# Go入门学习笔记

PPT

# 目录



## 简介

### Go语言特点

- 高性能、高并发
- 语法简单、学习曲线平缓
- 丰富的标准库
- 完善的工具链
- 静态链接
- 快速编译
- 跨平台
- 垃圾回收

## 入门

第一行package main代表这个文件属于main包的一部分，main包也就是程序的入口包。
第三行导入了标准库里面的FMT包。这个包主要是用来往屏幕输入输出字符串、格式化字符串
import下面是main函数，main函数的话里面调用了fmt.Println输出helloword
在FMT包里面还有很多的函数来做不同的输入输出格式化工作

```
package main

import (
        "fmt"
)

func main() {
        fmt.Println("hello world")
}
```

go语言是一门强类型语言，每一个变量都有它自己的变量类型。
常见的变量类型包括 字符串 整数 浮点型、布尔型等。
go 语言的字符串是内置类型，可以直接通过加号拼接，也能够直接用等于号去比较两个字符串。
在go语言里面，大部分运算符的使用和优先级都和C或者C++类似，这里就不再概述。
在go语言里面变量的声明有两种方式,一种是通过

```
var name string = ""
```

这种方式来声明变量，声明变量的时候，一般会自动去推导变量的类型，如果有需要，你也可以显式写出变量类型。另一种声明变量的方式是:

```
使用变量名 := 值
```

常量的话就是把 var 改成 const，值在一提的是go语言里面的常量，它没有确定的类型，会根据使用的上下文来自动确定类型。

go语言里面的ifese写法和C或者 C++类似，不同点1是许后面没有括号。不同点2是Golang里面的if，它必须后面接大括号，就是你不能像C或者C++一样，直接把if里面的语句写在同一行而省略大括号。

go语言只有for循环，没有while循环

go语言里面的 switch 分支结构。看起来也 C 或者 C++ 比较类似。同样的在 switch 后面的那个变量名不需要小括号包裹。
这里有个很大的一点不同的是，在c++里面，switch case如果不显式加brek的话会然后会继续往下跑完所有的case，而在go语言里面的话是不需要加break的，执行完当前case直接退出。
相比C或者C++，go语言里面的switch功能更强大。可以使用任意的变量类型，甚至可以用来取代任意的if else语句。你可以在switch后面不加任何的变量，然后在case里面写条件分支，这样代码相比你用多个if else代码逻辑会更为清晰。

定义数组
固定长度的数组在传参的时候是值拷贝，严格匹配数组类型

```
var a [5]int
var b [3][4]int
```

输出长度，适用于数组、切片、map等

```
len(a)
```

定义切片
slice在传参的时候是引用传递，在调用函数时改变slice内元素值，会改变原始的slice

```
//定义长度为3的切片
s := make([]string, 3)

//只声明，但没有给slice分配空间
var s []int
//需要使用make开辟空间
s := make([]int, 3)

//声明slice并分配空间，初始值是0
var s []int = make([]int, 3)
```

在切片后面追加元素

```
s = append(s, "a")
```

slice的容量，使用 `cap()` 获取，slice的 `len` 一定是小于等于容量的，slice中存在一个头指针和尾指针，当向slice中添加元素，尾指针位置向后移，当尾指针再向后移就超过cap大小以后，slice自动将cap大小乘2
如果不特别声明cap大小，定义时默认是 `len=cap`

```
var s []int = make([]int, 3, 5)
//输出slice容量大小
cap(s) //5
```

切片取元素
可以通过设置上下限来取slice当中的元素，取得是[下限:上限)，虽然取到了元素但两个slice指向的还是

同一段空间，利用 copy 函数可以将底层数组的slice一起进行拷贝

```
num := []int{0, 1, 2, 3, 4, 5}
//取元素，num和num1指向相同的地址空间，操作同一个slice
num1 := num[0:2]

//深拷贝
num2 := make([]int, 6)
//将num中的值拷贝到num2中，两个slice指向不同的地址空间
copy(num2, num)
```

