实验12 PBFT共识算法简单实现

【实验介绍】

本次实验涉及到实用拜占庭容错算法（PBFT, Practical Byzantine Fault Tolerance）的基础知识及共识流程，并需要读者填充共识阶段中的代码以实现PBFT算法。通过此次实验，读者将对拜占庭容错下最经典的算法有较全面的认识和理解，这也为未来深入学习区块链内核打下了基础。为了后续实验的顺利进行，接下来将介绍PBFT的相关知识及共识流程。

（1）实用拜占庭容错算法（PBFT）

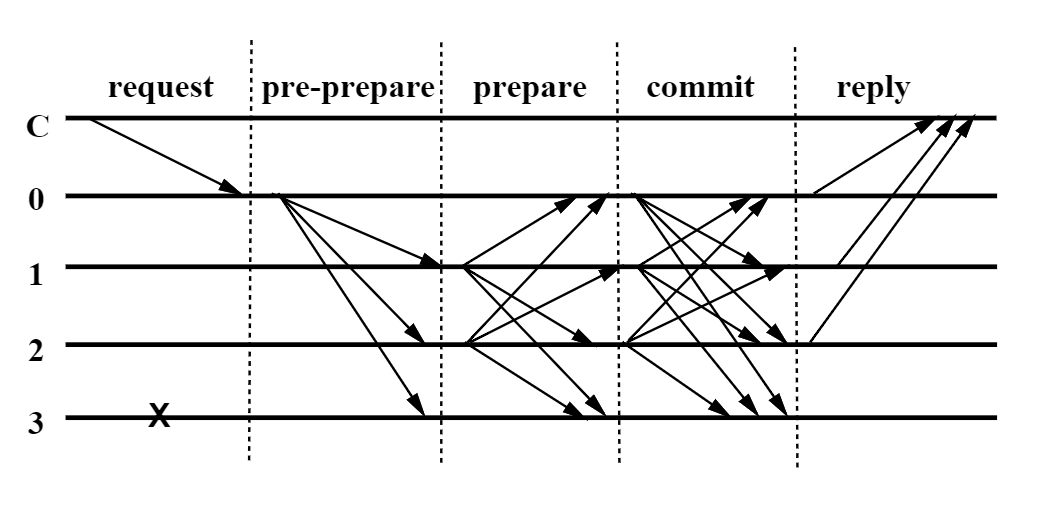
实用拜占庭容错算法于1999年被提出，它最杰出的贡献之一就是将拜占庭容错算法的时间复杂度降为了O(n2)，这使得其成为了第一个在实际系统中应用的BFT算法。PBFT应用于一个互不信任的多方网络中，可容忍1/3的恶意结点。假设有f个恶意结点，则总结点数须大于等于3f+1个才可使系统正常工作。总体上讲，每个结点在投票阶段确定收到 2f + 1个响应就可以进行下一阶段；客户端的请求发到 leader 结点，由其组织进pbft共识。下图为PBFT算法的时序图，其有三个阶段。其中0是leader结点，3是恶意结点。

图12-1 PBFT算法的共识时序

PBFT在某种程度上可以理解为是一种SMR算法。在该机制下，视图（view）是一个重要的概念。在一个view下，仅有一个主节点（leader），其余均为从节点。Leader负责将客户端的请求发送给从节点。而在BFT下，leader有可能是拜占庭的。这便要求从节点有义务检查leader的合法性，才出现异常时通过视图切换（view change）来确立新的leader。Leader的计算方法为：当前的视图号模节点的总数。接下来将介绍PBFT算法具体的共识流程：

1) REQUEST：

客户端向leader发送请求<request, o, t, c>。其中request包括请求的内容、内容的摘要及内容的签名，o表示进行的操作，t表示客户端赋予的时间戳，c表示客户端的标识。

2) PRE-PREPARE：

leader收到客户端的请求，根据签名即可判断请求是否合法。若合法则将其广播给所有从节点，广播的信息为<<pre-prepare, v, n, d>, m>。其中v是当前视图的编号，n为请求的编号，d是客户端请求内容的摘要，m为具体内容。

1. PREPARE：

从节点收到leader的消息后，需要进行如下三项校验：leader广播的消息的签名的正确性；从节点有没有收到一条在同一视图下且编号也是n，但是签名却不同的pre-prepare信息；m的摘要与d是否相同。

之后若请求合法，从节点则向其他节点发送<prepare, v, n, d, i>。其中v、n、d与先前表示的含义相同，而i是从节点的编号。

1. COMMIT：

leader和从节点收到prepare消息后，须完成如下的检验：从节点prepare消息签名的正确性；此时从节点是否收到同一视图下的n；已收到pre-prepare中的d和prepare消息中的d是否一致。

之后若从节点收到了2f+1个合法的prepare消息，则向所有节点发送形如<commit, v, n, d, i>的确认消息。其中v、n、d、i均与先前表示的含义相同。

