数据交换软件

v1.0

使用说明书

2018年6月4日

[1. 概述 4](#_Toc31074)

[2. 软件运行的软硬件环境 4](#_Toc3363)

[3. 安装使用说明 4](#_Toc16481)

[4. 软件接口api说明 7](#_Toc29394)

[func main() 7](#_Toc28660)

[func webserver() 7](#_Toc4195)

[func netsocket() 7](#_Toc10244)

[func dbblock() 7](#_Toc29954)

[func (web \*WEB)listen(conn \*net.Conn) 8](#_Toc5719)

[func (web \*WEB)recv(conn \*net.Conn) 8](#_Toc3484)

[func(web \*WEB)send(conn \*net.Conn) 8](#_Toc31094)

[func(web \*WEB)stop(conn \*net.Conn) 8](#_Toc22170)

[func (d \*Dialer) deadline(ctx context.Context, now time.Time) (earliest time.Time) 8](#_Toc23556)

[func (d \*Dialer) resolver() \*Resolver 9](#_Toc14449)

[func parseNetwork(ctx context.Context, network string, needsProto bool) (afnet string, proto int, err error) 9](#_Toc21403)

[func (r \*Resolver) resolveAddrList(ctx context.Context, op, network, addr string, hint Addr) (addrList, error) 9](#_Toc19840)

[func Dial(network, address string) (Conn, error) 10](#_Toc10139)

[func (d \*Dialer) Dial(network, address string) (Conn, error) 10](#_Toc1047)

[func (d \*Dialer) DialContext(ctx context.Context, network, address string) 10](#_Toc10148)

[func dialParallel(ctx context.Context, dp \*dialParam, primaries, fallbacks addrList) (Conn, error) 10](#_Toc26026)

[func dialSingle(ctx context.Context, dp \*dialParam, ra Addr) (c Conn, err error) 11](#_Toc30339)

[func (k \*contextKey) String() string { return "net/http context value " + k.name } 11](#_Toc21608)

[func hasPort(s string) bool { return strings.LastIndex(s, ":") > strings.LastIndex(s, "]") } 11](#_Toc30169)

[func isASCII(s string) bool 11](#_Toc23256)

[func refererForURL(lastReq, newReq \*url.URL) string 11](#_Toc8145)

[func (c \*Client) send(req \*Request, deadline time.Time) (resp \*Response, didTimeout func() bool, err error) 11](#_Toc17715)

[func send(ireq \*Request, rt RoundTripper, deadline time.Time) (resp \*Response, didTimeout func() bool, err error) 12](#_Toc18326)

[func setRequestCancel(req \*Request, rt RoundTripper, deadline time.Time) (stopTimer func(), didTimeout func() bool) 12](#_Toc14979)

[func basicAuth(username, password string) string 12](#_Toc15980)

[func Get(url string) (resp \*Response, err error) 12](#_Toc31703)

[func (c \*Client) checkRedirect(req \*Request, via []\*Request) error 12](#_Toc2708)

[func (c \*Client) makeHeadersCopier(ireq \*Request) func(\*Request) 13](#_Toc25104)

[func Post(url string, contentType string, body io.Reader) (resp \*Response, err error) 13](#_Toc1646)

[func (c \*Client) Post(url string, contentType string, body io.Reader) (resp \*Response, err error) 13](#_Toc14096)

[func PostForm(url string, data url.Values) (resp \*Response, err error) 13](#_Toc12650)

[func (c \*Client) PostForm(url string, data url.Values) (resp \*Response, err error) 13](#_Toc11594)

[func dbinit() 13](#_Toc2509)

[func (ns \*NullString) Scan(value interface{}) error 13](#_Toc4368)

[func (ns NullString) Value() (driver.Value, error) 14](#_Toc15700)

[func (n NullInt64) Value() (driver.Value, error) 14](#_Toc26895)

[func (db \*DB) maxIdleConnsLocked() int 14](#_Toc12346)

[func (db \*DB) SetMaxIdleConns(n int) 14](#_Toc10194)

[func (db \*DB) connectionCleaner(d time.Duration) 14](#_Toc5426)

[func (db \*DB) Stats() DBStats 14](#_Toc24137)

[func (db \*DB) maybeOpenNewConnections() 14](#_Toc26185)

[func (db \*DB) connectionOpener() 14](#_Toc8107)

[func (db \*DB) openNewConnection() 14](#_Toc19924)

[func (db \*DB) nextRequestKeyLocked() uint64 15](#_Toc2337)

[func (db \*DB) conn(ctx context.Context, strategy connReuseStrategy) 15](#_Toc18387)

[func (db \*DB) putConn(dc \*driverConn, err error) 15](#_Toc13346)

[func (db \*DB) putConnDBLocked(dc \*driverConn, err error) bool 15](#_Toc18292)

[func Marshal(v interface{}) ([]byte, error) 15](#_Toc20591)

[func Unmarshal(data []byte, v interface{}) error 15](#_Toc18841)

[5. Lib库接口api说明 15](#_Toc19201)

[http包 15](#_Toc17540)

[Net包 66](#_Toc20852)

[6. 开发举例说明 120](#_Toc9769)

# 概述

数据交换软件是一款数据交换的接口软件，它利用sql语言对关系型数据库的数据，通过交换转换成json格式的文本，然后通过socket通信传输给web调用，通过web的service的接口对外发布服务，使得可以兼容各类数据库，数据交换成统一的web接口命令，可以成立为永久性的数据交换接口使用。

# 软件运行的软硬件环境

硬件环境：

Intel cpu xen处理器，内存16g以上，硬盘空间为300G

软件环境：

Ubuntu Linux 16.04操作系统

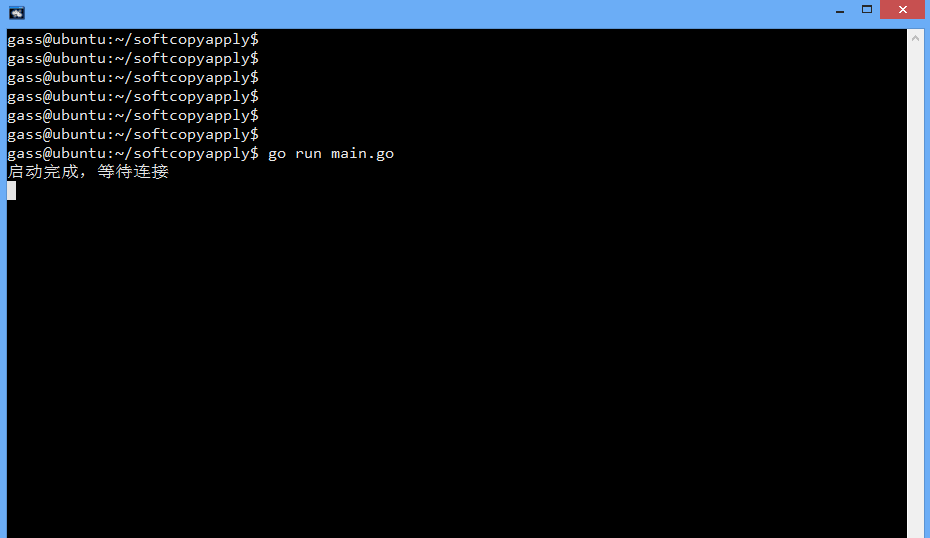
兼容windows系统，windows 2008以上服务器操作系统

编译环境为golang 1.9.0以上版本

# 安装使用说明

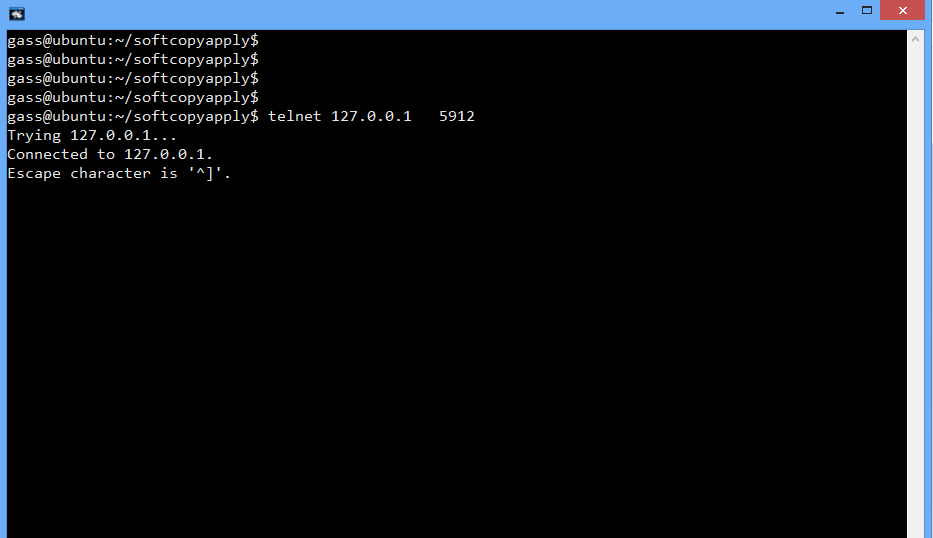
软件文件包括main.go文件和lib目录，lib目录下有client.go，decode.go，dial.go，encode.go，http.go，sql.go。Lib下目录文件为软件sdk开发库。主体文件为main.go。

运行程序在系统命令行下运行 go run main.go ,执行后如图：

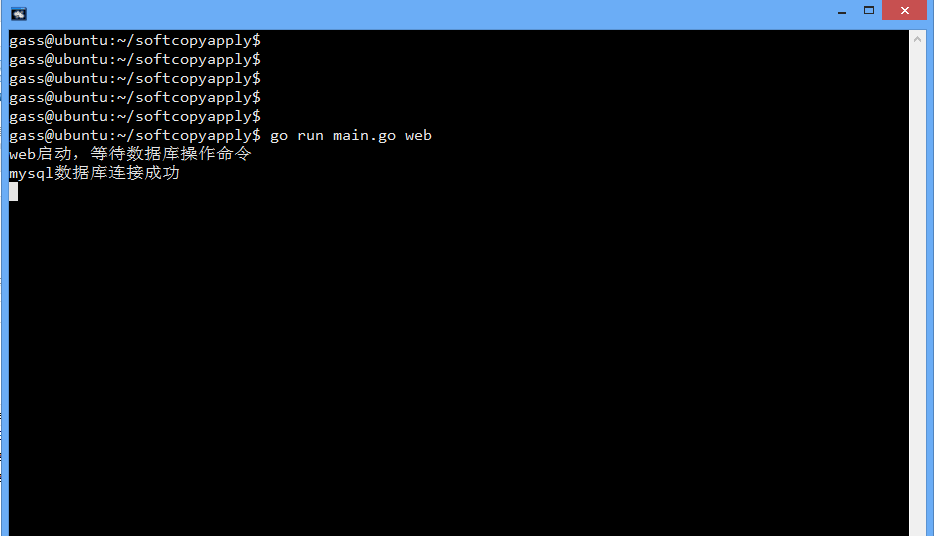


此种状态下，软件对外打开socket通信服务，外部命令可以通过json格式的文本输入进系统中。

如果想要查看服务端口打开与否，可以运行telnet 127.0.0.1 5912, 5912位本软件的缺省段阔，实际情况按照用户需求配置。如下图：

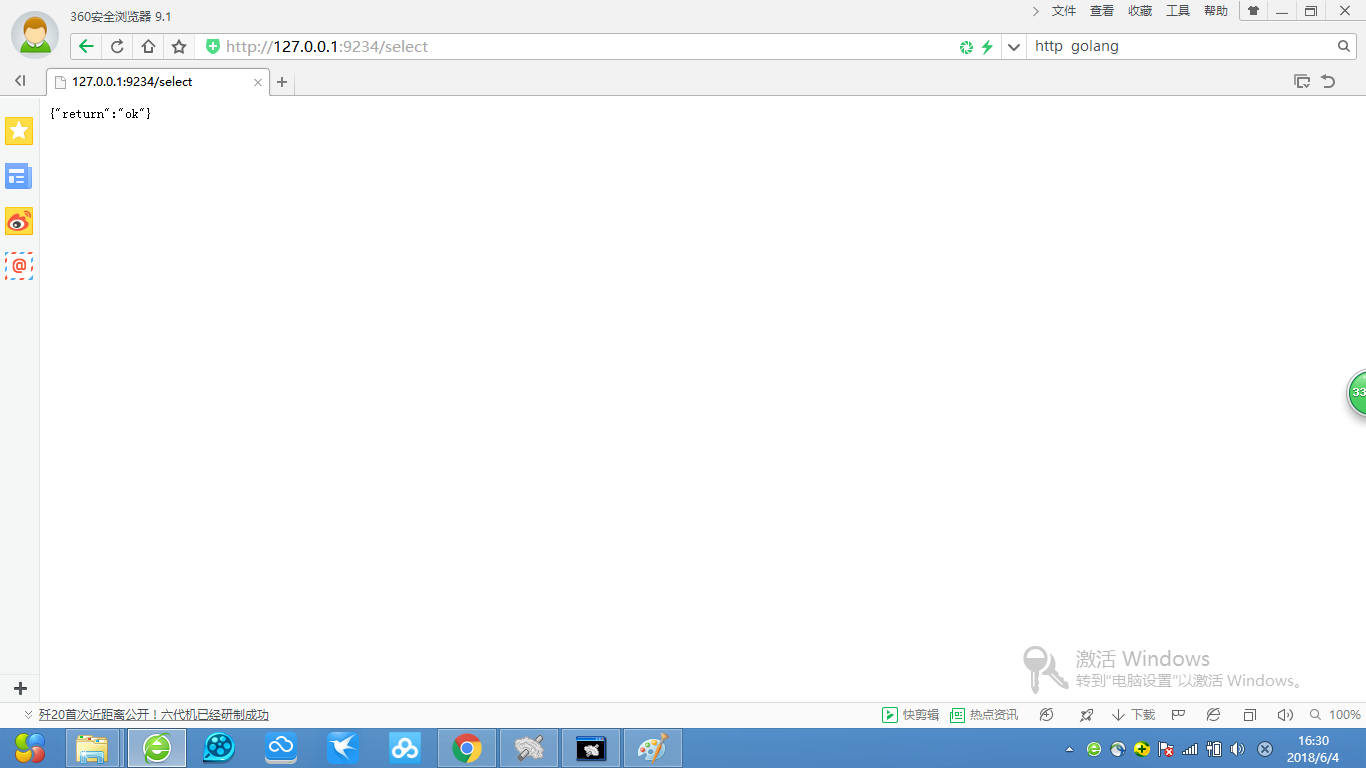


在命令行模式下，运行 go run main.go web ,如下图：



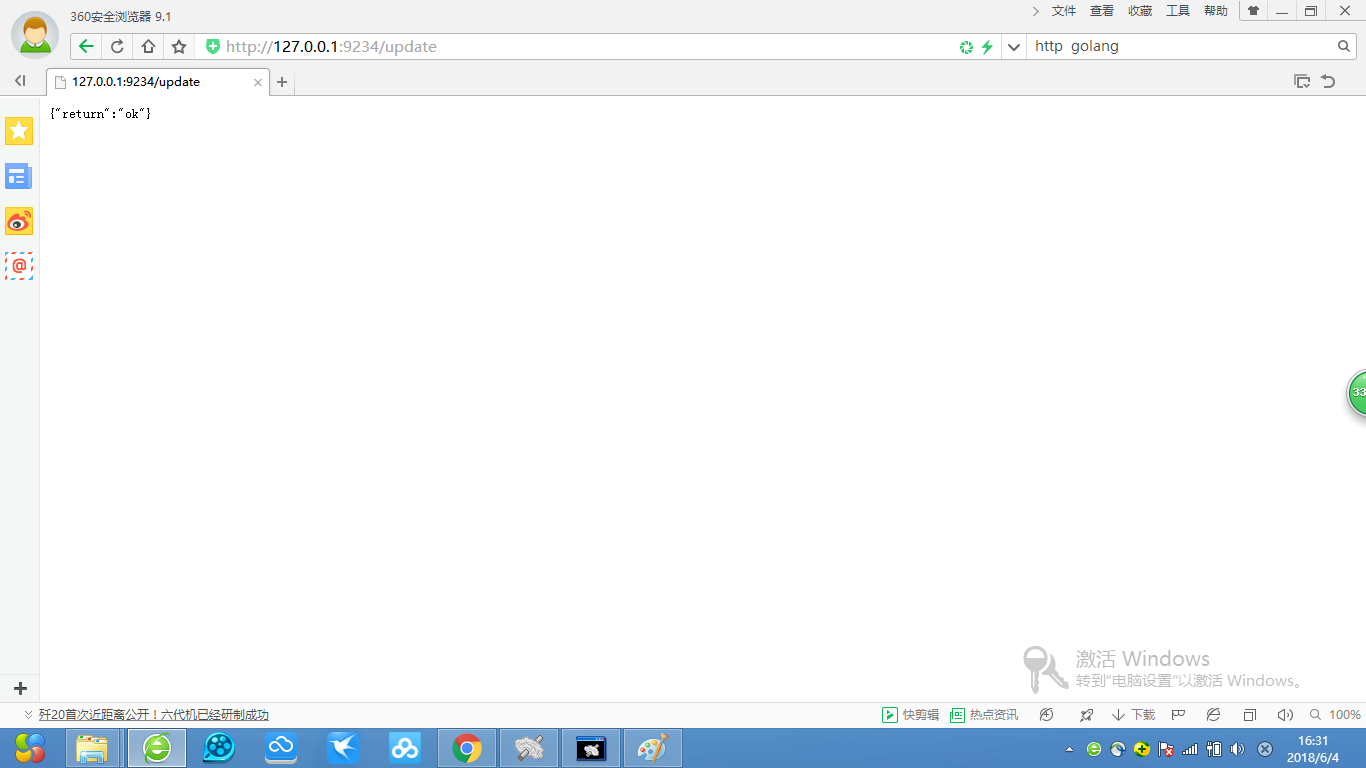
此种状态，软件打开web服务器，可以通过web调用把json格式文本输入软件系统，并主动连接后台数据库，对此数据进行操作处理。