定义map

map并不是C++当中的map，而是一种哈希映射的表，因此内部key的顺序并不一定是按照顺序排列的，使用map前必须得make初始化，否则不能直接使用

```
//定义一个空map，无法直接使用，得开辟一段空间
var mp map[string]int
//使用map前必须给map分配空间
mp = make(map[string]int)

m := make(map[string]int)

m2 := map[string]int{
        "one": 1,
        "two": 2
}

var m3 = map[string]int{
        "one": 1,
        "two": 2
}
```

删除map内元素

```
delete(m2, "one")
```

获取map内元素，如果存在元素ok为true，否则为false，此时v=0

```
v, ok := m["unknow"]
```

利用range遍历map，map是没有固定顺序遍历的，和c++有区别，返回两个值，第一个是索引，第二个是值

```
nums := []int{1, 2, 3}
for index, value := range nums {

}
for _, value := range nums {

}
m := map[string]string{"a": "A", "b": "B"}
for key, value := range m {

}
```

map传递的时候和slice一样，都是引用传递，传递一个指针，传递后的变量仍然可以修改原map的值

```
func init(m map[string]string) {
        //m和mp指向同一段地址空间
        m["one"] = "two"
}

func main() {
        mp := make(map[string]string)
        init(mp)
}
```

函数表示，在实际业务逻辑代码里几乎所有的函数都是返回两个值，第一个是真正的结果，第二个是一个错误信息

```
func 函数名(传入值的名 传入值的类型, ...) 返回值类型或空 {

}
```

go语言的指针并没有c++指针灵活，支持的操作有限

定义、修改结构体，结构体也可以传入指针，可以避免大结构体的拷贝开销，传入指针修改原始值，不传入指针只修改拷贝值，原始值不会改变
如果类的首字母大写，表示该类可以被外部的包调用，如果是小写，只能在当前包内使用，函数也是类似

```
type user struct {
        name     string
        password string
}

a := user{name: "wang", password: "1024"}
b := user{"wang", "1024"}
c := user{name: "wang"}
c.password = "1024"
var d user
d.name = "wang"
d.password = "1024"

func checkPassword(u user, password string) bool {
        return u.password == password
}

func checkPassword2(u *user, password string) bool {
        return u.password == password
}

fmt.Println(checkPassword(a, "haha"))   // false
fmt.Println(checkPassword2(&a, "haha")) // false
```

定义结构体函数

```
type user struct {
        name     string
        password string
}
//以下函数的u只是一个值拷贝，修改u的值不会改变原始对象的值
func (u user) checkPassword(password string) bool {
        return u.password == password
}
//以下函数的u是一个指针，修改u的值改变原始对象的值
func (u *user) resetPassword(password string) {
        u.password = password
}
```

go中类的继承

```go
type Human struct {
        name string
        sex string
}

func (this *Human) Eat() {

}

func (this *Human) Walk() {

}

type Superman struct {
        Human //Superman继承Human类的方法
        level int
}

//重定义父类的方法Eat
func (this *Superman) Eat() {

}

//子类新方法
func (this *Superman) Fly() {

}

func main() {
        h := Human{"zhang3", "female"}

        //定义一个子类对象
        s := Superman{Human{"li4", "male"}, 10}

        //子类可以直接使用父类的属性
        var t Superman
        t.name = "wang5"
        t.sex = "male"
        t.level = 11

        //父类方法
        s.Walk()
        //子类方法
```

```
        s.Fly()
        s.Eat()
}
```

go语言通过interface接口实现多态
interface本质是一个指针，类必须完全实现接口当中的属性，否则接口指针无法指向当前的类

```
type animal interface {
        Sleep()
        GetColor() string
        GetType() string
}

type Cat struct {
        color string
}

func (this *Cat) Sleep() {

}

func (this *Cat) GetColor() string {
        return this.color
}

func (this *Cat) GetType() string {
        return "Cat"
}

type Dog struct {
        color string
}

func (this *Dog) Sleep() {

}

func (this *Dog) GetColor() string {
        return this.color
}

func (this *Dod) GetType() string {
        return "Dog"
}

func show(ani animal) {
        fmt.Println("color = ", animal.GetColor())
}

func main() {
```

```
        var ani animal
        ani = &Cat{"red"}

        cat := Cat{"yellow"}
        dog := Dog{"green"}
        show(&cat)
        show(&dog)
}
```

