## 序言

本项目为清华大学《大数据系统软件》（分布式数据库系统）课程的实验框架，该框架基于MIT的6.824课程使用的RPC框架搭建。与原框架使用K-V存储模型不同，该框架了包含一个简单的二维表数据模型以及对应的单节点的基础读写功能。本课程的实验目的为让同学们自己动手实现一些分布式数据库系统中比较重要也比较常见的功能，例如数据划分，分布式查询以及数据副本，从而加深对这些功能的实现原理以及实际作用的理解，进而更好地掌握课程内容。服务于上述实验目的，本框架提供与分布式数据库系统关联较小，但会在实验过程中使用到的RPC，数据模型，单节点读写等功能的简单实现，来使得同学们可以在实验过程中专注于分布式数据库系统对应功能的开发而无需从头完成所有代码。

目录

[序言 1](#_Toc28520)

[背景介绍 1](#_Toc1876)

[RPC 1](#_Toc27073)

[数据模型 1](#_Toc10390)

[集群架构 2](#_Toc19381)

[使用示例 3](#_Toc19891)

[注意事项 5](#_Toc10011)

[在RPC中使用接口类型 5](#_Toc22633)

## 背景介绍

### RPC

RPC（Remote Procedure Call，远程过程调用），简单来说，就是支持从一台计算机调用另一条计算机提供的函数并接收响应返回结果的一类技术，通常用于设计各类分布式系统中节点的通信协议。在本地过程调用中，一个线程首先进行内存空间（栈帧）的分配，并拷贝函数参数到栈帧的对应位置，之后移动PC（程序计数器）至被调用函数的开头来开始被调用函数的执行，被调用函数执行完毕后将结果写入到内存中指定的位置，之后弹出栈帧并将PC移回到调用处的下一个位置，来完成整个调用过程。而在一般的RPC实现中，调用者（通常也称客户端）通过网络将编码后的函数名和参数等信息发送给被调用者（通常也称为服务器），服务器解析请求后执行对应函数，并将结果编码后通过网络返回给调用者。

### 数据模型

本框架提供一个简单的二维表数据模型来进行数据的读写。注意该模型尚且不是一个关系型表结构，因为它不支持主键等约束，我们欢迎各位同学对其进行扩展来得到一个完整的关系表，但这并不一定会在之后的实验中被使用。