在web服务器起来后，可以通过浏览器测试web通信是否正常，利用浏览器浏览127.0.0.1：9234/select，如下图：



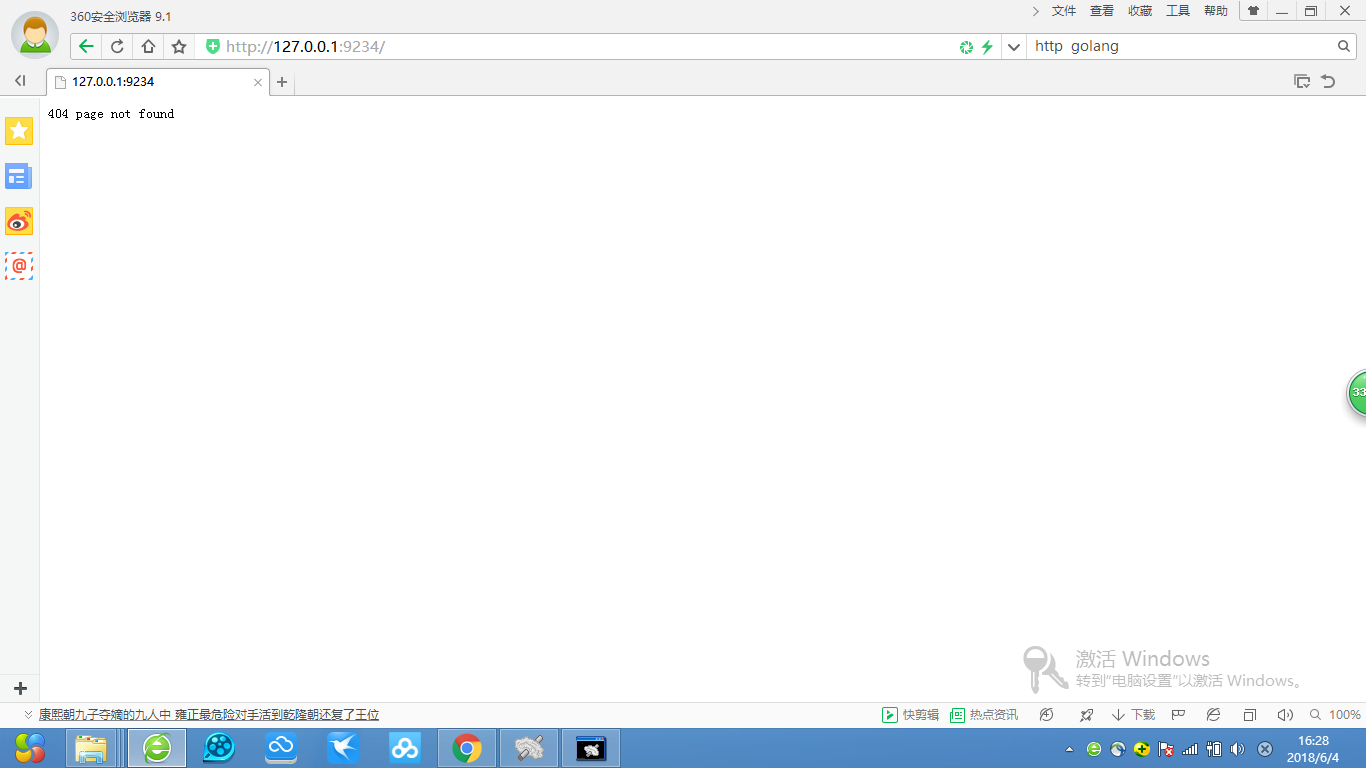
反馈结果为ok，代表select操作测试正常。

在web服务器起来后，可以通过浏览器测试web通信是否正常，利用浏览器浏览127.0.0.1：9234/update，如下图：



反馈结果为ok，代表update操作测试正常。

如果反馈结果为404错误，代表操作测试失败，如下图：



实际环境中，使用本软件，需要进行二次开发，利用本软件提供的api函数，进行接口应用开发，编译出实际使用的exe软件。

# 软件接口api说明

## func main()

软件的主函数，实现软件的主框架的主循环。

## func webserver()

将获取到的数据根据http协议进行解码，并启动web服务，实现web接口功能

## func netsocket()

在tcpip网络中实现socket通信，利用socket的接口应用实现网络数据交换。

## func dbblock()

兼容各类数据库封装的数据模块。本文例子都使用mysql。

## func (web \*WEB)listen(conn \*net.Conn)

Web服务器启动，使用此函数实现稳定的监听，保证web接口的有效使用

## func (web \*WEB)recv(conn \*net.Conn)

Web服务器的接受功能，通过web的服务器的conn接口，实现web接受数据功能

## func(web \*WEB)send(conn \*net.Conn)

Web服务器的发送共能，通过web服务器的conn接口，实现web的发送数据功能

## func(web \*WEB)stop(conn \*net.Conn)

启动web服务器后，停止web服务，并清扫内存残留。

## func (d \*Dialer) deadline(ctx context.Context, now time.Time) (earliest time.Time)

Socket通信建立后，设置socket链接的超时时间，切断上下文内容，反馈切断上下文的最早的时间。

## func (d \*Dialer) resolver() \*Resolver

解析网络socket通信中的域名服务

## func parseNetwork(ctx context.Context, network string, needsProto bool) (afnet string, proto int, err error)

处理socket通信中，网络上下文内容为可读字串，反馈网络名称协议和出错信息。

## func (r \*Resolver) resolveAddrList(ctx context.Context, op, network, addr string, hint Addr) (addrList, error)

解析网络socket通信中的域名服务，但是根据上下文进行批量操作，反馈地址序列和出错信息。

## func Dial(network, address string) (Conn, error)

Socket的网络连接，主动发起连接。反馈socket的连接和出错信息。

## func (d \*Dialer) Dial(network, address string) (Conn, error)

在socket连接对象中，主动发起连接。反馈socket的连接和出错信息。

## func (d \*Dialer) DialContext(ctx context.Context, network, address string)

根据上下文，在socket连接对象中，主动发起连接。反馈socket的连接和出错信息。

## func dialParallel(ctx context.Context, dp \*dialParam, primaries, fallbacks addrList) (Conn, error)

根据上下文，输入网络参数，地址列表，并行的发起连接，反馈socket的连接和出错信息。

## func dialSingle(ctx context.Context, dp \*dialParam, ra Addr) (c Conn, err error)

根据上下文，输入网络参数，发送单一连接，反馈socket连接和出错信息。

## func (k \*contextKey) String() string { return "net/http context value " + k.name }

在上下文的对象中，解析出可读字符串。

## func hasPort(s string) bool { return strings.LastIndex(s, ":") > strings.LastIndex(s, "]") }

根据输入字串，获取url字串中的端口port的数值。

## func isASCII(s string) bool

判断是否为ascii的字串，对输出为true，不对输出为false

## func refererForURL(lastReq, newReq \*url.URL) string

获取web通信中的refer的url，输出为字串。

## func (c \*Client) send(req \*Request, deadline time.Time) (resp \*Response, didTimeout func() bool, err error)

在客户端的对象中，发送请求，并根据超时时间限制发送时间，反馈成功为true，不成功为false，并在反馈输出反应对象。

## func send(ireq \*Request, rt RoundTripper, deadline time.Time) (resp \*Response, didTimeout func() bool, err error)

在客户端的对象中，发送请求，并根据超时时间限制发送时间，反馈成功为true，不成功为false，并在反馈输出反应对象。发送循环次数有设置，根据输入所决定。

## func setRequestCancel(req \*Request, rt RoundTripper, deadline time.Time) (stopTimer func(), didTimeout func() bool)

设置请求取消，反馈停止时间

## func basicAuth(username, password string) string

网络通信中认证，输入用户名和密码，输出sha256的数值

## func Get(url string) (resp \*Response, err error)

在socket通信中，发送get命令，获取响应。

## func (c \*Client) checkRedirect(req \*Request, via []\*Request) error

在客户端对象中，通过请求，检查重定向。

## func (c \*Client) makeHeadersCopier(ireq \*Request) func(\*Request)

备份制作一份数据包头内容。

## func Post(url string, contentType string, body io.Reader) (resp \*Response, err error)

Web通信中发送post命令。

## func (c \*Client) Post(url string, contentType string, body io.Reader) (resp \*Response, err error)

在客户端对象中，发送post命令。

## func PostForm(url string, data url.Values) (resp \*Response, err error)

Web通信中发送post命令，命令的格式为html中的格式。

## func (c \*Client) PostForm(url string, data url.Values) (resp \*Response, err error)

在客户端对象中，发送post命令，命令的格式为html中的Form格式。

## func dbinit()

数据库操作之前的初始化。

## func (ns \*NullString) Scan(value interface{}) error

扫描数据库，获取数据库结构。

## func (ns NullString) Value() (driver.Value, error)

根据驱动字串，获取驱动属于哪个数据库的操作。

## func (n NullInt64) Value() (driver.Value, error)

根据驱动数值，获取驱动属于哪个数据库的操作。

## func (db \*DB) maxIdleConnsLocked() int

获取数据库的最大连接锁定的数值。

## func (db \*DB) SetMaxIdleConns(n int)

设置数据库最大空闲连接数。

## func (db \*DB) connectionCleaner(d time.Duration)

在数据库对象中，清理连接。

## func (db \*DB) Stats() DBStats

获取数据库的连接状态信息。

## func (db \*DB) maybeOpenNewConnections()

在数据库对象中，打开新连接。

## func (db \*DB) connectionOpener()

在数据库对象中，获取打开的连接。

## func (db \*DB) openNewConnection()

数据库打开新连接。

## func (db \*DB) nextRequestKeyLocked() uint64

在数据库对象中，下次请求，并锁定。

## func (db \*DB) conn(ctx context.Context, strategy connReuseStrategy)

根据上下文和策略获取数据库的连接。

## func (db \*DB) putConn(dc \*driverConn, err error)

在驱动对象中，把各类数据库的驱动放置在统一的连接中。

## func (db \*DB) putConnDBLocked(dc \*driverConn, err error) bool

在驱动对象中，把各类数据库的驱动放置在统一的连接中并锁定连接。

## func Marshal(v interface{}) ([]byte, error)

json格式化字串，实现标准的json对象字符串。

## func Unmarshal(data []byte, v interface{}) error

把json字符串解码成json对象结构。

# **Lib库接口api说明**

## http包

import "net/http"

http包提供了HTTP客户端和服务端的实现。

Get、Head、Post和PostForm函数发出HTTP/ HTTPS请求。

resp, err := http.Get("<http://example.com/>")

...

resp, err := http.Post("<http://example.com/upload>", "image/jpeg", &buf)

...

resp, err := http.PostForm("<http://example.com/form>",

url.Values{"key": {"Value"}, "id": {"123"}})

程序在使用完回复后必须关闭回复的主体。

resp, err := http.Get("<http://example.com/>")

if err != nil {

// handle error

}

defer resp.Body.Close()

body, err := ioutil.ReadAll(resp.Body)

// ...

要管理HTTP客户端的头域、重定向策略和其他设置，创建一个Client：

client := &http.Client{

CheckRedirect: redirectPolicyFunc,

}

resp, err := client.Get("[http://example.com](http://example.com/)")

// ...

req, err := http.NewRequest("GET", "[http://example.com](http://example.com/)", nil)

// ...

req.Header.Add("If-None-Match", `W/"wyzzy"`)

resp, err := client.Do(req)

// ...

要管理代理、TLS配置、keep-alive、压缩和其他设置，创建一个Transport：

tr := &http.Transport{

TLSClientConfig: &tls.Config{RootCAs: pool},

DisableCompression: true,

}

client := &http.Client{Transport: tr}

resp, err := client.Get("[https://example.com](https://example.com/)")

Client和Transport类型都可以安全的被多个go程同时使用。出于效率考虑，应该一次建立、尽量重用。

ListenAndServe使用指定的监听地址和处理器启动一个HTTP服务端。处理器参数通常是nil，这表示采用包变量DefaultServeMux作为处理器。Handle和HandleFunc函数可以向DefaultServeMux添加处理器。

http.Handle("/foo", fooHandler)

http.HandleFunc("/bar", func(w http.ResponseWriter, r \*http.Request) {

fmt.Fprintf(w, "Hello, %q", html.EscapeString(r.URL.Path))

})

log.Fatal(http.ListenAndServe(":8080", nil))

要管理服务端的行为，可以创建一个自定义的Server：

s := &http.Server{

Addr: ":8080",

Handler: myHandler,

ReadTimeout: 10 \* time.Second,

WriteTimeout: 10 \* time.Second,

MaxHeaderBytes: 1 << 20,

}

log.Fatal(s.ListenAndServe())

· [Constants](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "pkg-constants)

· [Variables](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "pkg-variables)

· [type ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError)

* [func (err \*ProtocolError) Error() string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError.Error)

· [func CanonicalHeaderKey(s string) string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "CanonicalHeaderKey)

· [func DetectContentType(data []byte) string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "DetectContentType)

· [func ParseHTTPVersion(vers string) (major, minor int, ok bool)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ParseHTTPVersion)

· [func ParseTime(text string) (t time.Time, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ParseTime)

· [func StatusText(code int) string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "StatusText)

· [type ConnState](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ConnState)

* [func (c ConnState) String() string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ConnState.String)

· [type Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)

* [func (h Header) Get(key string) string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header.Get)
* [func (h Header) Set(key, value string)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header.Set)
* [func (h Header) Add(key, value string)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header.Add)
* [func (h Header) Del(key string)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header.Del)
* [func (h Header) Write(w io.Writer) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header.Write)
* [func (h Header) WriteSubset(w io.Writer, exclude map[string]bool) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header.WriteSubset)

· [type Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie)

* [func (c \*Cookie) String() string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie.String)

· [type CookieJar](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "CookieJar)

· [type Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)

* [func NewRequest(method, urlStr string, body io.Reader) (\*Request, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "NewRequest)
* [func ReadRequest(b \*bufio.Reader) (req \*Request, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ReadRequest)
* [func (r \*Request) ProtoAtLeast(major, minor int) bool](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.ProtoAtLeast)
* [func (r \*Request) UserAgent() string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.UserAgent)
* [func (r \*Request) Referer() string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.Referer)
* [func (r \*Request) AddCookie(c \*Cookie)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.AddCookie)
* [func (r \*Request) SetBasicAuth(username, password string)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.SetBasicAuth)
* [func (r \*Request) Write(w io.Writer) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.Write)
* [func (r \*Request) WriteProxy(w io.Writer) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.WriteProxy)
* [func (r \*Request) Cookies() []\*Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.Cookies)
* [func (r \*Request) Cookie(name string) (\*Cookie, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.Cookie)
* [func (r \*Request) ParseForm() error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.ParseForm)
* [func (r \*Request) ParseMultipartForm(maxMemory int64) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.ParseMultipartForm)
* [func (r \*Request) FormValue(key string) string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.FormValue)
* [func (r \*Request) PostFormValue(key string) string](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.PostFormValue)
* [func (r \*Request) FormFile(key string) (multipart.File, \*multipart.FileHeader, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.FormFile)
* [func (r \*Request) MultipartReader() (\*multipart.Reader, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request.MultipartReader)

· [type Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response)

* [func ReadResponse(r \*bufio.Reader, req \*Request) (\*Response, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ReadResponse)
* [func (r \*Response) ProtoAtLeast(major, minor int) bool](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response.ProtoAtLeast)
* [func (r \*Response) Cookies() []\*Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response.Cookies)
* [func (r \*Response) Location() (\*url.URL, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response.Location)
* [func (r \*Response) Write(w io.Writer) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response.Write)

· [type ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter)

· [type Flusher](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Flusher)

· [type CloseNotifier](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "CloseNotifier)

· [type Hijacker](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Hijacker)

· [type RoundTripper](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "RoundTripper)

· [type Transport](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport)

* [func (t \*Transport) RegisterProtocol(scheme string, rt RoundTripper)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport.RegisterProtocol)
* [func (t \*Transport) RoundTrip(req \*Request) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport.RoundTrip)
* [func (t \*Transport) CloseIdleConnections()](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport.CloseIdleConnections)
* [func (t \*Transport) CancelRequest(req \*Request)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport.CancelRequest)

· [type Client](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client)

* [func (c \*Client) Do(req \*Request) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client.Do)
* [func (c \*Client) Head(url string) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client.Head)
* [func (c \*Client) Get(url string) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client.Get)
* [func (c \*Client) Post(url string, bodyType string, body io.Reader) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client.Post)
* [func (c \*Client) PostForm(url string, data url.Values) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client.PostForm)

· [type Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)

* [func NotFoundHandler() Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "NotFoundHandler)
* [func RedirectHandler(url string, code int) Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "RedirectHandler)
* [func TimeoutHandler(h Handler, dt time.Duration, msg string) Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "TimeoutHandler)
* [func StripPrefix(prefix string, h Handler) Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "StripPrefix)

· [type HandlerFunc](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "HandlerFunc)

* [func (f HandlerFunc) ServeHTTP(w ResponseWriter, r \*Request)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "HandlerFunc.ServeHTTP)

· [type ServeMux](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux)

* [func NewServeMux() \*ServeMux](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "NewServeMux)
* [func (mux \*ServeMux) Handle(pattern string, handler Handler)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux.Handle)
* [func (mux \*ServeMux) HandleFunc(pattern string, handler func(ResponseWriter, \*Request))](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux.HandleFunc)
* [func (mux \*ServeMux) Handler(r \*Request) (h Handler, pattern string)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux.Handler)
* [func (mux \*ServeMux) ServeHTTP(w ResponseWriter, r \*Request)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux.ServeHTTP)

· [type Server](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server)

* [func (s \*Server) SetKeepAlivesEnabled(v bool)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server.SetKeepAlivesEnabled)
* [func (srv \*Server) Serve(l net.Listener) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server.Serve)
* [func (srv \*Server) ListenAndServe() error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server.ListenAndServe)
* [func (srv \*Server) ListenAndServeTLS(certFile, keyFile string) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server.ListenAndServeTLS)

· [type File](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "File)

· [type FileSystem](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "FileSystem)

· [type Dir](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Dir)

* [func (d Dir) Open(name string) (File, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Dir.Open)

· [func NewFileTransport(fs FileSystem) RoundTripper](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "NewFileTransport)

· [func FileServer(root FileSystem) Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "FileServer)

· [func SetCookie(w ResponseWriter, cookie \*Cookie)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "SetCookie)

· [func Redirect(w ResponseWriter, r \*Request, urlStr string, code int)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Redirect)

· [func NotFound(w ResponseWriter, r \*Request)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "NotFound)

· [func Error(w ResponseWriter, error string, code int)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Error)

· [func ServeContent(w ResponseWriter, req \*Request, name string, modtime time.Time, content io.ReadSeeker)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeContent)

· [func ServeFile(w ResponseWriter, r \*Request, name string)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeFile)

· [func MaxBytesReader(w ResponseWriter, r io.ReadCloser, n int64) io.ReadCloser](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "MaxBytesReader)

· [func ProxyURL(fixedURL \*url.URL) func(\*Request) (\*url.URL, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProxyURL)

· [func ProxyFromEnvironment(req \*Request) (\*url.URL, error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProxyFromEnvironment)

· [func Head(url string) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Head)

· [func Get(url string) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Get)

· [func Post(url string, bodyType string, body io.Reader) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Post)

· [func PostForm(url string, data url.Values) (resp \*Response, err error)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "PostForm)

· [func Handle(pattern string, handler Handler)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handle)

· [func HandleFunc(pattern string, handler func(ResponseWriter, \*Request))](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "HandleFunc)

· [func Serve(l net.Listener, handler Handler) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Serve)

· [func ListenAndServe(addr string, handler Handler) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ListenAndServe)

· [func ListenAndServeTLS(addr string, certFile string, keyFile string, handler Handler) error](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ListenAndServeTLS)

#### Examples

· [FileServer](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "example-FileServer)

· [FileServer (StripPrefix)](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "example-FileServer--StripPrefix)

· [Get](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "example-Get)

· [Hijacker](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "example-Hijacker)

· [ServeMux.Handle](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "example-ServeMux-Handle)

· [StripPrefix](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "example-StripPrefix)

const (

StatusContinue = 100

StatusSwitchingProtocols = 101

StatusOK = 200

StatusCreated = 201

StatusAccepted = 202

StatusNonAuthoritativeInfo = 203

StatusNoContent = 204

StatusResetContent = 205

StatusPartialContent = 206

StatusMultipleChoices = 300

StatusMovedPermanently = 301

StatusFound = 302

StatusSeeOther = 303

StatusNotModified = 304

StatusUseProxy = 305

StatusTemporaryRedirect = 307

StatusBadRequest = 400

StatusUnauthorized = 401

StatusPaymentRequired = 402

StatusForbidden = 403

StatusNotFound = 404

StatusMethodNotAllowed = 405

StatusNotAcceptable = 406

StatusProxyAuthRequired = 407

StatusRequestTimeout = 408

StatusConflict = 409

StatusGone = 410

StatusLengthRequired = 411

StatusPreconditionFailed = 412

StatusRequestEntityTooLarge = 413

StatusRequestURITooLong = 414

StatusUnsupportedMediaType = 415

StatusRequestedRangeNotSatisfiable = 416

StatusExpectationFailed = 417

StatusTeapot = 418

StatusInternalServerError = 500

StatusNotImplemented = 501

StatusBadGateway = 502

StatusServiceUnavailable = 503

StatusGatewayTimeout = 504

StatusHTTPVersionNotSupported = 505

)

HTTP状态码，参见[RFC 2616](http://tools.ietf.org/html/rfc2616)

const DefaultMaxHeaderBytes = 1 << 20 // 1 MB

DefaultMaxHeaderBytes是HTTP请求的头域最大允许长度。可以通过设置Server.MaxHeaderBytes字段来覆盖。

const DefaultMaxIdleConnsPerHost = 2

DefaultMaxIdleConnsPerHost是Transport的MaxIdleConnsPerHost的默认值。

const TimeFormat = "Mon, 02 Jan 2006 15:04:05 GMT"

TimeFormat是当解析或生产HTTP头域中的时间时，用与time.Parse或time.Format函数的时间格式。这种格式类似time.RFC1123但强制采用GMT时区。

### Variables

var (

ErrHeaderTooLong = &[ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError){"header too long"}

ErrShortBody = &[ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError){"entity body too short"}

ErrNotSupported = &[ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError){"feature not supported"}

ErrUnexpectedTrailer = &[ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError){"trailer header without chunked transfer encoding"}

ErrMissingContentLength = &[ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError){"missing ContentLength in HEAD response"}

ErrNotMultipart = &[ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError){"request Content-Type isn't multipart/form-data"}

ErrMissingBoundary = &[ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError){"no multipart boundary param in Content-Type"}

)

HTTP请求的解析错误。

var (

ErrWriteAfterFlush = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("Conn.Write called after Flush")

ErrBodyNotAllowed = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("http: request method or response status code does not allow body")

ErrHijacked = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("Conn has been hijacked")

ErrContentLength = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("Conn.Write wrote more than the declared Content-Length")

)

会被HTTP服务端返回的错误。

var DefaultClient = &[Client](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client){}

DefaultClient是用于包函数Get、Head和Post的默认Client。

var DefaultServeMux = [NewServeMux](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "NewServeMux)()

DefaultServeMux是用于Serve的默认ServeMux。

var ErrBodyReadAfterClose = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("http: invalid Read on closed Body")

在Resquest或Response的Body字段已经关闭后，试图从中读取时，就会返回ErrBodyReadAfterClose。这个错误一般发生在：HTTP处理器中调用完ResponseWriter 接口的WriteHeader或Write后从请求中读取数据的时候。

var ErrHandlerTimeout = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("http: Handler timeout")

在处理器超时以后调用ResponseWriter接口的Write方法，就会返回ErrHandlerTimeout。

var ErrLineTooLong = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("header line too long")

var ErrMissingFile = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("http: no such file")

当请求中没有提供给FormFile函数的文件字段名，或者该字段名不是文件字段时，该函数就会返回ErrMissingFile。

var ErrNoCookie = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("http: named cookie not present")

var ErrNoLocation = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("http: no Location header in response")

### type [ProtocolError](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "38" \o "View Source)

type ProtocolError struct {

ErrorString [string](http://godoc.org/builtin" \l "string)

}

HTTP请求解析错误。

#### func (\*ProtocolError) [Error](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "42" \o "View Source)

func (err \*[ProtocolError](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProtocolError)) Error() [string](http://godoc.org/builtin" \l "string)

### func [CanonicalHeaderKey](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "171" \o "View Source)

func CanonicalHeaderKey(s [string](http://godoc.org/builtin" \l "string)) [string](http://godoc.org/builtin" \l "string)

CanonicalHeaderKey函数返回头域（表示为Header类型）的键s的规范化格式。规范化过程中让单词首字母和'-'后的第一个字母大写，其余字母小写。例如，"accept-encoding"规范化为"Accept-Encoding"。

### func [DetectContentType](https://github.com/golang/go/blob/master/src/net/http/sniff.go?name=release" \l "21" \o "View Source)

func DetectContentType(data [][byte](http://godoc.org/builtin" \l "byte)) [string](http://godoc.org/builtin" \l "string)

DetectContentType函数实现了<http://mimesniff.spec.whatwg.org/>描述的算法，用于确定数据的Content-Type。函数总是返回一个合法的MIME类型；如果它不能确定数据的类型，将返回"application/octet-stream"。它最多检查数据的前512字节。

### func [ParseHTTPVersion](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "448" \o "View Source)

func ParseHTTPVersion(vers [string](http://godoc.org/builtin" \l "string)) (major, minor [int](http://godoc.org/builtin" \l "int), ok [bool](http://godoc.org/builtin" \l "bool))

ParseHTTPVersion解析HTTP版本字符串。如"HTTP/1.0"返回(1, 0, true)。

### func [ParseTime](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "79" \o "View Source)

func ParseTime(text [string](http://godoc.org/builtin" \l "string)) (t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time), err [error](http://godoc.org/builtin" \l "error))

ParseTime用3种格式TimeFormat, time.RFC850和time.ANSIC尝试解析一个时间头的值（如Date: header）。

### func [StatusText](https://github.com/golang/go/blob/master/src/net/http/status.go?name=release" \l "118" \o "View Source)

func StatusText(code [int](http://godoc.org/builtin" \l "int)) [string](http://godoc.org/builtin" \l "string)

StatusText返回HTTP状态码code对应的文本，如220对应"OK"。如果code是未知的状态码，会返回""。

### type [ConnState](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1614" \o "View Source)

type ConnState [int](http://godoc.org/builtin" \l "int)

ConnState代表一个客户端到服务端的连接的状态。本类型用于可选的Server.ConnState回调函数。

const (

// StateNew代表一个新的连接，将要立刻发送请求。

// 连接从这个状态开始，然后转变为StateAlive或StateClosed。

StateNew [ConnState](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ConnState) = [iota](http://godoc.org/builtin" \l "iota)

// StateActive代表一个已经读取了请求数据1到多个字节的连接。

// 用于StateAlive的Server.ConnState回调函数在将连接交付给处理器之前被触发，

// 等到请求被处理完后，Server.ConnState回调函数再次被触发。

// 在请求被处理后，连接状态改变为StateClosed、StateHijacked或StateIdle。

StateActive

// StateIdle代表一个已经处理完了请求、处在闲置状态、等待新请求的连接。

// 连接状态可以从StateIdle改变为StateActive或StateClosed。

StateIdle

// 代表一个被劫持的连接。这是一个终止状态，不会转变为StateClosed。

StateHijacked

// StateClosed代表一个关闭的连接。

// 这是一个终止状态。被劫持的连接不会转变为StateClosed。

StateClosed

)

#### func (ConnState) [String](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1655" \o "View Source)

func (c [ConnState](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ConnState)) String() [string](http://godoc.org/builtin" \l "string)

### type [Header](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "19" \o "View Source)

type Header map[[string](http://godoc.org/builtin" \l "string)][][string](http://godoc.org/builtin" \l "string)

Header代表HTTP头域的键值对。

#### func (Header) [Get](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "38" \o "View Source)

func (h [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)) Get(key [string](http://godoc.org/builtin" \l "string)) [string](http://godoc.org/builtin" \l "string)

Get返回键对应的第一个值，如果键不存在会返回""。如要获取该键对应的值切片，请直接用规范格式的键访问map。

#### func (Header) [Set](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "30" \o "View Source)

func (h [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)) Set(key, value [string](http://godoc.org/builtin" \l "string))

Set添加键值对到h，如键已存在则会用只有新值一个元素的切片取代旧值切片。

#### func (Header) [Add](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "23" \o "View Source)

func (h [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)) Add(key, value [string](http://godoc.org/builtin" \l "string))

Add添加键值对到h，如键已存在则会将新的值附加到旧值切片后面。

#### func (Header) [Del](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "51" \o "View Source)

func (h [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)) Del(key [string](http://godoc.org/builtin" \l "string))

Del删除键值对。

#### func (Header) [Write](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "56" \o "View Source)

func (h [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)) Write(w [io](http://godoc.org/io).[Writer](http://godoc.org/io" \l "Writer)) [error](http://godoc.org/builtin" \l "error)

Write以有线格式将头域写入w。

#### func (Header) [WriteSubset](https://github.com/golang/go/blob/master/src/net/http/header.go?name=release" \l "145" \o "View Source)

func (h [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)) WriteSubset(w [io](http://godoc.org/io).[Writer](http://godoc.org/io" \l "Writer), exclude map[[string](http://godoc.org/builtin" \l "string)][bool](http://godoc.org/builtin" \l "bool)) [error](http://godoc.org/builtin" \l "error)

WriteSubset以有线格式将头域写入w。当exclude不为nil时，如果h的键值对的键在exclude中存在且其对应值为真，该键值对就不会被写入w。

### type [Cookie](https://github.com/golang/go/blob/master/src/net/http/cookie.go?name=release" \l "23" \o "View Source)

type Cookie struct {

Name [string](http://godoc.org/builtin" \l "string)

