hyperledger/fabric-peer             "peer node start --pe"   6 minutes ago       Up 6 minutes
er3
e3862e48835a        hyperledger/fabric-peer             "peer node start --pe"   6 minutes ago       Up 6 minutes
er2
2a13d2982136        hyperledger/fabric-peer             "peer node start --pe"   6 minutes ago       Up 6 minutes
er1
6d15203369e2        hyperledger/fabric-peer             "peer node start --pe"   6 minutes ago       Up 6 minutes
er0
cdc4f3e58d19        hyperledger/fabric-couchdb          "tini -- /docker-entr"   6 minutes ago       Up 6 minutes
uchdb0
e294c72dc8be        hyperledger/fabric-orderer          "orderer"                6 minutes ago       Up 6 minutes
derer0
d1e251e11e30        hyperledger/fabric-couchdb          "tini -- /docker-entr"   6 minutes ago       Up 6 minutes
uchdb1
17a6650196af        hyperledger/fabric-couchdb          "tini -- /docker-entr"   6 minutes ago       Up 6 minutes
uchdb2
137f230e72c0        hyperledger/fabric-couchdb          "tini -- /docker-entr"   6 minutes ago       Up 6 minutes
uchdb3
```

执行 docker exec -it cli bash 进入 cli 容器。

在 cli 容器的 peer 目录下执行如下命令创建通道 goodchannel：

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig CORE_PEER_LOCALMSPID="OrdererMSP" peer channel create -o orderer0:7050 -c goodchannel -f crypto/orderer/channel.tx --tls $CORE_PEER_TLS_ENABLED --cafile $GOPATH/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem
```

创建通道后，在 cli 容器的目录 /opt/gopath/src/github.com/hyperledger/fabric/peer 下生成通道文件 goodchannel.block。

将 peer0,peer2,peer3 加入通道 goodchannel。

将 peer0 加入通道 goodchannel:

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer0/localMspConfig CORE_PEER_ADDRESS=peer0:7051 CORE_PEER_LOCALMSPID="Org0MSP" CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer0/localMspConfig/cacerts/peerOrg0.pem peer channel join -b goodchannel.block
```

将 peer2 加入通道 goodchannel:

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer2/localMspConfig CORE_PEER_ADDRESS=peer2:7051 CORE_PEER_LOCALMSPID="Org1MSP" CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer2/localMspConfig/cacerts/peerOrg1.pem peer channel join -b goodchannel.block
```

将 peer3 加入通道 goodchannel:

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer3/localMspConfig CORE_PEER_ADDRESS=peer3:7051
CORE_PEER_LOCALMSPID="Org1MSP"
CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer3/localMspConfig/cacerts/peerOrg1.pem peer channel join -b goodchannel.block
```

使用前面介绍的方法步骤针对 marbles02 chaincode 执行安装，实例化，调用和查询操作。加入通道命令执行后，用如下命令和 marbles02 chaincode 进行交互：

在 peer0 上安装 chaincode

```
CORE_PEER_MSPCONFIGPATH=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer0/localMspConfig CORE_PEER_ADDRESS=peer0:7051
CORE_PEER_LOCALMSPID="Org0MSP"
CORE_PEER_TLS_ROOTCERT_FILE=$GOPATH/src/github.com/hyperledger/fabric/peer/crypto/peer/peer0/localMspConfig/cacerts/peerOrg0.pem peer chaincode install -o orderer0:7050 -n marbles -v 1.0 -p github.com/hyperledger/fabric/examples/chaincode/go/marbles02
```

在 peer0 上实例化 chaincode（沿用前面的环境变量）

```
peer chaincode instantiate -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem -C goodchannel -n marbles -v 1.0 -p github.com/hyperledger/fabric/examples/chaincode/go/marbles02 -c '{"Args":["init"]}' -P "OR ('Org0MSP.member','Org1MSP.member')"
```

执行该语句可能遇到如下问题：

