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。

The TCP/IP Model The OSI Model **Application** DHCP, DNS, FTP, HTTP, HTTPS, POP, SMTP, SSH, etc... 6 Presentation **Application** Segment **TCP** IP Address: IPv4, IPv6 3 Frame **MAC Address Network Access** Ethernet cable, fibre, wireless, 1 Phisical coax, etc...

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

| | ТСР | 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 |

- 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

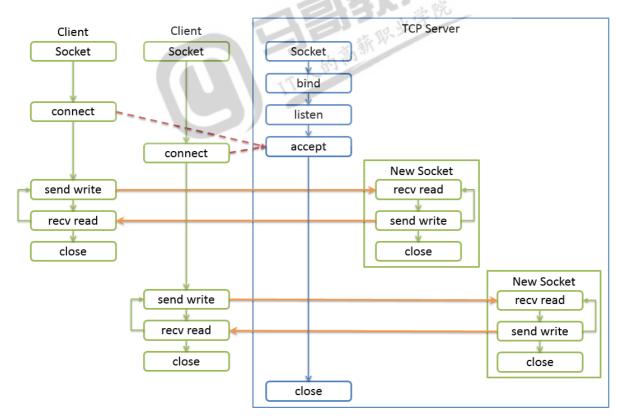
- TCP协议是流协议,也就是一大段数据看做字节流,一段段持续发送这些字节
- UDP协议是数据报协议,每一份数据封在一个单独的数据报中,一份一份发送数据

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。

```
1
   package main
 2
 3
    import (
        "fmt"
4
        "log"
5
6
        "net"
 7
    )
8
9
   func catchErr(err error) {
10
       if err != nil {
           log.Fatalln(err)
11
12
        }
13
   }
14
15
   func main() {
16
        raddr, err := net.ResolveTCPAddr("tcp", "127.0.0.1:9999")
        catchErr(err)
17
18
        // 1 创建socket 2 发起对服务器端的连接
19
        conn, err := net.DialTCP("tcp", nil, raddr)
20
        catchErr(err)
        fmt.Println(conn.LocalAddr()) // 看看客户端的端口
21
22
        // 4 关闭
23
        defer conn.Close()
24
        // 3 收发数据,客户端一般主动发数据
25
26
        buffer := make([]byte, 1024)
27
        var input string
        for i := 0; i < 3; i++ {
28
29
           fmt.Scanln(&input)
30
            _, err = conn.Write([]byte(input))
           catchErr(err)
31
           n, err := conn.Read(buffer)
32
           catchErr(err)
33
34
           fmt.Println(buffer[:n])
35
        }
36 }
```

Scan和Read函数都会阻塞,把它们放到协程中。

```
package main
 1
 2
 3
    import (
        "fmt"
 4
 5
        "log"
        "net"
 6
 7
        "runtime"
 8
        "time"
 9
    )
10
11
    func catchErr(err error) {
        if err != nil {
12
            log.Fatalln(err)
13
14
        }
    }
15
16
```

```
17
    func main() {
18
        raddr, err := net.ResolveTCPAddr("tcp", "127.0.0.1:9999")
        catchErr(err)
19
20
        // 1 创建socket 2 发起对服务器端的连接
21
        conn, err := net.DialTCP("tcp", nil, raddr)
22
        catchErr(err)
        fmt.Println(conn.LocalAddr()) // 看看客户端的端口
23
        // 4 关闭
24
        defer conn.Close()
25
26
27
        exit := make(chan struct{})
28
29
        // 3 收发数据,客户端一般主动发数据
30
        go func() {
            buffer := make([]byte, 1024)
31
32
            for {
                conn.SetReadDeadline(time.Now().Add(time.Second)) // 每次等1秒就超
33
    时
34
                n, err := conn.Read(buffer)
35
                if err != nil {
                    if _, ok := err.(*net.OpError); !ok {
36
37
                         exit <- struct{}{} // 退出,如果不是超时错误就退出
38
                         return
                                            // 协程结束
                    }
39
                    continue
40
41
                }
                fmt.Println(buffer[:n])
42
            }
43
        }()
44
45
46
        go func() {
47
            var input string
            for {
48
49
                fmt.Scanln(&input)
                if input == "exit" {
50
51
                    exit <- struct{}{}</pre>
52
                     return
53
                }
54
                _, err = conn.Write([]byte(input))
                if err != nil {
55
                    log.Println(err)
56
57
                     continue
58
                }
59
            }
        }()
60
61
        t := time.NewTicker(3 * time.Second)
62
63
        for {
64
            select {
65
            case <-exit:</pre>
66
                goto EXIT
            case <-t.C:
67
68
                fmt.Println(runtime.NumGoroutine(), "@@@")
69
            }
70
        }
```

```
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。