Value [string](http://godoc.org/builtin" \l "string)

Path [string](http://godoc.org/builtin" \l "string)

Domain [string](http://godoc.org/builtin" \l "string)

Expires [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)

RawExpires [string](http://godoc.org/builtin" \l "string)

// MaxAge=0表示未设置Max-Age属性

// MaxAge<0表示立刻删除该cookie，等价于"Max-Age: 0"

// MaxAge>0表示存在Max-Age属性，单位是秒

MaxAge [int](http://godoc.org/builtin" \l "int)

Secure [bool](http://godoc.org/builtin" \l "bool)

HttpOnly [bool](http://godoc.org/builtin" \l "bool)

Raw [string](http://godoc.org/builtin" \l "string)

Unparsed [][string](http://godoc.org/builtin" \l "string) // 未解析的“属性-值”对的原始文本

}

Cookie代表一个出现在HTTP回复的头域中Set-Cookie头的值里或者HTTP请求的头域中Cookie头的值里的HTTP cookie。

#### func (\*Cookie) [String](https://github.com/golang/go/blob/master/src/net/http/cookie.go?name=release" \l "136" \o "View Source)

func (c \*[Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie)) String() [string](http://godoc.org/builtin" \l "string)

String返回该cookie的序列化结果。如果只设置了Name和Value字段，序列化结果可用于HTTP请求的Cookie头或者HTTP回复的Set-Cookie头；如果设置了其他字段，序列化结果只能用于HTTP回复的Set-Cookie头。

### type [CookieJar](https://github.com/golang/go/blob/master/src/net/http/jar.go?name=release" \l "17" \o "View Source)

type CookieJar interface {

// SetCookies管理从u的回复中收到的cookie

// 根据其策略和实现，它可以选择是否存储cookie

SetCookies(u \*[url](http://godoc.org/net/url).[URL](http://godoc.org/net/url" \l "URL), cookies []\*[Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie))

// Cookies返回发送请求到u时应使用的cookie

// 本方法有责任遵守RFC 6265规定的标准cookie限制

Cookies(u \*[url](http://godoc.org/net/url).[URL](http://godoc.org/net/url" \l "URL)) []\*[Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie)

}

CookieJar管理cookie的存储和在HTTP请求中的使用。CookieJar的实现必须能安全的被多个go程同时使用。

net/http/cookiejar包提供了一个CookieJar的实现。

### type [Request](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "76" \o "View Source)

type Request struct {

// Method指定HTTP方法（GET、POST、PUT等）。对客户端，""代表GET。

Method [string](http://godoc.org/builtin" \l "string)

// URL在服务端表示被请求的URI，在客户端表示要访问的URL。

//

// 在服务端，URL字段是解析请求行的URI（保存在RequestURI字段）得到的，

// 对大多数请求来说，除了Path和RawQuery之外的字段都是空字符串。

// （参见RFC 2616, Section 5.1.2）

//

// 在客户端，URL的Host字段指定了要连接的服务器，

// 而Request的Host字段（可选地）指定要发送的HTTP请求的Host头的值。

URL \*[url](http://godoc.org/net/url).[URL](http://godoc.org/net/url" \l "URL)

// 接收到的请求的协议版本。本包生产的Request总是使用HTTP/1.1

Proto [string](http://godoc.org/builtin" \l "string) // "HTTP/1.0"

ProtoMajor [int](http://godoc.org/builtin" \l "int) // 1

ProtoMinor [int](http://godoc.org/builtin" \l "int) // 0

// Header字段用来表示HTTP请求的头域。如果头域（多行键值对格式）为：

// accept-encoding: gzip, deflate

// Accept-Language: en-us

// Connection: keep-alive

// 则：

// Header = map[string][]string{

// "Accept-Encoding": {"gzip, deflate"},

// "Accept-Language": {"en-us"},

// "Connection": {"keep-alive"},

// }

// HTTP规定头域的键名（头名）是大小写敏感的，请求的解析器通过规范化头域的键名来实现这点。

// 在客户端的请求，可能会被自动添加或重写Header中的特定的头，参见Request.Write方法。

Header [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)

// Body是请求的主体。

//

// 在客户端，如果Body是nil表示该请求没有主体买入GET请求。

// Client的Transport字段会负责调用Body的Close方法。

//

// 在服务端，Body字段总是非nil的；但在没有主体时，读取Body会立刻返回EOF。

// Server会关闭请求的主体，ServeHTTP处理器不需要关闭Body字段。

Body [io](http://godoc.org/io).[ReadCloser](http://godoc.org/io" \l "ReadCloser)

// ContentLength记录相关内容的长度。

// 如果为-1，表示长度未知，如果>=0，表示可以从Body字段读取ContentLength字节数据。

// 在客户端，如果Body非nil而该字段为0，表示不知道Body的长度。

ContentLength [int64](http://godoc.org/builtin" \l "int64)

// TransferEncoding按从最外到最里的顺序列出传输编码，空切片表示"identity"编码。

// 本字段一般会被忽略。当发送或接受请求时，会自动添加或移除"chunked"传输编码。

TransferEncoding [][string](http://godoc.org/builtin" \l "string)

// Close在服务端指定是否在回复请求后关闭连接，在客户端指定是否在发送请求后关闭连接。

Close [bool](http://godoc.org/builtin" \l "bool)

// 在服务端，Host指定URL会在其上寻找资源的主机。

// 根据RFC 2616，该值可以是Host头的值，或者URL自身提供的主机名。

// Host的格式可以是"host:port"。

//

// 在客户端，请求的Host字段（可选地）用来重写请求的Host头。

// 如过该字段为""，Request.Write方法会使用URL字段的Host。

Host [string](http://godoc.org/builtin" \l "string)

// Form是解析好的表单数据，包括URL字段的query参数和POST或PUT的表单数据。

// 本字段只有在调用ParseForm后才有效。在客户端，会忽略请求中的本字段而使用Body替代。

Form [url](http://godoc.org/net/url).[Values](http://godoc.org/net/url" \l "Values)

// PostForm是解析好的POST或PUT的表单数据。

// 本字段只有在调用ParseForm后才有效。在客户端，会忽略请求中的本字段而使用Body替代。

PostForm [url](http://godoc.org/net/url).[Values](http://godoc.org/net/url" \l "Values)

// MultipartForm是解析好的多部件表单，包括上传的文件。

// 本字段只有在调用ParseMultipartForm后才有效。

// 在客户端，会忽略请求中的本字段而使用Body替代。

MultipartForm \*[multipart](http://godoc.org/mime/multipart).[Form](http://godoc.org/mime/multipart" \l "Form)

// Trailer指定了会在请求主体之后发送的额外的头域。

//

// 在服务端，Trailer字段必须初始化为只有trailer键，所有键都对应nil值。

// （客户端会声明哪些trailer会发送）

// 在处理器从Body读取时，不能使用本字段。

// 在从Body的读取返回EOF后，Trailer字段会被更新完毕并包含非nil的值。

// （如果客户端发送了这些键值对），此时才可以访问本字段。

//

// 在客户端，Trail必须初始化为一个包含将要发送的键值对的映射。（值可以是nil或其终值）

// ContentLength字段必须是0或-1，以启用"chunked"传输编码发送请求。

// 在开始发送请求后，Trailer可以在读取请求主体期间被修改，

// 一旦请求主体返回EOF，调用者就不可再修改Trailer。

//

// 很少有HTTP客户端、服务端或代理支持HTTP trailer。

Trailer [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)

// RemoteAddr允许HTTP服务器和其他软件记录该请求的来源地址，一般用于日志。

// 本字段不是ReadRequest函数填写的，也没有定义格式。

// 本包的HTTP服务器会在调用处理器之前设置RemoteAddr为"IP:port"格式的地址。

// 客户端会忽略请求中的RemoteAddr字段。

RemoteAddr [string](http://godoc.org/builtin" \l "string)

// RequestURI是被客户端发送到服务端的请求的请求行中未修改的请求URI

// （参见RFC 2616, Section 5.1）

// 一般应使用URI字段，在客户端设置请求的本字段会导致错误。

RequestURI [string](http://godoc.org/builtin" \l "string)

// TLS字段允许HTTP服务器和其他软件记录接收到该请求的TLS连接的信息

// 本字段不是ReadRequest函数填写的。

// 对启用了TLS的连接，本包的HTTP服务器会在调用处理器之前设置TLS字段，否则将设TLS为nil。

// 客户端会忽略请求中的TLS字段。

TLS \*[tls](http://godoc.org/crypto/tls).[ConnectionState](http://godoc.org/crypto/tls" \l "ConnectionState)

}

Request类型代表一个服务端接受到的或者客户端发送出去的HTTP请求。

Request各字段的意义和用途在服务端和客户端是不同的。除了字段本身上方文档，还可参见Request.Write方法和RoundTripper接口的文档。

#### func [NewRequest](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "479" \o "View Source)

func NewRequest(method, urlStr [string](http://godoc.org/builtin" \l "string), body [io](http://godoc.org/io).[Reader](http://godoc.org/io" \l "Reader)) (\*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request), [error](http://godoc.org/builtin" \l "error))

NewRequest使用指定的方法、网址和可选的主题创建并返回一个新的\*Request。

如果body参数实现了io.Closer接口，Request返回值的Body 字段会被设置为body，并会被Client类型的Do、Post和PostFOrm方法以及Transport.RoundTrip方法关闭。

#### func [ReadRequest](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "549" \o "View Source)

func ReadRequest(b \*[bufio](http://godoc.org/bufio).[Reader](http://godoc.org/bufio" \l "Reader)) (req \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request), err [error](http://godoc.org/builtin" \l "error))

ReadRequest从b读取并解析出一个HTTP请求。（本函数主要用在服务端从下层获取请求）

#### func (\*Request) [ProtoAtLeast](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "232" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) ProtoAtLeast(major, minor [int](http://godoc.org/builtin" \l "int)) [bool](http://godoc.org/builtin" \l "bool)

ProtoAtLeast报告该请求使用的HTTP协议版本至少是major.minor。

#### func (\*Request) [UserAgent](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "238" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) UserAgent() [string](http://godoc.org/builtin" \l "string)

UserAgent返回请求中的客户端用户代理信息（请求的User-Agent头）。

#### func (\*Request) [Referer](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "279" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) Referer() [string](http://godoc.org/builtin" \l "string)

Referer返回请求中的访问来路信息。（请求的Referer头）

Referer在请求中就是拼错了的，这是HTTP早期就有的错误。该值也可以从用Header["Referer"]获取； 让获取Referer字段变成方法的好处是，编译器可以诊断使用正确单词拼法的req.Referrer()的程序，但却不能诊断使用Header["Referrer"]的程序。

#### func (\*Request) [AddCookie](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "262" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) AddCookie(c \*[Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie))

AddCookie向请求中添加一个cookie。按照[RFC 6265](http://tools.ietf.org/html/rfc6265) section 5.4的跪地，AddCookie不会添加超过一个Cookie头字段。这表示所有的cookie都写在同一行，用分号分隔（cookie内部用逗号分隔属性）。

#### func (\*Request) [SetBasicAuth](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "517" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) SetBasicAuth(username, password [string](http://godoc.org/builtin" \l "string))

SetBasicAuth使用提供的用户名和密码，采用HTTP基本认证，设置请求的Authorization头。HTTP基本认证会明码传送用户名和密码。

#### func (\*Request) [Write](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "350" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) Write(w [io](http://godoc.org/io).[Writer](http://godoc.org/io" \l "Writer)) [error](http://godoc.org/builtin" \l "error)

Write方法以有线格式将HTTP/1.1请求写入w（用于将请求写入下层TCPConn等）。本方法会考虑请求的如下字段：

Host

URL

Method (defaults to "GET")

Header

ContentLength

TransferEncoding

Body

如果存在Body，ContentLength字段<= 0且TransferEncoding字段未显式设置为["identity"]，Write方法会显式添加"Transfer-Encoding: chunked"到请求的头域。Body字段会在发送完请求后关闭。

#### func (\*Request) [WriteProxy](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "360" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) WriteProxy(w [io](http://godoc.org/io).[Writer](http://godoc.org/io" \l "Writer)) [error](http://godoc.org/builtin" \l "error)

WriteProxy类似Write但会将请求以HTTP代理期望的格式发送。

尤其是，按照[RFC 2616](http://tools.ietf.org/html/rfc2616) Section 5.1.2，WriteProxy会使用绝对URI（包括协议和主机名）来初始化请求的第1行（Request-URI行）。无论何种情况，WriteProxy都会使用r.Host或r.URL.Host设置Host头。

#### func (\*Request) [Cookies](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "243" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) Cookies() []\*[Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie)

Cookies解析并返回该请求的Cookie头设置的cookie。

#### func (\*Request) [Cookie](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "251" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) Cookie(name [string](http://godoc.org/builtin" \l "string)) (\*[Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie), [error](http://godoc.org/builtin" \l "error))

Cookie返回请求中名为name的cookie，如果未找到该cookie会返回nil, ErrNoCookie。

#### func (\*Request) [ParseForm](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "736" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) ParseForm() [error](http://godoc.org/builtin" \l "error)

ParseForm解析URL中的查询字符串，并将解析结果更新到r.Form字段。

对于POST或PUT请求，ParseForm还会将body当作表单解析，并将结果既更新到r.PostForm也更新到r.Form。解析结果中，POST或PUT请求主体要优先于URL查询字符串（同名变量，主体的值在查询字符串的值前面）。

如果请求的主体的大小没有被MaxBytesReader函数设定限制，其大小默认限制为开头10MB。

ParseMultipartForm会自动调用ParseForm。重复调用本方法是无意义的。

#### func (\*Request) [ParseMultipartForm](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "777" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) ParseMultipartForm(maxMemory [int64](http://godoc.org/builtin" \l "int64)) [error](http://godoc.org/builtin" \l "error)

ParseMultipartForm将请求的主体作为multipart/form-data解析。请求的整个主体都会被解析，得到的文件记录最多maxMemery字节保存在内存，其余部分保存在硬盘的temp文件里。如果必要，ParseMultipartForm会自行调用ParseForm。重复调用本方法是无意义的。

#### func (\*Request) [FormValue](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "812" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) FormValue(key [string](http://godoc.org/builtin" \l "string)) [string](http://godoc.org/builtin" \l "string)

FormValue返回key为键查询r.Form字段得到结果[]string切片的第一个值。POST和PUT主体中的同名参数优先于URL查询字符串。如果必要，本函数会隐式调用ParseMultipartForm和ParseForm。

#### func (\*Request) [PostFormValue](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "825" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) PostFormValue(key [string](http://godoc.org/builtin" \l "string)) [string](http://godoc.org/builtin" \l "string)

PostFormValue返回key为键查询r.PostForm字段得到结果[]string切片的第一个值。如果必要，本函数会隐式调用ParseMultipartForm和ParseForm。

#### func (\*Request) [FormFile](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "837" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) FormFile(key [string](http://godoc.org/builtin" \l "string)) ([multipart](http://godoc.org/mime/multipart).[File](http://godoc.org/mime/multipart" \l "File), \*[multipart](http://godoc.org/mime/multipart).[FileHeader](http://godoc.org/mime/multipart" \l "FileHeader), [error](http://godoc.org/builtin" \l "error))

FormFile返回以key为键查询r.MultipartForm字段得到结果中的第一个文件和它的信息。如果必要，本函数会隐式调用ParseMultipartForm和ParseForm。查询失败会返回ErrMissingFile错误。

#### func (\*Request) [MultipartReader](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "295" \o "View Source)

func (r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) MultipartReader() (\*[multipart](http://godoc.org/mime/multipart).[Reader](http://godoc.org/mime/multipart" \l "Reader), [error](http://godoc.org/builtin" \l "error))

如果请求是multipart/form-data POST请求，MultipartReader返回一个multipart.Reader接口，否则返回nil和一个错误。使用本函数代替ParseMultipartForm，可以将r.Body作为流处理。

### type [Response](https://github.com/golang/go/blob/master/src/net/http/response.go?name=release" \l "29" \o "View Source)

type Response struct {

Status [string](http://godoc.org/builtin" \l "string) // 例如"200 OK"

StatusCode [int](http://godoc.org/builtin" \l "int) // 例如200

Proto [string](http://godoc.org/builtin" \l "string) // 例如"HTTP/1.0"

ProtoMajor [int](http://godoc.org/builtin" \l "int) // 例如1

ProtoMinor [int](http://godoc.org/builtin" \l "int) // 例如0

// Header保管头域的键值对。

// 如果回复中有多个头的键相同，Header中保存为该键对应用逗号分隔串联起来的这些头的值

// （参见RFC 2616 Section 4.2）

// 被本结构体中的其他字段复制保管的头（如ContentLength）会从Header中删掉。

//

// Header中的键都是规范化的，参见CanonicalHeaderKey函数

Header [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)

// Body代表回复的主体。

// Client类型和Transport类型会保证Body字段总是非nil的，即使回复没有主体或主体长度为0。

// 关闭主体是调用者的责任。

// 如果服务端采用"chunked"传输编码发送的回复，Body字段会自动进行解码。

Body [io](http://godoc.org/io).[ReadCloser](http://godoc.org/io" \l "ReadCloser)

// ContentLength记录相关内容的长度。

// 其值为-1表示长度未知（采用chunked传输编码）

// 除非对应的Request.Method是"HEAD"，其值>=0表示可以从Body读取的字节数

ContentLength [int64](http://godoc.org/builtin" \l "int64)

// TransferEncoding按从最外到最里的顺序列出传输编码，空切片表示"identity"编码。

TransferEncoding [][string](http://godoc.org/builtin" \l "string)

// Close记录头域是否指定应在读取完主体后关闭连接。（即Connection头）

// 该值是给客户端的建议，Response.Write方法的ReadResponse函数都不会关闭连接。

Close [bool](http://godoc.org/builtin" \l "bool)

// Trailer字段保存和头域相同格式的trailer键值对，和Header字段相同类型

Trailer [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)

// Request是用来获取此回复的请求

// Request的Body字段是nil（因为已经被用掉了）

// 这个字段是被Client类型发出请求并获得回复后填充的

Request \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)

// TLS包含接收到该回复的TLS连接的信息。 对未加密的回复，本字段为nil。

// 返回的指针是被（同一TLS连接接收到的）回复共享的，不应被修改。

TLS \*[tls](http://godoc.org/crypto/tls).[ConnectionState](http://godoc.org/crypto/tls" \l "ConnectionState)

}

Response代表一个HTTP请求的回复。

#### func [ReadResponse](https://github.com/golang/go/blob/master/src/net/http/response.go?name=release" \l "116" \o "View Source)

func ReadResponse(r \*[bufio](http://godoc.org/bufio).[Reader](http://godoc.org/bufio" \l "Reader), req \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) (\*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), [error](http://godoc.org/builtin" \l "error))

ReadResponse从r读取并返回一个HTTP 回复。req参数是可选的，指定该回复对应的请求（即是对该请求的回复）。如果是nil，将假设请求是GET请求。客户端必须在结束resp.Body的读取后关闭它。读取完毕并关闭后，客户端可以检查resp.Trailer字段获取回复的trailer的键值对。（本函数主要用在客户端从下层获取回复）

#### func (\*Response) [ProtoAtLeast](https://github.com/golang/go/blob/master/src/net/http/response.go?name=release" \l "184" \o "View Source)

func (r \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response)) ProtoAtLeast(major, minor [int](http://godoc.org/builtin" \l "int)) [bool](http://godoc.org/builtin" \l "bool)

ProtoAtLeast报告该回复使用的HTTP协议版本至少是major.minor。

#### func (\*Response) [Cookies](https://github.com/golang/go/blob/master/src/net/http/response.go?name=release" \l "89" \o "View Source)

func (r \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response)) Cookies() []\*[Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie)

Cookies解析并返回该回复中的Set-Cookie头设置的cookie。

#### func (\*Response) [Location](https://github.com/golang/go/blob/master/src/net/http/response.go?name=release" \l "99" \o "View Source)

func (r \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response)) Location() (\*[url](http://godoc.org/net/url).[URL](http://godoc.org/net/url" \l "URL), [error](http://godoc.org/builtin" \l "error))

Location返回该回复的Location头设置的URL。相对地址的重定向会相对于该回复对应的请求来确定绝对地址。如果回复中没有Location头，会返回nil, ErrNoLocation。

#### func (\*Response) [Write](https://github.com/golang/go/blob/master/src/net/http/response.go?name=release" \l "203" \o "View Source)

func (r \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response)) Write(w [io](http://godoc.org/io).[Writer](http://godoc.org/io" \l "Writer)) [error](http://godoc.org/builtin" \l "error)

Write以有线格式将回复写入w（用于将回复写入下层TCPConn等）。本方法会考虑如下字段：

StatusCode

ProtoMajor

ProtoMinor

Request.Method

TransferEncoding

Trailer

Body

ContentLength

Header（不规范的键名和它对应的值会导致不可预知的行为）

Body字段在发送完回复后会被关闭。

### type [ResponseWriter](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "51" \o "View Source)

type ResponseWriter interface {

// Header返回一个Header类型值，该值会被WriteHeader方法发送。

// 在调用WriteHeader或Write方法后再改变该对象是没有意义的。

Header() [Header](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Header)

// WriteHeader该方法发送HTTP回复的头域和状态码。

// 如果没有被显式调用，第一次调用Write时会触发隐式调用WriteHeader(http.StatusOK)

// WriterHeader的显式调用主要用于发送错误码。

WriteHeader([int](http://godoc.org/builtin" \l "int))

// Write向连接中写入作为HTTP的一部分回复的数据。

// 如果被调用时还未调用WriteHeader，本方法会先调用WriteHeader(http.StatusOK)

// 如果Header中没有"Content-Type"键，

// 本方法会使用包函数DetectContentType检查数据的前512字节，将返回值作为该键的值。

Write([][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

}

ResponseWriter接口被HTTP处理器用于构造HTTP回复。

### type [Flusher](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "79" \o "View Source)

type Flusher interface {

// Flush将缓冲中的所有数据发送到客户端

Flush()

}

HTTP处理器ResponseWriter接口参数的下层如果实现了Flusher接口，可以让HTTP处理器将缓冲中的数据发送到客户端。

注意：即使ResponseWriter接口的下层支持Flush方法，如果客户端是通过HTTP代理连接的，缓冲中的数据也可能直到回复完毕才被传输到客户端。

### type [CloseNotifier](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "100" \o "View Source)

type CloseNotifier interface {

// CloseNotify返回一个通道，该通道会在客户端连接丢失时接收到唯一的值

CloseNotify() <-chan [bool](http://godoc.org/builtin" \l "bool)

}

HTTP处理器ResponseWriter接口参数的下层如果实现了CloseNotifier接口，可以让用户检测下层的连接是否停止。如果客户端在回复准备好之前关闭了连接，该机制可以用于取消服务端耗时较长的操作。

### type [Hijacker](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "86" \o "View Source)

type Hijacker interface {

// Hijack让调用者接管连接，返回连接和关联到该连接的一个缓冲读写器。

// 调用本方法后，HTTP服务端将不再对连接进行任何操作，

// 调用者有责任管理、关闭返回的连接。

Hijack() ([net](http://godoc.org/net).[Conn](http://godoc.org/net" \l "Conn), \*[bufio](http://godoc.org/bufio).[ReadWriter](http://godoc.org/bufio" \l "ReadWriter), [error](http://godoc.org/builtin" \l "error))

