浙江大學

硕 士 研 究 生 读 书 报 告

题目 Fabric 网络部署及链码开发

作者姓名 付炬

作者学号 22051198

指导教师 李启雷

学科专业 软件工程

所在学院 软件学院

提交日期 二○二一 年 一 月

Fabric network deployment and chaincode development

A Dissertation Submitted to

Zhejiang University

in partial fulfillment of the requirements for

the degree of

Master of Engineering

Major Subject: Software Engineering

Advisor: QiLei Li

By

ju fu

Zhejiang University, P.R. China

2021

摘要

本文首先从共享账本、智能合约、共识机制、隐私保护四个方面简要介绍 Hyperledger - fabric 的基本特性，这为后文网络的搭建、链码开发风险分析提供了背景。本文的第二部分参考项目的官方文档列举了搭建 Fabric 网络的具体步骤，同时给出了实际操作过程中可能的部署优化技巧，具有很好的实操性。

本文的第三部分是读书报告的关键之处，其首先介绍了联盟链 Hyperledger-fabric 的 chaincode 和公链以太坊的智能合约之间在开发语言上的区别，这一区别正是造成 chaincode 开发漏洞较多的原因之一。紧接着本文罗列了 chaincode 开发过程中潜在的风险，对于部分风险还给出了具体的 chaincode 代码。

关键词：Hyperledger, Fabric，网络搭建，链码部署，链码开发

Abstract

This article first briefly introduces the basic characteristics of Hyperledger-fabric from the four aspects of shared ledger, smart contract, consensus mechanism, and privacy protection. This provides a background for the construction of the network and the risk analysis of chaincode development. The second part of this article refers to the official documents of the project listing the specific steps to build a Fabric network, and at the same time, it gives possible deployment optimization techniques during the actual operation, which has good practicality.

The third part of this article is the key point of the book report. It first introduces the difference in development language between the Hyperledger-fabric chaincode of the alliance chain and the smart contract of the public chain Ethereum. This difference is what caused the chaincode development loopholes. One of the more reasons. Then this article lists the potential risks in the chaincode development process, and gives specific chaincode codes for some of the risks..

Keywords： Fabric, network construction, chaincode deployment, chaincode development

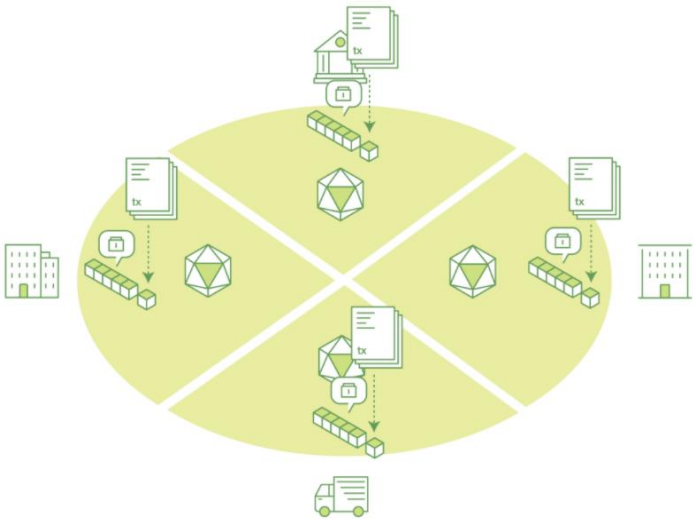
1.Hyperledger Fabric 简介

Hyperledger Fabric 是一个提供分布式账本解决方案的平台。Hyperledger Fabric 由模块化架构支撑，并具备极佳的保密性、可伸缩性、灵活性和可扩展性。Hyperledger Fabric 被设计成支持不同的模块组件直接拔插启用，并能适应在经济生态系统中错综复杂的各种场景。

1.1 共享账本

Hyperledger Fabric 包含一个账本子系统，这个子系统包含两个组件：世界状态(world state)和交易记录。在 Hyperledger Fabric 网络中的每一个参与者都拥有一个账本的副本。