1. REPLY：

当节点收到commit消息，须完成如下的检验：从节点commit消息签名的正确性；此时从节点是否收到同一视图下的n；计算d的摘要与m的摘要是否相同。

之后若从节点收到了2f+1个合法的commit消息，表示现已完成确认，故向客户端发送形如<reply, v, t, c, i, r>的消息，其中r表示请求操作的返回结果。若客户端收到了f+1个一致的reply消息，表明此请求已完成了共识。

【实验要求】

了解PBFT算法的原理及共识流程，并能对给出的不完整代码进行补充。

【实验准备】

1. 熟悉PBFT算法的相关知识及基本流程。

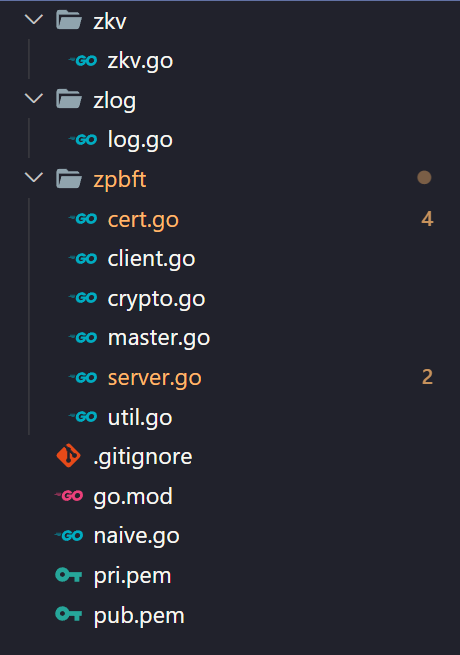
2. 熟悉本次实验所给代码。

【实验过程】

1. 实验代码介绍

本实验是一个简单的分布式系统，节点有三种角色：master、server、client。master负责登记server的注册信息（地址和公钥），并将所有server信息发给每个节点，然后server间互相建立P2P连接，等待client的请求到达。Client也需要连接master获取server集群信息，当用户输入命令后，构建请求发给server中的leader，由leader发起共识，在集群中达成一致后，执行该命令，这样所有的server将保持一致的副本状态（每个server拥有独自的kv数据库，但只要按同样顺序执行同样的命令序列，数据库的状态将保持一致）。

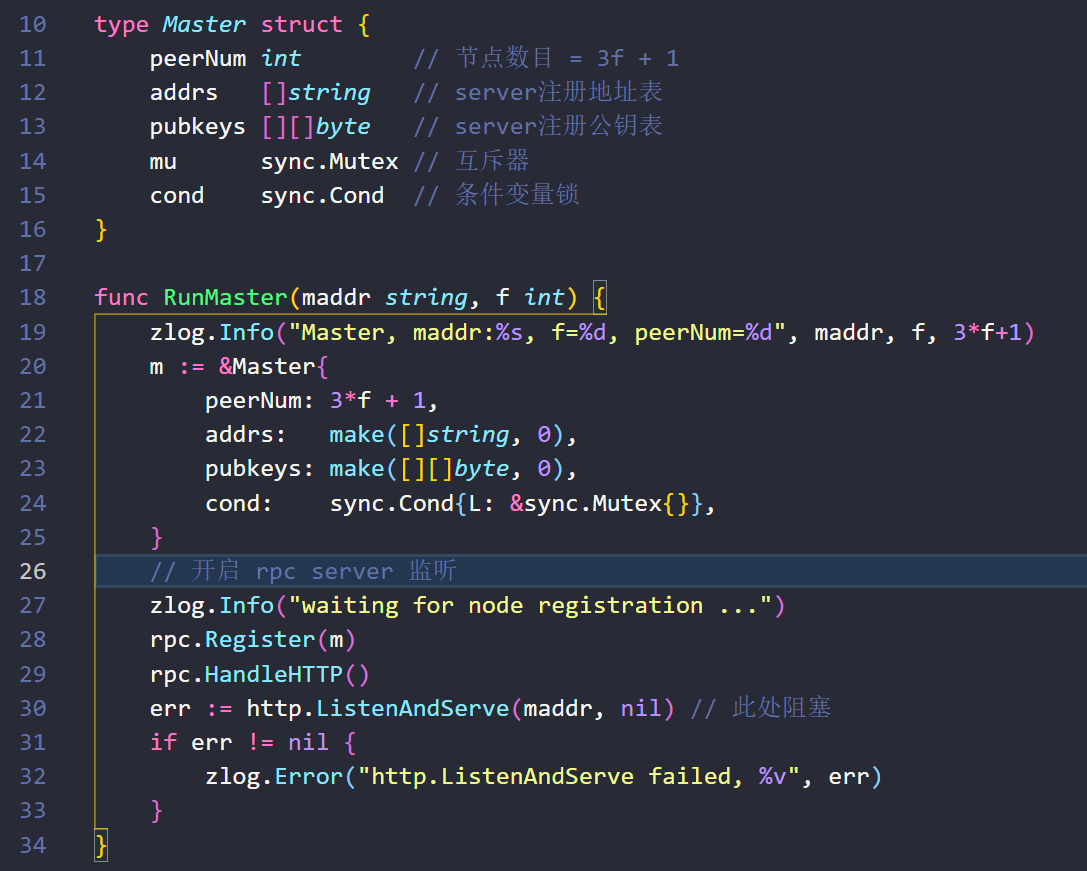
实验代码框架如下图所示，zkv是简单的kv数据库，zlog是分级别打印日志，zpbft是共识协议实现，其中有master、server、client的实现代码框架。