`interface{}` 是万能数据类型，可以用 `interface{}` 类型引用任意的数据类型，在go当中的诸如 `int` 、 `string` 等内部都实现了 `interface{}`

```
func myFunc(x interface{}) {
        //interface{}提供了类型断言机制
        //判断x是不是string类型，ok为true就是string类型，否则不是
        value, ok := x.(string)
}
```

错误处理在go语言里面符合语言习惯的做法就是使用一个单独的返回值来传递错误信息。
不同于Java使用的异常。go语言的处理方式，能够很清晰地知道哪个函数返回了错误，并且能用简单的if else来处理错误。
在函数里面，我们可以在那个函数的返回值类型里面，后面加一个 error， 就代表这个函数可能会返回错误。那么在函数实现的时候，return需要同时return两个值，如果出现错误的话，那么可以return nil和一个err，如果没有的话，那么返回原本的结果和nil

在go语言中，可以使用%v来打印任意类型变量，不需要区分数字和字符串，也可以使用%+v和%#v来打印更为详细的信息，使用%T来打印变量的类型

在go语言里面的JSON操作非常简单，对于一个已有的结构体，我们可以什么都不做，只要保证每个字段的第一个字母是大写，也就是是公开字段，那么这个结构体就能用JSON.marshal 去序列化，变成一个JSON的字符串。序列化之后的字符串也能够用 JSON.Unmarshal 去反序列化到一个空的变量里面。
这样默认序列化出来的字符串的话，它的风格是大写字母开头，而不是下划线。我们可以在后面用json tag等语法来去修改输出JSON结果里面的字段名。
结构体标签应用：

- json编解码
- orm映射关系

```
type userInfo struct {
        Name  string
        Age   int `json:"age"`
        Hobby []string
}

a := userInfo{Name: "wang", Age: 18, Hobby: []string{"Golang", "TypeScript"}}
buf, err := json.Marshal(a)
if err != nil {
        panic(err)
}
fmt.Println(buf)         // [123 34 78 97...]
fmt.Println(string(buf)) // {"Name":"wang","age":18,"Hobby":["Golang","TypeScript"]}

buf, err = json.MarshalIndent(a, "", "\t")
if err != nil {
    panic(err)
}
fmt.Println(string(buf))

var b userInfo
err = json.Unmarshal(buf, &b)
if err != nil {
    panic(err)
}
fmt.Printf("%#v\n", b) // main.userInfo{Name:"wang", Age:18, Hobby:[]string{"Golang", "TypeScri
```

go语言中最常用的时间函数就是time.Now()

在go语言中，关于字符串和数字类型之间的转换都在strconv包内，这个包是string convert缩写，如果输入不合法，其中的函数都会返回err

在go语言中，使用os.Args来得到程序执行时指定的命令行参数，使用os.Getenv("PATH")获取环境变量，os.Setenv("xxx", "xxxx")来设置环境变量

在go语言中，可以给import的包起别名

```
//给包起别名 下面就可以直接使用 别名.func 进行调用
import 别名 "fmt"
```

如果import的包暂时不调用里面的函数，但需要包内进行初始化操作，可以在包前面加 _

```
import _ "fmt"
```

如果要把包内的所有函数全部加载到当前代码内，在包名前面加 . ，那么包内所有的函数便都加载到当前代码内了，执行函数的时候不需要使用 包名+函数名 的方式，而是直接使用函数名调用，不推荐这样使用，因为可能会产生冲突

```
import . "fmt"
```

defer的顺序相当于一个队列，当代码一行行执行的时候，在当前函数内先遇到的defer最后执行，defer是在当前函数内的return之后进行执行，也就是说defer一定是在当前函数生命周期的最后才会执行，执行完生命周期结束