}

HTTP处理器ResponseWriter接口参数的下层如果实现了Hijacker接口，可以让HTTP处理器接管该连接。

Example

### type [RoundTripper](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "82" \o "View Source)

type RoundTripper interface {

// RoundTrip执行单次HTTP事务，接收并发挥请求req的回复。

// RoundTrip不应试图解析/修改得到的回复。

// 尤其要注意，只要RoundTrip获得了一个回复，不管该回复的HTTP状态码如何，

// 它必须将返回值err设置为nil。

// 非nil的返回值err应该留给获取回复失败的情况。

// 类似的，RoundTrip不能试图管理高层次的细节，如重定向、认证、cookie。

//

// 除了从请求的主体读取并关闭主体之外，RoundTrip不应修改请求，包括（请求的）错误。

// RoundTrip函数接收的请求的URL和Header字段可以保证是（被）初始化了的。

RoundTrip(\*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) (\*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), [error](http://godoc.org/builtin" \l "error))

}

RoundTripper接口是具有执行单次HTTP事务的能力（接收指定请求的回复）的接口。

RoundTripper接口的类型必须可以安全的被多线程同时使用。

### type [Transport](https://github.com/golang/go/blob/master/src/net/http/transport.go?name=release" \l "49" \o "View Source)

type Transport struct {

// Proxy指定一个对给定请求返回代理的函数。

// 如果该函数返回了非nil的错误值，请求的执行就会中断并返回该错误。

// 如果Proxy为nil或返回nil的\*URL置，将不使用代理。

Proxy func(\*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) (\*[url](http://godoc.org/net/url).[URL](http://godoc.org/net/url" \l "URL), [error](http://godoc.org/builtin" \l "error))

// Dial指定创建TCP连接的拨号函数。如果Dial为nil，会使用net.Dial。

Dial func(network, addr [string](http://godoc.org/builtin" \l "string)) ([net](http://godoc.org/net).[Conn](http://godoc.org/net" \l "Conn), [error](http://godoc.org/builtin" \l "error))

// TLSClientConfig指定用于tls.Client的TLS配置信息。

// 如果该字段为nil，会使用默认的配置信息。

TLSClientConfig \*[tls](http://godoc.org/crypto/tls).[Config](http://godoc.org/crypto/tls" \l "Config)

// TLSHandshakeTimeout指定等待TLS握手完成的最长时间。零值表示不设置超时。

TLSHandshakeTimeout [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration)

// 如果DisableKeepAlives为真，会禁止不同HTTP请求之间TCP连接的重用。

DisableKeepAlives [bool](http://godoc.org/builtin" \l "bool)

// 如果DisableCompression为真，会禁止Transport在请求中没有Accept-Encoding头时，

// 主动添加"Accept-Encoding: gzip"头，以获取压缩数据。

// 如果Transport自己请求gzip并得到了压缩后的回复，它会主动解压缩回复的主体。

// 但如果用户显式的请求gzip压缩数据，Transport是不会主动解压缩的。

DisableCompression [bool](http://godoc.org/builtin" \l "bool)

// 如果MaxIdleConnsPerHost!=0，会控制每个主机下的最大闲置连接。

// 如果MaxIdleConnsPerHost==0，会使用DefaultMaxIdleConnsPerHost。

MaxIdleConnsPerHost [int](http://godoc.org/builtin" \l "int)

// ResponseHeaderTimeout指定在发送完请求（包括其可能的主体）之后，

// 等待接收服务端的回复的头域的最大时间。零值表示不设置超时。

// 该时间不包括获取回复主体的时间。

ResponseHeaderTimeout [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration)

// 内含隐藏或非导出字段

}

Transport类型实现了RoundTripper接口，支持http、https和http/https代理。Transport类型可以缓存连接以在未来重用。

var DefaultTransport [RoundTripper](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "RoundTripper) = &[Transport](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport){

Proxy: [ProxyFromEnvironment](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ProxyFromEnvironment),

Dial: (&[net](http://godoc.org/net).[Dialer](http://godoc.org/net" \l "Dialer){

Timeout: 30 \* [time](http://godoc.org/time).[Second](http://godoc.org/time" \l "Second),

KeepAlive: 30 \* [time](http://godoc.org/time).[Second](http://godoc.org/time" \l "Second),

}).Dial,

TLSHandshakeTimeout: 10 \* [time](http://godoc.org/time).[Second](http://godoc.org/time" \l "Second),

}

DefaultTransport是被包变量DefaultClient使用的默认RoundTripper接口。它会根据需要创建网络连接，并缓存以便在之后的请求中重用这些连接。它使用环境变量$HTTP\_PROXY和$NO\_PROXY（或$http\_proxy和$no\_proxy）指定的HTTP代理。

#### func (\*Transport) [RegisterProtocol](https://github.com/golang/go/blob/master/src/net/http/transport.go?name=release" \l "217" \o "View Source)

func (t \*[Transport](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport)) RegisterProtocol(scheme [string](http://godoc.org/builtin" \l "string), rt [RoundTripper](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "RoundTripper))

RegisterProtocol注册一个新的名为scheme的协议。t会将使用scheme协议的请求转交给rt。rt有责任模拟HTTP请求的语义。

RegisterProtocol可以被其他包用于提供"ftp"或"file"等协议的实现。

#### func (\*Transport) [RoundTrip](https://github.com/golang/go/blob/master/src/net/http/transport.go?name=release" \l "164" \o "View Source)

func (t \*[Transport](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport)) RoundTrip(req \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

RoundTrip方法实现了RoundTripper接口。

高层次的HTTP客户端支持（如管理cookie和重定向）请参见Get、Post等函数和Client类型。

#### func (\*Transport) [CloseIdleConnections](https://github.com/golang/go/blob/master/src/net/http/transport.go?name=release" \l "236" \o "View Source)

func (t \*[Transport](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport)) CloseIdleConnections()

CloseIdleConnections关闭所有之前的请求建立但目前处于闲置状态的连接。本方法不会中断正在使用的连接。

#### func (\*Transport) [CancelRequest](https://github.com/golang/go/blob/master/src/net/http/transport.go?name=release" \l "251" \o "View Source)

func (t \*[Transport](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Transport)) CancelRequest(req \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request))

CancelRequest通过关闭请求所在的连接取消一个执行中的请求。

### type [Client](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "35" \o "View Source)

type Client struct {

// Transport指定执行独立、单次HTTP请求的机制。

// 如果Transport为nil，则使用DefaultTransport。

Transport [RoundTripper](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "RoundTripper)

// CheckRedirect指定处理重定向的策略。

// 如果CheckRedirect不为nil，客户端会在执行重定向之前调用本函数字段。

// 参数req和via是将要执行的请求和已经执行的请求（切片，越新的请求越靠后）。

// 如果CheckRedirect返回一个错误，本类型的Get方法不会发送请求req，

// 而是返回之前得到的最后一个回复和该错误。（包装进url.Error类型里）

//

// 如果CheckRedirect为nil，会采用默认策略：连续10此请求后停止。

CheckRedirect func(req \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request), via []\*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) [error](http://godoc.org/builtin" \l "error)

// Jar指定cookie管理器。

// 如果Jar为nil，请求中不会发送cookie，回复中的cookie会被忽略。

Jar [CookieJar](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "CookieJar)

// Timeout指定本类型的值执行请求的时间限制。

// 该超时限制包括连接时间、重定向和读取回复主体的时间。

// 计时器会在Head、Get、Post或Do方法返回后继续运作并在超时后中断回复主体的读取。

//

// Timeout为零值表示不设置超时。

//

// Client实例的Transport字段必须支持CancelRequest方法，

// 否则Client会在试图用Head、Get、Post或Do方法执行请求时返回错误。

// 本类型的Transport字段默认值（DefaultTransport）支持CancelRequest方法。

Timeout [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration)

}

Client类型代表HTTP客户端。它的零值（DefaultClient）是一个可用的使用DefaultTransport的客户端。

Client的Transport字段一般会含有内部状态（缓存TCP连接），因此Client类型值应尽量被重用而不是每次需要都创建新的。Client类型值可以安全的被多个go程同时使用。

Client类型的层次比RoundTripper接口（如Transport）高，还会管理HTTP的cookie和重定向等细节。

#### func (\*Client) [Do](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "148" \o "View Source)

func (c \*[Client](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client)) Do(req \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

Do方法发送请求，返回HTTP回复。它会遵守客户端c设置的策略（如重定向、cookie、认证）。

如果客户端的策略（如重定向）返回错误或存在HTTP协议错误时，本方法将返回该错误；如果回应的状态码不是2xx，本方法并不会返回错误。

如果返回值err为nil，resp.Body总是非nil的，调用者应该在读取完resp.Body后关闭它。如果返回值resp的主体未关闭，c下层的RoundTripper接口（一般为Transport类型）可能无法重用resp主体下层保持的TCP连接去执行之后的请求。

请求的主体，如果非nil，会在执行后被c.Transport关闭，即使出现错误。

一般应使用Get、Post或PostForm方法代替Do方法。

#### func (\*Client) [Head](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "462" \o "View Source)

func (c \*[Client](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client)) Head(url [string](http://godoc.org/builtin" \l "string)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

Head向指定的URL发出一个HEAD请求，如果回应的状态码如下，Head会在调用c.CheckRedirect后执行重定向：

301 (Moved Permanently)

302 (Found)

303 (See Other)

307 (Temporary Redirect)

#### func (\*Client) [Get](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "270" \o "View Source)

func (c \*[Client](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client)) Get(url [string](http://godoc.org/builtin" \l "string)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

Get向指定的URL发出一个GET请求，如果回应的状态码如下，Get会在调用c.CheckRedirect后执行重定向：

301 (Moved Permanently)

302 (Found)

303 (See Other)

307 (Temporary Redirect)

如果c.CheckRedirect执行失败或存在HTTP协议错误时，本方法将返回该错误；如果回应的状态码不是2xx，本方法并不会返回错误。如果返回值err为nil，resp.Body总是非nil的，调用者应该在读取完resp.Body后关闭它。

#### func (\*Client) [Post](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "411" \o "View Source)

func (c \*[Client](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client)) Post(url [string](http://godoc.org/builtin" \l "string), bodyType [string](http://godoc.org/builtin" \l "string), body [io](http://godoc.org/io).[Reader](http://godoc.org/io" \l "Reader)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

Post向指定的URL发出一个POST请求。bodyType为POST数据的类型， body为POST数据，作为请求的主体。如果参数body实现了io.Closer接口，它会在发送请求后被关闭。调用者有责任在读取完返回值resp的主体后关闭它。

#### func (\*Client) [PostForm](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "436" \o "View Source)

func (c \*[Client](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Client)) PostForm(url [string](http://godoc.org/builtin" \l "string), data [url](http://godoc.org/net/url).[Values](http://godoc.org/net/url" \l "Values)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

PostForm向指定的URL发出一个POST请求，url.Values类型的data会被编码为请求的主体。POST数据的类型一般会设为"application/x-www-form-urlencoded"。如果返回值err为nil，resp.Body总是非nil的，调用者应该在读取完resp.Body后关闭它。

### type [Handler](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "45" \o "View Source)

type Handler interface {

ServeHTTP([ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request))

}

实现了Handler接口的对象可以注册到HTTP服务端，为特定的路径及其子树提供服务。

ServeHTTP应该将回复的头域和数据写入ResponseWriter接口然后返回。返回标志着该请求已经结束，HTTP服务端可以转移向该连接上的下一个请求。

#### func [NotFoundHandler](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1253" \o "View Source)

func NotFoundHandler() [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)

NotFoundHandler返回一个简单的请求处理器，该处理器会对每个请求都回复"404 page not found"。

#### func [RedirectHandler](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1360" \o "View Source)

func RedirectHandler(url [string](http://godoc.org/builtin" \l "string), code [int](http://godoc.org/builtin" \l "int)) [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)

RedirectHandler返回一个请求处理器，该处理器会对每个请求都使用状态码code重定向到网址url。

#### func [TimeoutHandler](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1860" \o "View Source)

func TimeoutHandler(h [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler), dt [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration), msg [string](http://godoc.org/builtin" \l "string)) [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)

TimeoutHandler返回一个采用指定时间限制的请求处理器。

返回的Handler会调用h.ServeHTTP去处理每个请求，但如果某一次调用耗时超过了时间限制，该处理器会回复请求状态码503 Service Unavailable，并将msg作为回复的主体（如果msg为空字符串，将发送一个合理的默认信息）。在超时后，h对它的ResponseWriter接口参数的写入操作会返回ErrHandlerTimeout。

#### func [StripPrefix](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1260" \o "View Source)

func StripPrefix(prefix [string](http://godoc.org/builtin" \l "string), h [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)) [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)

StripPrefix返回一个处理器，该处理器会将请求的URL.Path字段中给定前缀prefix去除后再交由h处理。StripPrefix会向URL.Path字段中没有给定前缀的请求回复404 page not found。

Example

### type [HandlerFunc](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1231" \o "View Source)

type HandlerFunc func([ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request))

HandlerFunc type是一个适配器，通过类型转换让我们可以将普通的函数作为HTTP处理器使用。如果f是一个具有适当签名的函数，HandlerFunc(f)通过调用f实现了Handler接口。

#### func (HandlerFunc) [ServeHTTP](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1234" \o "View Source)

func (f [HandlerFunc](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "HandlerFunc)) ServeHTTP(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request))

ServeHTTP方法会调用f(w, r)

### type [ServeMux](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1391" \o "View Source)

type ServeMux struct {

// 内含隐藏或非导出字段

}

ServeMux类型是HTTP请求的多路转接器。它会将每一个接收的请求的URL与一个注册模式的列表进行匹配，并调用和URL最匹配的模式的处理器。

模式是固定的、由根开始的路径，如"/favicon.ico"，或由根开始的子树，如"/images/"（注意结尾的斜杠）。较长的模式优先于较短的模式，因此如果模式"/images/"和"/images/thumbnails/"都注册了处理器，后一个处理器会用于路径以"/images/thumbnails/"开始的请求，前一个处理器会接收到其余的路径在"/images/"子树下的请求。

注意，因为以斜杠结尾的模式代表一个由根开始的子树，模式"/"会匹配所有的未被其他注册的模式匹配的路径，而不仅仅是路径"/"。

模式也能（可选地）以主机名开始，表示只匹配该主机上的路径。指定主机的模式优先于一般的模式，因此一个注册了两个模式"/codesearch"和"codesearch.google.com/"的处理器不会接管目标为"<http://www.google.com/>"的请求。

ServeMux还会注意到请求的URL路径的无害化，将任何路径中包含"."或".."元素的请求重定向到等价的没有这两种元素的URL。（参见path.Clean函数）

#### func [NewServeMux](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1404" \o "View Source)

func NewServeMux() \*[ServeMux](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux)

NewServeMux创建并返回一个新的\*ServeMux

#### func (\*ServeMux) [Handle](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1516" \o "View Source)

func (mux \*[ServeMux](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux)) Handle(pattern [string](http://godoc.org/builtin" \l "string), handler [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler))

Handle注册HTTP处理器handler和对应的模式pattern。如果该模式已经注册有一个处理器，Handle会panic。

Example

#### func (\*ServeMux) [HandleFunc](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1554" \o "View Source)

func (mux \*[ServeMux](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux)) HandleFunc(pattern [string](http://godoc.org/builtin" \l "string), handler func([ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)))

HandleFunc注册一个处理器函数handler和对应的模式pattern。

#### func (\*ServeMux) [Handler](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1468" \o "View Source)

func (mux \*[ServeMux](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux)) Handler(r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) (h [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler), pattern [string](http://godoc.org/builtin" \l "string))

Handler根据r.Method、r.Host和r.URL.Path等数据，返回将用于处理该请求的HTTP处理器。它总是返回一个非nil的处理器。如果路径不是它的规范格式，将返回内建的用于重定向到等价的规范路径的处理器。

Handler也会返回匹配该请求的的已注册模式；在内建重定向处理器的情况下，pattern会在重定向后进行匹配。如果没有已注册模式可以应用于该请求，本方法将返回一个内建的"404 page not found"处理器和一个空字符串模式。

#### func (\*ServeMux) [ServeHTTP](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1502" \o "View Source)

func (mux \*[ServeMux](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ServeMux)) ServeHTTP(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request))

ServeHTTP将请求派遣到与请求的URL最匹配的模式对应的处理器。

### type [Server](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1581" \o "View Source)

type Server struct {

Addr [string](http://godoc.org/builtin" \l "string) // 监听的TCP地址，如果为空字符串会使用":http"

Handler [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler) // 调用的处理器，如为nil会调用http.DefaultServeMux

ReadTimeout [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration) // 请求的读取操作在超时前的最大持续时间

WriteTimeout [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration) // 回复的写入操作在超时前的最大持续时间

MaxHeaderBytes [int](http://godoc.org/builtin" \l "int) // 请求的头域最大长度，如为0则用DefaultMaxHeaderBytes

TLSConfig \*[tls](http://godoc.org/crypto/tls).[Config](http://godoc.org/crypto/tls" \l "Config) // 可选的TLS配置，用于ListenAndServeTLS方法

// TLSNextProto（可选地）指定一个函数来在一个NPN型协议升级出现时接管TLS连接的所有权。

// 映射的键为商谈的协议名；映射的值为函数，该函数的Handler参数应处理HTTP请求，

// 并且初始化Handler.ServeHTTP的\*Request参数的TLS和RemoteAddr字段（如果未设置）。

// 连接在函数返回时会自动关闭。

TLSNextProto map[[string](http://godoc.org/builtin" \l "string)]func(\*[Server](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server), \*[tls](http://godoc.org/crypto/tls).[Conn](http://godoc.org/crypto/tls" \l "Conn), [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler))

// ConnState字段指定一个可选的回调函数，该函数会在一个与客户端的连接改变状态时被调用。

// 参见ConnState类型和相关常数获取细节。

ConnState func([net](http://godoc.org/net).[Conn](http://godoc.org/net" \l "Conn), [ConnState](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ConnState))

// ErrorLog指定一个可选的日志记录器，用于记录接收连接时的错误和处理器不正常的行为。

// 如果本字段为nil，日志会通过log包的标准日志记录器写入os.Stderr。

ErrorLog \*[log](http://godoc.org/log).[Logger](http://godoc.org/log" \l "Logger)

// 内含隐藏或非导出字段

}

Server类型定义了运行HTTP服务端的参数。Server的零值是合法的配置。

#### func (\*Server) [SetKeepAlivesEnabled](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1733" \o "View Source)

func (s \*[Server](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server)) SetKeepAlivesEnabled(v [bool](http://godoc.org/builtin" \l "bool))

SetKeepAlivesEnabled控制是否允许HTTP闲置连接重用（keep-alive）功能。默认该功能总是被启用的。只有资源非常紧张的环境或者服务端在关闭进程中时，才应该关闭该功能。

#### func (\*Server) [Serve](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1694" \o "View Source)

func (srv \*[Server](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server)) Serve(l [net](http://godoc.org/net).[Listener](http://godoc.org/net" \l "Listener)) [error](http://godoc.org/builtin" \l "error)

Serve会接手监听器l收到的每一个连接，并为每一个连接创建一个新的服务go程。该go程会读取请求，然后调用srv.Handler回复请求。

#### func (\*Server) [ListenAndServe](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1679" \o "View Source)

func (srv \*[Server](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server)) ListenAndServe() [error](http://godoc.org/builtin" \l "error)

ListenAndServe监听srv.Addr指定的TCP地址，并且会调用Serve方法接收到的连接。如果srv.Addr为空字符串，会使用":http"。

#### func (\*Server) [ListenAndServeTLS](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1823" \o "View Source)

func (srv \*[Server](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Server)) ListenAndServeTLS(certFile, keyFile [string](http://godoc.org/builtin" \l "string)) [error](http://godoc.org/builtin" \l "error)

ListenAndServeTLS监听srv.Addr确定的TCP地址，并且会调用Serve方法处理接收到的连接。必须提供证书文件和对应的私钥文件。如果证书是由权威机构签发的，certFile参数必须是顺序串联的服务端证书和CA证书。如果srv.Addr为空字符串，会使用":https"。

### type [File](https://github.com/golang/go/blob/master/src/net/http/fs.go?name=release" \l "58" \o "View Source)

type File interface {

[io](http://godoc.org/io).[Closer](http://godoc.org/io" \l "Closer)

[io](http://godoc.org/io).[Reader](http://godoc.org/io" \l "Reader)

Readdir(count [int](http://godoc.org/builtin" \l "int)) ([][os](http://godoc.org/os).[FileInfo](http://godoc.org/os" \l "FileInfo), [error](http://godoc.org/builtin" \l "error))

Seek(offset [int64](http://godoc.org/builtin" \l "int64), whence [int](http://godoc.org/builtin" \l "int)) ([int64](http://godoc.org/builtin" \l "int64), [error](http://godoc.org/builtin" \l "error))

Stat() ([os](http://godoc.org/os).[FileInfo](http://godoc.org/os" \l "FileInfo), [error](http://godoc.org/builtin" \l "error))

}

File是被FileSystem接口的Open方法返回的接口类型，可以被FileServer等函数用于文件访问服务。

该接口的方法的行为应该和\*os.File类型的同名方法相同。

### type [FileSystem](https://github.com/golang/go/blob/master/src/net/http/fs.go?name=release" \l "50" \o "View Source)

type FileSystem interface {

Open(name [string](http://godoc.org/builtin" \l "string)) ([File](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "File), [error](http://godoc.org/builtin" \l "error))

}

FileSystem接口实现了对一系列命名文件的访问。文件路径的分隔符为'/'，不管主机操作系统的惯例如何。

### type [Dir](https://github.com/golang/go/blob/master/src/net/http/fs.go?name=release" \l "29" \o "View Source)

type Dir [string](http://godoc.org/builtin" \l "string)

Dir使用限制到指定目录树的本地文件系统实现了http.FileSystem接口。空Dir被视为"."，即代表当前目录。

#### func (Dir) [Open](https://github.com/golang/go/blob/master/src/net/http/fs.go?name=release" \l "31" \o "View Source)

func (d [Dir](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Dir)) Open(name [string](http://godoc.org/builtin" \l "string)) ([File](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "File), [error](http://godoc.org/builtin" \l "error))

### func [NewFileTransport](https://github.com/golang/go/blob/master/src/net/http/filetransport.go?name=release" \l "30" \o "View Source)

func NewFileTransport(fs [FileSystem](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "FileSystem)) [RoundTripper](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "RoundTripper)

NewFileTransport返回一个RoundTripper接口，使用FileSystem接口fs提供文件访问服务。 返回的RoundTripper接口会忽略接收的请求的URL主机及其他绝大多数属性。

NewFileTransport函数的典型使用情况是给Transport类型的值注册"file"协议，如下所示：

t := &http.Transport{}

t.RegisterProtocol("file", http.NewFileTransport(http.Dir("/")))

c := &http.Client{Transport: t}

res, err := c.Get("file:///etc/passwd")

...

