全部

网络编程入门, 100行实现Websocket转TCP代理

路奇 2024-05-06 Go 知识点 100行代码

> *"本文发表于入职啦(公众号: ruzhila) 大家可以访问入职啦学习更多的编程实战。"*

🎉 用Go实现Websocket转TCP代理，浏览器可以访问TCP Socket，网络编程入门 实用项目 👏 🎉

项目地址

代码已经开源， websockify-go 👏 欢迎Star

代码运行效果：

```
[mpi@mpis-Mac-mini cmd % ./websockify -h
Usage: ./websockify:
  -addr string
        address to listen on (default ":8080")
  -cert string
        SSL cert.pem
  -key string
        SSL key.pem
  -target string
        target address (default "localhost:5900")
  -url string
        url path to proxy, e.g. /vnc
[mpi@mpis-Mac-mini cmd % ./websockify
 2024/05/06 19:34:40 Proxying / to localhost:5900 on :8080
```

> *"所有的项目都在github上开源：100-line-code 欢迎Star 👏"*

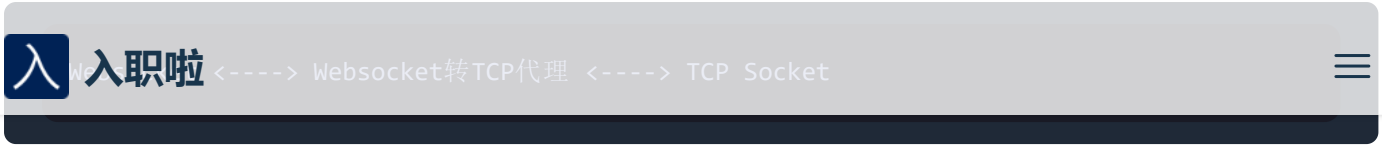
用100行代码的不同语言（Java、Python、Go、Javascript、Rust）实现项目，通过讲解项目的实现，帮助大家学习编程

我们会定期在群里分享最新的项目实战代码，包括不同语言的实现

老师还会详细讲解代码优化的思路，扫码加入实战群：

为什么需要Websocket转TCP

Websocket是一种全双工通信协议，可以在浏览器和服务器之间建立持久连接，用于实时通信。但是Websocket只支持文本和二进制数据，无法直接访问TCP Socket。 如果要在网页上访问TCP Socket，就需要一个代理服务器，将Websocket请求转发到TCP Socket， 这就是Websocket转TCP代理:



典型的场景：

- 在浏览器上访问VNC远程桌面
- 在浏览器上访问SSH终端
- 在浏览器上访问数据库管理工具
- 在浏览器上访问自定义TCP服务, 比如游戏服务器或者IM服务器

novnc/websockify是一个非常流行的Websocket转TCP代理，但是它是用Python实现的，**性能不够好**，代码也很复杂

用Go实现一个Websocket转TCP代理，性能更好，更稳定

Go实现Websocket转TCP代理

Go特别适合网络编程，标准库提供了丰富的网络库，不仅性能好，而且代码简洁，只需要一个可执行文件就可以运行，不需要依赖其他库 这次要通过100行代码实现一个Websocket转TCP代理：