GMP调度器的设计策略

- 复用线程
  - work strealing机制：如果某个执行器的本地队列空闲，而其他执行器的本地队列还有协程在等待，那么它会偷取其他执行器的协程来执行
  - hand off机制：如果某个执行器在执行协程的时候突然协程阻塞了，那么会再创建或唤醒一个线程，该执行器带着本地队列到新线程上继续执行，而阻塞的协程则与以前的线程进行绑定，等待阻塞结束，物理CPU切换到其他线程进行执行。如果阻塞的协程执行完毕了，那么其绑定的线程就会睡眠或销毁，如果阻塞的协程阻塞结束还未完成，那么它会加入其他队列中
- 利用并行：GOMAXPROCS的个数=CPU核数/2，不会使用全部CPU，让剩余CPU分配给其他进程
- 抢占：goroutine使用CPU时，如果又来了一个goroutine，它们执行时是有并发特点的，没有优先级，那么老goroutine执行完最多10ms后，CPU就会被新goroutine进行抢占，不需要等待老goroutine主动释放
- 全局G队列：当执行器的本地队列都为空，而全局队列不为空时，work stealing机制就会偷取全局G队列中的协程来执行，但如果本地队列不为空，优先还是偷取本地队列的协程执行，因为偷取全局G队列会进行加锁和解锁，速度较慢

goroutine，其返回值需要用channel接收，不支持直接赋值

```
go func(传参) 返回值 {

        func() {
                //调用下列函数退出当前goroutine
                runtime.Goexit()
        }
}(传参)
```

channel的定义

```
//make(chan Type)
make(chan int) //等价于make(chan int, 0) 即无缓冲管道
make(chan int, 5) //有缓冲管道

c := make(chan int)
c <- num // 向管道中存数据

//从管道中提取数据
<- c //无接收
x := <- c //有接收
x, ok := <- c//接收数据并检测是否读成功
```

channel特点：当channel已满，再向channel内发送数据会阻塞。当channel为空，再从里拿数据会阻塞
如果channel不再接收数据，且channel不关闭，但仍然有数值在准备从channel内拿数据，就会造成死锁
如果channel关闭了，还向channel发数据会报panic错，关闭channel后，如果channel内还有数据，接收方是可以继续从channel内接收数据的，直到channel内没有数据
对于 `nil channel` 无论收发数据都会阻塞
使用 `close(channel名称)` 来关闭channel

channel和range配合
使用range来遍历channel内的数据

```
c := make(chan int)

for data := range c {

}
```

channel和select配合
单流程下一个go只能监控一个channel的状态，select可以完成监控多个channel的状态

```
func fibo(c, quit chan int) {
        x, y := 1, 1
        for {
                select {
                        case c <- x:
                                x = y
                                y = x + y
                        case <- quit:
                                fmt.Println("quit")
                                return
                }
        }
}

func main() {
        c := make(chan int)
        quit := make(chan int)

        go func() {
                for i := 0; i < 10; i++ {
                        fmt.Println(<- c)
                }
                quit <- 0
        }()
        fibo(c, quit)
}
```

GOPATH弊端

- 无版本控制概念
- 无法同步一致第三方版本号
- 无法指定当前项目引用的第三方版本号

Go Modules模式
go必须1.11版本及以上
命令

| 命令 | 作用 |
| --- | --- |
| `go mod init` | 生成go.mod文件 |
| `go mod download` | 下载go.mod文件中指明的所有依赖 |

| 命令 | 作用 |
| --- | --- |
| `go mod tidy` | 整理现有的依赖 |
| `go mod graph` | 查看现有的依赖结构 |
| `go mod edit` | 编辑go.mod文件 |
| `go mod vendor` | 导出项目所有的依赖到vendor目录 |
| `go mod verify` | 校验一个模块是否被篡改过 |
| `go mod why` | 查看为什么需要依赖某模块 |

