1、基本介绍

Package `http` `provides HTTP client and server implementations`.
net/http标准库实现了客户端和服务器。里面有`两个重要的类型Client和Server`，还有Request和Response类型。goweb框架都是基于net/http标准库实现的。

type Client

func (c *Client) CloseIdleConnections()

func (c *Client) Do(req *Request) (*Response, error)

func (c *Client) Get(url string) (resp *Response, err error)

func (c *Client) Head(url string) (resp *Response, err error)

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

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

---

type Server

func (srv *Server) Close() error

func (srv *Server) ListenAndServe() error

func (srv *Server) ListenAndServeTLS(certFile, keyFile string) error

func (srv *Server) RegisterOnShutdown(f func())

func (srv *Server) Serve(l net.Listener) error

func (srv *Server) ServeTLS(l net.Listener, certFile, keyFile string) error

func (srv *Server) SetKeepAlivesEnabled(v bool)

func (srv *Server) Shutdown(ctx context.Context) error

type Request

func NewRequest(method, url string, body io.Reader) (*Request, error)

func NewRequestWithContext(ctx context.Context, method, url string, body io.Reader) (*Request, error)

func ReadRequest(b *bufio.Reader) (*Request, error)

func (r *Request) AddCookie(c *Cookie)

func (r *Request) BasicAuth() (username, password string, ok bool)

func (r *Request) Clone(ctx context.Context) *Request

func (r *Request) Context() context.Context

func (r *Request) Cookie(name string) (*Cookie, error)

func (r *Request) Cookies() []*Cookie

func (r *Request) FormFile(key string) (multipart.File, *multipart.FileHeader, error)

func (r *Request) FormValue(key string) string

func (r *Request) MultipartReader() (*multipart.Reader, error)

func (r *Request) ParseForm() error

func (r *Request) ParseMultipartForm(maxMemory int64) error

func (r *Request) PostFormValue(key string) string

func (r *Request) ProtoAtLeast(major, minor int) bool

func (r *Request) Referer() string

func (r *Request) SetBasicAuth(username, password string)

func (r *Request) UserAgent() string

func (r *Request) WithContext(ctx context.Context) *Request

func (r *Request) Write(w io.Writer) error

func (r *Request) WriteProxy(w io.Writer) error

---

type Response

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

如果不需要模拟客户端，可以直接使用包中的函数直接获取一个响应数据

func Head(url string) (resp *Response, err error)

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

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

func ReadResponse(r *bufio.Reader, req *Request) (*Response, error)

func (r *Response) Cookies() []*Cookie

func (r *Response) Location() (*url.URL, error)

func (r *Response) ProtoAtLeast(major, minor int) bool

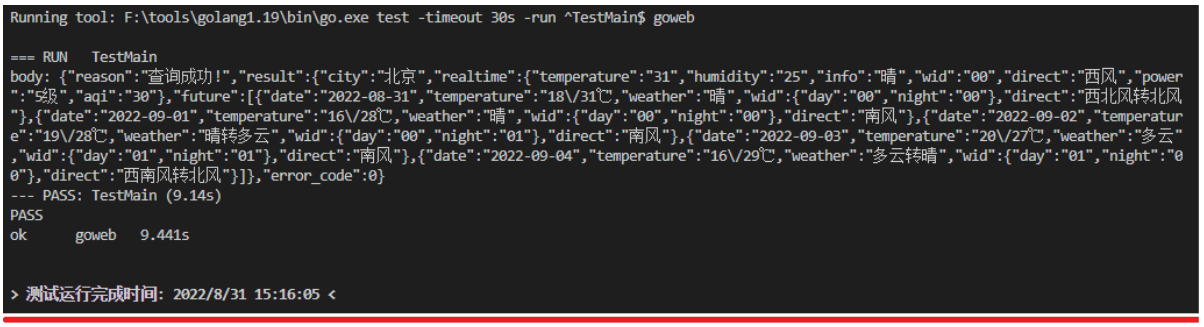
func (r *Response) Write(w io.Writer) error