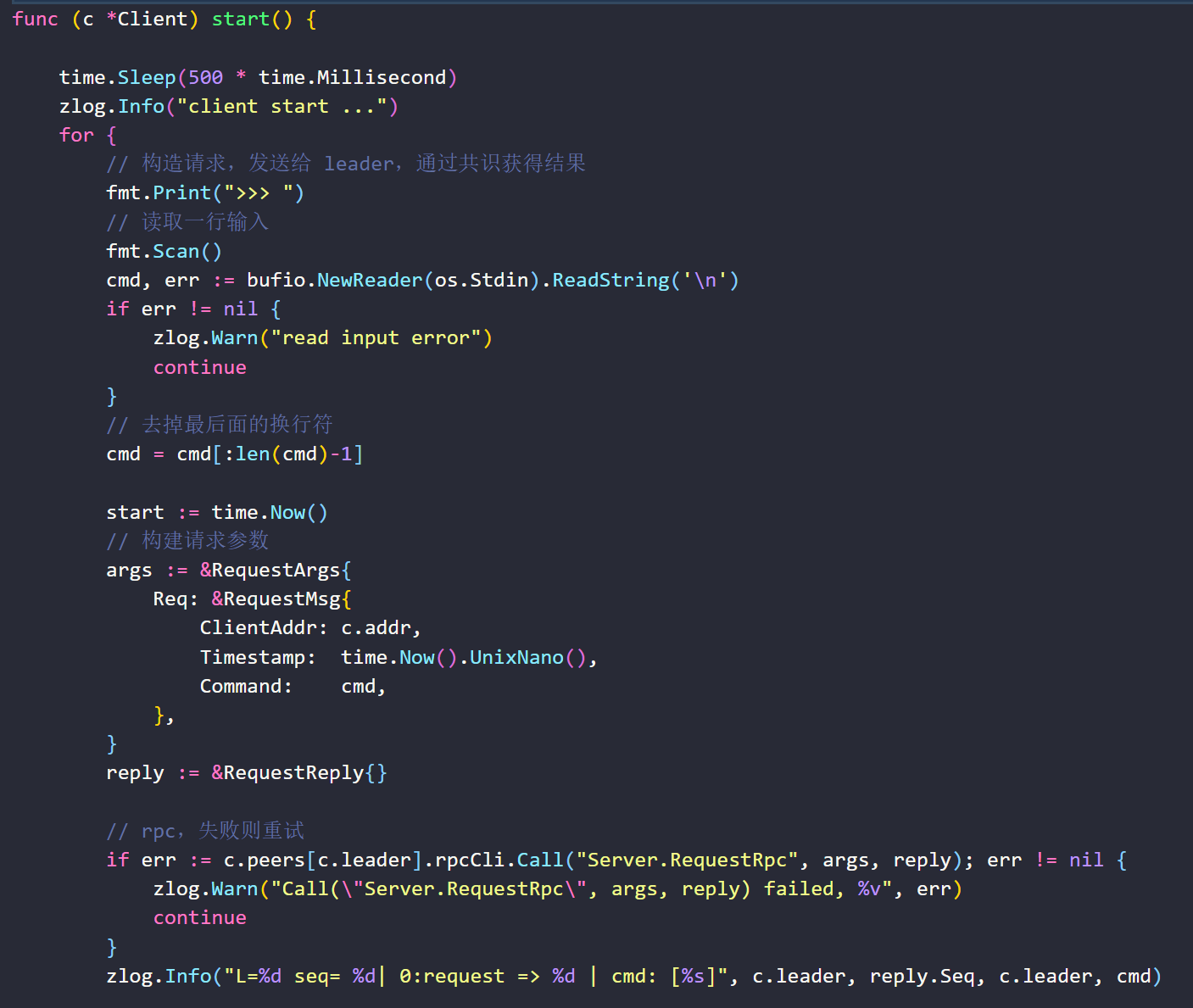
主函数代码在naive.go中，其解析命令行参数（下图红框注释部分），通过指定不同的role参数选择不同身份运行。