Golang生态

- Web框架
  - Beego：初学者可看，文档丰富
  - gin：主流且轻量级框架
  - echo：轻量级框架
  - Iris：重量级框架
- 微服务架构
  - go kit：适合更加灵活、自定义化的微服务项目
  - Istio：适合大型的微服务项目
- 容器编排
  - Kubernetes：主流
  - swarm
- 服务发现
  - consul
- 存储引擎
  - k/v存储--etcd：类似于redis，比redis好
  - 分布式存储--tidb：分布式SQL数据库
- 静态建站
  - hugo
- 中间件
  - 消息队列
    - nsq
  - TCP长链接框架(轻量级服务器)
    - zinx
  - Leaf(游戏服务器)
  - RPC框架

        - gRPC
        - gRPC
    - redis集群
        - codis
- 爬虫框架
    - go query

# 实操

猜数字游戏

```go
package main

import (
	"bufio"
	"fmt"
	"math/rand"
	"os"
	"strconv"
	"strings"
	"time"
)

func main() {
	maxNum := 100
	rand.Seed(time.Now().UnixNano())
	secretNumber := rand.Intn(maxNum)
	// fmt.Println("The secret number is ", secretNumber)

	fmt.Println("Please input your guess")
	reader := bufio.NewReader(os.Stdin)
	for {
		input, err := reader.ReadString('\n')
		if err != nil {
			fmt.Println("An error occured while reading input. Please try again", er
			continue
		}
		input = strings.Trim(input, "\r\n")

		guess, err := strconv.Atoi(input)
		if err != nil {
			fmt.Println("Invalid input. Please enter an integer value")
			continue
		}
		fmt.Println("You guess is", guess)
		if guess > secretNumber {
			fmt.Println("Your guess is bigger than the secret number. Please try aga
		} else if guess < secretNumber {
			fmt.Println("Your guess is smaller than the secret number. Please try ag
		} else {
			fmt.Println("Correct, you Legend!")
			break
		}
```

```
        }
    }
```

在线翻译词典

```
package main

import (
        "bytes"
        "encoding/json"
        "fmt"
        "io/ioutil"
        "log"
        "net/http"
        "os"
)

type DictRequest struct {
        TransType string `json:"trans_type"`
        Source    string `json:"source"`
        UserID    string `json:"user_id"`
}

type DictResponse struct {
        Rc   int `json:"rc"`
        Wiki struct {
                KnownInLaguages int `json:"known_in_laguages"`
                Description     struct {
                        Source string      `json:"source"`
                        Target interface{} `json:"target"`
                } `json:"description"`
                ID   string `json:"id"`
                Item struct {
                        Source string `json:"source"`
                        Target string `json:"target"`
                } `json:"item"`
                ImageURL  string `json:"image_url"`
                IsSubject string `json:"is_subject"`
                Sitelink  string `json:"sitelink"`
        } `json:"wiki"`
        Dictionary struct {
                Prons struct {
                        EnUs string `json:"en-us"`
                        En   string `json:"en"`
                } `json:"prons"`
                Explanations []string      `json:"explanations"`
                Synonym      []string      `json:"synonym"`
                Antonym      []string      `json:"antonym"`
```

```
                WqxExample   [][]string    `json:"wqx_example"`
                Entry        string        `json:"entry"`
                Type         string        `json:"type"`
                Related      []interface{} `json:"related"`
                Source       string        `json:"source"`
        } `json:"dictionary"`
}

func query(word string) {
        client := &http.Client{}
        request := DictRequest{TransType: "en2zh", Source: word}
        buf, err := json.Marshal(request)
        if err != nil {
                log.Fatal(err)
        }
        var data = bytes.NewReader(buf)
        req, err := http.NewRequest("POST", "https://api.interpreter.caiyunai.com/v1/dict", data
        if err != nil {
                log.Fatal(err)
        }
        req.Header.Set("Connection", "keep-alive")
        req.Header.Set("DNT", "1")
        req.Header.Set("os-version", "")
        req.Header.Set("sec-ch-ua-mobile", "?0")
        req.Header.Set("User-Agent", "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebK
        req.Header.Set("app-name", "xy")
        req.Header.Set("Content-Type", "application/json;charset=UTF-8")
        req.Header.Set("Accept", "application/json, text/plain, */*")
        req.Header.Set("device-id", "")
        req.Header.Set("os-type", "web")
        req.Header.Set("X-Authorization", "token:qgemv4jr1y38jyq6vhvi")
        req.Header.Set("Origin", "https://fanyi.caiyunapp.com")
        req.Header.Set("Sec-Fetch-Site", "cross-site")
        req.Header.Set("Sec-Fetch-Mode", "cors")
        req.Header.Set("Sec-Fetch-Dest", "empty")
        req.Header.Set("Referer", "https://fanyi.caiyunapp.com/")
        req.Header.Set("Accept-Language", "zh-CN,zh;q=0.9")
        req.Header.Set("Cookie", "_ym_uid=16456948721020430059; _ym_d=1645694872")
        resp, err := client.Do(req)
        if err != nil {
                log.Fatal(err)
        }
        defer resp.Body.Close()
```

```
        bodyText, err := ioutil.ReadAll(resp.Body)
        if err != nil {
                log.Fatal(err)
        }
        if resp.StatusCode != 200 {
                log.Fatal("bad StatusCode:", resp.StatusCode, "body", string(bodyText))
        }
        var dictResponse DictResponse
        err = json.Unmarshal(bodyText, &dictResponse)
        if err != nil {
                log.Fatal(err)
        }
        fmt.Println(word, "UK:", dictResponse.Dictionary.Prons.En, "US:", dictResponse.Dictionar
        for _, item := range dictResponse.Dictionary.Explanations {
                fmt.Println(item)
        }
}

func main() {
        if len(os.Args) != 2 {
                fmt.Fprintf(os.Stderr, `usage: simpleDict WORD
example: simpleDict hello
                `)
                os.Exit(1)
        }
        word := os.Args[1]
        query(word)
}
```

