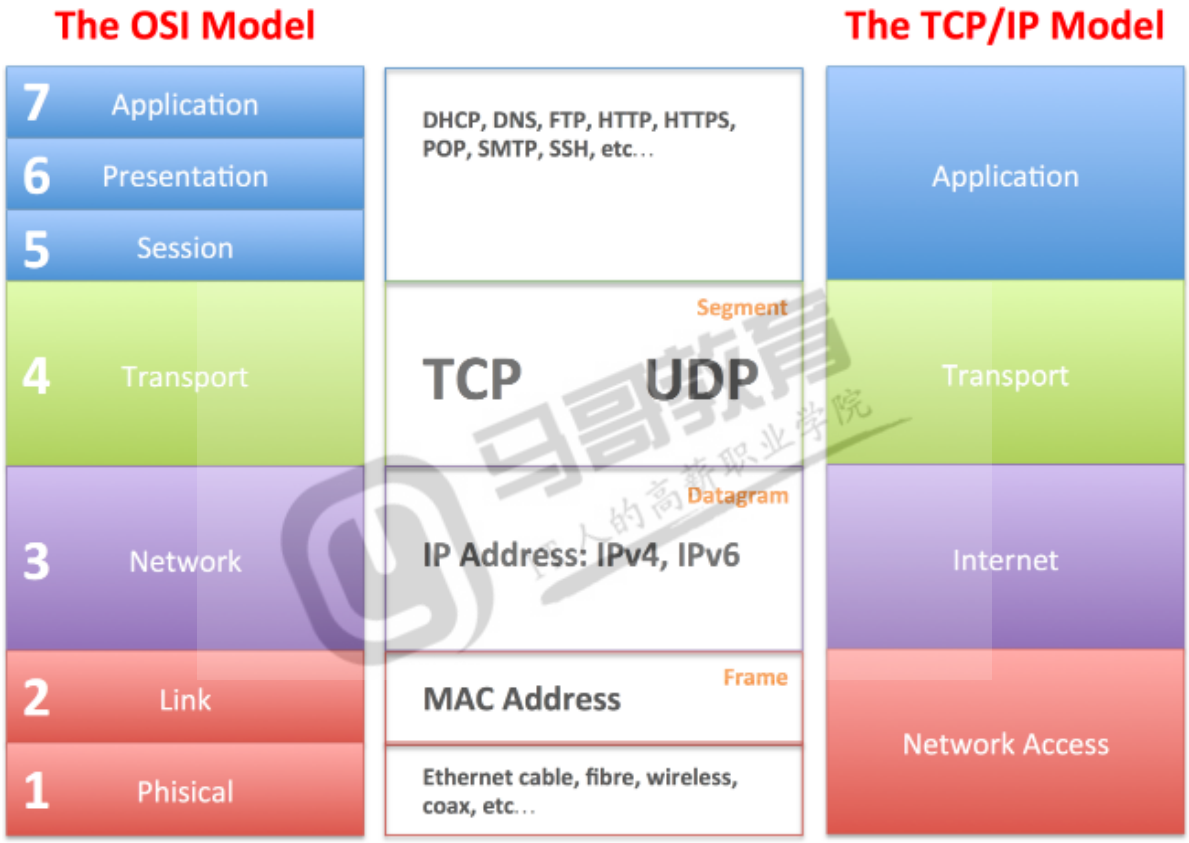
TCP/IP协议栈

TCP/IP（Transmission Control Protocol/Internet Protocol），传输控制协议/因特网互联协议。

它是一个包含很多工作在不同层的协议的协议族，其中最著名的2个协议分别是TCP和IP协议。

它最早起源于美国国防部（缩写为DoD）的ARPA网项目，1982年应用于美国所有军事网络。IBM、AT&T、DEC从1984年起就开始使用TCP/IP协议。TCP/IP更加广泛的传播是在1989年，加州大学伯克利分校在BSD中加入了该协议。微软是在Win95中增加。