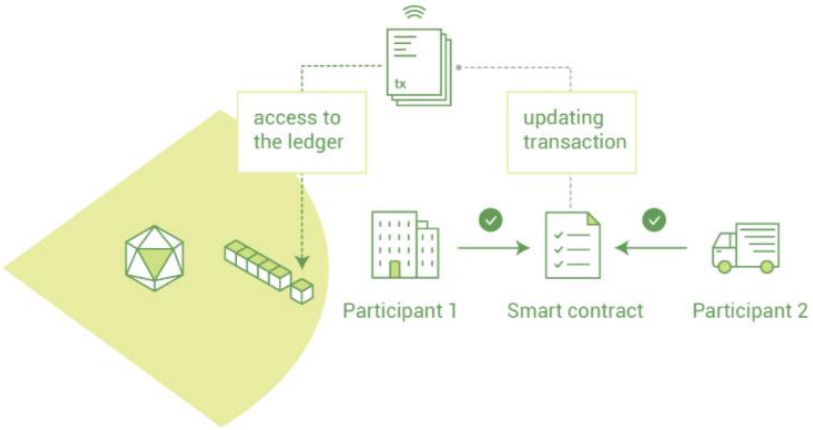
世界状态组件描述了账本在特定时间点的状态，它是账本的数据库。交易记录组件记录了产生世界状态当前值的所有交易，它是世界状态的更新历史。账本则是世界状态数据库和交易历史记录的集合。



1.2 智能合约

为了持续的进行信息的更新，以及对账本进行管理（写入交易，进行查询等），区块链网络引入了智能合约来实现对账本的访问和控制，还可以通过智能合约来自动执行由参与者定义的特定交易操作。

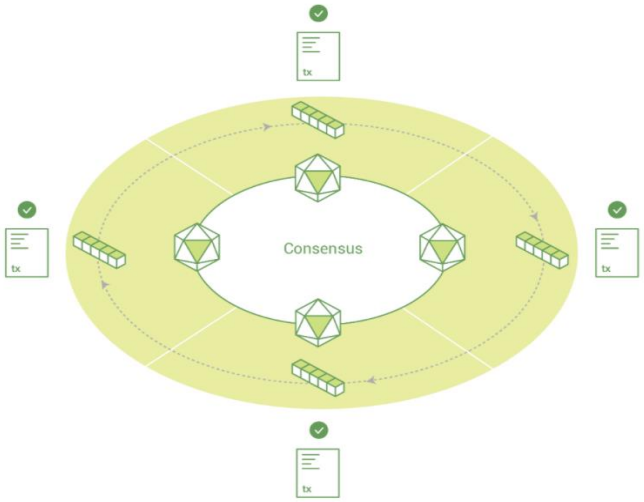
Hyperledger Fabric 智能合约被称为 chaincode，当一个区块链外部的一个应用程序需要访问账本时，就会调用 chaincode。大多数情况下，chaincode 只会访问账本的数据库组件和世界状态(world state)（比如查询），但不会查询交易记录。



1.3 共识机制

保持网络中所有账本交易的同步流程，就是共识。共识保证了账本只会在交易双方都确认后才进行更新。同时在账本更新时，交易双方能够在账本中的相同位置，更新一个相同的交易信息。

Hyperledger Fabric 被设计为允许网络构建者依据业务需求来选择采用的共识机制。好比考虑隐私性，就会有一连串的需求，从高度结构化的网络或是更加点对点的网络。



1.4 隐私保护

根据网络的需求，在一个 Business-to-Business（B2B）网络中的参与者会对信息共享的程度极为敏感。

Hyperledger Fabric 支持构建隐私保护严格的网络，也支持构建相对开放的网络。例如，通过构建 channel 通道，允许参与者为交易新建一个单独的账本，只有在同一个 channel 中的参与者，才会拥有该 channel 中的账本，而其他不在此 channel 中的参与者则看不到这个账本。

2.Fabric 网络的部署

基于 Ubuntu18.04 搭建最新版本的 Fabric 网络，成功运行官方提供的 test-network 测试网络。

2.1 搭建 Fabric 网络的主要步骤

2.1.1 搭建开发环境

```
//(1)更新系统
sudo apt-get update

//(2)安装基础工具软件
sudo apt-get install -y build-essential git make curl unzip g++ libtool

//(3)安装 docker（要求 v17.06.2-ce 及以上版本）和 docker-compose
sudo apt-get install docker docker-compose

//(4)安装 go 和 node.js
sudo apt-get install go node.js
```

