TVM-Light（Fabric）

说明文档

|  |  |
| --- | --- |
| 版本 | 作者 |
| V1.0 | 倪明 |

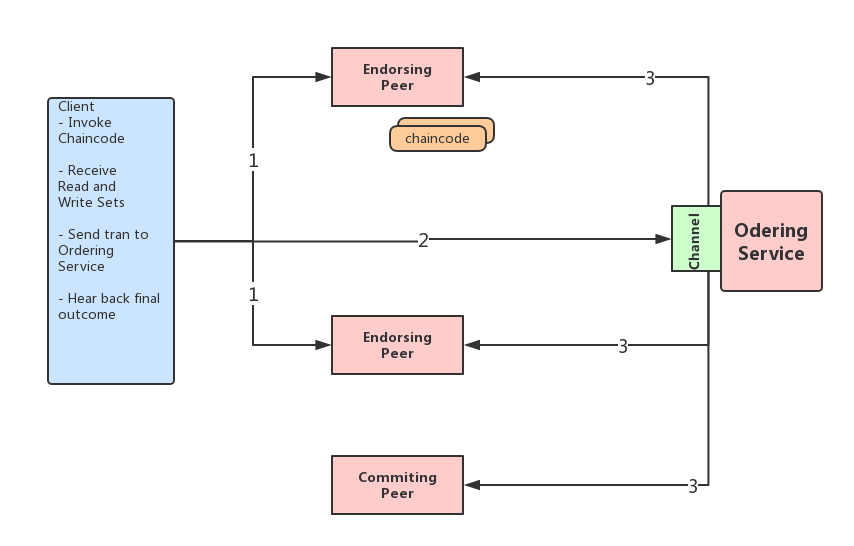
# 项目描述

## 1.1 背景描述

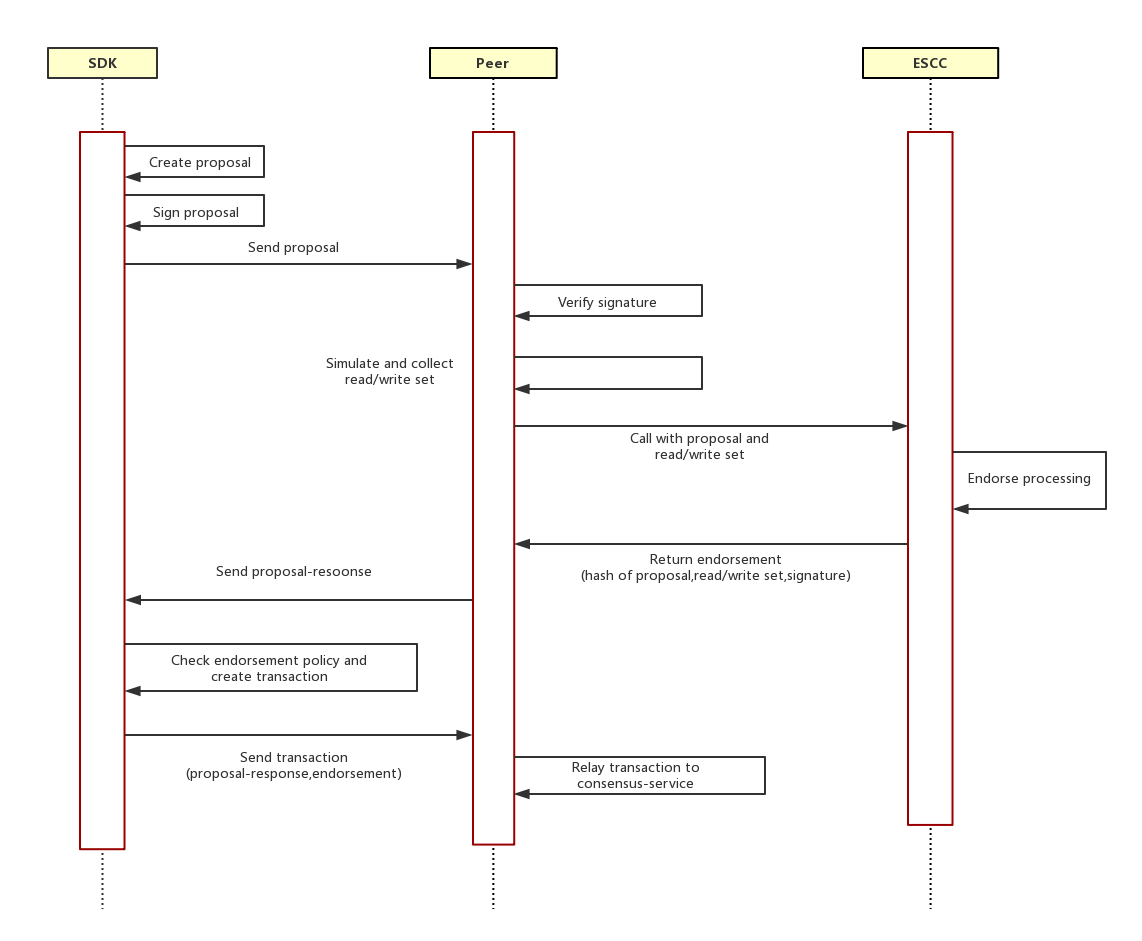
为了支持主网的智能合约应用，为了能够快速开发，部署合约，同时与TM开发语言一致，初版选用了Hyperledger Fabric 作为智能合约的执行载体，同时基于TM与Hyperledger Fabric，开发了tvm-light中间服务，用来执行tm与fabric的交互。