### func [FileServer](https://github.com/golang/go/blob/master/src/net/http/fs.go?name=release" \l "435" \o "View Source)

func FileServer(root [FileSystem](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "FileSystem)) [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)

FileServer返回一个使用FileSystem接口root提供文件访问服务的HTTP处理器。要使用操作系统的FileSystem接口实现，可使用http.Dir：

http.Handle("/", http.FileServer(http.Dir("/tmp")))

Example

Example (StripPrefix)

### func [ProxyURL](https://github.com/golang/go/blob/master/src/net/http/transport.go?name=release" \l "140" \o "View Source)

func ProxyURL(fixedURL \*[url](http://godoc.org/net/url).[URL](http://godoc.org/net/url" \l "URL)) func(\*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) (\*[url](http://godoc.org/net/url).[URL](http://godoc.org/net/url" \l "URL), [error](http://godoc.org/builtin" \l "error))

ProxyURL返回一个代理函数（用于Transport类型），该函数总是返回同一个URL。

### func [ProxyFromEnvironment](https://github.com/golang/go/blob/master/src/net/http/transport.go?name=release" \l "115" \o "View Source)

func ProxyFromEnvironment(req \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)) (\*[url](http://godoc.org/net/url).[URL](http://godoc.org/net/url" \l "URL), [error](http://godoc.org/builtin" \l "error))

ProxyFromEnvironment使用环境变量$HTTP\_PROXY和$NO\_PROXY(或$http\_proxy和$no\_proxy)的配置返回用于req的代理。如果代理环境不合法将返回错误；如果环境未设定代理或者给定的request不应使用代理时，将返回(nil, nil)；如果req.URL.Host字段是"localhost"（可以有端口号，也可以没有），也会返回(nil, nil)。

### func [SetCookie](https://github.com/golang/go/blob/master/src/net/http/cookie.go?name=release" \l "129" \o "View Source)

func SetCookie(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), cookie \*[Cookie](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Cookie))

SetCookie在w的头域中添加Set-Cookie头，该HTTP头的值为cookie。

### func [Redirect](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1276" \o "View Source)

func Redirect(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request), urlStr [string](http://godoc.org/builtin" \l "string), code [int](http://godoc.org/builtin" \l "int))

Redirect回复请求一个重定向地址urlStr和状态码code。该重定向地址可以是相对于请求r的相对地址。

### func [NotFound](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1249" \o "View Source)

func NotFound(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request))

NotFound回复请求404状态码（not found：目标未发现）。

### func [Error](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1242" \o "View Source)

func Error(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), error [string](http://godoc.org/builtin" \l "string), code [int](http://godoc.org/builtin" \l "int))

Error使用指定的错误信息和状态码回复请求，将数据写入w。错误信息必须是明文。

### func [ServeContent](https://github.com/golang/go/blob/master/src/net/http/fs.go?name=release" \l "113" \o "View Source)

func ServeContent(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), req \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request), name [string](http://godoc.org/builtin" \l "string), modtime [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time), content [io](http://godoc.org/io).[ReadSeeker](http://godoc.org/io" \l "ReadSeeker))

ServeContent使用提供的ReadSeeker的内容回复请求。ServeContent比起io.Copy函数的主要优点，是可以处理范围类请求（只要一部分内容）、设置MIME类型，处理If-Modified-Since请求。

如果未设定回复的Content-Type头，本函数首先会尝试从name的文件扩展名推断数据类型；如果失败，会用读取content的第1块数据并提供给DetectContentType推断类型；之后会设置Content-Type头。参数name不会用于别的地方，甚至于它可以是空字符串，也永远不会发送到回复里。

如果modtime不是Time零值，函数会在回复的头域里设置Last-Modified头。如果请求的头域包含If-Modified-Since头，本函数会使用modtime参数来确定是否应该发送内容。如果调用者设置了w的ETag头，ServeContent会使用它处理包含If-Range头和If-None-Match头的请求。

参数content的Seek方法必须有效：函数使用Seek来确定它的大小。

注意：本包File接口和\*os.File类型都实现了io.ReadSeeker接口。

### func [ServeFile](https://github.com/golang/go/blob/master/src/net/http/fs.go?name=release" \l "419" \o "View Source)

func ServeFile(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), r \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request), name [string](http://godoc.org/builtin" \l "string))

ServeFile回复请求name指定的文件或者目录的内容。

### func [MaxBytesReader](https://github.com/golang/go/blob/master/src/net/http/request.go?name=release" \l "637" \o "View Source)

func MaxBytesReader(w [ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), r [io](http://godoc.org/io).[ReadCloser](http://godoc.org/io" \l "ReadCloser), n [int64](http://godoc.org/builtin" \l "int64)) [io](http://godoc.org/io).[ReadCloser](http://godoc.org/io" \l "ReadCloser)

MaxBytesReader类似io.LimitReader，但它是用来限制接收到的请求的Body的大小的。不同于io.LimitReader，本函数返回一个ReadCloser，返回值的Read方法在读取的数据超过大小限制时会返回非EOF错误，其Close方法会关闭下层的io.ReadCloser接口r。

MaxBytesReader预防客户端因为意外或者蓄意发送的“大”请求，以避免尺寸过大的请求浪费服务端资源。

### func [Head](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "450" \o "View Source)

func Head(url [string](http://godoc.org/builtin" \l "string)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

Head向指定的URL发出一个HEAD请求，如果回应的状态码如下，Head会在调用c.CheckRedirect后执行重定向：

301 (Moved Permanently)

302 (Found)

303 (See Other)

307 (Temporary Redirect)

Head是对包变量DefaultClient的Head方法的包装。

### func [Get](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "251" \o "View Source)

func Get(url [string](http://godoc.org/builtin" \l "string)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

Get向指定的URL发出一个GET请求，如果回应的状态码如下，Get会在调用c.CheckRedirect后执行重定向：

301 (Moved Permanently)

302 (Found)

303 (See Other)

307 (Temporary Redirect)

如果c.CheckRedirect执行失败或存在HTTP协议错误时，本方法将返回该错误；如果回应的状态码不是2xx，本方法并不会返回错误。如果返回值err为nil，resp.Body总是非nil的，调用者应该在读取完resp.Body后关闭它。

Get是对包变量DefaultClient的Get方法的包装。

Example

### func [Post](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "401" \o "View Source)

func Post(url [string](http://godoc.org/builtin" \l "string), bodyType [string](http://godoc.org/builtin" \l "string), body [io](http://godoc.org/io).[Reader](http://godoc.org/io" \l "Reader)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

Post向指定的URL发出一个POST请求。bodyType为POST数据的类型， body为POST数据，作为请求的主体。如果参数body实现了io.Closer接口，它会在发送请求后被关闭。调用者有责任在读取完返回值resp的主体后关闭它。

Post是对包变量DefaultClient的Post方法的包装。

### func [PostForm](https://github.com/golang/go/blob/master/src/net/http/client.go?name=release" \l "427" \o "View Source)

func PostForm(url [string](http://godoc.org/builtin" \l "string), data [url](http://godoc.org/net/url).[Values](http://godoc.org/net/url" \l "Values)) (resp \*[Response](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Response), err [error](http://godoc.org/builtin" \l "error))

PostForm向指定的URL发出一个POST请求，url.Values类型的data会被编码为请求的主体。如果返回值err为nil，resp.Body总是非nil的，调用者应该在读取完resp.Body后关闭它。

PostForm是对包变量DefaultClient的PostForm方法的包装。

### func [Handle](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1561" \o "View Source)

func Handle(pattern [string](http://godoc.org/builtin" \l "string), handler [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler))

Handle注册HTTP处理器handler和对应的模式pattern（注册到DefaultServeMux）。如果该模式已经注册有一个处理器，Handle会panic。ServeMux的文档解释了模式的匹配机制。

### func [HandleFunc](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1566" \o "View Source)

func HandleFunc(pattern [string](http://godoc.org/builtin" \l "string), handler func([ResponseWriter](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "ResponseWriter), \*[Request](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Request)))

HandleFunc注册一个处理器函数handler和对应的模式pattern（注册到DefaultServeMux）。ServeMux的文档解释了模式的匹配机制。

### func [Serve](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1574" \o "View Source)

func Serve(l [net](http://godoc.org/net).[Listener](http://godoc.org/net" \l "Listener), handler [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)) [error](http://godoc.org/builtin" \l "error)

Serve会接手监听器l收到的每一个连接，并为每一个连接创建一个新的服务go程。该go程会读取请求，然后调用handler回复请求。handler参数一般会设为nil，此时会使用DefaultServeMux。

### func [ListenAndServe](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1776" \o "View Source)

func ListenAndServe(addr [string](http://godoc.org/builtin" \l "string), handler [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)) [error](http://godoc.org/builtin" \l "error)

ListenAndServe监听TCP地址addr，并且会使用handler参数调用Serve函数处理接收到的连接。handler参数一般会设为nil，此时会使用DefaultServeMux。

一个简单的服务端例子：

package main

import (

"io"

"net/http"

"log"

)

// hello world, the web server

func HelloServer(w http.ResponseWriter, req \*http.Request) {

io.WriteString(w, "hello, world!\n")

}

func main() {

http.HandleFunc("/hello", HelloServer)

err := http.ListenAndServe(":12345", nil)

if err != nil {

log.Fatal("ListenAndServe: ", err)

}

}

### func [ListenAndServeTLS](https://github.com/golang/go/blob/master/src/net/http/server.go?name=release" \l "1809" \o "View Source)

func ListenAndServeTLS(addr [string](http://godoc.org/builtin" \l "string), certFile [string](http://godoc.org/builtin" \l "string), keyFile [string](http://godoc.org/builtin" \l "string), handler [Handler](https://studygolang.com/static/pkgdoc/pkg/net_http.htm" \l "Handler)) [error](http://godoc.org/builtin" \l "error)

ListenAndServeTLS函数和ListenAndServe函数的行为基本一致，除了它期望HTTPS连接之外。此外，必须提供证书文件和对应的私钥文件。如果证书是由权威机构签发的，certFile参数必须是顺序串联的服务端证书和CA证书。如果srv.Addr为空字符串，会使用":https"。

一个简单的服务端例子：

import (

"log"

"net/http"

)

func handler(w http.ResponseWriter, req \*http.Request) {

w.Header().Set("Content-Type", "text/plain")

w.Write([]byte("This is an example server.\n"))

}

func main() {

http.HandleFunc("/", handler)

log.Printf("About to listen on 10443. Go to <https://127.0.0.1:10443/>")

err := http.ListenAndServeTLS(":10443", "cert.pem", "key.pem", nil)

if err != nil {

log.Fatal(err)

}

}

程序员可以使用crypto/tls包的generate\_cert.go文件来生成cert.pem和key.pem两个文件。

## Net包

import "net"

net包提供了可移植的网络I/O接口，包括TCP/IP、UDP、域名解析和Unix域socket。

虽然本包提供了对网络原语的访问，大部分使用者只需要Dial、Listen和Accept函数提供的基本接口；以及相关的Conn和Listener接口。crypto/tls包提供了相同的接口和类似的Dial和Listen函数。

Dial函数和服务端建立连接：

conn, err := net.Dial("tcp", "google.com:80")

if err != nil {

// handle error

}

fmt.Fprintf(conn, "GET / HTTP/1.0\r\n\r\n")

status, err := bufio.NewReader(conn).ReadString('\n')

// ...

Listen函数创建的服务端：

ln, err := net.Listen("tcp", ":8080")

if err != nil {

// handle error

}

for {

conn, err := ln.Accept()

if err != nil {

// handle error

continue

}

go handleConnection(conn)

}

### Index

### [返回首页](https://studygolang.com/static/pkgdoc/main.html)

· [Constants](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "pkg-constants)

· [Variables](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "pkg-variables)

· [type ParseError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ParseError)

* [func (e \*ParseError) Error() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ParseError.Error)

· [type Error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Error)

· [type InvalidAddrError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InvalidAddrError)

* [func (e InvalidAddrError) Error() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InvalidAddrError.Error)
* [func (e InvalidAddrError) Temporary() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InvalidAddrError.Temporary)
* [func (e InvalidAddrError) Timeout() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InvalidAddrError.Timeout)

· [type UnknownNetworkError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnknownNetworkError)

* [func (e UnknownNetworkError) Error() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnknownNetworkError.Error)
* [func (e UnknownNetworkError) Temporary() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnknownNetworkError.Temporary)
* [func (e UnknownNetworkError) Timeout() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnknownNetworkError.Timeout)

· [type DNSConfigError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSConfigError)

* [func (e \*DNSConfigError) Error() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSConfigError.Error)
* [func (e \*DNSConfigError) Temporary() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSConfigError.Temporary)
* [func (e \*DNSConfigError) Timeout() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSConfigError.Timeout)

· [type DNSError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSError)

* [func (e \*DNSError) Error() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSError.Error)
* [func (e \*DNSError) Temporary() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSError.Temporary)
* [func (e \*DNSError) Timeout() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSError.Timeout)

· [type AddrError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "AddrError)

* [func (e \*AddrError) Error() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "AddrError.Error)
* [func (e \*AddrError) Temporary() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "AddrError.Temporary)
* [func (e \*AddrError) Timeout() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "AddrError.Timeout)

· [type OpError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "OpError)

* [func (e \*OpError) Error() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "OpError.Error)
* [func (e \*OpError) Temporary() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "OpError.Temporary)
* [func (e \*OpError) Timeout() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "OpError.Timeout)

· [func SplitHostPort(hostport string) (host, port string, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "SplitHostPort)

· [func JoinHostPort(host, port string) string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "JoinHostPort)

· [type HardwareAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "HardwareAddr)

* [func ParseMAC(s string) (hw HardwareAddr, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ParseMAC)
* [func (a HardwareAddr) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "HardwareAddr.String)

· [type Flags](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Flags)

* [func (f Flags) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Flags.String)

· [type Interface](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface)

* [func InterfaceByIndex(index int) (\*Interface, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InterfaceByIndex)
* [func InterfaceByName(name string) (\*Interface, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InterfaceByName)
* [func (ifi \*Interface) Addrs() ([]Addr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface.Addrs)
* [func (ifi \*Interface) MulticastAddrs() ([]Addr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface.MulticastAddrs)

· [func Interfaces() ([]Interface, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interfaces)

· [func InterfaceAddrs() ([]Addr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InterfaceAddrs)

· [type IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

* [func IPv4(a, b, c, d byte) IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPv4)
* [func ParseIP(s string) IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ParseIP)
* [func (ip IP) IsGlobalUnicast() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.IsGlobalUnicast)
* [func (ip IP) IsLinkLocalUnicast() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.IsLinkLocalUnicast)
* [func (ip IP) IsInterfaceLocalMulticast() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.IsInterfaceLocalMulticast)
* [func (ip IP) IsLinkLocalMulticast() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.IsLinkLocalMulticast)
* [func (ip IP) IsMulticast() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.IsMulticast)
* [func (ip IP) IsLoopback() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.IsLoopback)
* [func (ip IP) IsUnspecified() bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.IsUnspecified)
* [func (ip IP) DefaultMask() IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.DefaultMask)
* [func (ip IP) Equal(x IP) bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.Equal)
* [func (ip IP) To16() IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.To16)
* [func (ip IP) To4() IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.To4)
* [func (ip IP) Mask(mask IPMask) IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.Mask)
* [func (ip IP) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.String)
* [func (ip IP) MarshalText() ([]byte, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.MarshalText)
* [func (ip \*IP) UnmarshalText(text []byte) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP.UnmarshalText)

· [type IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask)

* [func IPv4Mask(a, b, c, d byte) IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPv4Mask)
* [func CIDRMask(ones, bits int) IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "CIDRMask)
* [func (m IPMask) Size() (ones, bits int)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask.Size)
* [func (m IPMask) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask.String)

· [type IPNet](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPNet)

* [func ParseCIDR(s string) (IP, \*IPNet, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ParseCIDR)
* [func (n \*IPNet) Contains(ip IP) bool](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPNet.Contains)
* [func (n \*IPNet) Network() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPNet.Network)
* [func (n \*IPNet) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPNet.String)

· [type Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

· [type Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn)

* [func Dial(network, address string) (Conn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Dial)
* [func DialTimeout(network, address string, timeout time.Duration) (Conn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DialTimeout)
* [func Pipe() (Conn, Conn)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Pipe)

· [type PacketConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "PacketConn)

* [func ListenPacket(net, laddr string) (PacketConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ListenPacket)

· [type Dialer](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Dialer)

* [func (d \*Dialer) Dial(network, address string) (Conn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Dialer.Dial)

· [type Listener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Listener)

* [func Listen(net, laddr string) (Listener, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Listen)

· [type IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr)

* [func ResolveIPAddr(net, addr string) (\*IPAddr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ResolveIPAddr)
* [func (a \*IPAddr) Network() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr.Network)
* [func (a \*IPAddr) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr.String)

· [type TCPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPAddr)

* [func ResolveTCPAddr(net, addr string) (\*TCPAddr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ResolveTCPAddr)
* [func (a \*TCPAddr) Network() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPAddr.Network)
* [func (a \*TCPAddr) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPAddr.String)

· [type UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr)

* [func ResolveUDPAddr(net, addr string) (\*UDPAddr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ResolveUDPAddr)
* [func (a \*UDPAddr) Network() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr.Network)
* [func (a \*UDPAddr) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr.String)

· [type UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr)

* [func ResolveUnixAddr(net, addr string) (\*UnixAddr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ResolveUnixAddr)
* [func (a \*UnixAddr) Network() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr.Network)
* [func (a \*UnixAddr) String() string](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr.String)

· [type IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn)

* [func DialIP(netProto string, laddr, raddr \*IPAddr) (\*IPConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DialIP)
* [func ListenIP(netProto string, laddr \*IPAddr) (\*IPConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ListenIP)
* [func (c \*IPConn) LocalAddr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.LocalAddr)
* [func (c \*IPConn) RemoteAddr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.RemoteAddr)
* [func (c \*IPConn) SetReadBuffer(bytes int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.SetReadBuffer)
* [func (c \*IPConn) SetWriteBuffer(bytes int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.SetWriteBuffer)
* [func (c \*IPConn) SetDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.SetDeadline)
* [func (c \*IPConn) SetReadDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.SetReadDeadline)
* [func (c \*IPConn) SetWriteDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.SetWriteDeadline)
* [func (c \*IPConn) Read(b []byte) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.Read)
* [func (c \*IPConn) ReadFrom(b []byte) (int, Addr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.ReadFrom)
* [func (c \*IPConn) ReadFromIP(b []byte) (int, \*IPAddr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.ReadFromIP)
* [func (c \*IPConn) ReadMsgIP(b, oob []byte) (n, oobn, flags int, addr \*IPAddr, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.ReadMsgIP)
* [func (c \*IPConn) Write(b []byte) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.Write)
* [func (c \*IPConn) WriteTo(b []byte, addr Addr) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.WriteTo)
* [func (c \*IPConn) WriteToIP(b []byte, addr \*IPAddr) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.WriteToIP)
* [func (c \*IPConn) WriteMsgIP(b, oob []byte, addr \*IPAddr) (n, oobn int, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.WriteMsgIP)
* [func (c \*IPConn) Close() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.Close)
* [func (c \*IPConn) File() (f \*os.File, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn.File)

· [type TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn)

* [func DialTCP(net string, laddr, raddr \*TCPAddr) (\*TCPConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DialTCP)
* [func (c \*TCPConn) LocalAddr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.LocalAddr)
* [func (c \*TCPConn) RemoteAddr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.RemoteAddr)
* [func (c \*TCPConn) SetReadBuffer(bytes int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetReadBuffer)
* [func (c \*TCPConn) SetWriteBuffer(bytes int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetWriteBuffer)
* [func (c \*TCPConn) SetDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetDeadline)
* [func (c \*TCPConn) SetReadDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetReadDeadline)
* [func (c \*TCPConn) SetWriteDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetWriteDeadline)
* [func (c \*TCPConn) SetKeepAlive(keepalive bool) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetKeepAlive)
* [func (c \*TCPConn) SetKeepAlivePeriod(d time.Duration) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetKeepAlivePeriod)
* [func (c \*TCPConn) SetLinger(sec int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetLinger)
* [func (c \*TCPConn) SetNoDelay(noDelay bool) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.SetNoDelay)
* [func (c \*TCPConn) Read(b []byte) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.Read)
* [func (c \*TCPConn) ReadFrom(r io.Reader) (int64, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.ReadFrom)
* [func (c \*TCPConn) Write(b []byte) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.Write)
* [func (c \*TCPConn) Close() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.Close)
* [func (c \*TCPConn) CloseRead() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.CloseRead)
* [func (c \*TCPConn) CloseWrite() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.CloseWrite)
* [func (c \*TCPConn) File() (f \*os.File, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn.File)

· [type UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn)

* [func DialUDP(net string, laddr, raddr \*UDPAddr) (\*UDPConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DialUDP)
* [func ListenMulticastUDP(net string, ifi \*Interface, gaddr \*UDPAddr) (\*UDPConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ListenMulticastUDP)
* [func ListenUDP(net string, laddr \*UDPAddr) (\*UDPConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ListenUDP)
* [func (c \*UDPConn) LocalAddr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.LocalAddr)
* [func (c \*UDPConn) RemoteAddr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.RemoteAddr)
* [func (c \*UDPConn) SetReadBuffer(bytes int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.SetReadBuffer)
* [func (c \*UDPConn) SetWriteBuffer(bytes int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.SetWriteBuffer)
* [func (c \*UDPConn) SetDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.SetDeadline)
* [func (c \*UDPConn) SetReadDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.SetReadDeadline)
* [func (c \*UDPConn) SetWriteDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.SetWriteDeadline)
* [func (c \*UDPConn) Read(b []byte) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.Read)
* [func (c \*UDPConn) ReadFrom(b []byte) (int, Addr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.ReadFrom)
* [func (c \*UDPConn) ReadFromUDP(b []byte) (n int, addr \*UDPAddr, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.ReadFromUDP)
* [func (c \*UDPConn) ReadMsgUDP(b, oob []byte) (n, oobn, flags int, addr \*UDPAddr, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.ReadMsgUDP)
* [func (c \*UDPConn) Write(b []byte) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.Write)
* [func (c \*UDPConn) WriteTo(b []byte, addr Addr) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.WriteTo)
* [func (c \*UDPConn) WriteToUDP(b []byte, addr \*UDPAddr) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.WriteToUDP)
* [func (c \*UDPConn) WriteMsgUDP(b, oob []byte, addr \*UDPAddr) (n, oobn int, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.WriteMsgUDP)
* [func (c \*UDPConn) Close() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.Close)
* [func (c \*UDPConn) File() (f \*os.File, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn.File)

· [type UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)

* [func DialUnix(net string, laddr, raddr \*UnixAddr) (\*UnixConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DialUnix)
* [func ListenUnixgram(net string, laddr \*UnixAddr) (\*UnixConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ListenUnixgram)
* [func (c \*UnixConn) LocalAddr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.LocalAddr)
* [func (c \*UnixConn) RemoteAddr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.RemoteAddr)
* [func (c \*UnixConn) SetReadBuffer(bytes int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.SetReadBuffer)
* [func (c \*UnixConn) SetWriteBuffer(bytes int) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.SetWriteBuffer)
* [func (c \*UnixConn) SetDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.SetDeadline)
* [func (c \*UnixConn) SetReadDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.SetReadDeadline)
* [func (c \*UnixConn) SetWriteDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.SetWriteDeadline)
* [func (c \*UnixConn) Read(b []byte) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.Read)
* [func (c \*UnixConn) ReadFrom(b []byte) (int, Addr, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.ReadFrom)
* [func (c \*UnixConn) ReadFromUnix(b []byte) (n int, addr \*UnixAddr, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.ReadFromUnix)
* [func (c \*UnixConn) ReadMsgUnix(b, oob []byte) (n, oobn, flags int, addr \*UnixAddr, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.ReadMsgUnix)
* [func (c \*UnixConn) Write(b []byte) (int, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.Write)
* [func (c \*UnixConn) WriteTo(b []byte, addr Addr) (n int, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.WriteTo)
* [func (c \*UnixConn) WriteToUnix(b []byte, addr \*UnixAddr) (n int, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.WriteToUnix)
* [func (c \*UnixConn) WriteMsgUnix(b, oob []byte, addr \*UnixAddr) (n, oobn int, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.WriteMsgUnix)
* [func (c \*UnixConn) Close() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.Close)
* [func (c \*UnixConn) CloseRead() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.CloseRead)
* [func (c \*UnixConn) CloseWrite() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.CloseWrite)
* [func (c \*UnixConn) File() (f \*os.File, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn.File)

· [type TCPListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener)

* [func ListenTCP(net string, laddr \*TCPAddr) (\*TCPListener, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ListenTCP)
* [func (l \*TCPListener) Addr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener.Addr)
* [func (l \*TCPListener) SetDeadline(t time.Time) error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener.SetDeadline)
* [func (l \*TCPListener) Accept() (Conn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener.Accept)
* [func (l \*TCPListener) AcceptTCP() (\*TCPConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener.AcceptTCP)
* [func (l \*TCPListener) Close() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener.Close)
* [func (l \*TCPListener) File() (f \*os.File, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener.File)

· [type UnixListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener)

* [func ListenUnix(net string, laddr \*UnixAddr) (\*UnixListener, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ListenUnix)
* [func (l \*UnixListener) Addr() Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener.Addr)
* [func (l \*UnixListener) SetDeadline(t time.Time) (err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener.SetDeadline)
* [func (l \*UnixListener) Accept() (c Conn, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener.Accept)
* [func (l \*UnixListener) AcceptUnix() (\*UnixConn, error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener.AcceptUnix)
* [func (l \*UnixListener) Close() error](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener.Close)
* [func (l \*UnixListener) File() (f \*os.File, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener.File)

· [func FileConn(f \*os.File) (c Conn, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "FileConn)

· [func FilePacketConn(f \*os.File) (c PacketConn, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "FilePacketConn)

· [func FileListener(f \*os.File) (l Listener, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "FileListener)

· [type MX](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "MX)

· [type NS](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "NS)

· [type SRV](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "SRV)

· [func LookupPort(network, service string) (port int, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupPort)

· [func LookupCNAME(name string) (cname string, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupCNAME)

· [func LookupHost(host string) (addrs []string, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupHost)

· [func LookupIP(host string) (addrs []IP, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupIP)

· [func LookupAddr(addr string) (name []string, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupAddr)

· [func LookupMX(name string) (mx []\*MX, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupMX)

· [func LookupNS(name string) (ns []\*NS, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupNS)

· [func LookupSRV(service, proto, name string) (cname string, addrs []\*SRV, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupSRV)

· [func LookupTXT(name string) (txt []string, err error)](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "LookupTXT)

#### Examples

### [返回首页](https://studygolang.com/static/pkgdoc/main.html)

· [Listener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "example-Listener)

const (

IPv4len = 4

IPv6len = 16

)

IP address lengths (bytes).