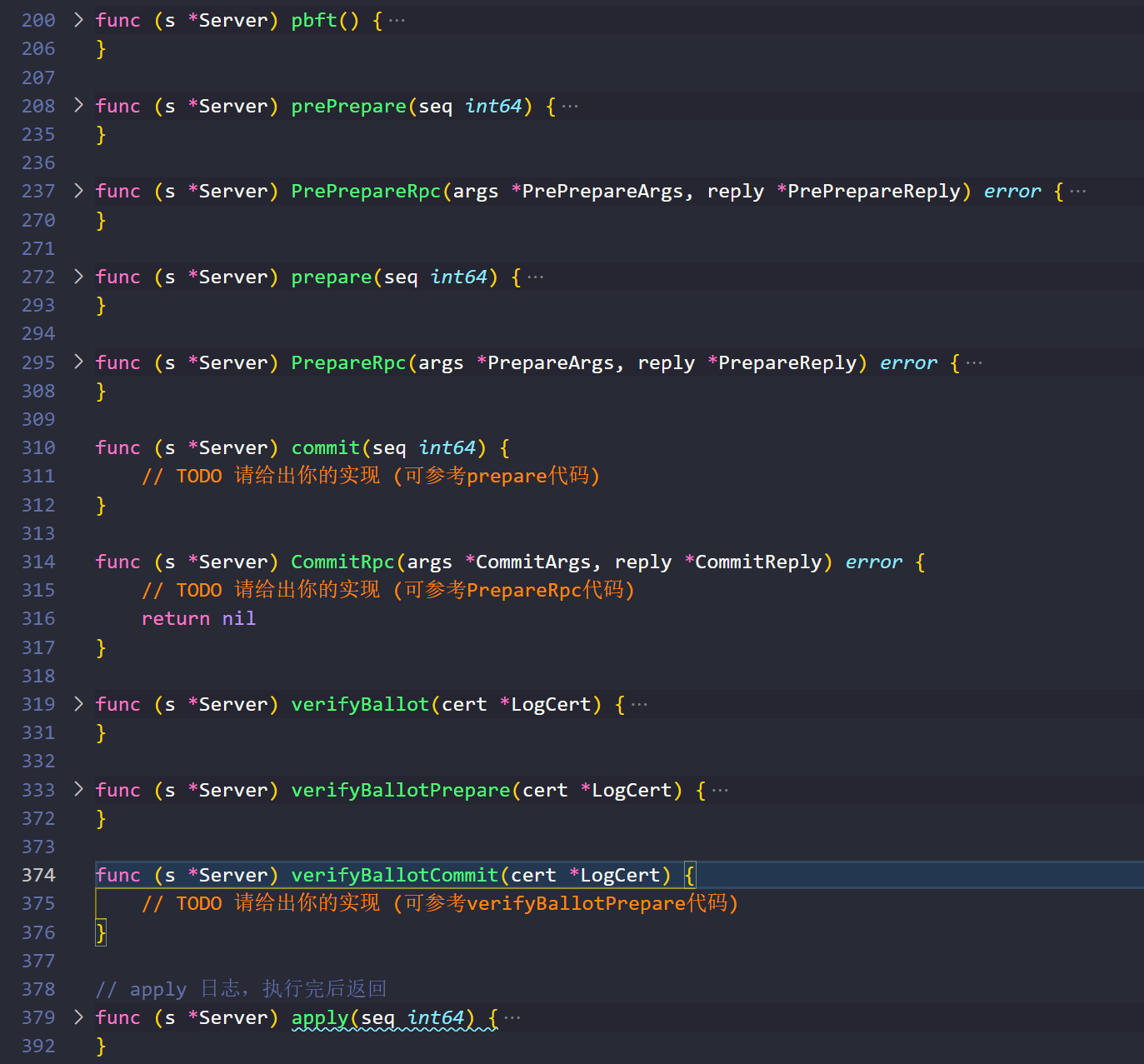
master.go提供了master功能的全部实现，其具有两个rpc，一个提供给server进行注册，一个提供给client获取server信息，所以master要先启动，这两个rpc会阻塞直至所有server都注册成功后再返回。master需要f参数（即pbft中的容错数）来计算集群需要的server数（3f+1），本实验f默认为1，需要启动四个server去注册，如果f=2，则需要7个。



client.go提供了client功能的全部实现，在连接master获取server信息后，就可以接受用户的命令，构造请求发送给leader，等server进行共识后，server会发送reply给client，client确定接受到f+1个reply消息后就确定该命令执行成功，打印输出结果。下图为client接受命令构造请求通过RPC发送给leader，其他逻辑见代码。



server.go提供了server端的大部分代码，在向master注册信息返回并与其他节点建立连接后，leader等待client请求到达，然后发起一轮pbft共识，经过三阶段的消息交互，执行该命令，改变本节点的zkv数据库状态，并将结果异步返回给client。server的具体逻辑见代码，下图是你需要实现的三个函数，注释给出了相关提示。

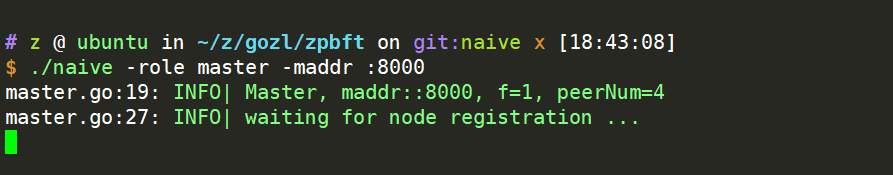


2. 执行实验代码

实现完上面提及的三个函数后，在项目目录下执行 go build naive.go 进行编译，生成可执行文件。(本实验请在linux环境下进行，其他系统因为换行符格式存在命令解析问题）