## 1.2 前期调研

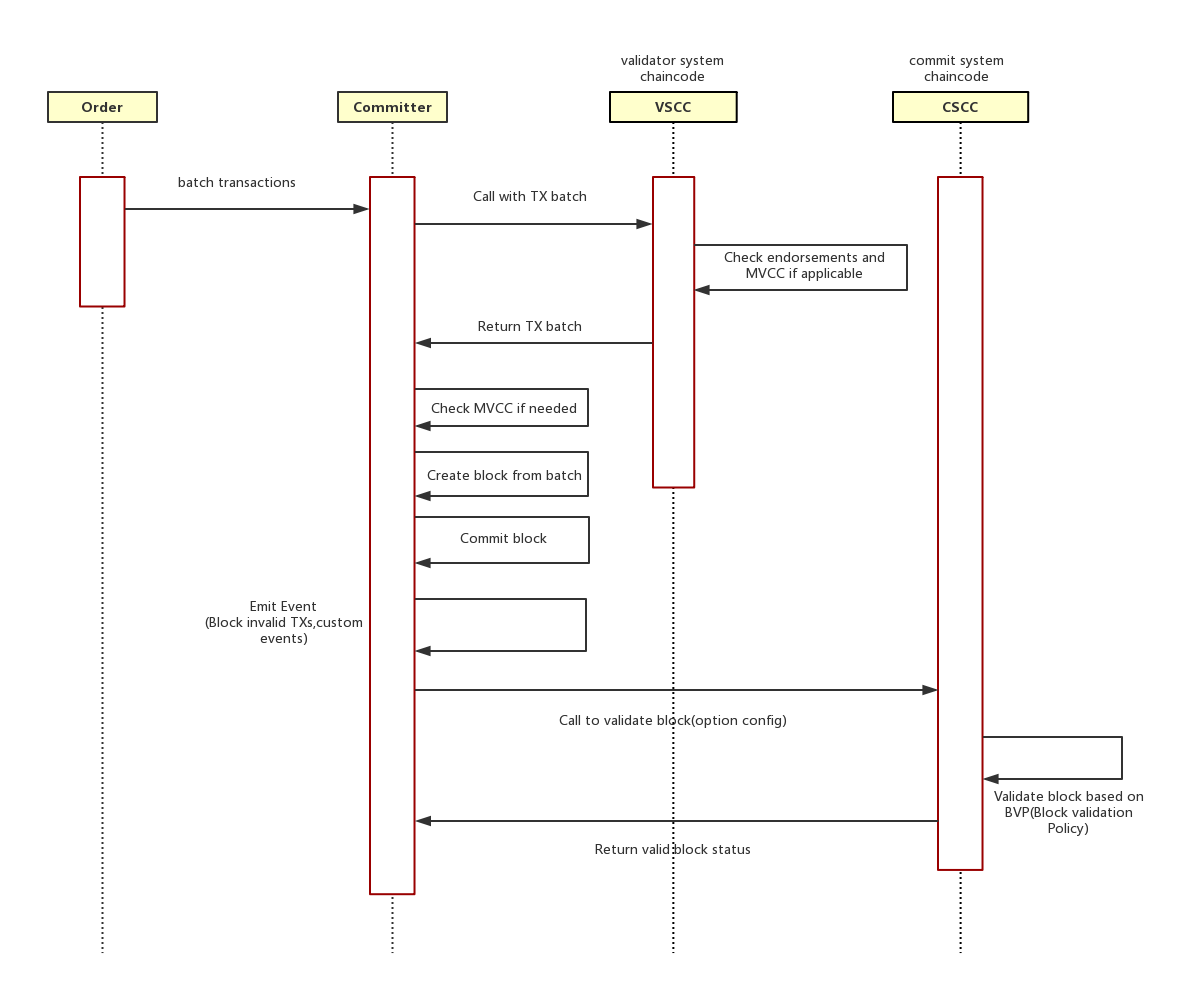
主要调研对象包括fabric整体流程，总结了调用时序以及提交交易的时序，如下图。