### Variables

var (

IPv4bcast = [IPv4](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPv4)(255, 255, 255, 255) // 广播地址

IPv4allsys = [IPv4](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPv4)(224, 0, 0, 1) // 所有主机和路由器

IPv4allrouter = [IPv4](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPv4)(224, 0, 0, 2) // 所有路由器

IPv4zero = [IPv4](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPv4)(0, 0, 0, 0) // 本地地址，只能作为源地址（曾用作广播地址）

)

常用的IPv4地址。

var (

IPv6zero = [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP){0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

IPv6unspecified = [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP){0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

IPv6loopback = [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP){0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}

IPv6interfacelocalallnodes = [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP){0xff, 0x01, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0x01}

IPv6linklocalallnodes = [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP){0xff, 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0x01}

IPv6linklocalallrouters = [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP){0xff, 0x02, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0x02}

)

常用的IPv6地址。

var (

ErrWriteToConnected = [errors](http://godoc.org/errors).[New](http://godoc.org/errors" \l "New)("use of WriteTo with pre-connected connection")

)

很多OpError类型的错误会包含本错误。

### type [ParseError](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "634" \o "View Source)

type ParseError struct {

Type [string](http://godoc.org/builtin" \l "string)

Text [string](http://godoc.org/builtin" \l "string)

}

ParseError代表一个格式错误的字符串，Type为期望的格式。

#### func (\*ParseError) [Error](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "639" \o "View Source)

func (e \*[ParseError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "ParseError)) Error() [string](http://godoc.org/builtin" \l "string)

### type [Error](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "209" \o "View Source)

type Error interface {

[error](http://godoc.org/builtin" \l "error)

Timeout() [bool](http://godoc.org/builtin" \l "bool) // 错误是否为超时？

Temporary() [bool](http://godoc.org/builtin" \l "bool) // 错误是否是临时的？

}

Error代表一个网络错误。

### type [UnknownNetworkError](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "373" \o "View Source)

type UnknownNetworkError [string](http://godoc.org/builtin" \l "string)

#### func (UnknownNetworkError) [Error](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "375" \o "View Source)

func (e [UnknownNetworkError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnknownNetworkError)) Error() [string](http://godoc.org/builtin" \l "string)

#### func (UnknownNetworkError) [Temporary](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "376" \o "View Source)

func (e [UnknownNetworkError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnknownNetworkError)) Temporary() [bool](http://godoc.org/builtin" \l "bool)

#### func (UnknownNetworkError) [Timeout](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "377" \o "View Source)

func (e [UnknownNetworkError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnknownNetworkError)) Timeout() [bool](http://godoc.org/builtin" \l "bool)

### type [InvalidAddrError](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "379" \o "View Source)

type InvalidAddrError [string](http://godoc.org/builtin" \l "string)

#### func (InvalidAddrError) [Error](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "381" \o "View Source)

func (e [InvalidAddrError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InvalidAddrError)) Error() [string](http://godoc.org/builtin" \l "string)

#### func (InvalidAddrError) [Temporary](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "383" \o "View Source)

func (e [InvalidAddrError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InvalidAddrError)) Temporary() [bool](http://godoc.org/builtin" \l "bool)

#### func (InvalidAddrError) [Timeout](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "382" \o "View Source)

func (e [InvalidAddrError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "InvalidAddrError)) Timeout() [bool](http://godoc.org/builtin" \l "bool)

### type [DNSConfigError](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "386" \o "View Source)

type DNSConfigError struct {

Err [error](http://godoc.org/builtin" \l "error)

}

DNSConfigError代表读取主机DNS配置时出现的错误。

#### func (\*DNSConfigError) [Error](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "390" \o "View Source)

func (e \*[DNSConfigError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSConfigError)) Error() [string](http://godoc.org/builtin" \l "string)

#### func (\*DNSConfigError) [Temporary](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "395" \o "View Source)

func (e \*[DNSConfigError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSConfigError)) Temporary() [bool](http://godoc.org/builtin" \l "bool)

#### func (\*DNSConfigError) [Timeout](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "394" \o "View Source)

func (e \*[DNSConfigError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSConfigError)) Timeout() [bool](http://godoc.org/builtin" \l "bool)

### type [DNSError](https://github.com/golang/go/blob/master/src/net/dnsclient.go?name=release" \l "13" \o "View Source)

type DNSError struct {

Err [string](http://godoc.org/builtin" \l "string) // 错误的描述

Name [string](http://godoc.org/builtin" \l "string) // 查询的名称

Server [string](http://godoc.org/builtin" \l "string) // 使用的服务器

IsTimeout [bool](http://godoc.org/builtin" \l "bool)

}

DNSError代表DNS查询的错误。

#### func (\*DNSError) [Error](https://github.com/golang/go/blob/master/src/net/dnsclient.go?name=release" \l "20" \o "View Source)

func (e \*[DNSError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSError)) Error() [string](http://godoc.org/builtin" \l "string)

#### func (\*DNSError) [Temporary](https://github.com/golang/go/blob/master/src/net/dnsclient.go?name=release" \l "33" \o "View Source)

func (e \*[DNSError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSError)) Temporary() [bool](http://godoc.org/builtin" \l "bool)

#### func (\*DNSError) [Timeout](https://github.com/golang/go/blob/master/src/net/dnsclient.go?name=release" \l "32" \o "View Source)

func (e \*[DNSError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "DNSError)) Timeout() [bool](http://godoc.org/builtin" \l "bool)

### type [AddrError](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "349" \o "View Source)

type AddrError struct {

Err [string](http://godoc.org/builtin" \l "string)

Addr [string](http://godoc.org/builtin" \l "string)

}

#### func (\*AddrError) [Error](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "354" \o "View Source)

func (e \*[AddrError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "AddrError)) Error() [string](http://godoc.org/builtin" \l "string)

#### func (\*AddrError) [Temporary](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "365" \o "View Source)

func (e \*[AddrError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "AddrError)) Temporary() [bool](http://godoc.org/builtin" \l "bool)

#### func (\*AddrError) [Timeout](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "369" \o "View Source)

func (e \*[AddrError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "AddrError)) Timeout() [bool](http://godoc.org/builtin" \l "bool)

### type [OpError](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "292" \o "View Source)

type OpError struct {

// Op是出现错误的操作，如"read"或"write"

Op [string](http://godoc.org/builtin" \l "string)

// Net是错误所在的网络类型，如"tcp"或"udp6"

Net [string](http://godoc.org/builtin" \l "string)

// Addr是出现错误的网络地址

Addr [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

// Err是操作中出现的错误

Err [error](http://godoc.org/builtin" \l "error)

}

OpError是经常被net包的函数返回的错误类型。它描述了该错误的操作、网络类型和网络地址。

#### func (\*OpError) [Error](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "308" \o "View Source)

func (e \*[OpError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "OpError)) Error() [string](http://godoc.org/builtin" \l "string)

#### func (\*OpError) [Temporary](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "327" \o "View Source)

func (e \*[OpError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "OpError)) Temporary() [bool](http://godoc.org/builtin" \l "bool)

#### func (\*OpError) [Timeout](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "338" \o "View Source)

func (e \*[OpError](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "OpError)) Timeout() [bool](http://godoc.org/builtin" \l "bool)

### func [SplitHostPort](https://github.com/golang/go/blob/master/src/net/ipsock.go?name=release" \l "143" \o "View Source)

func SplitHostPort(hostport [string](http://godoc.org/builtin" \l "string)) (host, port [string](http://godoc.org/builtin" \l "string), err [error](http://godoc.org/builtin" \l "error))

函数将格式为"host:port"、"[host]:port"或"[ipv6-host%zone]:port"的网络地址分割为host或ipv6-host%zone和port两个部分。Ipv6的文字地址或者主机名必须用方括号括起来，如"[::1]:80"、"[ipv6-host]:http"、"[ipv6-host%zone]:80"。

### func [JoinHostPort](https://github.com/golang/go/blob/master/src/net/ipsock.go?name=release" \l "223" \o "View Source)

func JoinHostPort(host, port [string](http://godoc.org/builtin" \l "string)) [string](http://godoc.org/builtin" \l "string)

函数将host和port合并为一个网络地址。一般格式为"host:port"；如果host含有冒号或百分号，格式为"[host]:port"。

### type [HardwareAddr](https://github.com/golang/go/blob/master/src/net/mac.go?name=release" \l "14" \o "View Source)

type HardwareAddr [][byte](http://godoc.org/builtin" \l "byte)

HardwareAddr类型代表一个硬件地址（MAC地址）。

#### func [ParseMAC](https://github.com/golang/go/blob/master/src/net/mac.go?name=release" \l "39" \o "View Source)

func ParseMAC(s [string](http://godoc.org/builtin" \l "string)) (hw [HardwareAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "HardwareAddr), err [error](http://godoc.org/builtin" \l "error))

ParseMAC函数使用如下格式解析一个IEEE 802 MAC-48、EUI-48或EUI-64硬件地址：

01:23:45:67:89:ab

01:23:45:67:89:ab:cd:ef

01-23-45-67-89-ab

01-23-45-67-89-ab-cd-ef

0123.4567.89ab

0123.4567.89ab.cdef

#### func (HardwareAddr) [String](https://github.com/golang/go/blob/master/src/net/mac.go?name=release" \l "16" \o "View Source)

func (a [HardwareAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "HardwareAddr)) String() [string](http://godoc.org/builtin" \l "string)

### type [Flags](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "28" \o "View Source)

type Flags [uint](http://godoc.org/builtin" \l "uint)

const (

FlagUp [Flags](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Flags) = 1 << [iota](http://godoc.org/builtin" \l "iota) // 接口在活动状态

FlagBroadcast // 接口支持广播

FlagLoopback // 接口是环回的

FlagPointToPoint // 接口是点对点的

FlagMulticast // 接口支持组播

)

#### func (Flags) [String](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "46" \o "View Source)

func (f [Flags](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Flags)) String() [string](http://godoc.org/builtin" \l "string)

### type [Interface](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "20" \o "View Source)

type Interface struct {

Index [int](http://godoc.org/builtin" \l "int) // 索引，>=1的整数

MTU [int](http://godoc.org/builtin" \l "int) // 最大传输单元

Name [string](http://godoc.org/builtin" \l "string) // 接口名，例如"en0"、"lo0"、"eth0.100"

HardwareAddr [HardwareAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "HardwareAddr) // 硬件地址，IEEE MAC-48、EUI-48或EUI-64格式

Flags [Flags](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Flags) // 接口的属性，例如FlagUp、FlagLoopback、FlagMulticast

}

Interface类型代表一个网络接口（系统与网络的一个接点）。包含接口索引到名字的映射，也包含接口的设备信息。

#### func [InterfaceByIndex](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "91" \o "View Source)

func InterfaceByIndex(index [int](http://godoc.org/builtin" \l "int)) (\*[Interface](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface), [error](http://godoc.org/builtin" \l "error))

InterfaceByIndex返回指定索引的网络接口。

#### func [InterfaceByName](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "112" \o "View Source)

func InterfaceByName(name [string](http://godoc.org/builtin" \l "string)) (\*[Interface](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface), [error](http://godoc.org/builtin" \l "error))

InterfaceByName返回指定名字的网络接口。

#### func (\*Interface) [Addrs](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "63" \o "View Source)

func (ifi \*[Interface](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface)) Addrs() ([][Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr), [error](http://godoc.org/builtin" \l "error))

Addrs方法返回网络接口ifi的一或多个接口地址。

#### func (\*Interface) [MulticastAddrs](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "72" \o "View Source)

func (ifi \*[Interface](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface)) MulticastAddrs() ([][Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr), [error](http://godoc.org/builtin" \l "error))

MulticastAddrs返回网络接口ifi加入的多播组地址。

### func [Interfaces](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "80" \o "View Source)

func Interfaces() ([][Interface](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface), [error](http://godoc.org/builtin" \l "error))

Interfaces返回该系统的网络接口列表。

### func [InterfaceAddrs](https://github.com/golang/go/blob/master/src/net/interface.go?name=release" \l "86" \o "View Source)

func InterfaceAddrs() ([][Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr), [error](http://godoc.org/builtin" \l "error))

InterfaceAddrs返回该系统的网络接口的地址列表。

### type [IP](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "32" \o "View Source)

type IP [][byte](http://godoc.org/builtin" \l "byte)

IP类型是代表单个IP地址的[]byte切片。本包的函数都可以接受4字节（IPv4）和16字节（IPv6）的切片作为输入。

注意，IP地址是IPv4地址还是IPv6地址是语义上的属性，而不取决于切片的长度：16字节的切片也可以是IPv4地址。

#### func [IPv4](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "45" \o "View Source)

func IPv4(a, b, c, d [byte](http://godoc.org/builtin" \l "byte)) [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

IPv4返回包含一个IPv4地址a.b.c.d的IP地址（16字节格式）。

#### func [ParseIP](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "648" \o "View Source)

func ParseIP(s [string](http://godoc.org/builtin" \l "string)) [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

ParseIP将s解析为IP地址，并返回该地址。如果s不是合法的IP地址文本表示，ParseIP会返回nil。

字符串可以是小数点分隔的IPv4格式（如"74.125.19.99"）或IPv6格式（如"2001:4860:0:2001::68"）格式。

#### func (IP) [IsGlobalUnicast](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "161" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) IsGlobalUnicast() [bool](http://godoc.org/builtin" \l "bool)

如果ip是全局单播地址，则返回真。

#### func (IP) [IsLinkLocalUnicast](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "152" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) IsLinkLocalUnicast() [bool](http://godoc.org/builtin" \l "bool)

如果ip是链路本地单播地址，则返回真。

#### func (IP) [IsInterfaceLocalMulticast](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "137" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) IsInterfaceLocalMulticast() [bool](http://godoc.org/builtin" \l "bool)

如果ip是接口本地组播地址，则返回真。

#### func (IP) [IsLinkLocalMulticast](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "143" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) IsLinkLocalMulticast() [bool](http://godoc.org/builtin" \l "bool)

如果ip是链路本地组播地址，则返回真。

#### func (IP) [IsMulticast](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "128" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) IsMulticast() [bool](http://godoc.org/builtin" \l "bool)

如果ip是组播地址，则返回真。

#### func (IP) [IsLoopback](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "120" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) IsLoopback() [bool](http://godoc.org/builtin" \l "bool)

如果ip是环回地址，则返回真。

#### func (IP) [IsUnspecified](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "112" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) IsUnspecified() [bool](http://godoc.org/builtin" \l "bool)

如果ip是未指定地址，则返回真。

#### func (IP) [DefaultMask](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "215" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) DefaultMask() [IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask)

函数返回IP地址ip的默认子网掩码。只有IPv4有默认子网掩码；如果ip不是合法的IPv4地址，会返回nil。

#### func (IP) [Equal](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "355" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) Equal(x [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) [bool](http://godoc.org/builtin" \l "bool)

如果ip和x代表同一个IP地址，Equal会返回真。代表同一地址的IPv4地址和IPv6地址也被认为是相等的。

#### func (IP) [To16](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "195" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) To16() [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

To16将一个IP地址转换为16字节表示。如果ip不是一个IP地址（长度错误），To16会返回nil。

#### func (IP) [To4](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "180" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) To4() [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

To4将一个IPv4地址转换为4字节表示。如果ip不是IPv4地址，To4会返回nil。

#### func (IP) [Mask](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "239" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) Mask(mask [IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask)) [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

Mask方法认为mask为ip的子网掩码，返回ip的网络地址部分的ip。（主机地址部分都置0）

#### func (IP) [String](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "261" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) String() [string](http://godoc.org/builtin" \l "string)

String返回IP地址ip的字符串表示。如果ip是IPv4地址，返回值的格式为点分隔的，如"74.125.19.99"；否则表示为IPv6格式，如"2001:4860:0:2001::68"。

#### func (IP) [MarshalText](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "326" \o "View Source)

func (ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) MarshalText() ([][byte](http://godoc.org/builtin" \l "byte), [error](http://godoc.org/builtin" \l "error))

MarshalText实现了encoding.TextMarshaler接口，返回值和String方法一样。

#### func (\*IP) [UnmarshalText](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "338" \o "View Source)

func (ip \*[IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) UnmarshalText(text [][byte](http://godoc.org/builtin" \l "byte)) [error](http://godoc.org/builtin" \l "error)

UnmarshalText实现了encoding.TextUnmarshaler接口。IP地址字符串应该是ParseIP函数可以接受的格式。

### type [IPMask](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "35" \o "View Source)

type IPMask [][byte](http://godoc.org/builtin" \l "byte)

IPMask代表一个IP地址的掩码。

#### func [IPv4Mask](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "59" \o "View Source)

func IPv4Mask(a, b, c, d [byte](http://godoc.org/builtin" \l "byte)) [IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask)

IPv4Mask返回一个4字节格式的IPv4掩码a.b.c.d。

#### func [CIDRMask](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "71" \o "View Source)

func CIDRMask(ones, bits [int](http://godoc.org/builtin" \l "int)) [IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask)

CIDRMask返回一个IPMask类型值，该返回值总共有bits个字位，其中前ones个字位都是1，其余字位都是0。

#### func (IPMask) [Size](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "412" \o "View Source)

func (m [IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask)) Size() (ones, bits [int](http://godoc.org/builtin" \l "int))

Size返回m的前导的1字位数和总字位数。如果m不是规范的子网掩码（字位：/^1+0+$/），将返会(0, 0)。

#### func (IPMask) [String](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "421" \o "View Source)

func (m [IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask)) String() [string](http://godoc.org/builtin" \l "string)

String返回m的十六进制格式，没有标点。

### type [IPNet](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "38" \o "View Source)

type IPNet struct {

IP [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP) // 网络地址

Mask [IPMask](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPMask) // 子网掩码

}

IPNet表示一个IP网络。

#### func [ParseCIDR](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "663" \o "View Source)

func ParseCIDR(s [string](http://godoc.org/builtin" \l "string)) ([IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP), \*[IPNet](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPNet), [error](http://godoc.org/builtin" \l "error))

ParseCIDR将s作为一个CIDR（无类型域间路由）的IP地址和掩码字符串，如"192.168.100.1/24"或"2001:DB8::/48"，解析并返回IP地址和IP网络，参见[RFC 4632](http://tools.ietf.org/html/rfc4632)和[RFC 4291](http://tools.ietf.org/html/rfc4291)。

本函数会返回IP地址和该IP所在的网络和掩码。例如，ParseCIDR("192.168.100.1/16")会返回IP地址192.168.100.1和IP网络192.168.0.0/16。

#### func (\*IPNet) [Contains](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "456" \o "View Source)

func (n \*[IPNet](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPNet)) Contains(ip [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)) [bool](http://godoc.org/builtin" \l "bool)

Contains报告该网络是否包含地址ip。

#### func (\*IPNet) [Network](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "474" \o "View Source)

func (n \*[IPNet](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPNet)) Network() [string](http://godoc.org/builtin" \l "string)

Network返回网络类型名："ip+net"，注意该类型名是不合法的。

#### func (\*IPNet) [String](https://github.com/golang/go/blob/master/src/net/ip.go?name=release" \l "482" \o "View Source)

func (n \*[IPNet](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPNet)) String() [string](http://godoc.org/builtin" \l "string)

String返回n的CIDR表示，如"192.168.100.1/24"或"2001:DB8::/48"，参见[RFC 4632](http://tools.ietf.org/html/rfc4632)和[RFC 4291](http://tools.ietf.org/html/rfc4291)。如果n的Mask字段不是规范格式，它会返回一个包含n.IP.String()、斜线、n.Mask.String()（此时表示为无标点十六进制格式）的字符串，如"192.168.100.1/c000ff00"。

### type [Addr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "54" \o "View Source)

type Addr interface {

Network() [string](http://godoc.org/builtin" \l "string) // 网络名

String() [string](http://godoc.org/builtin" \l "string) // 字符串格式的地址

}

Addr代表一个网络终端地址。

### type [Conn](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "62" \o "View Source)

type Conn interface {

// Read从连接中读取数据

// Read方法可能会在超过某个固定时间限制后超时返回错误，该错误的Timeout()方法返回真

Read(b [][byte](http://godoc.org/builtin" \l "byte)) (n [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

// Write从连接中写入数据

// Write方法可能会在超过某个固定时间限制后超时返回错误，该错误的Timeout()方法返回真

Write(b [][byte](http://godoc.org/builtin" \l "byte)) (n [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

// Close方法关闭该连接

// 并会导致任何阻塞中的Read或Write方法不再阻塞并返回错误

Close() [error](http://godoc.org/builtin" \l "error)

// 返回本地网络地址

LocalAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

// 返回远端网络地址

RemoteAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

// 设定该连接的读写deadline，等价于同时调用SetReadDeadline和SetWriteDeadline

// deadline是一个绝对时间，超过该时间后I/O操作就会直接因超时失败返回而不会阻塞

// deadline对之后的所有I/O操作都起效，而不仅仅是下一次的读或写操作

// 参数t为零值表示不设置期限

SetDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

// 设定该连接的读操作deadline，参数t为零值表示不设置期限

SetReadDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

// 设定该连接的写操作deadline，参数t为零值表示不设置期限

// 即使写入超时，返回值n也可能>0，说明成功写入了部分数据

SetWriteDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

}

Conn接口代表通用的面向流的网络连接。多个线程可能会同时调用同一个Conn的方法。

#### func [Dial](https://github.com/golang/go/blob/master/src/net/dial.go?name=release" \l "142" \o "View Source)

func Dial(network, address [string](http://godoc.org/builtin" \l "string)) ([Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn), [error](http://godoc.org/builtin" \l "error))

在网络network上连接地址address，并返回一个Conn接口。可用的网络类型有：

"tcp"、"tcp4"、"tcp6"、"udp"、"udp4"、"udp6"、"ip"、"ip4"、"ip6"、"unix"、"unixgram"、"unixpacket"

对TCP和UDP网络，地址格式是host:port或[host]:port，参见函数JoinHostPort和SplitHostPort。

Dial("tcp", "12.34.56.78:80")

Dial("tcp", "google.com:http")

Dial("tcp", "[2001:db8::1]:http")

Dial("tcp", "[fe80::1%lo0]:80")

对IP网络，network必须是"ip"、"ip4"、"ip6"后跟冒号和协议号或者协议名，地址必须是IP地址字面值。

Dial("ip4:1", "127.0.0.1")

