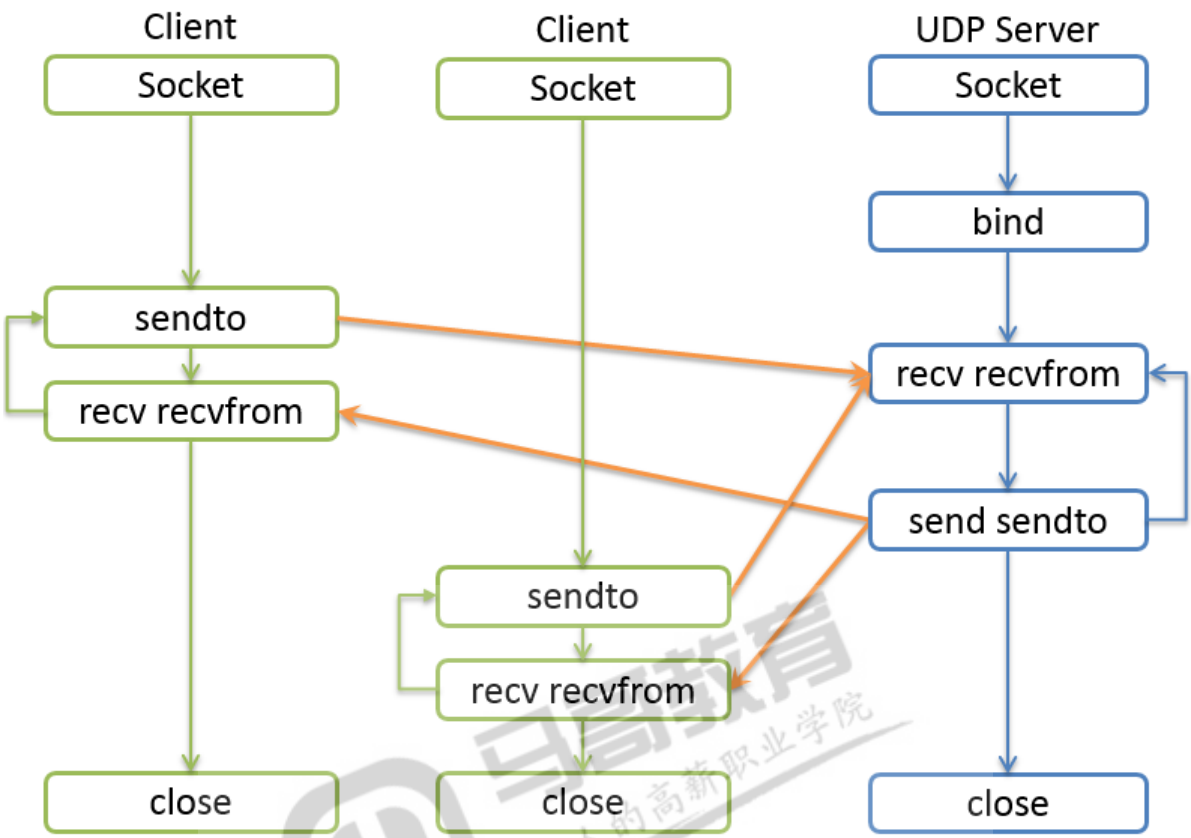
UDP编程

服务器端编程



- 创建socket对象。socket.SOCK_DGRAM
- 绑定IP和Port，bind()方法
- 传输数据
  - 接收数据，socket.recvfrom(bufsize[, flags])，获得一个二元组(string, address)
  - 发送数据，socket.sendto(string, address) 发给某地址某信息
- 释放资源

```go
package main

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

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

func main() {
    laddr, err := net.ResolveUDPAddr("udp", "127.0.0.1:9999")
```

```
    catchErr(err)
    fmt.Println(laddr)
    // 1 socket 2 bind
    server, err := net.ListenUDP("udp", laddr)
    catchErr(err)
    defer server.Close()
    // 不需要accept

    // 3 收发
    buffer := make([]byte, 1024)
    n, raddr, err := server.ReadFromUDP(buffer)
    fmt.Println(err)
    msg := fmt.Sprintf("Client=%v, data=%s", raddr, string(buffer[:n]))
    fmt.Println(msg)
    server.WriteToUDP([]byte(msg), raddr)
}
```

使用协程改造一下。

```
package main

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

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

func main() {
    laddr, err := net.ResolveUDPAddr("udp", "127.0.0.1:9999")
    catchErr(err)
    fmt.Println(laddr)
    // 1 socket 2 bind
    server, err := net.ListenUDP("udp", laddr)
    catchErr(err)
    defer server.Close()
    // 不需要accept

    // 3 收发
    buffer := make([]byte, 1024)
    exit := make(chan struct{})

    go func() {
        for {
            server.SetReadDeadline(time.Now().Add(time.Second))
            n, raddr, err := server.ReadFromUDP(buffer)
            fmt.Printf("%T, %[1]v\n", err)
            if err != nil {
```

```
                if _, ok := err.(*net.OpError); !ok {
                    exit <- struct{}{}
                    return
                }
                continue
            }
            msg := fmt.Sprintf("Client=%v, data=%s", raddr,
string(buffer[:n]))
            fmt.Println(msg)
            server.WriteToUDP([]byte(msg), raddr)
        }
    }()

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

EXIT:
    fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
}
```

客户端编程

- 创建socket对象。socket.SOCK_DGRAM
- 传输数据
  - 接收数据，socket.recvfrom(bufsize[, flags])，获得一个二元组(string, address)
  - 发送数据，socket.sendto(string, address) 发给某地址某信息
- 释放资源

```
package main

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

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

func main() {
```

```
    raddr, err := net.ResolveUDPAddr("udp", "127.0.0.1:9999")
    catchErr(err)

    // 1 socket
    conn, err := net.DialUDP("udp", nil, raddr)
    catchErr(err)
    defer conn.Close()
    // 不需要accept

    // 2 收发
    exit := make(chan struct{})

    go func() {
        buffer := make([]byte, 1024)
        for {
            conn.SetReadDeadline(time.Now().Add(time.Second))
            n, _, err := conn.ReadFromUDP(buffer)
            if err != nil {
                if _, ok := err.(*net.OpError); !ok {
                    exit <- struct{}{}
                    return
                }
                continue
            }
            fmt.Println(buffer[:n])
        }
    }()

    go func() {
        var input string
        for {
            fmt.Scanln(&input)
            if input == "exit" {
                exit <- struct{}{}
                return
            }
            i, err2 := conn.Write([]byte(input)) // 要用Write，因为DialUDP已经知道了对端
            fmt.Println(i, err2)
        }

    }()

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

EXIT:
    fmt.Println("~~~~~~~~~~~~~~~~~~~~~~~~~~")
```

```
72 }
```