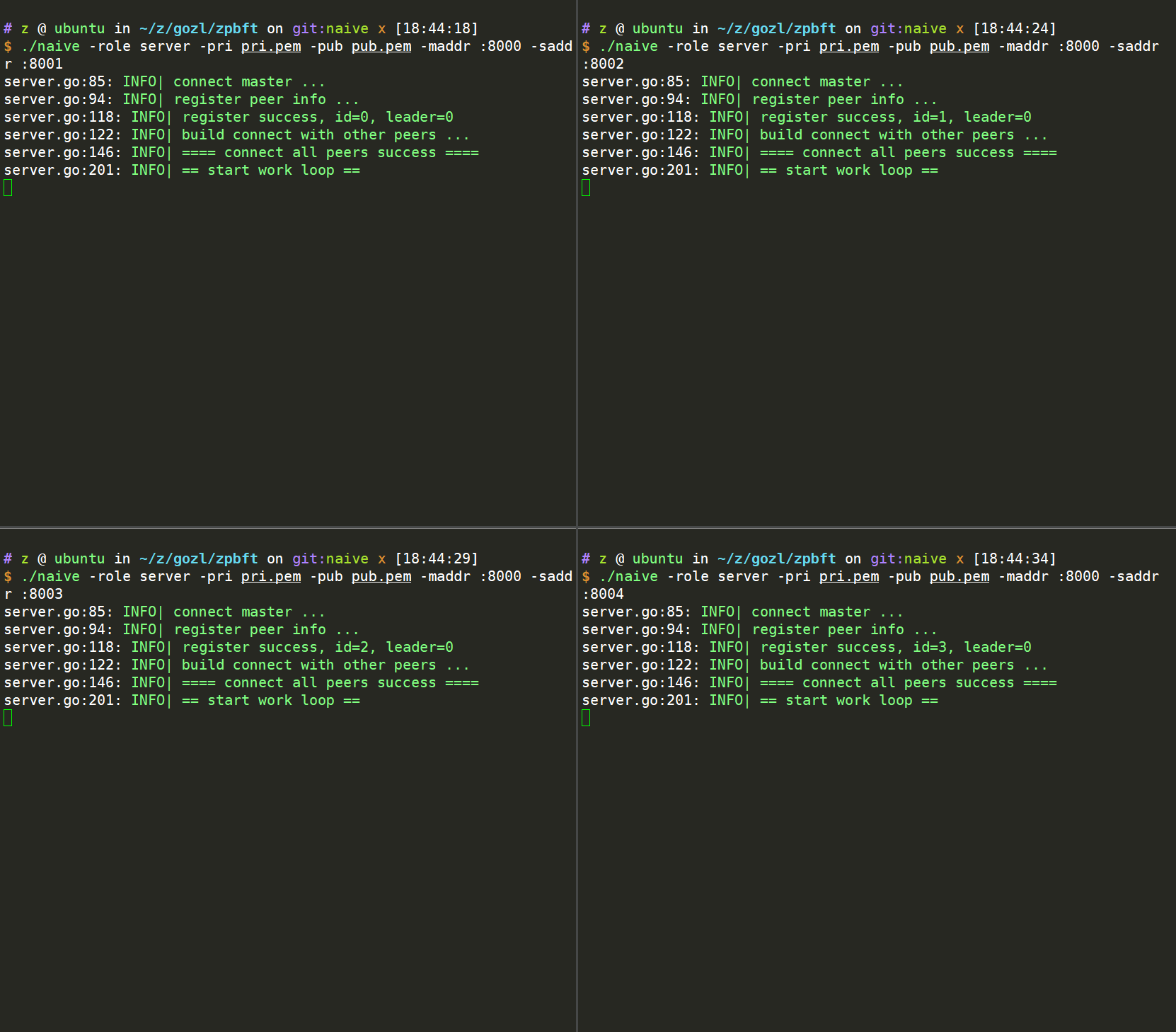
先启动master，f参数默认为1可不写。

./naive -role master -maddr :8000



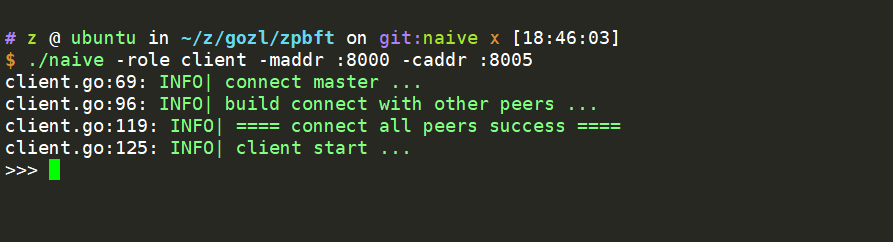
再启动四个server，下图可以看到id=0，addr=localhost:8001的server成为leader。（默认id=0为leader，但id的分配与谁先注册到master有关)