Dial("ip6:ospf", "::1")

对Unix网络，地址必须是文件系统路径。

#### func [DialTimeout](https://github.com/golang/go/blob/master/src/net/dial.go?name=release" \l "149" \o "View Source)

func DialTimeout(network, address [string](http://godoc.org/builtin" \l "string), timeout [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration)) ([Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn), [error](http://godoc.org/builtin" \l "error))

DialTimeout类似Dial但采用了超时。timeout参数如果必要可包含名称解析。

#### func [Pipe](https://github.com/golang/go/blob/master/src/net/pipe.go?name=release" \l "18" \o "View Source)

func Pipe() ([Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn), [Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn))

Pipe创建一个内存中的同步、全双工网络连接。连接的两端都实现了Conn接口。一端的读取对应另一端的写入，直接将数据在两端之间作拷贝；没有内部缓冲。

### type [PacketConn](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "218" \o "View Source)

type PacketConn interface {

// ReadFrom方法从连接读取一个数据包，并将有效信息写入b

// ReadFrom方法可能会在超过某个固定时间限制后超时返回错误，该错误的Timeout()方法返回真

// 返回写入的字节数和该数据包的来源地址

ReadFrom(b [][byte](http://godoc.org/builtin" \l "byte)) (n [int](http://godoc.org/builtin" \l "int), addr [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr), err [error](http://godoc.org/builtin" \l "error))

// WriteTo方法将有效数据b写入一个数据包发送给addr

// WriteTo方法可能会在超过某个固定时间限制后超时返回错误，该错误的Timeout()方法返回真

// 在面向数据包的连接中，写入超时非常罕见

WriteTo(b [][byte](http://godoc.org/builtin" \l "byte), addr [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)) (n [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

// Close方法关闭该连接

// 会导致任何阻塞中的ReadFrom或WriteTo方法不再阻塞并返回错误

Close() [error](http://godoc.org/builtin" \l "error)

// 返回本地网络地址

LocalAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

// 设定该连接的读写deadline

SetDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

// 设定该连接的读操作deadline，参数t为零值表示不设置期限

// 如果时间到达deadline，读操作就会直接因超时失败返回而不会阻塞

SetReadDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

// 设定该连接的写操作deadline，参数t为零值表示不设置期限

// 如果时间到达deadline，写操作就会直接因超时失败返回而不会阻塞

// 即使写入超时，返回值n也可能>0，说明成功写入了部分数据

SetWriteDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

}

PacketConn接口代表通用的面向数据包的网络连接。多个线程可能会同时调用同一个Conn的方法。

#### func [ListenPacket](https://github.com/golang/go/blob/master/src/net/dial.go?name=release" \l "285" \o "View Source)

func ListenPacket(net, laddr [string](http://godoc.org/builtin" \l "string)) ([PacketConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "PacketConn), [error](http://godoc.org/builtin" \l "error))

ListenPacket函数监听本地网络地址laddr。网络类型net必须是面向数据包的网络类型：

"ip"、"ip4"、"ip6"、"udp"、"udp4"、"udp6"、或"unixgram"。laddr的格式参见Dial函数。

### type [Dialer](https://github.com/golang/go/blob/master/src/net/dial.go?name=release" \l "17" \o "View Source)

type Dialer struct {

// Timeout是dial操作等待连接建立的最大时长，默认值代表没有超时。

// 如果Deadline字段也被设置了，dial操作也可能更早失败。

// 不管有没有设置超时，操作系统都可能强制执行它的超时设置。

// 例如，TCP（系统）超时一般在3分钟左右。

Timeout [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration)

// Deadline是一个具体的时间点期限，超过该期限后，dial操作就会失败。

// 如果Timeout字段也被设置了，dial操作也可能更早失败。

// 零值表示没有期限，即遵守操作系统的超时设置。

Deadline [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)

// LocalAddr是dial一个地址时使用的本地地址。

// 该地址必须是与dial的网络相容的类型。

// 如果为nil，将会自动选择一个本地地址。

LocalAddr [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

// DualStack允许单次dial操作在网络类型为"tcp"，

// 且目的地是一个主机名的DNS记录具有多个地址时，

// 尝试建立多个IPv4和IPv6连接，并返回第一个建立的连接。

DualStack [bool](http://godoc.org/builtin" \l "bool)

// KeepAlive指定一个活动的网络连接的生命周期；如果为0，会禁止keep-alive。

// 不支持keep-alive的网络连接会忽略本字段。

KeepAlive [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration)

}

Dialer类型包含与某个地址建立连接时的参数。

每一个字段的零值都等价于没有该字段。因此调用Dialer零值的Dial方法等价于调用Dial函数。

#### func (\*Dialer) [Dial](https://github.com/golang/go/blob/master/src/net/dial.go?name=release" \l "158" \o "View Source)

func (d \*[Dialer](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Dialer)) Dial(network, address [string](http://godoc.org/builtin" \l "string)) ([Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn), [error](http://godoc.org/builtin" \l "error))

Dial在指定的网络上连接指定的地址。参见Dial函数获取网络和地址参数的描述。

### type [Listener](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "266" \o "View Source)

type Listener interface {

// Addr返回该接口的网络地址

Addr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

// Accept等待并返回下一个连接到该接口的连接

Accept() (c [Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn), err [error](http://godoc.org/builtin" \l "error))

// Close关闭该接口，并使任何阻塞的Accept操作都会不再阻塞并返回错误。

Close() [error](http://godoc.org/builtin" \l "error)

}

Listener是一个用于面向流的网络协议的公用的网络监听器接口。多个线程可能会同时调用一个Listener的方法。

Example

#### func [Listen](https://github.com/golang/go/blob/master/src/net/dial.go?name=release" \l "261" \o "View Source)

func Listen(net, laddr [string](http://godoc.org/builtin" \l "string)) ([Listener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Listener), [error](http://godoc.org/builtin" \l "error))

返回在一个本地网络地址laddr上监听的Listener。网络类型参数net必须是面向流的网络：

"tcp"、"tcp4"、"tcp6"、"unix"或"unixpacket"。参见Dial函数获取laddr的语法。

### type [IPAddr](https://github.com/golang/go/blob/master/src/net/iprawsock.go?name=release" \l "8" \o "View Source)

type IPAddr struct {

IP [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

Zone [string](http://godoc.org/builtin" \l "string) // IPv6范围寻址域

}

IPAddr代表一个IP终端的地址。

#### func [ResolveIPAddr](https://github.com/golang/go/blob/master/src/net/iprawsock.go?name=release" \l "36" \o "View Source)

func ResolveIPAddr(net, addr [string](http://godoc.org/builtin" \l "string)) (\*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr), [error](http://godoc.org/builtin" \l "error))

ResolveIPAddr将addr作为一个格式为"host"或"ipv6-host%zone"的IP地址来解析。 函数会在参数net指定的网络类型上解析，net必须是"ip"、"ip4"或"ip6"。

#### func (\*IPAddr) [Network](https://github.com/golang/go/blob/master/src/net/iprawsock.go?name=release" \l "14" \o "View Source)

func (a \*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr)) Network() [string](http://godoc.org/builtin" \l "string)

Network返回地址的网络类型："ip"。

#### func (\*IPAddr) [String](https://github.com/golang/go/blob/master/src/net/iprawsock.go?name=release" \l "16" \o "View Source)

func (a \*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr)) String() [string](http://godoc.org/builtin" \l "string)

### type [TCPAddr](https://github.com/golang/go/blob/master/src/net/tcpsock.go?name=release" \l "8" \o "View Source)

type TCPAddr struct {

IP [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

Port [int](http://godoc.org/builtin" \l "int)

Zone [string](http://godoc.org/builtin" \l "string) // IPv6范围寻址域

}

TCPAddr代表一个TCP终端地址。

#### func [ResolveTCPAddr](https://github.com/golang/go/blob/master/src/net/tcpsock.go?name=release" \l "41" \o "View Source)

func ResolveTCPAddr(net, addr [string](http://godoc.org/builtin" \l "string)) (\*[TCPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPAddr), [error](http://godoc.org/builtin" \l "error))

ResolveTCPAddr将addr作为TCP地址解析并返回。参数addr格式为"host:port"或"[ipv6-host%zone]:port"，解析得到网络名和端口名；net必须是"tcp"、"tcp4"或"tcp6"。

IPv6地址字面值/名称必须用方括号包起来，如"[::1]:80"、"[ipv6-host]:http"或"[ipv6-host%zone]:80"。

#### func (\*TCPAddr) [Network](https://github.com/golang/go/blob/master/src/net/tcpsock.go?name=release" \l "15" \o "View Source)

func (a \*[TCPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPAddr)) Network() [string](http://godoc.org/builtin" \l "string)

返回地址的网络类型，"tcp"。

#### func (\*TCPAddr) [String](https://github.com/golang/go/blob/master/src/net/tcpsock.go?name=release" \l "17" \o "View Source)

func (a \*[TCPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPAddr)) String() [string](http://godoc.org/builtin" \l "string)

### type [UDPAddr](https://github.com/golang/go/blob/master/src/net/udpsock.go?name=release" \l "8" \o "View Source)

type UDPAddr struct {

IP [IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP)

Port [int](http://godoc.org/builtin" \l "int)

Zone [string](http://godoc.org/builtin" \l "string) // IPv6范围寻址域

}

UDPAddr代表一个UDP终端地址。

#### func [ResolveUDPAddr](https://github.com/golang/go/blob/master/src/net/udpsock.go?name=release" \l "41" \o "View Source)

func ResolveUDPAddr(net, addr [string](http://godoc.org/builtin" \l "string)) (\*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr), [error](http://godoc.org/builtin" \l "error))

ResolveTCPAddr将addr作为TCP地址解析并返回。参数addr格式为"host:port"或"[ipv6-host%zone]:port"，解析得到网络名和端口名；net必须是"udp"、"udp4"或"udp6"。

IPv6地址字面值/名称必须用方括号包起来，如"[::1]:80"、"[ipv6-host]:http"或"[ipv6-host%zone]:80"。

#### func (\*UDPAddr) [Network](https://github.com/golang/go/blob/master/src/net/udpsock.go?name=release" \l "15" \o "View Source)

func (a \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr)) Network() [string](http://godoc.org/builtin" \l "string)

返回地址的网络类型，"udp"。

#### func (\*UDPAddr) [String](https://github.com/golang/go/blob/master/src/net/udpsock.go?name=release" \l "17" \o "View Source)

func (a \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr)) String() [string](http://godoc.org/builtin" \l "string)

### type [UnixAddr](https://github.com/golang/go/blob/master/src/net/unixsock.go?name=release" \l "8" \o "View Source)

type UnixAddr struct {

Name [string](http://godoc.org/builtin" \l "string)

Net [string](http://godoc.org/builtin" \l "string)

}

UnixAddr代表一个Unix域socket终端地址。

#### func [ResolveUnixAddr](https://github.com/golang/go/blob/master/src/net/unixsock.go?name=release" \l "36" \o "View Source)

func ResolveUnixAddr(net, addr [string](http://godoc.org/builtin" \l "string)) (\*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr), [error](http://godoc.org/builtin" \l "error))

ResolveUnixAddr将addr作为Unix域socket地址解析，参数net指定网络类型："unix"、"unixgram"或"unixpacket"。

#### func (\*UnixAddr) [Network](https://github.com/golang/go/blob/master/src/net/unixsock.go?name=release" \l "15" \o "View Source)

func (a \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr)) Network() [string](http://godoc.org/builtin" \l "string)

返回地址的网络类型，"unix"，"unixgram"或"unixpacket"。

#### func (\*UnixAddr) [String](https://github.com/golang/go/blob/master/src/net/unixsock.go?name=release" \l "19" \o "View Source)

func (a \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr)) String() [string](http://godoc.org/builtin" \l "string)

### type [IPConn](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "62" \o "View Source)

type IPConn struct {

// 内含隐藏或非导出字段

}

IPConn类型代表IP网络连接，实现了Conn和PacketConn接口。

#### func [DialIP](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "184" \o "View Source)

func DialIP(netProto [string](http://godoc.org/builtin" \l "string), laddr, raddr \*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr)) (\*[IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn), [error](http://godoc.org/builtin" \l "error))

DialIP在网络协议netProto上连接本地地址laddr和远端地址raddr，netProto必须是"ip"、"ip4"或"ip6"后跟冒号和协议名或协议号。

#### func [ListenIP](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "212" \o "View Source)

func ListenIP(netProto [string](http://godoc.org/builtin" \l "string), laddr \*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr)) (\*[IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn), [error](http://godoc.org/builtin" \l "error))

ListenIP创建一个接收目的地是本地地址laddr的IP数据包的网络连接，返回的\*IPConn的ReadFrom和WriteTo方法可以用来发送和接收IP数据包。（每个包都可获取来源址或者设置目标地址）

#### func (\*IPConn) [LocalAddr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "142" \o "View Source)

func (c \*IPConn) LocalAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

LocalAddr返回本地网络地址

#### func (\*IPConn) [RemoteAddr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "150" \o "View Source)

func (c \*IPConn) RemoteAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

RemoteAddr返回远端网络地址

#### func (\*IPConn) [SetReadBuffer](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "183" \o "View Source)

func (c \*IPConn) SetReadBuffer(bytes [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetReadBuffer设置该连接的系统接收缓冲

#### func (\*IPConn) [SetWriteBuffer](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "192" \o "View Source)

func (c \*IPConn) SetWriteBuffer(bytes [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetWriteBuffer设置该连接的系统发送缓冲

#### func (\*IPConn) [SetDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "158" \o "View Source)

func (c \*IPConn) SetDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetDeadline设置读写操作绝对期限，实现了Conn接口的SetDeadline方法

#### func (\*IPConn) [SetReadDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "166" \o "View Source)

func (c \*IPConn) SetReadDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetReadDeadline设置读操作绝对期限，实现了Conn接口的SetReadDeadline方法

#### func (\*IPConn) [SetWriteDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "174" \o "View Source)

func (c \*IPConn) SetWriteDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetWriteDeadline设置写操作绝对期限，实现了Conn接口的SetWriteDeadline方法

#### func (\*IPConn) [Read](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "118" \o "View Source)

func (c \*IPConn) Read(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

Read实现Conn接口Read方法

#### func (\*IPConn) [ReadFrom](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "98" \o "View Source)

func (c \*[IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn)) ReadFrom(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr), [error](http://godoc.org/builtin" \l "error))

ReadFrom实现PacketConn接口ReadFrom方法。注意本方法有bug，应避免使用。

#### func (\*IPConn) [ReadFromIP](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "75" \o "View Source)

func (c \*[IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn)) ReadFromIP(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), \*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr), [error](http://godoc.org/builtin" \l "error))

ReadFromIP从c读取一个IP数据包，将有效负载拷贝到b，返回拷贝字节数和数据包来源地址。

ReadFromIP方法会在超过一个固定的时间点之后超时，并返回一个错误。注意本方法有bug，应避免使用。

#### func (\*IPConn) [ReadMsgIP](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "110" \o "View Source)

func (c \*[IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn)) ReadMsgIP(b, oob [][byte](http://godoc.org/builtin" \l "byte)) (n, oobn, flags [int](http://godoc.org/builtin" \l "int), addr \*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr), err [error](http://godoc.org/builtin" \l "error))

ReadMsgIP从c读取一个数据包，将有效负载拷贝进b，相关的带外数据拷贝进oob，返回拷贝进b的字节数，拷贝进oob的字节数，数据包的flag，数据包来源地址和可能的错误。

#### func (\*IPConn) [Write](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "126" \o "View Source)

func (c \*IPConn) Write(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

Write实现Conn接口Write方法

#### func (\*IPConn) [WriteTo](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "150" \o "View Source)

func (c \*[IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn)) WriteTo(b [][byte](http://godoc.org/builtin" \l "byte), addr [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

WriteTo实现PacketConn接口WriteTo方法

#### func (\*IPConn) [WriteToIP](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "132" \o "View Source)

func (c \*[IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn)) WriteToIP(b [][byte](http://godoc.org/builtin" \l "byte), addr \*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

WriteToIP通过c向地址addr发送一个数据包，b为包的有效负载，返回写入的字节。

WriteToIP方法会在超过一个固定的时间点之后超时，并返回一个错误。在面向数据包的连接上，写入超时是十分罕见的。

#### func (\*IPConn) [WriteMsgIP](https://github.com/golang/go/blob/master/src/net/iprawsock_posix.go?name=release" \l "164" \o "View Source)

func (c \*[IPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPConn)) WriteMsgIP(b, oob [][byte](http://godoc.org/builtin" \l "byte), addr \*[IPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IPAddr)) (n, oobn [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

WriteMsgIP通过c向地址addr发送一个数据包，b和oob分别为包有效负载和对应的带外数据，返回写入的字节数（包数据、带外数据）和可能的错误。

#### func (\*IPConn) [Close](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "134" \o "View Source)

func (c \*IPConn) Close() [error](http://godoc.org/builtin" \l "error)

Close关闭连接

#### func (\*IPConn) [File](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "206" \o "View Source)

func (c \*IPConn) File() (f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File), err [error](http://godoc.org/builtin" \l "error))

File方法设置下层的os.File为阻塞模式并返回其副本。

使用者有责任在用完后关闭f。关闭c不影响f，关闭f也不影响c。返回的os.File类型文件描述符和原本的网络连接是不同的。试图使用该副本修改本体的属性可能会（也可能不会）得到期望的效果。

### type [TCPConn](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "57" \o "View Source)

type TCPConn struct {

// 内含隐藏或非导出字段

}

TCPConn代表一个TCP网络连接，实现了Conn接口。

#### func [DialTCP](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "143" \o "View Source)

func DialTCP(net [string](http://godoc.org/builtin" \l "string), laddr, raddr \*[TCPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPAddr)) (\*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn), [error](http://godoc.org/builtin" \l "error))

DialTCP在网络协议net上连接本地地址laddr和远端地址raddr。net必须是"tcp"、"tcp4"、"tcp6"；如果laddr不是nil，将使用它作为本地地址，否则自动选择一个本地地址。

#### func (\*TCPConn) [LocalAddr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "142" \o "View Source)

func (c \*TCPConn) LocalAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

LocalAddr返回本地网络地址

#### func (\*TCPConn) [RemoteAddr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "150" \o "View Source)

func (c \*TCPConn) RemoteAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

RemoteAddr返回远端网络地址

#### func (\*TCPConn) [SetReadBuffer](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "183" \o "View Source)

func (c \*TCPConn) SetReadBuffer(bytes [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetReadBuffer设置该连接的系统接收缓冲

#### func (\*TCPConn) [SetWriteBuffer](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "192" \o "View Source)

func (c \*TCPConn) SetWriteBuffer(bytes [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetWriteBuffer设置该连接的系统发送缓冲

#### func (\*TCPConn) [SetDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "158" \o "View Source)

func (c \*TCPConn) SetDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetDeadline设置读写操作期限，实现了Conn接口的SetDeadline方法

#### func (\*TCPConn) [SetReadDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "166" \o "View Source)

func (c \*TCPConn) SetReadDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetReadDeadline设置读操作期限，实现了Conn接口的SetReadDeadline方法

#### func (\*TCPConn) [SetWriteDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "174" \o "View Source)

func (c \*TCPConn) SetWriteDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetWriteDeadline设置写操作期限，实现了Conn接口的SetWriteDeadline方法

#### func (\*TCPConn) [SetKeepAlive](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "114" \o "View Source)

func (c \*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn)) SetKeepAlive(keepalive [bool](http://godoc.org/builtin" \l "bool)) [error](http://godoc.org/builtin" \l "error)

SetKeepAlive设置操作系统是否应该在该连接中发送keepalive信息

#### func (\*TCPConn) [SetKeepAlivePeriod](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "122" \o "View Source)

func (c \*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn)) SetKeepAlivePeriod(d [time](http://godoc.org/time).[Duration](http://godoc.org/time" \l "Duration)) [error](http://godoc.org/builtin" \l "error)

SetKeepAlivePeriod设置keepalive的周期，超出会断开

#### func (\*TCPConn) [SetLinger](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "105" \o "View Source)

func (c \*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn)) SetLinger(sec [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetLinger设定当连接中仍有数据等待发送或接受时的Close方法的行为。

如果sec < 0（默认），Close方法立即返回，操作系统停止后台数据发送；如果 sec == 0，Close立刻返回，操作系统丢弃任何未发送或未接收的数据；如果sec > 0，Close方法阻塞最多sec秒，等待数据发送或者接收，在一些操作系统中，在超时后，任何未发送的数据会被丢弃。

#### func (\*TCPConn) [SetNoDelay](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "133" \o "View Source)

func (c \*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn)) SetNoDelay(noDelay [bool](http://godoc.org/builtin" \l "bool)) [error](http://godoc.org/builtin" \l "error)

SetNoDelay设定操作系统是否应该延迟数据包传递，以便发送更少的数据包（Nagle's算法）。默认为真，即数据应该在Write方法后立刻发送。

#### func (\*TCPConn) [Read](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "118" \o "View Source)

func (c \*TCPConn) Read(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

Read实现了Conn接口Read方法

#### func (\*TCPConn) [Write](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "126" \o "View Source)

func (c \*TCPConn) Write(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

Write实现了Conn接口Write方法

#### func (\*TCPConn) [ReadFrom](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "68" \o "View Source)

func (c \*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn)) ReadFrom(r [io](http://godoc.org/io).[Reader](http://godoc.org/io" \l "Reader)) ([int64](http://godoc.org/builtin" \l "int64), [error](http://godoc.org/builtin" \l "error))

ReadFrom实现了io.ReaderFrom接口的ReadFrom方法

#### func (\*TCPConn) [Close](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "134" \o "View Source)

func (c \*TCPConn) Close() [error](http://godoc.org/builtin" \l "error)

Close关闭连接

#### func (\*TCPConn) [CloseRead](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "77" \o "View Source)

func (c \*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn)) CloseRead() [error](http://godoc.org/builtin" \l "error)

CloseRead关闭TCP连接的读取侧（以后不能读取），应尽量使用Close方法。

#### func (\*TCPConn) [CloseWrite](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "86" \o "View Source)

func (c \*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn)) CloseWrite() [error](http://godoc.org/builtin" \l "error)

CloseWrite关闭TCP连接的写入侧（以后不能写入），应尽量使用Close方法。

#### func (\*TCPConn) [File](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "206" \o "View Source)

func (c \*TCPConn) File() (f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File), err [error](http://godoc.org/builtin" \l "error))

File方法设置下层的os.File为阻塞模式并返回其副本。

使用者有责任在用完后关闭f。关闭c不影响f，关闭f也不影响c。返回的os.File类型文件描述符和原本的网络连接是不同的。试图使用该副本修改本体的属性可能会（也可能不会）得到期望的效果。

### type [UDPConn](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "50" \o "View Source)

type UDPConn struct {

// 内含隐藏或非导出字段

}

UDPConn代表一个UDP网络连接，实现了Conn和PacketConn接口。

#### func [DialUDP](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "165" \o "View Source)

func DialUDP(net [string](http://godoc.org/builtin" \l "string), laddr, raddr \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr)) (\*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn), [error](http://godoc.org/builtin" \l "error))

DialTCP在网络协议net上连接本地地址laddr和远端地址raddr。net必须是"udp"、"udp4"、"udp6"；如果laddr不是nil，将使用它作为本地地址，否则自动选择一个本地地址。

#### func [ListenUDP](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "192" \o "View Source)

func ListenUDP(net [string](http://godoc.org/builtin" \l "string), laddr \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr)) (\*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn), [error](http://godoc.org/builtin" \l "error))

ListenUDP创建一个接收目的地是本地地址laddr的UDP数据包的网络连接。net必须是"udp"、"udp4"、"udp6"；如果laddr端口为0，函数将选择一个当前可用的端口，可以用Listener的Addr方法获得该端口。返回的\*UDPConn的ReadFrom和WriteTo方法可以用来发送和接收UDP数据包（每个包都可获得来源地址或设置目标地址）。