socket5代理

```go
package main

import (
	"bufio"
	"context"
	"encoding/binary"
	"errors"
	"fmt"
	"io"
	"log"
	"net"
)

const socks5Ver = 0x05
const cmdBind = 0x01
const atypeIPV4 = 0x01
const atypeHOST = 0x03
const atypeIPV6 = 0x04

func main() {
	server, err := net.Listen("tcp", "127.0.0.1:1080")
	if err != nil {
		panic(err)
	}
	for {
		client, err := server.Accept()
		if err != nil {
			log.Printf("Accept failed %v", err)
			continue
		}
		go process(client)
	}
}

func process(conn net.Conn) {
	defer conn.Close()
	reader := bufio.NewReader(conn)
	err := auth(reader, conn)
	if err != nil {
		log.Printf("client %v auth failed:%v", conn.RemoteAddr(), err)
		return
	}
	err = connect(reader, conn)
```

```
        if err != nil {
                log.Printf("client %v auth failed:%v", conn.RemoteAddr(), err)
                return
        }
}

func auth(reader *bufio.Reader, conn net.Conn) (err error) {
        // +----+----------+----------+
        // |VER | NMETHODS | METHODS  |
        // +----+----------+----------+
        // | 1  |    1     | 1 to 255 |
        // +----+----------+----------+
        // VER: 协议版本，socks5为0x05
        // NMETHODS: 支持认证的方法数量
        // METHODS: 对应NMETHODS，NMETHODS的值为多少，METHODS就有多少个字节。RFC预定义了一些值的含义
        // X’00’ NO AUTHENTICATION REQUIRED
        // X’02’ USERNAME/PASSWORD

        ver, err := reader.ReadByte()
        if err != nil {
                return fmt.Errorf("read ver failed:%w", err)
        }
        if ver != socks5Ver {
                return fmt.Errorf("not supported ver:%v", ver)
        }
        methodSize, err := reader.ReadByte()
        if err != nil {
                return fmt.Errorf("read methodSize failed:%w", err)
        }
        method := make([]byte, methodSize)
        _, err = io.ReadFull(reader, method)
        if err != nil {
                return fmt.Errorf("read method failed:%w", err)
        }

        // +----+--------+
        // |VER | METHOD |
        // +----+--------+
        // | 1  |   1    |
        // +----+--------+
        _, err = conn.Write([]byte{socks5Ver, 0x00})
        if err != nil {
                return fmt.Errorf("write failed:%w", err)
```

```
        }
        return nil
}

func connect(reader *bufio.Reader, conn net.Conn) (err error) {
        // +----+-----+-------+------+----------+----------+
        // |VER | CMD |  RSV  | ATYP | DST.ADDR | DST.PORT |
        // +----+-----+-------+------+----------+----------+
        // | 1  |  1  | X'00' |  1   | Variable |    2     |
        // +----+-----+-------+------+----------+----------+
        // VER 版本号，socks5的值为0x05
        // CMD 0x01表示CONNECT请求
        // RSV 保留字段，值为0x00