./naive -role server -pri pri.pem -pub pub.pem -maddr :8000 -saddr :8001



最后启动client。

./naive -role client -maddr :8000 -caddr :8005



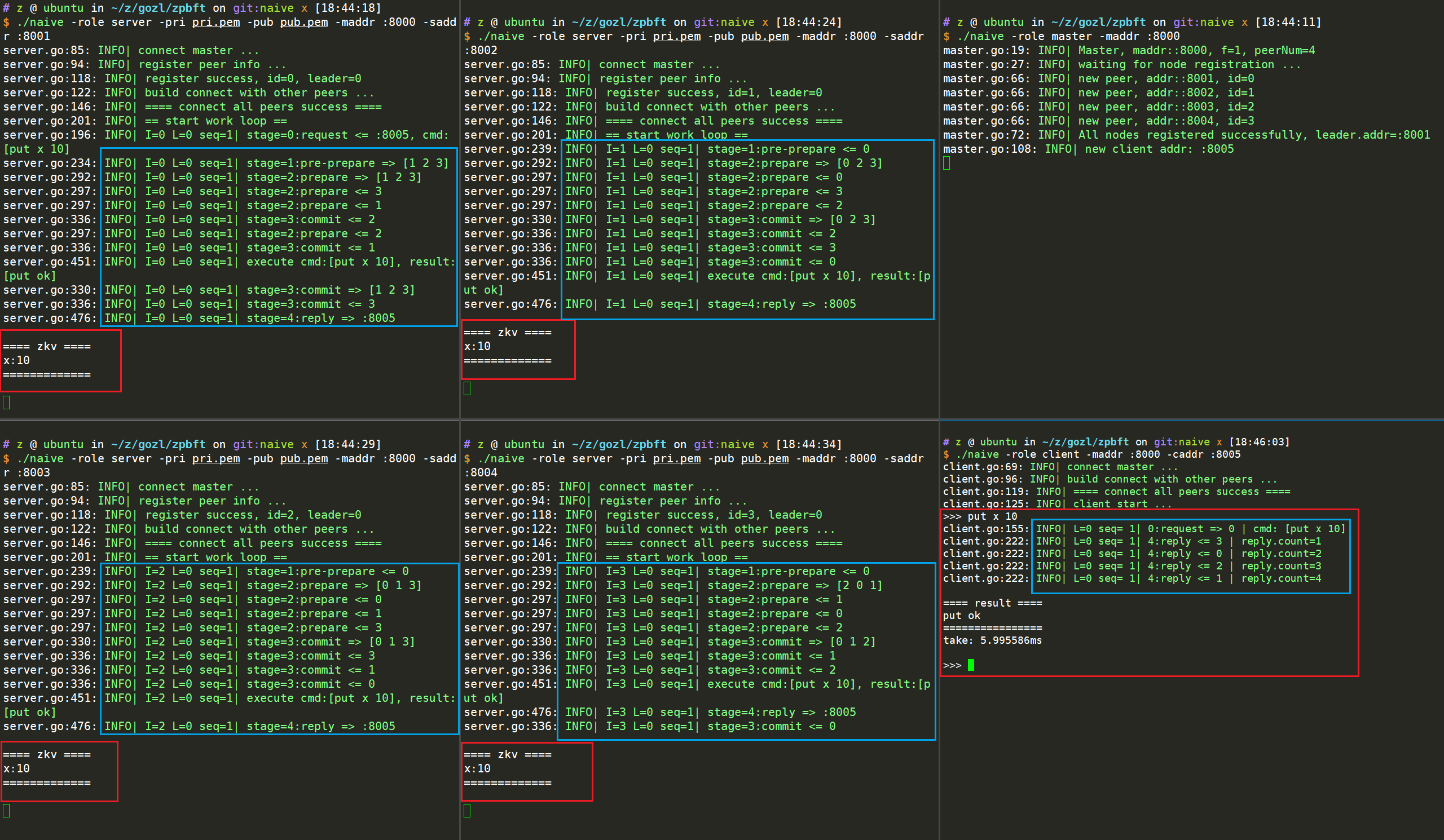
此时master端输出了相关注册信息：



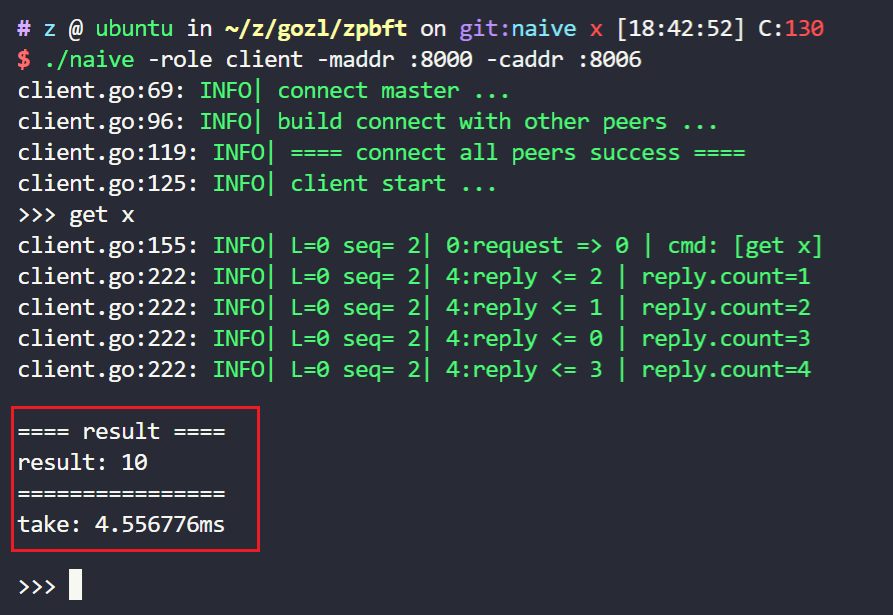
