分布式系统通信

前面提到分布式系统是由很多服务器节点共同组成的，这些服务器有些可能是在同城同数据中心，有些可能是同城不同数据中心，还有些可能在不同城市的不同数据中心。 而要使这些服务器之间相互协作，就不可避免得要使用网络进行通信。而讲到网络通信，又不可避免要提到 TCP/IP 网络协议。

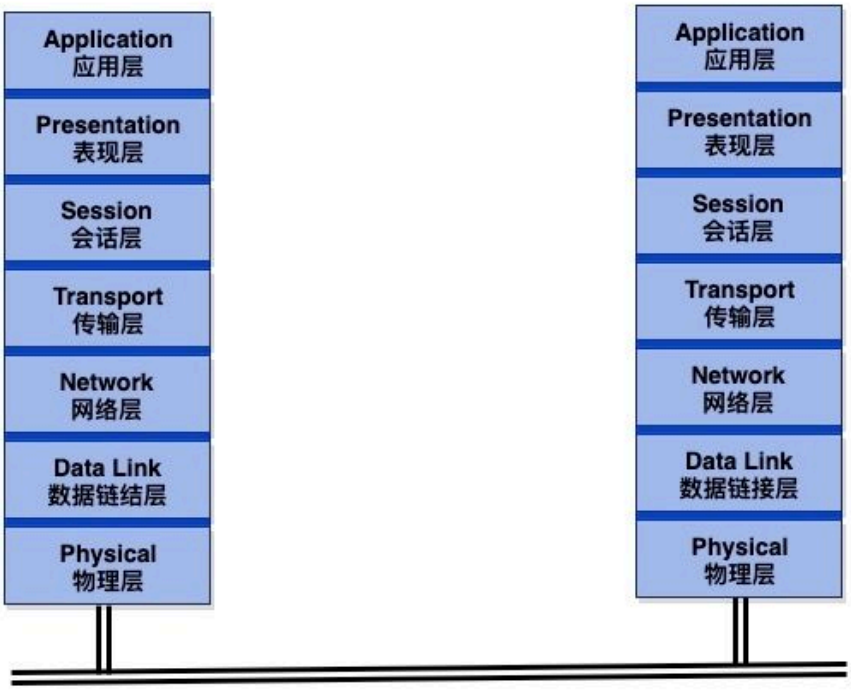
协议层

分布式系统是复杂的。在分布式系统中，往往有多台计算机相关联，他们之间必须以某种方式进行连接。 程序分别部署在不同服务器上，并且在这些不同服务器上的服务要 相互协作来完成分布式任务。

处理复杂问题最简单的方式就是将一个大而复杂的系统切割为多个小而简单的系统。其中每一部分都有自己的结构， 但是必须定义接口让这些独立的应用之间可以相互交流。 在分布式系统中，将此部分成为协议层，并且有清晰的定义。 他们遵循同一个技术栈，每一层都会同时跟上下层进行通信，层间的通信被协议所定义。依照上述逻辑，我们来 了解下 ISO OSI Protocol，也就是大名鼎鼎的七层网络协议。