```
Error: Error endorsing chaincode: rpc error: code = 2 desc = Error starting container: Failed to generate platform-specific docker build: Error returned from build: 2 "# github.com/hyperledger/fabric/examples/chaincode/go/marbles02
chaincode/input/src/github.com/hyperledger/fabric/examples/chaincode/go/marbles02/marbles_chaincode.go:370: assignment count mismatch: 2 = 3
```

单独 go build 这个 go 文件，发现报告 ltdl.h: no such file or directory 错误，究其根源，原来是缺少第三方包，如果在 centos 操作系统中，只需安装：

```
sodo yum install libtool-ltdl-devel
```

安装后，go build 没问题，但是这里的 peer chaincode instatiate 还是报同样的错误。根据错误信息提示，是代码版本不一致造成:**接收调用函数的返回值的个数与实际函数的返回值个数不一致**。于是决定手动重新生成镜像。

手动 make docker 生成镜像，报错：

```
cp: cannot stat ‘build/docker/gotools/bin/protoc-gen-go’: No such file or directory
```

直接 go get github.com/golang/protobuf/protoc-gen-go 安装包，然后到$GOPATH/bin 目录下拷贝 protoc-gen-go 文件到 build/docker/gotools/bin/即可。

再次执行 make docker,报错：
Creating build/javashim.tar.bz2
tar (child): bzip2: Cannot exec: No such file or directory
tar (child): Error is not recoverable: exiting now
分析原因是未安装 bzip2 导致，安装 bzip2
sudo yum install bzip2
幸运地是，bzip2 是个小软件，安装顺利。

再次执行 make docker,报错：
make: stat: examples/chaincode/go/utxo/util/dah.proto: Permission denied
make: *** No rule to make target `examples/chaincode/go/utxo/util/dah.proto', needed by `build/protos.tar.bz2'. Stop.
直接修改 examples/chaincode/go 目录的权限即可，修改命令：
chmod -R 777 examples/chaincode/go

再次执行 make docker,报错：
The command '/bin/sh -c curl -sSL https://services.gradle.org/distributions/gradle-2.12-bin.zip > /tmp/gradle-2.12-bin.zip' returned a non-zero code: 56
make: *** [build/image/javaenv/.dummy-x86_64-1.0.0-snapshot-a076bba] Error 56
此问题还在调查中.....

在 peer0 上实例化 chaincode 成功后，执行下列命令移动弹珠：
```
peer chaincode invoke -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem -C goodchannel -n marbles -c '{"Args":["initMarble","marble1","blue","35","tom"]}'
peer chaincode invoke -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem -C goodchannel -n marbles -c '{"Args":["initMarble","marble2","red","50","tom"]}'
peer chaincode invoke -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem -C goodchannel -n marbles -c '{"Args":["initMarble","marble3","blue","70","tom"]}'
peer chaincode invoke -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfig/cacerts/ordererOrg0.pem -C goodchannel -n marbles -c '{"Args":["transferMarble","marble2","jerry"]}'
peer chaincode invoke -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfi
```

```
g/cacerts/ordererOrg0.pem -C goodchannel -n marbles -c
'{"Args":["transferMarblesBasedOnColor","blue","jerry"]}'
peer chaincode invoke -o orderer0:7050 --tls $CORE_PEER_TLS_ENABLED --cafile
/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/orderer/localMspConfi
g/cacerts/ordererOrg0.pem -C goodchannel -n marbles -c
'{"Args":["delete","marble1"]}'
```

如果你激活了端口映射，可以通过 couchdb 的 web 访问接口 Fauxton 通过浏览器来查看状态数据，对于运行在 vagrant 环境的容器，可以通过如下 URL 访问：
http://localhost:15984/_utils

对于非 vagrant 环境的容器，使用 CouchDB 容器指定的端口，可以通过如下 URL 访问：
http://localhost:5984/_utils
你应该可以看到一名字为 goodchannel 的状态数据库和里面的 documents。