本实验的zkv数据库支持四种操作。

usage: get <key>, put <key> <val>, del <key>, all

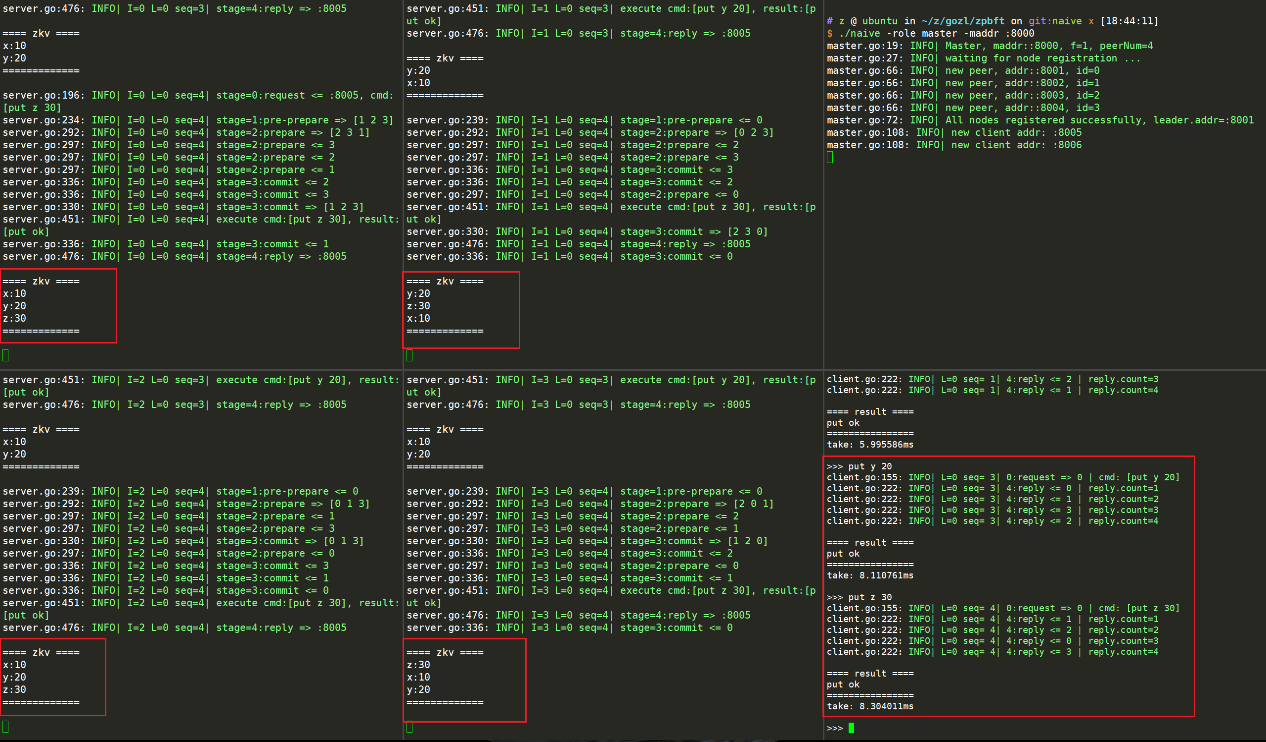
Client输入命令put x 10，如下图所示，右上角为master，右下角为client，左四个框为server，蓝色框内容为pbft共识协议消息交互过程，=》和《=为发送方向，红色框为执行结果，四个server都执行了该命令，对本命令达成共识。