整体流程图



调用服务时序图



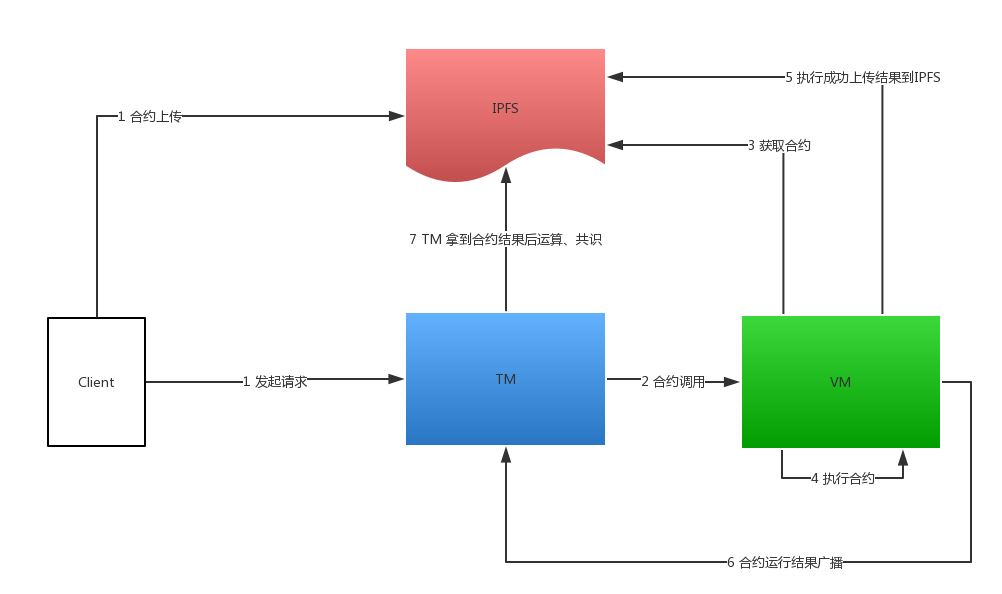
交易提交时序

Fabric自己本身通过配置可以自己实现状态同步，但是不是去中心化的过程，peer节点需要将自己的节点信息，如channelid等注册到中心化的kafka节点，这与我们的去中心化思想矛盾。

基于此，所以决定使用fabric的singlepeer模式，各节点的同步由上层TM来分发合约调用消息。

## 1.3 Fabric轻量改造

Fabric 默认使用的是leveldb，为了可以使节点快速同步状态，使用了couchdb，通过对数据文件拷贝，可以直接进行快速同步。



# Tvm-light 工程描述

## 2.1 文件说明

文件结构：



* config: 配置文件

1. common.go: 基本对象，这里存放的是通用的返回对象
2. env.go: 项目的配置环境，与config.yml一一对应
3. singlepeer: fabric单机单节点的配置项，需要copy到fabric工程目录下

* contract: 合约相关

1. base: 存放了一个基础合约(baseContract.go)，用来存放本地hash以及ipfs地址;一个样例合约(sample.go)
2. contract.go: 合约执行

* proto: grpc相关，为了支持grpc通信服务
* python-script: 俊清的代码，之前用作钱包与dag交互使用
* server: http/main.go，http接口的入口代码
* trans:

1. consensus\_service.go 虚拟机同步相关的代码
2. contract\_service.go 合约操作的前置逻辑处理代码

* utils: 常用的工具类文件
* validate: 验证代码，目前只是个空壳，没有加逻辑进去
* config.yml: 配置文件

## 2.2 运行环境

* golang
* docker
* docker-compose
* ipfs

## 2.3 部署

* Fabric v1.4.0
* $GOPATH = /opt/gopath/

注意：此处如果修改需要对应修改~/fabric/singlepeer/docker-compose-cli.yaml 对应的配置路径

* 创建如下目录

/trias-contract/contract/  
/var/hyperledger/data/  
/trias-contract/backup/

* 启动fabric

docker-compose -f /opt/gopath/src/github.com/hyperledger/fabric/singlepeer/docker-compose-cli.yaml up -d

* 启动tvm-light

1. main.go文件在~/tvm-light/server/http/main.go下
2. go build –o main main.go
3. cp main (程序目录，自定义，服务器不一定要pull完整的tvm-light项目，直接运行可执行文件即可);同理 cp ~/tvm-light/config.yml 到自定义目录
4. nohup ./main &即可后台启动

* 初始化

1. 需要安装baseContract，按照接口内的install, instantiate来即可，instantiate部分参数为（剩下参数详细参照接口文档内描述）：

{

……

"command": "{\"Args\":[\"init\"]}",

"contractName": "basic\_trias",

"contractType": "fabric",

"operation": "instantiate",

……

}

## 2.4 接口

接口详见~tvm-light/README.md,有详细描述

## 2.5 未来发展

目前是提供的http接口，存在一定的网络开销以及安全隐患，由于tm也是go语言实现，可以尝试将两个项目合并，这样降低安全风险以及减少网络通信的消耗。

更远的目标，可以尝试着借鉴Fabric或者其他智能合约的做法，自己编写一套合约的执行代码，完成与整个项目的无缝融合。