#### func [ListenMulticastUDP](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "212" \o "View Source)

func ListenMulticastUDP(net [string](http://godoc.org/builtin" \l "string), ifi \*[Interface](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Interface), gaddr \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr)) (\*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn), [error](http://godoc.org/builtin" \l "error))

ListenMulticastUDP接收目的地是ifi接口上的组地址gaddr的UDP数据包。它指定了使用的接口，如果ifi是nil，将使用默认接口。

#### func (\*UDPConn) [LocalAddr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "142" \o "View Source)

func (c \*UDPConn) LocalAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

LocalAddr返回本地网络地址

#### func (\*UDPConn) [RemoteAddr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "150" \o "View Source)

func (c \*UDPConn) RemoteAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

RemoteAddr返回远端网络地址

#### func (\*UDPConn) [SetReadBuffer](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "183" \o "View Source)

func (c \*UDPConn) SetReadBuffer(bytes [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetReadBuffer设置该连接的系统接收缓冲

#### func (\*UDPConn) [SetWriteBuffer](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "192" \o "View Source)

func (c \*UDPConn) SetWriteBuffer(bytes [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetWriteBuffer设置该连接的系统发送缓冲

#### func (\*UDPConn) [SetDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "158" \o "View Source)

func (c \*UDPConn) SetDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetDeadline设置读写操作期限，实现了Conn接口的SetDeadline方法

#### func (\*UDPConn) [SetReadDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "166" \o "View Source)

func (c \*UDPConn) SetReadDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetReadDeadline设置读操作期限，实现了Conn接口的SetReadDeadline方法

#### func (\*UDPConn) [SetWriteDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "174" \o "View Source)

func (c \*UDPConn) SetWriteDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetWriteDeadline设置写操作期限，实现了Conn接口的SetWriteDeadline方法

#### func (\*UDPConn) [Read](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "118" \o "View Source)

func (c \*UDPConn) Read(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

Read实现Conn接口Read方法

#### func (\*UDPConn) [ReadFrom](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "78" \o "View Source)

func (c \*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn)) ReadFrom(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr), [error](http://godoc.org/builtin" \l "error))

ReadFrom实现PacketConn接口ReadFrom方法

#### func (\*UDPConn) [ReadFromUDP](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "63" \o "View Source)

func (c \*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn)) ReadFromUDP(b [][byte](http://godoc.org/builtin" \l "byte)) (n [int](http://godoc.org/builtin" \l "int), addr \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr), err [error](http://godoc.org/builtin" \l "error))

ReadFromUDP从c读取一个UDP数据包，将有效负载拷贝到b，返回拷贝字节数和数据包来源地址。

ReadFromUDP方法会在超过一个固定的时间点之后超时，并返回一个错误。

#### func (\*UDPConn) [ReadMsgUDP](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "91" \o "View Source)

func (c \*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn)) ReadMsgUDP(b, oob [][byte](http://godoc.org/builtin" \l "byte)) (n, oobn, flags [int](http://godoc.org/builtin" \l "int), addr \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr), err [error](http://godoc.org/builtin" \l "error))

ReadMsgUDP从c读取一个数据包，将有效负载拷贝进b，相关的带外数据拷贝进oob，返回拷贝进b的字节数，拷贝进oob的字节数，数据包的flag，数据包来源地址和可能的错误。

#### func (\*UDPConn) [Write](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "126" \o "View Source)

func (c \*UDPConn) Write(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

Write实现Conn接口Write方法

#### func (\*UDPConn) [WriteTo](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "131" \o "View Source)

func (c \*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn)) WriteTo(b [][byte](http://godoc.org/builtin" \l "byte), addr [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

WriteTo实现PacketConn接口WriteTo方法

#### func (\*UDPConn) [WriteToUDP](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "113" \o "View Source)

func (c \*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn)) WriteToUDP(b [][byte](http://godoc.org/builtin" \l "byte), addr \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

WriteToUDP通过c向地址addr发送一个数据包，b为包的有效负载，返回写入的字节。

WriteToUDP方法会在超过一个固定的时间点之后超时，并返回一个错误。在面向数据包的连接上，写入超时是十分罕见的。

#### func (\*UDPConn) [WriteMsgUDP](https://github.com/golang/go/blob/master/src/net/udpsock_posix.go?name=release" \l "145" \o "View Source)

func (c \*[UDPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPConn)) WriteMsgUDP(b, oob [][byte](http://godoc.org/builtin" \l "byte), addr \*[UDPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UDPAddr)) (n, oobn [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

WriteMsgUDP通过c向地址addr发送一个数据包，b和oob分别为包有效负载和对应的带外数据，返回写入的字节数（包数据、带外数据）和可能的错误。

#### func (\*UDPConn) [Close](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "134" \o "View Source)

func (c \*UDPConn) Close() [error](http://godoc.org/builtin" \l "error)

Close关闭连接

#### func (\*UDPConn) [File](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "206" \o "View Source)

func (c \*UDPConn) File() (f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File), err [error](http://godoc.org/builtin" \l "error))

File方法设置下层的os.File为阻塞模式并返回其副本。

使用者有责任在用完后关闭f。关闭c不影响f，关闭f也不影响c。返回的os.File类型文件描述符和原本的网络连接是不同的。试图使用该副本修改本体的属性可能会（也可能不会）得到期望的效果。

### type [UnixConn](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "110" \o "View Source)

type UnixConn struct {

// 内含隐藏或非导出字段

}

UnixConn代表Unix域socket连接，实现了Conn和PacketConn接口。

#### func [DialUnix](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "240" \o "View Source)

func DialUnix(net [string](http://godoc.org/builtin" \l "string), laddr, raddr \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr)) (\*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn), [error](http://godoc.org/builtin" \l "error))

DialUnix在网络协议net上连接本地地址laddr和远端地址raddr。net必须是"unix"、"unixgram"、"unixpacket"，如果laddr不是nil将使用它作为本地地址，否则自动选择一个本地地址。

#### func [ListenUnixgram](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "359" \o "View Source)

func ListenUnixgram(net [string](http://godoc.org/builtin" \l "string), laddr \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr)) (\*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn), [error](http://godoc.org/builtin" \l "error))

ListenUnixgram接收目的地是本地地址laddr的Unix datagram网络连接。net必须是"unixgram"，返回的\*UnixConn的ReadFrom和WriteTo方法可以用来发送和接收数据包（每个包都可获取来源址或者设置目标地址）。

#### func (\*UnixConn) [LocalAddr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "142" \o "View Source)

func (c \*UnixConn) LocalAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

LocalAddr返回本地网络地址

#### func (\*UnixConn) [RemoteAddr](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "150" \o "View Source)

func (c \*UnixConn) RemoteAddr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

RemoteAddr返回远端网络地址

#### func (\*UnixConn) [SetReadBuffer](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "183" \o "View Source)

func (c \*UnixConn) SetReadBuffer(bytes [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetReadBuffer设置该连接的系统接收缓冲

#### func (\*UnixConn) [SetWriteBuffer](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "192" \o "View Source)

func (c \*UnixConn) SetWriteBuffer(bytes [int](http://godoc.org/builtin" \l "int)) [error](http://godoc.org/builtin" \l "error)

SetWriteBuffer设置该连接的系统发送缓冲

#### func (\*UnixConn) [SetDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "158" \o "View Source)

func (c \*UnixConn) SetDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetDeadline设置读写操作期限，实现了Conn接口的SetDeadline方法

#### func (\*UnixConn) [SetReadDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "166" \o "View Source)

func (c \*UnixConn) SetReadDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetReadDeadline设置读操作期限，实现了Conn接口的SetReadDeadline方法

#### func (\*UnixConn) [SetWriteDeadline](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "174" \o "View Source)

func (c \*UnixConn) SetWriteDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

SetWriteDeadline设置写操作期限，实现了Conn接口的SetWriteDeadline方法

#### func (\*UnixConn) [Read](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "118" \o "View Source)

func (c \*UnixConn) Read(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

Read实现了Conn接口Read方法

#### func (\*UnixConn) [ReadFrom](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "138" \o "View Source)

func (c \*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)) ReadFrom(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr), [error](http://godoc.org/builtin" \l "error))

ReadFrom实现PacketConn接口ReadFrom方法

#### func (\*UnixConn) [ReadFromUnix](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "123" \o "View Source)

func (c \*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)) ReadFromUnix(b [][byte](http://godoc.org/builtin" \l "byte)) (n [int](http://godoc.org/builtin" \l "int), addr \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr), err [error](http://godoc.org/builtin" \l "error))

ReadFromUnix从c读取一个UDP数据包，将有效负载拷贝到b，返回拷贝字节数和数据包来源地址。

ReadFromUnix方法会在超过一个固定的时间点之后超时，并返回一个错误。

#### func (\*UnixConn) [ReadMsgUnix](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "150" \o "View Source)

func (c \*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)) ReadMsgUnix(b, oob [][byte](http://godoc.org/builtin" \l "byte)) (n, oobn, flags [int](http://godoc.org/builtin" \l "int), addr \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr), err [error](http://godoc.org/builtin" \l "error))

ReadMsgUnix从c读取一个数据包，将有效负载拷贝进b，相关的带外数据拷贝进oob，返回拷贝进b的字节数，拷贝进oob的字节数，数据包的flag，数据包来源地址和可能的错误。

#### func (\*UnixConn) [Write](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "126" \o "View Source)

func (c \*UnixConn) Write(b [][byte](http://godoc.org/builtin" \l "byte)) ([int](http://godoc.org/builtin" \l "int), [error](http://godoc.org/builtin" \l "error))

Write实现了Conn接口Write方法

#### func (\*UnixConn) [WriteTo](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "188" \o "View Source)

func (c \*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)) WriteTo(b [][byte](http://godoc.org/builtin" \l "byte), addr [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)) (n [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

WriteTo实现PacketConn接口WriteTo方法

#### func (\*UnixConn) [WriteToUnix](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "170" \o "View Source)

func (c \*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)) WriteToUnix(b [][byte](http://godoc.org/builtin" \l "byte), addr \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr)) (n [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

WriteToUnix通过c向地址addr发送一个数据包，b为包的有效负载，返回写入的字节。

WriteToUnix方法会在超过一个固定的时间点之后超时，并返回一个错误。在面向数据包的连接上，写入超时是十分罕见的。

#### func (\*UnixConn) [WriteMsgUnix](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "202" \o "View Source)

func (c \*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)) WriteMsgUnix(b, oob [][byte](http://godoc.org/builtin" \l "byte), addr \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr)) (n, oobn [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

WriteMsgUnix通过c向地址addr发送一个数据包，b和oob分别为包有效负载和对应的带外数据，返回写入的字节数（包数据、带外数据）和可能的错误。

#### func (\*UnixConn) [Close](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "134" \o "View Source)

func (c \*UnixConn) Close() [error](http://godoc.org/builtin" \l "error)

Close关闭连接

#### func (\*UnixConn) [CloseRead](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "221" \o "View Source)

func (c \*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)) CloseRead() [error](http://godoc.org/builtin" \l "error)

CloseRead关闭TCP连接的读取侧（以后不能读取），应尽量使用Close方法

#### func (\*UnixConn) [CloseWrite](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "230" \o "View Source)

func (c \*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn)) CloseWrite() [error](http://godoc.org/builtin" \l "error)

CloseWrite关闭TCP连接的写入侧（以后不能写入），应尽量使用Close方法

#### func (\*UnixConn) [File](https://github.com/golang/go/blob/master/src/net/net.go?name=release" \l "206" \o "View Source)

func (c \*UnixConn) File() (f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File), err [error](http://godoc.org/builtin" \l "error))

File方法设置下层的os.File为阻塞模式并返回其副本。

使用者有责任在用完后关闭f。关闭c不影响f，关闭f也不影响c。返回的os.File类型文件描述符和原本的网络连接是不同的。试图使用该副本修改本体的属性可能会（也可能不会）得到期望的效果。

### type [TCPListener](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "224" \o "View Source)

type TCPListener struct {

// 内含隐藏或非导出字段

}

TCPListener代表一个TCP网络的监听者。使用者应尽量使用Listener接口而不是假设（网络连接为）TCP。

#### func [ListenTCP](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "285" \o "View Source)

func ListenTCP(net [string](http://godoc.org/builtin" \l "string), laddr \*[TCPAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPAddr)) (\*[TCPListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener), [error](http://godoc.org/builtin" \l "error))

ListenTCP在本地TCP地址laddr上声明并返回一个\*TCPListener，net参数必须是"tcp"、"tcp4"、"tcp6"，如果laddr的端口字段为0，函数将选择一个当前可用的端口，可以用Listener的Addr方法获得该端口。

#### func (\*TCPListener) [Addr](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "261" \o "View Source)

func (l \*[TCPListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener)) Addr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

Addr返回l监听的的网络地址，一个\*TCPAddr。

#### func (\*TCPListener) [SetDeadline](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "265" \o "View Source)

func (l \*[TCPListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener)) SetDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) [error](http://godoc.org/builtin" \l "error)

设置监听器执行的期限，t为Time零值则会关闭期限限制。

#### func (\*TCPListener) [Accept](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "243" \o "View Source)

func (l \*[TCPListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener)) Accept() ([Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn), [error](http://godoc.org/builtin" \l "error))

Accept用于实现Listener接口的Accept方法；他会等待下一个呼叫，并返回一个该呼叫的Conn接口。

#### func (\*TCPListener) [AcceptTCP](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "230" \o "View Source)

func (l \*[TCPListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener)) AcceptTCP() (\*[TCPConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPConn), [error](http://godoc.org/builtin" \l "error))

AcceptTCP接收下一个呼叫，并返回一个新的\*TCPConn。

#### func (\*TCPListener) [Close](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "253" \o "View Source)

func (l \*[TCPListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener)) Close() [error](http://godoc.org/builtin" \l "error)

Close停止监听TCP地址，已经接收的连接不受影响。

#### func (\*TCPListener) [File](https://github.com/golang/go/blob/master/src/net/tcpsock_posix.go?name=release" \l "279" \o "View Source)

func (l \*[TCPListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "TCPListener)) File() (f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File), err [error](http://godoc.org/builtin" \l "error))

File方法返回下层的os.File的副本，并将该副本设置为阻塞模式。

使用者有责任在用完后关闭f。关闭c不影响f，关闭f也不影响c。返回的os.File类型文件描述符和原本的网络连接是不同的。试图使用该副本修改本体的属性可能会（也可能不会）得到期望的效果。

### type [UnixListener](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "260" \o "View Source)

type UnixListener struct {

// 内含隐藏或非导出字段

}

UnixListener代表一个Unix域scoket的监听者。使用者应尽量使用Listener接口而不是假设（网络连接为）Unix域scoket。

#### func [ListenUnix](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "267" \o "View Source)

func ListenUnix(net [string](http://godoc.org/builtin" \l "string), laddr \*[UnixAddr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixAddr)) (\*[UnixListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener), [error](http://godoc.org/builtin" \l "error))

ListenTCP在Unix域scoket地址laddr上声明并返回一个\*UnixListener，net参数必须是"unix"或"unixpacket"。

#### func (\*UnixListener) [Addr](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "335" \o "View Source)

func (l \*[UnixListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener)) Addr() [Addr](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Addr)

Addr返回l的监听的Unix域socket地址

#### func (\*UnixListener) [SetDeadline](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "339" \o "View Source)

func (l \*[UnixListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener)) SetDeadline(t [time](http://godoc.org/time).[Time](http://godoc.org/time" \l "Time)) (err [error](http://godoc.org/builtin" \l "error))

设置监听器执行的期限，t为Time零值则会关闭期限限制

#### func (\*UnixListener) [Accept](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "303" \o "View Source)

func (l \*[UnixListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener)) Accept() (c [Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn), err [error](http://godoc.org/builtin" \l "error))

Accept用于实现Listener接口的Accept方法；他会等待下一个呼叫，并返回一个该呼叫的Conn接口。

#### func (\*UnixListener) [AcceptUnix](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "285" \o "View Source)

func (l \*[UnixListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener)) AcceptUnix() (\*[UnixConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixConn), [error](http://godoc.org/builtin" \l "error))

AcceptUnix接收下一个呼叫，并返回一个新的\*UnixConn。

#### func (\*UnixListener) [Close](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "313" \o "View Source)

func (l \*[UnixListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener)) Close() [error](http://godoc.org/builtin" \l "error)

Close停止监听Unix域socket地址，已经接收的连接不受影响。

#### func (\*UnixListener) [File](https://github.com/golang/go/blob/master/src/net/unixsock_posix.go?name=release" \l "353" \o "View Source)

func (l \*[UnixListener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "UnixListener)) File() (f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File), err [error](http://godoc.org/builtin" \l "error))

File方法返回下层的os.File的副本，并将该副本设置为阻塞模式。

使用者有责任在用完后关闭f。关闭c不影响f，关闭f也不影响c。返回的os.File类型文件描述符和原本的网络连接是不同的。试图使用该副本修改本体的属性可能会（也可能不会）得到期望的效果。

### func [FileConn](https://github.com/golang/go/blob/master/src/net/file_unix.go?name=release" \l "82" \o "View Source)

func FileConn(f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File)) (c [Conn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Conn), err [error](http://godoc.org/builtin" \l "error))

FileConn返回一个下层为文件f的网络连接的拷贝。调用者有责任在结束程序前关闭f。关闭c不会影响f，关闭f也不会影响c。本函数与各种实现了Conn接口的类型的File方法是对应的。

### func [FilePacketConn](https://github.com/golang/go/blob/master/src/net/file_unix.go?name=release" \l "124" \o "View Source)

func FilePacketConn(f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File)) (c [PacketConn](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "PacketConn), err [error](http://godoc.org/builtin" \l "error))

FilePacketConn函数返回一个下层为文件f的数据包网络连接的拷贝。调用者有责任在结束程序前关闭f。关闭c不会影响f，关闭f也不会影响c。本函数与各种实现了PacketConn接口的类型的File方法是对应的。

### func [FileListener](https://github.com/golang/go/blob/master/src/net/file_unix.go?name=release" \l "105" \o "View Source)

func FileListener(f \*[os](http://godoc.org/os).[File](http://godoc.org/os" \l "File)) (l [Listener](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "Listener), err [error](http://godoc.org/builtin" \l "error))

FileListener返回一个下层为文件f的网络监听器的拷贝。调用者有责任在使用结束后改变l。关闭l不会影响f，关闭f也不会影响l。本函数与各种实现了Listener接口的类型的File方法是对应的。

### type [MX](https://github.com/golang/go/blob/master/src/net/dnsclient.go?name=release" \l "225" \o "View Source)

type MX struct {

Host [string](http://godoc.org/builtin" \l "string)

Pref [uint16](http://godoc.org/builtin" \l "uint16)

}

MX代表一条DNS MX记录（邮件交换记录），根据收信人的地址后缀来定位邮件服务器。

### type [NS](https://github.com/golang/go/blob/master/src/net/dnsclient.go?name=release" \l "249" \o "View Source)

type NS struct {

Host [string](http://godoc.org/builtin" \l "string)

}

NS代表一条DNS NS记录（域名服务器记录），指定该域名由哪个DNS服务器来进行解析。

### type [SRV](https://github.com/golang/go/blob/master/src/net/dnsclient.go?name=release" \l "166" \o "View Source)

type SRV struct {

Target [string](http://godoc.org/builtin" \l "string)

Port [uint16](http://godoc.org/builtin" \l "uint16)

Priority [uint16](http://godoc.org/builtin" \l "uint16)

Weight [uint16](http://godoc.org/builtin" \l "uint16)

}

SRV代表一条DNS SRV记录（资源记录），记录某个服务由哪台计算机提供。

### func [LookupPort](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "93" \o "View Source)

func LookupPort(network, service [string](http://godoc.org/builtin" \l "string)) (port [int](http://godoc.org/builtin" \l "int), err [error](http://godoc.org/builtin" \l "error))

LookupPort函数查询指定网络和服务的（默认）端口。

### func [LookupCNAME](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "101" \o "View Source)

func LookupCNAME(name [string](http://godoc.org/builtin" \l "string)) (cname [string](http://godoc.org/builtin" \l "string), err [error](http://godoc.org/builtin" \l "error))

LookupCNAME函数查询name的规范DNS名（但该域名未必可以访问）。如果调用者不关心规范名可以直接调用LookupHost或者LookupIP；这两个函数都会在查询时考虑到规范名。

### func [LookupHost](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "24" \o "View Source)

func LookupHost(host [string](http://godoc.org/builtin" \l "string)) (addrs [][string](http://godoc.org/builtin" \l "string), err [error](http://godoc.org/builtin" \l "error))

LookupHost函数查询主机的网络地址序列。

### func [LookupIP](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "30" \o "View Source)

func LookupIP(host [string](http://godoc.org/builtin" \l "string)) (addrs [][IP](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "IP), err [error](http://godoc.org/builtin" \l "error))

LookupIP函数查询主机的ipv4和ipv6地址序列。

### func [LookupAddr](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "135" \o "View Source)

func LookupAddr(addr [string](http://godoc.org/builtin" \l "string)) (name [][string](http://godoc.org/builtin" \l "string), err [error](http://godoc.org/builtin" \l "error))

LookupAddr查询某个地址，返回映射到该地址的主机名序列，本函数和LookupHost不互为反函数。

### func [LookupMX](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "119" \o "View Source)

func LookupMX(name [string](http://godoc.org/builtin" \l "string)) (mx []\*[MX](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "MX), err [error](http://godoc.org/builtin" \l "error))

LookupMX函数返回指定主机的按Pref字段排好序的DNS MX记录。

### func [LookupNS](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "124" \o "View Source)

func LookupNS(name [string](http://godoc.org/builtin" \l "string)) (ns []\*[NS](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "NS), err [error](http://godoc.org/builtin" \l "error))

LookupNS函数返回指定主机的DNS NS记录。

### func [LookupSRV](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "114" \o "View Source)

func LookupSRV(service, proto, name [string](http://godoc.org/builtin" \l "string)) (cname [string](http://godoc.org/builtin" \l "string), addrs []\*[SRV](https://studygolang.com/static/pkgdoc/pkg/net.htm" \l "SRV), err [error](http://godoc.org/builtin" \l "error))

LookupSRV函数尝试执行指定服务、协议、主机的SRV查询。协议proto为"tcp" 或"udp"。返回的记录按Priority字段排序，同一优先度按Weight字段随机排序。

LookupSRV函数按照[RFC 2782](http://tools.ietf.org/html/rfc2782)的规定构建用于查询的DNS名。也就是说，它会查询\_service.\_proto.name。为了适应将服务的SRV记录发布在非规范名下的情况，如果service和proto参数都是空字符串，函数会直接查询name。

### func [LookupTXT](https://github.com/golang/go/blob/master/src/net/lookup.go?name=release" \l "129" \o "View Source)

func LookupTXT(name [string](http://godoc.org/builtin" \l "string)) (txt [][string](http://godoc.org/builtin" \l "string), err [error](http://godoc.org/builtin" \l "error))

LookupTXT函数返回指定主机的DNS TXT记录。

# **开发举例说明**

先生成main.go的空文件，拷贝整个lib目录的文件到根目录中。在main.go文件中，写入如下：

Package main

package main

import(

"crypto/tls"

"encoding/base64"

"errors"

"fmt"

"io"

"io/ioutil"

"log"

"net/url"

"sort"

"strings"

"sync"

"time"

)

var Input\_read = flag.String("r", "020304F", "input reader id")

var Input\_write = flag.String("w", "020304F", "input writer id")

var Input\_file = flag.String("f", "filename", "input filename")

var Input\_scan = flag.String("s", "FFF", "input id for scan")

var Input\_daemon = flag.String("d", "ddd", "no input")

var wg sync.WaitGroup

func main(){

wg.Add(1)

go webserver()

wg.Add(1)

go netsocket()

wg.Add(1)

go dbblock()

wg.wait()

}

在命令行模式下。运行编译go build main.go后，会生成文件main.exe 。 开发完成。