ISO OSI Protocol

尽管OSI（Open Systems Interconnect）协议并没有完整实现，但 OSI 协议已经是事实上的通信标准协议。

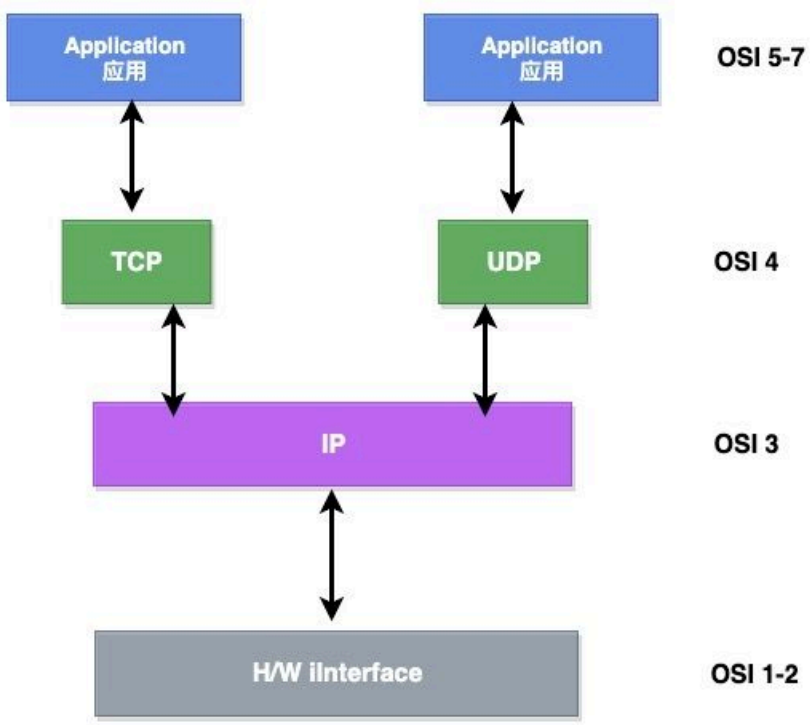


OSI 层

每一层的功能分别如下：

- 网络层提供交互和路由
- 传输层提供数据的交换
- 会话层建立连接，管理和终止应用之间的连接
- 表现层提供不同数据的不同表现形式
- 应用层支持应用程序和终端用户

TCP/IP 协议



从上图我们可以清晰的看到，TCP/UDP 都是属于传输层的协议，也就是四层协议。 在本书中，由于 ETCD 都是采用的 TCP 的传输协议， 所以在这里我们详细探讨下 Go 语言实现的 TCP 协议，以及如何利用 net 包进行 TCP 编程，这对后面理解 etcd 中的网络模型非常重要。

Go 语言网络编程之 TCP

Server

```
1  package main
2
3  import (
4      "fmt"
5      "net"
6  )
7  var c chan string
8
9  func main() {
10     c = make(chan string, 1)
11     server()
```

```
}
func server() {
    ln, err := net.Listen("tcp", ":8888")
    if err != nil {
        fmt.Println(err)
    }
    for {
        conn, err := ln.Accept()
        if err != nil {
            fmt.Println(err)
        }
        go handleConnection(conn)
    }
}
func handleConnection(conn net.Conn) {
    var buf [512]byte
    for {
        n, err := conn.Read(buf[0:])
        if err != nil {
            fmt.Println(err)
        }
        value := (string(buf[0:n]))
        c <- value
        _, err2 := conn.Write(buf[0:n])
        if err2 != nil {
            fmt.Println(err2)
        }
        go consumer()
        // select {
        // case <-c:
        //     fmt.Println("--->", value)
        // }
    }
}

func consumer() {
    fmt.Println("--->", <-c)
}
```

以上程序是一个简单的 Go 语言 TCP Server, 服务器端通过 net.Listen("tcp", ":8888") 监听 8888 端口。 然后有个 for 循环挂起，用来跟客户端进行数据交换。 handlerConnection 实现了数据处理的逻辑。 在数据处理逻辑中，服务器会将客户端发送过来的数据放到通道c中，然后调用协程进行异步消费。

Client

```
package main

import (
    "net"
    "fmt"
    "bufio"
    "os"
)

func main(){
    client()
}

func client(){
```

```
    conn, err := net.Dial("tcp", "localhost:8888")
    if err != nil{
        fmt.Println(err)
        return
    }
    in := bufio.NewReader(os.Stdin)

    for {
        inputV, _ := in.ReadString('\n')
        _, err := conn.Write([]byte(inputV))
        if err != nil{
            fmt.Println(err)
        }
    }

}
```

客户端就更简单了，客户端首先通过 net.Dail("tcp", "localhost:8888") 与服务器建立连接，然后有个有个 for 循环一直接受客户端的输入，并将客户端输入传送到服务器。 这里最后客户端并没有处理服务器最后传过来的数据。 当然也可以用 conn.Read() 方法来读取服务器传输过来的数据。

RPC&gRPC

// TODO

HTTP

// TODO