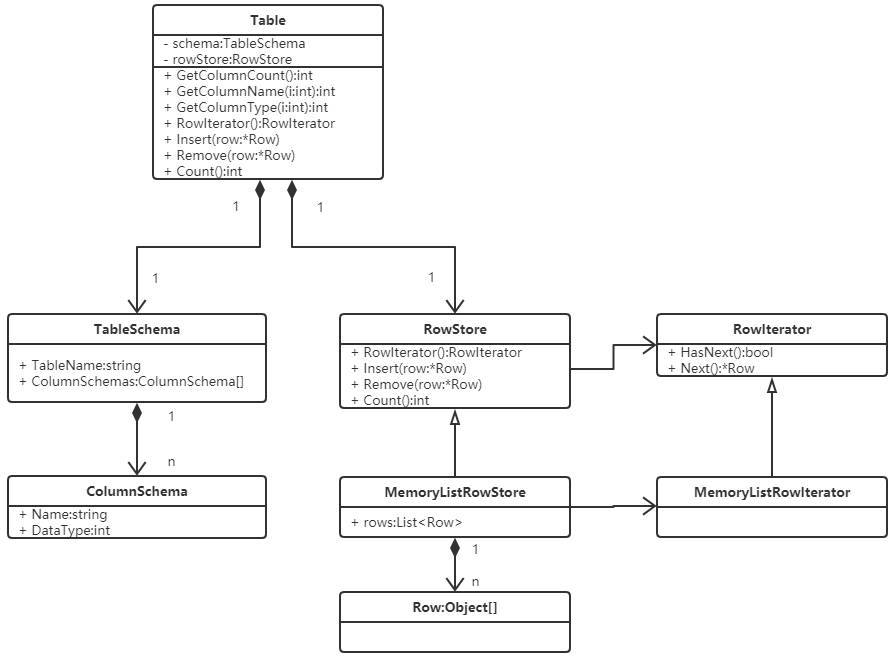


图 1 数据模型类图

数据模型的具体结构如图 1所示，由于这个数据模型整体比较简单，大部分函数与功能的含义可以从其名字推知，我们接下来只介绍一些注意要点。

* Table和RowStore类只提供以迭代器方式实现的遍历接口，结果以插入的顺序返回，你可以扩展更多的读接口以方便实现进阶功能。
* 注意参数的类型，例如Insert和Remove接口要求的是传引用（为了减少不必要的对象拷贝），但是在内部RowStore会对插入的Row对象进行拷贝，因此插入后再对参数进行修改不会影响Table中的对应内容。
* 因为没有主键约束，相同的行可以重复插入，就像你可以在Excel中插入若干完全相同的行一样。
* Remove()只会删除第一个与参数相同的行。
* 虽然有DataType字段，但目前并不会对插入内容进行类型检查。
* 提供的数据结构目前不支持并发修改，我们暂时也不对此进行要求。

### 集群架构

本框架提供一个相对中心化的集群架构，其结构如图 2所示。集群中包含若干个数据库节点，一个协调者以及一个客户端，它们之间通过网络相连。本框架中使用的网络并非真实网络，而是一个基于队列和选择器实现的网络模拟器，这允许我们在一台机器上运行一个集群，并且可以向这个集群注入各类网络故障。通常数据库节点对客户端是不可见的，客户端只能向协调者节点发送请求，但在本实验中为了调试目的，我们没有阻止客户端向任意数据库节点发送请求。你需要完善协调者的对应逻辑，让其可以正确地将各类请求分散、复制到各个节点上，并向客户端返回一个最终的结果。尽管我们使用了一个单独的协调者节点，但这并不意味着你必须采用中心化的算法和架构，你可以将协调者单纯地作为一个反向代理，它只负责将请求转发给任意节点，数据库节点通过去中心化的算法完成请求。

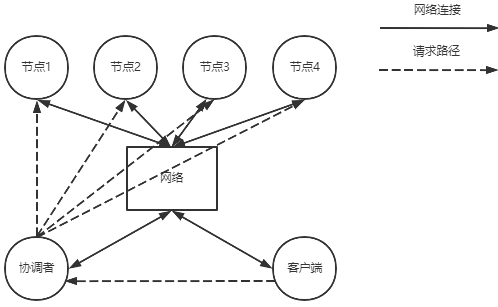


图 2 集群架构

## 使用示例

接下来我们根据代码中的HelloWorld示例，说明如何在已有框架的基础上扩展RPC。

首先我们选择或者新建一个类，在其中增加我们要实现的方法。在给出例子中，我们选择Node类（models/node.go）来添加要支持的方法SayHello，如图 3所示，添加的函数包含两个参数args和reply，它将args和节点自身的名字放入到一个字符串中并通过reply返回给客户端。注意在本框架中，所有支持RPC的函数都应该包含且只包含两个参数，并且第一个参数是实际用到的参数，第二个参数是参数形式的返回值。

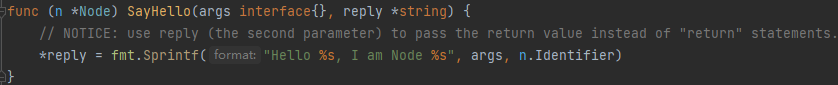


图 3 在Node类中增加SayHello函数

接下来在main/cluster.go中，我们创建一个Node对象，并将其转化为一个Service对象。一个Service对象可以认为是一个方法的列表，只有其中的方法才可以通过RPC被调用。这个转换的过程使用了Go的反射机制来获取一个对象的方法。

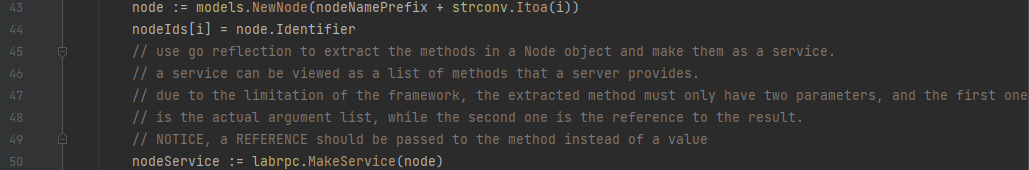


图 4 将Node对象转换为Service对象

然后我们创建一个Server对象，并将上面创建的Service对象注册到Network对象中（Network对象应该一开始就创建好）。注册Server时，第一个参数是其网络地址，我们在之后通过它来选择调用哪一个Server的方法。