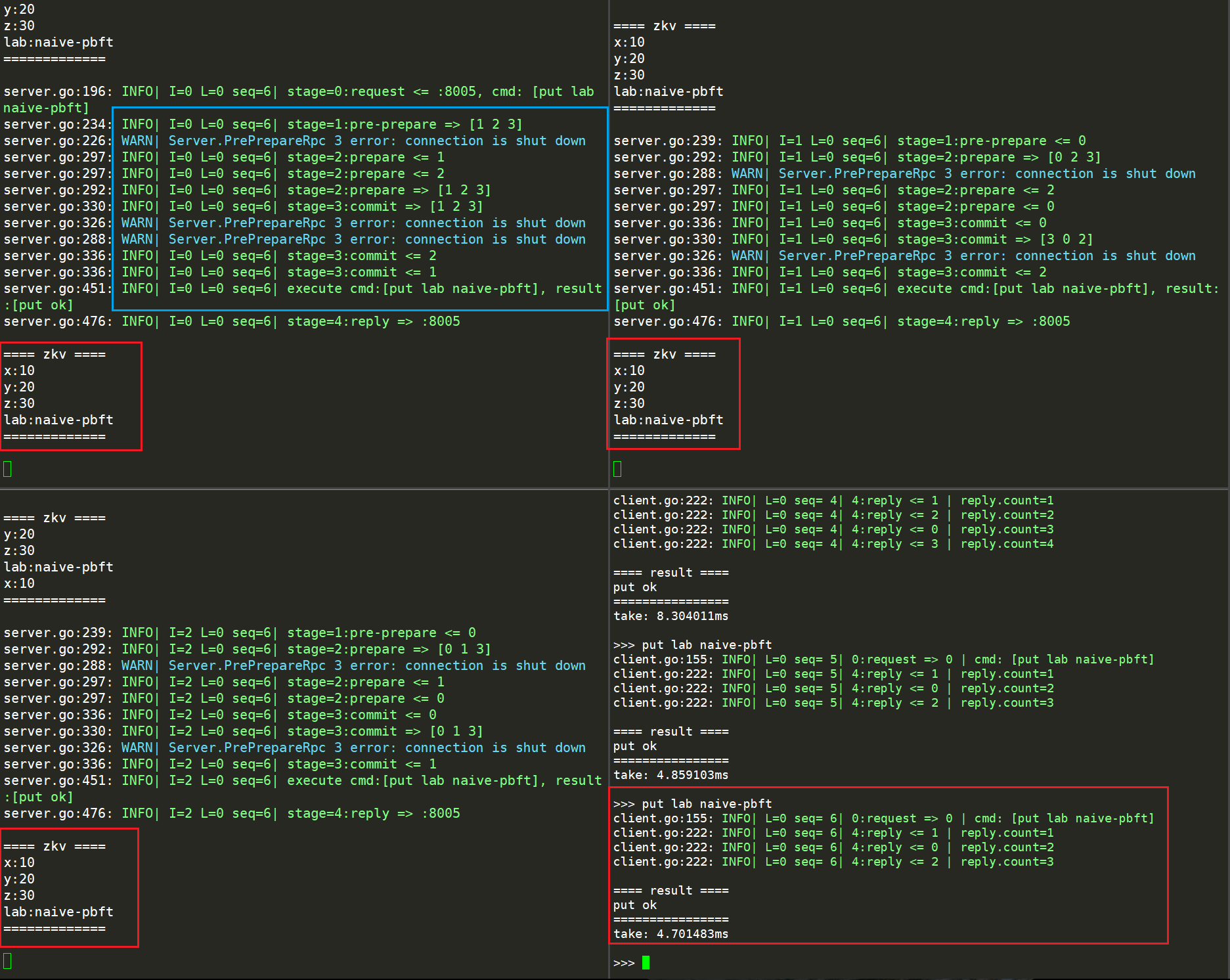
然后通过get x命令可以获取结果，这里我们启动另一个客户端来查看。



此时回去查看server，会发现server又根据这个客户端的命令达成共识。



此时关闭id=3的server（右下角的server），在故障节点为1的情况下仍然能达成共识，见下图，如果故障节点为2，则无法达成共识。



其他的操作可见演示视频。

【实验小结】

本次实验介绍了实用拜占庭容错（PBFT）算法的背景知识、基本概念及共识流程，其中对共识流程的5个子阶段(REQUEST、PRE-PREPARE、PREPARE、COMMIT、REPLY)展开详细讲解，并描述了主从节点在不同子阶段的行为以及它们如何实现对客户端请求的成功共识。除此之外，实验实现了简单的PBFT算法并要求读者对代码中缺失的部分进行补全。这加深了读者对拜占庭环境下容错机制的理解，也为进一步学习区块链系统内核打下了坚实的基础。

【习题】

1.请问在节点数量很大时PBFT算法的性能如何？

2.PBFT算法解决了什么问题？请描述其五个阶段。

3.以太坊、Fabric等区块链平台均开始使用Golang，经过使用你觉得Golang的优势在哪里？

【参考文献】

1. Castro M, Liskov B. Practical Byzantine fault tolerance and proactive recovery[J]. ACM Transactions on Computer Systems(TOCS), 2002, 20(4): 398-461.
2. Lampotr L. The Part-Time Parliament[J]. ACM Transactions on Computer Systems, 1998, 16(2): 133-169.
3. Lamport L, Shostak R, Pease M. The Byzantine Generals Problem[J]. ACM Transactions on Programming Languages and Systems, 1982, 4(3): 382-401.