- 要支持SSL
- 要支持自定义URL路径
- 要支持自定义目标地址
- 完整的单元测试和标准的服务程序架构

代码

```
package websockifygo

import (
    "crypto/tls"
    "io"
    "log"
    "net"
    "net/http"
    "strings"

    "github.com/gorilla/websocket"
)

type WSproxy struct {
    URL      string
    Target   string
    KeyPem   string
    CertPem string
}

var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
}

func (s *WSproxy) ServeHTTP(w http.ResponseWriter, r *http.Request) {
    if r.URL.Path != s.URL || r.Method != http.MethodGet {
        http.Error(w, "Not Found", http.StatusNotFound)
        return
    }
    conn, err := upgrader.Upgrade(w, r, nil)
    if err != nil {
        http.Error(w, "Bad Request", http.StatusBadRequest)
        return
    }
    defer conn.Close()
    target, err := net.Dial("tcp", s.Target)
    if err != nil {
        http.Error(w, "Bad Gateway", http.StatusBadGateway)
        return
    }
    defer target.Close()
    errChan := make(chan error, 2)
    go func() {
        for {
            messageType, message, err := conn.ReadMessage()
            if err != nil {
                errChan <- err
                return
            }
            if messageType != websocket.BinaryMessage && messageType != websocket.TextMessage {
                continue
            }
            if _, err = target.Write(message); err != nil {
                errChan <- err
                return
            }
        }
    }()
    go func() {
        buf := make([]byte, 1024)
        var n int
```

```
        for {
            if n, err = target.Read(buf); err != nil {
                errChan <- err
                return
            }
            if err = conn.WriteMessage(websocket.BinaryMessage, buf[:n]); err != nil {
                errChan <- err
                return
            }
        }
    }()
    log.Println("proxying", s.URL, "to", s.Target, "remote", conn.RemoteAddr())
    err = <-errChan
    if err != io.EOF {
        log.Println("proxy error:", err)
    }
}

func (prx *WSproxy) Serve(addr string) error {
    proxyNetListner, err := net.Listen("tcp", addr)
    if err != nil {
        return err
    }
    httpsrv := &http.Server{
        Handler: prx,
    }
    if strings.HasSuffix(addr, "443") && prx.CertPem != "" && prx.KeyPem != "" {
        httpsrv.TLSConfig = &tls.Config{
            MinVersion:       tls.VersionTLS12,
            CurvePreferences: []tls.CurveID{tls.CurveP521, tls.CurveP384, tls.CurveP256},
        }
        httpsrv.TLSNextProto = make(map[string]func(*http.Server, *tls.Conn, http.Handler))
        return httpsrv.ServeTLS(proxyNetListner, prx.CertPem, prx.KeyPem)
    } else {
        return httpsrv.Serve(proxyNetListner)
    }
}

```

代码解析

代码结构

- cmd/main.go 是程序的入口，解析命令行参数，启动服务
- websockify.go 是Websocket转TCP代理的核心代码
- websockify_test.go 是单元测试代码
- Dockerfile 是构建Docker镜像的文件，方便部署可以直接上生产系统
-

启动一个HTTP服务器

```
81  func (prx *WSproxy) Serve(addr string) error {
82      proxyNetListner, err := net.Listen("tcp", addr)
83      if err != nil {
84          return err
85      }
86      httpsrv := &http.Server{
87          Handler: prx,
88      }
89      if strings.HasSuffix(addr, "443") && prx.CertPem != "" && prx.KeyPem != "" {
90          httpsrv.TLSConfig = &tls.Config{
91              MinVersion:       tls.VersionTLS12,
92              CurvePreferences: []tls.CurveID{tls.CurveP521, tls.CurveP384, tls.CurveP256},
93          }
94          httpsrv.TLSNextProto = make(map[string]func(*http.Server, *tls.Conn, http.Handler))
95          return httpsrv.ServeTLS(proxyNetListner, prx.CertPem, prx.KeyPem)
96      } else {
97          return httpsrv.Serve(proxyNetListner)
98      }
99  }
```

- **82行：** 监听一个HTTP端口
- **89-95行：** 如果是443并且有SSL证书，就启动一个HTTPS服务器
- **86行：** 利用标准库创建一个能自定义处理的HTTP Server

处理Websocket请求

采用了 gorilla/websocket 库来处理Websocket请求，这个库是Go语言中最流行的Websocket库

所有的业务逻辑都在 ServeHTTP 函数：

```
26  func (s *WSproxy) ServeHTTP(w http.ResponseWriter, r *http.Request) {
27      if r.URL.Path != s.URL || r.Method != http.MethodGet {
28          http.Error(w, "Not Found", http.StatusNotFound)
29          return
30      }
31      conn, err := upgrader.Upgrade(w, r, nil)
32      if err != nil {
33          http.Error(w, "Bad Request", http.StatusBadRequest)
34          return
35      }
36      defer conn.Close()
37      target, err := net.Dial("tcp", s.Target)
38      if err != nil {
39          http.Error(w, "Bad Gateway", http.StatusBadGateway)
40          return
41      }
42      defer target.Close()
43      errChan := make(chan error, 2)
44      go func() {
45          for {
46              messageType, message, err := conn.ReadMessage()
47              if err != nil {
48                  errChan <- err
49                  return
50              }
51              if messageType != websocket.BinaryMessage && messageType != websocket.TextMessage {
52                  continue
53              }
54              if _, err = target.Write(message); err != nil {
55                  errChan <- err
56                  return
57              }
58          }
59      }()
60      go func() {
61          buf := make([]byte, 1024)
62          var n int
63          for {
64              if n, err = target.Read(buf); err != nil {
65                  errChan <- err
66                  return
67              }
68              if err = conn.WriteMessage(websocket.BinaryMessage, buf[:n]); err != nil {
69                  errChan <- err
70                  return
71              }
72          }
73      }()
74      log.Println("proxying", s.URL, "to", s.Target, "remote", conn.RemoteAddr())
75      err = <-errChan
76      if err != io.EOF {
77          log.Println("proxy error:", err)
78      }
79  }
```