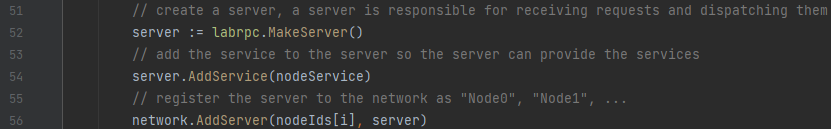
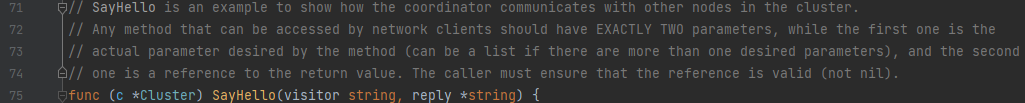


图 5 创建Server并注册Service和Server

上面添加的方法是为了协调者和其他节点准备的，通过类似的过程，我们为Cluster对象（协调者）添加由客户端调用的SayHello方法。该方法调用集群中的每一个节点的SayHello方法，并将结果打出来，最后向客户端返回一句来自协调者的问候。



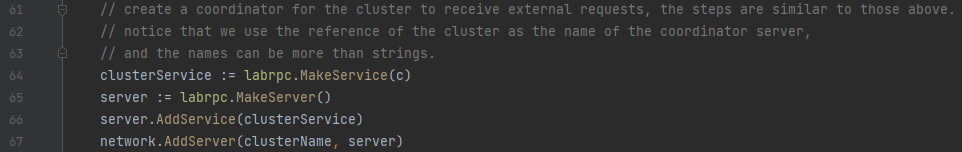
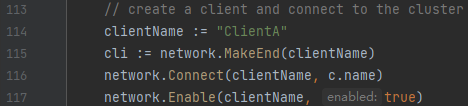


图 6 创建协调者对应的server

最后，我们创建一个客户端连接，将其连接到协调者对应的Server并激活，再发送SayHello的请求。因为我们将方法添加到了Cluster类中，并且函数名为SayHello，所以Call的第一个参数为“Cluster.SayHello”。运行Cluster.go，最终输出如所示，每个节点以及协调者都进行了各自的问候。



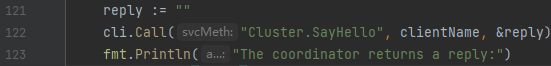


图 7 建立客户端连接并发送请求

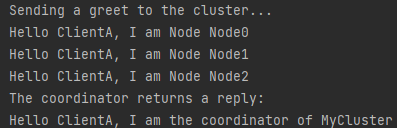
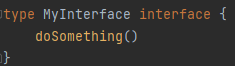


图 8 运行cluster.go的结果

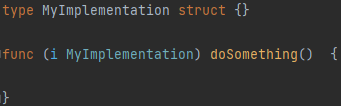
## 注意事项

### 在RPC中使用接口类型

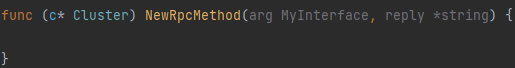
如果你需要添加一个新的RPC接口，并且其参数是一个接口类型，那么你需要使用引用包labgob，并使用labgob.Register方法，将一个该接口实现类型的对象注册到gob模块中，使得gob可以获取该类型的序列化和反序列化方式。例如定义了下面的接口：



以及如下的实现：



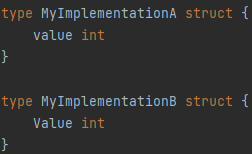
如果在RPC接口中使用了接口类型：



在调用上述接口前，需要先注册接口的实现（只需要注册一次，无需每次调用前都注册）：



注意实现类型中应该只包含被输出的字段，即首字母大写的字段，例如



MyImplementationA不可以被使用为RPC参数而MyImplementationB可以，因为MyImplementationA包含首字母没有大写的字段。

### GoLand的使用

如果你使用GoLand作为IDE，由于项目中在引用时使用了相对路径，你需要取消勾选模块集成。并且在测试时，需要以包（文件夹）为单位进行测试，不能仅以文件为单位进行测试。