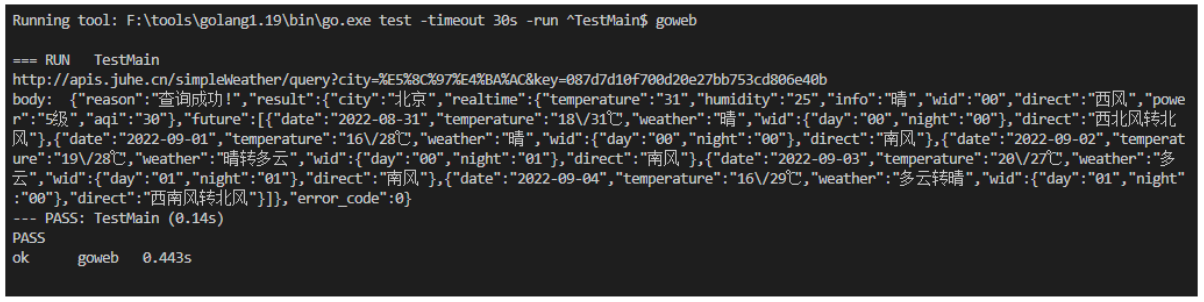
2、Get请求

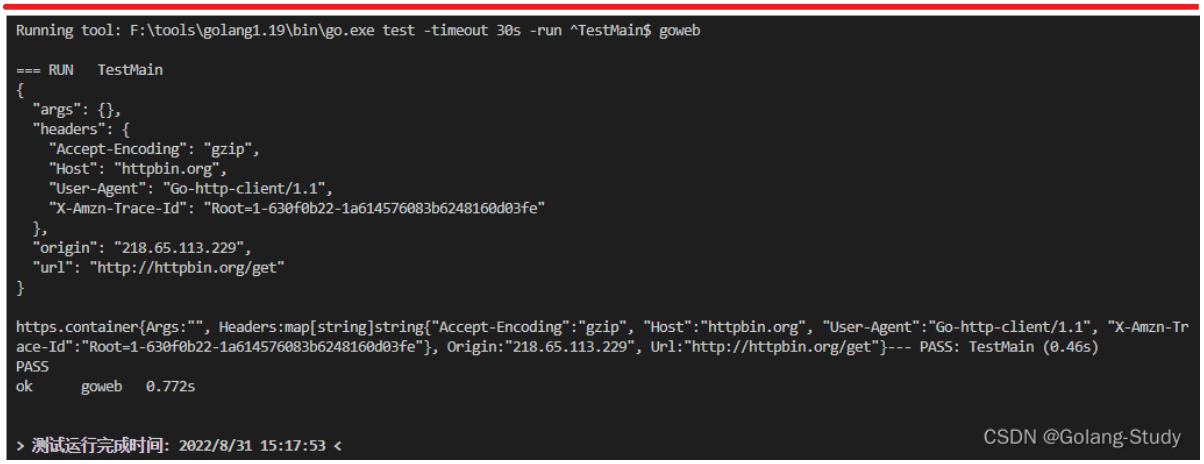
通过包中的Get函数能够直接获取响应数据。

```
package https

import (
    "encoding/json"
    "fmt"
    "io"
    "log"
    "net/http"
    "net/url"
)

// 1、测试http.Get
func TestGet() {

    url := "http://apis.juhe.cn/simpleWeather/query?
key=087d7d10f700d20e27bb753cd806e40b&city=北京"

    // 向url发送一个请求，并且获取响应Response
    res, err := http.Get(url)
    if err != nil {
        log.Fatalf("err")
    }
    defer res.Body.Close()

    body, _ := io.ReadAll(res.Body) // 读取全部数据
    fmt.Printf("body: %v\n", string(body))
}

// 2、Get请求，把一些参数做成变量，而不是拼接在url后面
func TestGetPara() {

    params := url.Values{}
    URL, err := url.Parse("http://apis.juhe.cn/simpleWeather/query")
    if err != nil {
        return
    }

    params.Set("key", "087d7d10f700d20e27bb753cd806e40b")
    params.Set("city", "北京")

    // 如果参数中中文参数，这个方法会进行URLEncode
    URL.RawQuery = params.Encode()
    urlPath := URL.String()

    fmt.Println(urlPath)

    // 发送Get请求
    response, err := http.Get(urlPath)
    if err != nil {
        log.Fatal("err")
    }
```

```
    defer response.Body.Close()

    body, _ := io.ReadAll(response.Body)
    fmt.Println("body: ", string(body))
}

// 3、解析json类型的返回结果
func TestJson() {

    // 预定义一个结构体
    type container struct {
        Args    string            `json:"args`
        Headers map[string]string `json:"headers`
        Origin  string            `json:"origin`
        Url     string            `json:"url"`
    }

    response, err := http.Get("http://httpbin.org/get")
    if err != nil {
        return
    }
    defer response.Body.Close()

    body, _ := io.ReadAll(response.Body)
    fmt.Println(string(body))

    var result container
    json.Unmarshal(body, &result) // 反序列化，结果的json数据转结构体
    fmt.Printf("%#v", result)
}
```