2.1.2 安装示例、二进制和 Docker 镜像

Fabric 源码中提供了一个引导脚本 bootstrap.sh，该脚本能帮助用户下载示例文件 fabric-samples、docker 镜像、fabric 二进制文件并自动编译。

```
//(1)创建项目文件夹并进入
mkdir -p go/src/github.com/hyperledger/
cd go/src/github.com/hyperledger/

//(2)拉取 fabric 源码 (优化)
git clone https://github.com/hyperledger/fabric.git

//(3)进入脚本文件夹并运行之 （优化）
cd ~/go/src/github.com/hyperledger/fabric/scripts
sudo ./bootstrap.sh

//(4)fabric-samples 的 bin 路径添加进 PATH 环境变量
sudo vim /etc/profile
export PATH=$GOPATH/src/github.com/hyperledger/fabric/scripts/fabric-samples/bin:$PATH  # 保存退出
source /etc/profile #执行
```

2.1.3 运行 fabric 的测试网络

```
//(1)进入到测试网络示例文件夹并启动测试网络
cd ~/go/src/github.com/hyperledger/fabric/scripts/fabric-samples/test-network
sudo ./network.sh up

//(2)创建 channel 通道
sudo ./network.sh createChannel

//(3)在刚刚创建的通道中部署默认的链码 fabcar
sudo ./network.sh deployCC -ccn mychaincode -ccp ~/fabcar -ccv 1 -ccl javascript
```

2.2 fabric 搭建部署过程中的优化技巧

2.2.1 更换 Ubuntu 的下载源为 aliyun，加速基础开发环境的搭建

2.2.2 使用阿里云加速器提升获取 Docker 官方镜像的速度

```
sudo mkdir -p /etc/docker
sudo tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://azkmbcz4.mirror.aliyuncs.com"]
}
EOF
sudo systemctl daemon-reload
sudo systemctl restart docker
```

2.2.3 修改 bootstrap.sh 引导脚本加速 fabric 网络的搭建过程

```
//修改 cloneSamplesRepo()函数中  拉取 fabric-samples 的链接,改为从 gitee.com 码云拉取
git clone -b master https://gitee.com/nonces/fabric-samples.git && cd fabric-samples
//修改 pullBinaries()函数中 fabric 和 fabric-ca 的下载地址
download "${BINARY_FILE}" "https://github.91chifun.workers.dev//https://github.com/hyperledger/fabric/releases/download/v${VERSION}/${BINARY_FILE}"
```

3.Fabric 的 chaincode

3.1 chaincode 简介

Chaincode 是一段由 Go 语言编写（支持其他编程语言，如 Java），并能实现预定义接口的程序。Chaincode 运行在一个受保护的 Docker 容器当中，与背书节点的运行互相隔离。Chaincode 可通过应用提交的交易对账本状态初始化并进行管理

一段 chaincode 通常处理由网络中的成员一致认可的业务逻辑，故我们很可能用“智能合约”来代指 chaincode。一段 chiancode 创建的（账本）状态是与其他 chaincode 互相隔离的，故而不能被其他 chaincode 直接访问。不过，如果是在相同的网络中，一段 chiancode 在获取相应许可后则可以调用其他 chiancode 来访问它的账本。

3.2 开发 chaincode 过程中的风险

3.2.1 Fabric 的 chaincode 开发支持多种语言

Hyperledger Fabric 是第一个使用通用编程语言实现智能合约（在 Hyperledger Fabric 中称为链码）的区块链框架，而其他流行的区块链框架（如以太坊）的智能合约都是使用 DSL（domain-specific languages）实现的。

Fabric 的 chaincode 可以使用 Go、Node.js 和 Java 等通用编程语言实现的。一方面，这是有利的，因为开发人员已经熟悉标准的编程语言，并且可能已经存在验证工具。另一方面，通用编程语言的缺点是它们最初不是专为开发智能合约而设计的，它们不同于 DSL 为了编写安全的智能合约而增添某些特性和限制。

允许使用通用编程语言编写 chaincode，在带来灵活性的同时也增添了开发风险。基于 DSL 编写智能合约的以太坊已有比较成熟的漏洞检测工具，而目前帮助检测 chaincode 漏洞的工具则相对欠缺。