TCP/IP协议，共定义了四层：网络访问层、Internet层、传输层、应用层。



TCP/IP协议是事实标准。目前局域网和广域网基本上也都用该协议。

传输层协议

	TCP	UDP
连接类型	面向连接	无连接
可靠性	可靠	不可靠
有序	数据包有序号	没有包序
使用场景	大多数场合，数据不能出任何问题	视频、音频

连接

- TCP需要通讯双方预先建立连接，需要三次握手、四次断开
- UDP不需要预先建立连接

可靠性

- TCP需要确定每一个包是否收到，丢包重发，效率低一些
- UDP尽最大努力交付数据，不需要确认数据包，丢包无法知道，也不重复，效率高一些

有序

- TCP包有序号，可以进行顺序控制。第一个包序号随机生成，之后的序号都和它有关
- UDP包无序，无法纠正，只能在应用层进行验证

数据

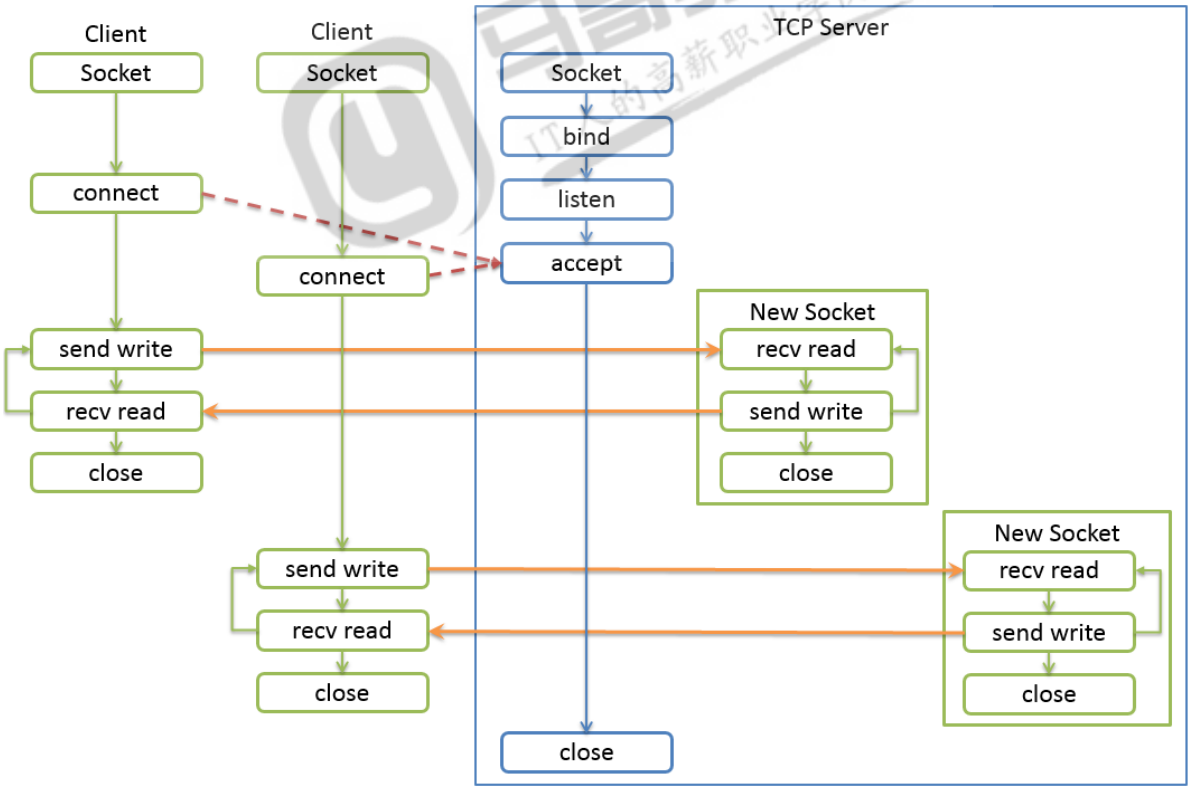
- TCP协议是流协议，也就是一大段数据看做字节流，一段段持续发送这些字节
- UDP协议是数据报协议，每一份数据封在一个单独的数据报中，一份一份发送数据

TCP编程

服务器端编程

在9-4课件中已经讲过了。

客户端编程



1. 创建socket对象
2. 随机选择端口即可以向服务器端发起连接
3. 连接成功后，就可以收发操作
4. 关闭

上面是socket编程的通用套路，Go语言也基本如此。

```go
package main

import (
    "fmt"
    "log"
    "net"
)

func catchErr(err error) {
    if err != nil {
        log.Fatalln(err)
    }
}

func main() {
    raddr, err := net.ResolveTCPAddr("tcp", "127.0.0.1:9999")
    catchErr(err)
    // 1 创建socket 2 发起对服务器端的连接
    conn, err := net.DialTCP("tcp", nil, raddr)
    catchErr(err)
    fmt.Println(conn.LocalAddr()) // 看看客户端的端口
    // 4 关闭
    defer conn.Close()

    // 3 收发数据，客户端一般主动发数据
    buffer := make([]byte, 1024)
    var input string
    for i := 0; i < 3; i++ {
        fmt.Scanln(&input)
        _, err = conn.Write([]byte(input))
        catchErr(err)
        n, err := conn.Read(buffer)
        catchErr(err)
        fmt.Println(buffer[:n])
    }
}
```

Scan和Read函数都会阻塞，把它们放到协程中。

```go
package main

import (
    "fmt"
    "log"
    "net"
    "runtime"
    "time"
)

func catchErr(err error) {
    if err != nil {
        log.Fatalln(err)
    }
}

```

```go
func main() {
    raddr, err := net.ResolveTCPAddr("tcp", "127.0.0.1:9999")
    catchErr(err)
    // 1 创建socket 2 发起对服务器端的连接
    conn, err := net.DialTCP("tcp", nil, raddr)
    catchErr(err)
    fmt.Println(conn.LocalAddr()) // 看看客户端的端口
    // 4 关闭
    defer conn.Close()

    exit := make(chan struct{})

    // 3 收发数据，客户端一般主动发数据
    go func() {
        buffer := make([]byte, 1024)
        for {
            conn.SetReadDeadline(time.Now().Add(time.Second)) // 每次等1秒就超时
            n, err := conn.Read(buffer)
            if err != nil {
                if _, ok := err.(*net.OpError); !ok {
                    exit <- struct{}{} // 退出，如果不是超时错误就退出
                    return             // 协程结束
                }
                continue
            }
            fmt.Println(buffer[:n])
        }
    }()

    go func() {
        var input string
        for {
            fmt.Scanln(&input)
            if input == "exit" {
                exit <- struct{}{}
                return
            }
            _, err = conn.Write([]byte(input))
            if err != nil {
                log.Println(err)
                continue
            }
        }
    }()

    t := time.NewTicker(3 * time.Second)
    for {
        select {
        case <-exit:
            goto EXIT
        case <-t.C:
            fmt.Println(runtime.NumGoroutine(), "@@@")
        }
    }
```

```
71
72  EXIT:
73      fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~~~")
74  }
```

上例代码依然简陋，目的在于掌握socket编程和协程怎么结合使用。