- **31行：** 升级HTTP连接为Websocket连接, Websocket协议是基于`GET`这个HTTP请求升级为Websocket协议
- **36行：** 利用`defer`关键字，当链接断开的时候，关闭TCP连接，确保不会泄露资源
- **44-59行：** 读取Websocket数据，转发到TCP Socket

一个链接启动了两个协程

其实从 ServerHTTP 这个函数开始，系统库就为每个请求创建了一个协程，这个协程负责处理这个请求

在这个函数里面又启动了两个协程：

- 一个负责读取Websocket数据
- 一个负责读取TCP Socket数据

这样就实现了一个链接同时处理两个方向的数据流

- **43和75行** 创建了能接受err的channel，当处理数据从协程出现错误的时候，就会发送一个错误到这个channel，然后当前的主协程就会退出关闭链接

单元测试

golang的单元测试非常的方便，只需要在文件名后面加上 _test.go ，然后在函数名前面加上 Test 就可以了

```go
func TestProxy(t *testing.T) {
    // create a example echo server
    echoServer, err := net.Listen("tcp", "127.0.0.1:12340")
    if err != nil {
        t.Fatal(err)
    }

    go func() {
        defer echoServer.Close()
        for {
            conn, err := echoServer.Accept()
            if err != nil {
                log.Fatal(err)
            }
            go func() {
                defer conn.Close()
                buf := make([]byte, 1024)
                for {
                    n, err := conn.Read(buf)
                    if err != nil {
                        return
                    }
                    fmt.Println("echo:", string(buf[:n]))
                    conn.Write(buf[:n])
                }
            }()
        }
    }()

    // create a proxy server
    wsp := &WSproxy{
        URL:    "/echo",
        Target: "127.0.0.1:12340",
    }

    go func() {
        wsp.Serve("127.0.0.1:12341")
    }()

    // wait for the proxy server to start
    time.Sleep(100 * time.Millisecond)

    {
        // create a client
        r, err := http.Get("http://127.0.0.1:12341/")
        assert.Nil(t, err)
        assert.Equal(t, http.StatusNotFound, r.StatusCode)
    }
    {
        r, err := http.Get("http://127.0.0.1:12341/echo")
        assert.Nil(t, err)
        assert.Equal(t, http.StatusBadRequest, r.StatusCode)
    }
    {
        // websocket client
        conn, _, err := websocket.DefaultDialer.Dial("ws://127.0.0.1:12341/echo", nil)
```

```
57          assert.Nil(t, err)
58          defer conn.Close()
59          conn.WriteMessage(websocket.BinaryMessage, []byte("hello"))
60          _, message, err := conn.ReadMessage()
61          assert.Nil(t, err)
62          assert.Equal(t, "hello", string(message))
63      }
64  }
65
```

单元测试的逻辑：

- **2-28行** 创建一个测试用的TCP服务器，写了一个Echo服务器，就是把接收到的数据原样返回
- **31-41行** 创建一个Websockify代理服务器
- **43-63行** 启动一个Websocket客户端测试链接是否能正常工作

标准的服务器程序架构

这个项目提供了 `Dockerfile`，现在Docker是最主要的线上程序交付方式，强烈建议用Docker部署服务

基于 `flag` 这个库，来处理命令行参数，这是Go语言标准库提供的命令行参数解析库

程序能支持的命令行参数来控制服务的启动

总结

通过Websockify这个项目，学习了Go语言的网络编程，Websocket是一个非常重要的协议，可以用于实时通信，实现一个Websockify的转发代理，对于内部项目的开发非常有用

Docker也是非常重要的技术，提供了Dockerfile，可以直接部署到生产环境

过去的演示项目都没提供单元测试，提供了一个非常实用的单元测试流程，大家可以参考即便是复杂的流程也是可以通过单元测试来验证的

交流

我们构建了一个**100行代码项目**的实战群，大家可以扫码加入，一起学习编程

也可以访问入职啦学习更多的编程实战

所有的代码都在github上开源：100-line-code 欢迎Star 👏

← 上一篇文章

超级简单，100行Java实现NIO HTTP客户端，无第三方依赖

→ 下一篇文章

100行Python代码实现打砖块游戏，无需第三方依赖

浙公网安备33010602013604号

浙ICP备2023036691号-1

意见反馈或举报邮箱：**kui@fourz.cn**　隐私条款