Category	Risks	Rationale	Source
Non-determinism Arising From Language Instructions	Global Variable	Global variable can be changed inherently.	GitHub
	KVS Structure Iteration	Key value is returned in a random order.	GitHub, [26]
	Reified Object Addresses	Addresses of memory depend on the environment.	GitHub
	Concurrency of Program	Concurrency can cause non-deterministic behavior such as race condition.	GitHub
	Generating Random Number	Each peer obtains different result of generating random number.	GitHub, Blog, [26]
	System Timestamp	It is difficult to ensure timestamp functions are executed at the same time for each peer.	GitHub, Blog, [26]
Non-determinism Caused From Accessing Outside of Blockchain	Web Service	Need to ensure that the results of calling the web service are not different among peers.	GitHub, Blog
	System Command Execution	Need to ensure that the results of calling the system command are not different among peers.	GitHub
	External File Accessing	Need to ensure that the results of accessing the file are not different among peers.	GitHub
	External Library Calling	Need to consider the behavior of the external library.	GitHub
State Database Specification	Range Query Risk	Two functions to call range query cause phantom read of the ledger.	Document, Blog
Fabric Specification	Field Declarations	It is not guaranteed that every transaction is executed on every peer. Hence, the value of a variable declared at structure field may not sustain same value among peers.	ChaincodeScanner
	Cross Channel Chaincode Invocation	No data will be committed in the other channel.	Blog
	Read Your Write	Fabric does not support Read-Your-Write semantic.	Document
Common Practices	Unhandled Errors	Should not ignore return values related to errors.	Document, ChaincodeScanner
	Unchecked Input Arguments	Should check input arguments.	Document, ChaincodeScanner

Risks in the chaincode development process

3.2.1 chaincode 潜在风险示例

Case1: Random Number Generation

```
// Something like a lottery application
// Users predict a number

// User's prediction
pred := arg[0]

// Answer
rand.Seed(seed)
sel := rand.Intn(10)

if pred == sel {
   PayPrizeToUser(user, prize)
}
```

解释：client 将初始 transaction 发送给若干个背书结点（endorsing peer），每一个背书结点都会执行该合约来模拟该 transaction 的过程，但由于生成数的随机性，不同的背书结点得到的结果可能是不一样的，这就造成了不一致性。

Case2：Read Your Write

```
1  // At the initial point: {key: "key", value
     : 0}
2  val := 1
3  // Update the value from 0 to 1
4  err := stub.PutState("key", val)
5  if err != nil {
6     fmt.Printf("Error is happened. %s", err)
7  }
8  // The method returns 0, not 1
9  ret, err := stub.GetState("key")
10 if err != nil {
11    fmt.Printf("Error is happened. %s", err)
12 }
```

解释：在 Hyperledger Fabric 中，不支持“读即写”语义。 因此，如果事务读取密钥的值，则即使在发出读取之前事务已更新密钥的值，也将返回处于提交状态的值。这段代码第 2 行将 val 的值从 0 修改为 1，第 9 行读取却依然还是 ledger 帐本中存储的 0 值。

Case3: Unhandled Errors

```
1  key := "keystring"
2  // Unhandled error is happened
3  ret, _ := stub.GetState(key)
4
5  // Unhandled error is not happened
6  ret, err := stub.GetState(key)
7  if err != nil {
8     fmt.Printf("Error is happened. %s", err)
9  }
```

Case4: Unchecked Input Arguments

```
// Unchecked input arguments is happened
ret, err := stub.GetState(args[0])

// Unchecked input arguments is not happened
if len(args) != 2 {
    return shim.Error("Incorrect number of arguments. Expecting 2")
}
ret, err := stub.GetState(args[0])
```

解释：case3 和 case4 是开发人员规范编码的体现，这种编码素养不仅仅局限于 chaincode 的开发；与此同时，类似的编码不规范均可通过开发工具提醒解决。

参考文献

[1] K. Yamashita, Y. Nomura, E. Zhou, B. Pi and S. Jun, "Potential Risks of Hyperledger Fabric Smart Contracts," 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), Hangzhou, China, 2019, pp. 1-10, doi: 10.1109/IWBOSE.2019.8666486.