3、Post请求

通过PostForm和Post函数发送Post请求。

```
// Post请求PostForm函数使用
func TestPost() {

    path := "http://apis.juhe.cn/simpleWeather/query"
    urlValues := url.Values{}
    urlValues.Add("key", "087d7d10f700d20e27bb753cd806e40b")
    urlValues.Add("city", "北京")

    // 发送Post请求，Post是用于提交数据的请求方式，多用于表单
    result, err := http.PostForm(path, urlValues)
    if err != nil {
        log.Fatal("err")
    }
    defer result.Body.Close()

    body, _ := io.ReadAll(result.Body)
    fmt.Println(string(body))
}

// Post请求的另一种方式Post函数使用
func TestPost2() {
```

```
    urlValues := url.Values{
        "username": {"百里守约"},
        "age":      {"20"},
    }

    reqBody := urlValues.Encode() //解析中文

    // Post函数，第一个是地址，第二个是文本类型，第三个是Reader对象，读取请求体(表单数据)
    response, _ := http.Post("http://httpbin.org/post", "text/html",
strings.NewReader(reqBody))
    body, _ := io.ReadAll(response.Body)
    fmt.Println(string(body))
}

// 发送Json数据的Post请求
func TestPostJson() {

    data := make(map[string]interface{})
    data["site"] = "www.baidu.com"
    data["name"] = "马化腾"

    byteData, _ := json.Marshal(data) // 序列化操作
    response, _ := http.Post("http://httpbin.org/post", "application/json",
bytes.NewReader(byteData))
    body, _ := io.ReadAll(response.Body)
    fmt.Println(string(body))
}
```

4、使用Client类型自定义请求

当需要模拟浏览器客户端，比如爬虫的时候就需要使用。

```
func TestClient() {

    client := &http.Client{
        Timeout: time.Second * 3,
    }

    url := "http://apis.juhe.cn/simpleWeather/query?
key=087d7d10f700d20e27bb753cd806e40b&city=北京"
    request, err := http.NewRequest(http.MethodGet, url, nil)
    if err != nil {
        log.Fatal("err")
    }
    // 添加请求头信息
    request.Header.Add("referer", "http://apis.juhe.cn/")

    // Do 发送 HTTP 请求并返回 HTTP 响应，遵循客户端上配置的策略（例如重定向、cookie、身份验
证）。
    response, err2 := client.Do(request)
    if err2 != nil {
        log.Fatal("err2")
    }
    defer response.Body.Close()
```

```
    // 读取请求体的所有内容
    body, _ := io.ReadAll(response.Body)
    fmt.Printf("body: %v\n", string(body))

}
```

5、服务端

```
func TestHttpServer() {

    // 请求处理函数
    // 第一参数是往请求的客户端写数据的流对象，第二个则是监测到的请求对象
    caller := func(response http.ResponseWriter, request *http.Request) {

        // 读取当前路径下的http.html文件，然后将其相应给客户端
        file, err := os.OpenFile("F:/tools/golang/goweb/https/http.html",
os.O_RDONLY, 0600)
        if err != nil {
            log.Fatal("err")
        }
        defer file.Close()
        bytedata, _ := io.ReadAll(file) // 读取文件中的所有数据

        io.WriteString(response, string(bytedata))
    }

    /// 响应路径
    http.HandleFunc("/hello", caller)
    // 设置监听端口，并监听
    err := http.ListenAndServe(":8888", nil)
    if err != nil {
        log.Fatal("err")
    }
}
```