用正确的姿势开发 Hyperledger Fabric 系列

【Node.js SDK 篇】

功夫小猫

tanzhiguo@cn.ibm.com

计划要写一系列关于 Hyperledger Fabric 开发相关的技术文章，基于版本 Fabric v1.2.0，面向有一定区块链开发基础的编程人员，侧重标准和规范方面，如果您有任何疑问、建议，欢迎通过公众号下方回复！这是系列的第一篇，关于 Node.js SDK 的开发。

0x00 环境准备

官方要求 node 的 runtime 需要 LTS version 8.9.0 或者更高，但不支持 v9.0，golang 版本 1.10.x，docker 版本 17.06.2-ce 或者更新，当然上述版本主要是针对 build 的时候。

0x01 下载框架和示例

```
tams-MacBook-Pro:hyperledger tam$ git clone https://github.com/hyperledger/fabric.git

tams-MacBook-Pro:hyperledger tam$ git clone https://github.com/hyperledger/fabric-ca.git

tams-MacBook-Pro:hyperledger tam$ git clone https://github.com/hyperledger/fabric-samples.git
```

注意：fabric 以及 fabric ca 需要放到 gopath 下，samples 可以任意

git checkout tags/v1.2.0 -b tag-v1.2.0

为每个 Repo checkout 到准确的 1.2.0 tag

0x02 启动 Blockchain 网络

您可以启动 sample 下的 basic-network 或者是 first-network，这里不详细介绍，后面会有单独的文章说明。

0x03 进一步封装 HFC

HFC 的 api 的返回，都提供了 Promise 方式，我们这里采用 async/await 形式的写法「Node 7.6 版本原生支持这种写法」，会大大节省代码量以及提高可读性，比如调用 enroll 的函数可以这样写

```
const enrollUser = async function(username, org) {
  try {
    const client = await initClient(org)
    const admins = hfc.getConfigSetting('admins')
    const adminUserObj = await client.setUserContext({ username: admins[0].username, password: admins[0].secret })
    const caClient = client.getCertificateAuthority()
    const secret = await caClient.register({
      enrollmentID: username,
      affiliation: org.toLowerCase() + '.department1'
    }, adminUserObj)
    const user = await client.setUserContext({ username: username, password: secret })
    return user
  } catch (error) {
    return error
  }
}
```