        // ATYP 目标地址类型，DST.ADDR的数据对应这个字段的类型。
        //   0x01表示IPv4地址，DST.ADDR为4个字节
        //   0x03表示域名，DST.ADDR是一个可变长度的域名
        // DST.ADDR 一个可变长度的值
        // DST.PORT 目标端口，固定2个字节

        buf := make([]byte, 4)
        _, err = io.ReadFull(reader, buf)
        if err != nil {
                return fmt.Errorf("read header failed:%w", err)
        }
        ver, cmd, atyp := buf[0], buf[1], buf[3]
        if ver != socks5Ver {
                return fmt.Errorf("not supported ver:%v", ver)
        }
        if cmd != cmdBind {
                return fmt.Errorf("not supported cmd:%v", cmd)
        }
        addr := ""
        switch atyp {
        case atypeIPV4:
                _, err = io.ReadFull(reader, buf)
                if err != nil {
                        return fmt.Errorf("read atyp failed:%w", err)
                }
                addr = fmt.Sprintf("%d.%d.%d.%d", buf[0], buf[1], buf[2], buf[3])
        case atypeHOST:
                hostSize, err := reader.ReadByte()
                if err != nil {
                        return fmt.Errorf("read hostSize failed:%w", err)
```

```
        }
        host := make([]byte, hostSize)
        _, err = io.ReadFull(reader, host)
        if err != nil {
                return fmt.Errorf("read host failed:%w", err)
        }
        addr = string(host)
case atypeIPV6:
        return errors.New("IPv6: no supported yet")
default:
        return errors.New("invalid atyp")
}
_, err = io.ReadFull(reader, buf[:2])
if err != nil {
        return fmt.Errorf("read port failed:%w", err)
}
port := binary.BigEndian.Uint16(buf[:2])

dest, err := net.Dial("tcp", fmt.Sprintf("%v:%v", addr, port))
if err != nil {
        return fmt.Errorf("dial dst failed:%w", err)
}
defer dest.Close()
log.Println("dial", addr, port)

// +----+-----+-------+------+----------+----------+
// |VER | REP |  RSV  | ATYP | BND.ADDR | BND.PORT |
// +----+-----+-------+------+----------+----------+
// | 1  |  1  | X'00' |  1   | Variable |    2     |
// +----+-----+-------+------+----------+----------+
// VER socks版本，这里为0x05
// REP Relay field,内容取值如下 X’00’ succeeded
// RSV 保留字段
// ATYPE 地址类型
// BND.ADDR 服务绑定的地址
// BND.PORT 服务绑定的端口DST.PORT
_, err = conn.Write([]byte{0x05, 0x00, 0x00, 0x01, 0, 0, 0, 0, 0, 0})
if err != nil {
        return fmt.Errorf("write failed: %w", err)
}
ctx, cancel := context.WithCancel(context.Background())
defer cancel()
```

```
        go func() {
                _, _ = io.Copy(dest, reader)
                cancel()
        }()
        go func() {
                _, _ = io.Copy(conn, dest)
                cancel()
        }()

        <-ctx.Done()
        return nil
}
```