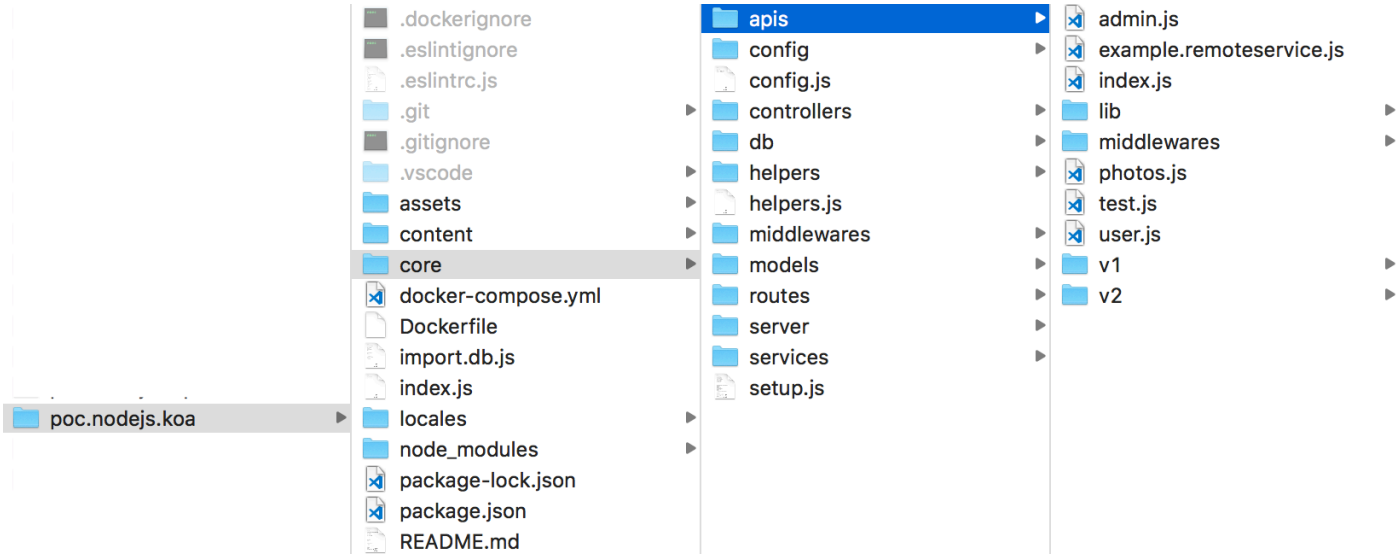
0x04 应用层后端

对于 Node 开发，常见的可以使用 express 框架或者 koa 框架，这里我们以 koa 为例，enroll 操作后才可以进行针对 channel 和 chaincode 进行调用，那么很明显，判断是否 enroll 可以通过中间件来完成，假设参数已经写进来 session 中，可以这样写，

```
router.get('/fabric', isEnrolled(), fabric.enroll)
```

```
module.exports.isEnrolled = function() {
  return async function(ctx, next) {
    const { org, username } = ctx.session.user
    const client = await fabricsdk.initClient(org)
    const user = await client.getUserContext(username, true)
    if (user && user.isEnrolled()) {
      await next()
    } else {
      ctx.status = 403
    }
  }
}
```

当然，做为标准的开发，Fabric 接口部分可能仅仅是整个应用层的一部分 api，那么我们可以把这些 api 单独提出来，目的是和业务层面的 api 相区分，而整个工程也要保持标准的开发模式，包括目录的层次结构，甚至是代码规范标准，比如使用 ESlint 约束代码可读性，对于 team 协作是非常必要的，比如我的工程是这样的，



0x05 应用层前端

至于前端的开发，就可以比较随意了，这与你是否在做区块链开发并没有直接关系，你可以使用 Vue/React/Angular/Ember 等框架技术，感兴趣可以搜索公众号历史文章，对于前端框架有相关的比较说明。

0x06 容器化

最后一个事情，那就是容器化，意义不言而喻的，为将来 scale 做准备，保证所有非状态的服务都跑在容器中，对于应用层的后端，可以通过 alpine 做为基础镜像，至于应用层前端，实际上在 build 成静态文件资源之后，只需要把这些静态资源放到应用服务器下就可以了，当然最好的方式就是使用 Nginx，也可以把前端和后端打包在一起，像 koa 和 express 都支持 static 方式，这就要看项目的规划，前端和后端的生命周期是否一致，假设你的后端服务还有很多针对其他模块访问的需要，那么还是建议分开比较好些，后端 build image 的 Dockerfile 大致是这样，

```
# Based on the node 4.7.3 image: https://github.com/nodejs/docker-node
# build docker : docker build -t tabenren/poc:node-koa-v-2.0.0 .

FROM node:8-alpine

ENV PORT 3000
ENV MONGO mongodb://database:27017/poc

# add more DNS
RUN echo "nameserver 8.8.8.8" >> /etc/resolv.conf
RUN echo "nameserver 8.8.4.4" >> /etc/resolv.conf

EXPOSE $PORT

RUN mkdir -p /apps/www

WORKDIR /apps/www

COPY ./ /apps/www/
RUN npm install

CMD [ "node", "index" ]
```

假如你希望应用崩溃后可以自动重启，那么可以使用类似 forever 的 npm，这些策略上的问题和开发普通的应用是一致的，另外需要注意开发过程中可能要考虑容器化的需要，比如 database 的地址要读 env 变量等类似问题。

0x07 小结

Fabric 官方提供 sample 只能做为借鉴，但并不适合完全照搬的，比如 balance-transfer 这个例子，是一个完整的工程，设计了日志、util 封装等等，还使用的 jwt，这并不适合你的工程完全照搬过来，你可能有自己的框架技术或者相关策略，况且 balance-transfer 里面有一些设计并不合理的地方，比如，该设计成中间件的却写在了 controller 里，也缺少必要的设计，比如，做为应用层一定会有存储的需要，那么框架需要设计存储部分，比如说我选用了 mongo 和 mongoose，怎么样把 Fabric api 设计得更小、更灵活，可能是应用层面需要注意的问题。

这是一系列关于 Hyperledger Fabric 开发相关的技术文章，欢迎您在公众号下方反馈。