

利用websockify将websocket通信转换成tcp

原创

AlbertS

于 2023-12-13 22:30:05 发布 阅读量2.9k 收藏 5 点赞数 1

版权

分类专栏：

Nginx

DevOps

Python

文章标签：

websocket

tcp/ip

网络协议

NVC

websockify



开放原子开发者工作坊 文章已被社区收录

加入社区



Python 同时被 3 个专栏收录 ▾

3 订阅

21 篇文章

订阅专栏

文章目录

前言

websockify

websockify 介绍

websockify 使用

探索的过程

提供基础TCP服务

测试可用

实现Websocket客户端

开始测试websockify功能

再次启动websockify

单独实现一个js版本websocket客户端

什么是VNC

前言

目前遇到一个问题，原本的服务都是利用tcp 通信的，但是某些平台只支持websocket，是不是要从头实现一套websocket网络收发逻辑呢？短时间内有没有替代方案呢？在经过一番寻找之后发现了websockify这个神器，可以将websocket协议转换成 tcp协议 ，从而实现不修改应用服务就支持tcp协议的目的，接下来就说说它怎么用，以及探索的过程。

websockify

websockify 介绍

websockify 是一个用于将 WebSocket 流量转发到其他协议的工具，它的主要用途之一是将 WebSockets 转发到基于 TCP 的协议，例如 VNC (Virtual Network Computing) ，以下是它的一些关键特点和用途：

- **WebSocket 转发：** websockify 允许你将 WebSocket 流量转发到其他类型的网络服务，使得你可以通过 WebSocket 连接访问这些服务。
- **VNC 转发：** websockify 最常用于将 WebSocket 转发到 VNC 服务器。这使得你可以通过 Web 浏览器访问运行 VNC 服务的远程计算机的桌面。
- **安全性：** websockify 支持加密通信，因此可以在安全连接上运行，提供更好的数据保护。
- **协议适配：** websockify 提供了一个通用的桥接机制，允许你将 WebSocket 连接转发到支持其他协议的服务，而不仅仅是 VNC。
- **JavaScript 客户端：** websockify 还包括一个 JavaScript 客户端库，可以直接在浏览器中使用，无需额外的插件。

使用 websockify 的典型场景包括在 Web 浏览器中访问远程计算机的桌面，或者通过 WebSocket 连接到其他需要 TCP 连接的服务，在实际应用

中，你可以通过命令行使用 websockify，也可以将其嵌入到其他应用程序中。

websockify 使用

在Ubuntu 系统下可以直接使用以下安装：

```
$ sudo apt install websockify
```

 AI助手

下面是一个使用示例，作用是将原本连接到8765的websocket请求转换成tcp请求，TCP端口4321

```
$ websockify 8765 127.0.0.1:12346
```

 AI助手

启动后原本仅支持TCP协议的服务就可以通过websockify实现对websocket的支持了

探索的过程

虽然是短短的一条 `websockify 8765 127.0.0.1:12346` 命令就可以实现从websocket到tcp的转换，但是探索的过程却不顺利，在发现 `websockify` 这个神器之后，首先是测试它是否能满足我们的需求，是否好用，但是直接在原有的服务上测试太费时间，所以尝试写了几个模拟的脚本。

提供基础TCP服务

最基本的我们需要一个很能提供TCP服务的程序，这里我使用python写了一个回显的服务器 echoserver.py，绑定本地的12346端口

```
import socket

def main():
    # 创建一个TCP套接字
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

 AI助手

```
# 绑定IP地址和端口
server_socket.bind(('127.0.0.1', 12346))

# 监听连接
server_socket.listen(5)

print("服务器已启动，等待客户端连接...")

while True:
    # 接受客户端连接
    client_socket, client_address = server_socket.accept()
    print(f"客户端{client_address}已连接")

    # 接收客户端发送的数据
    data = client_socket.recv(1024)
    print(f"收到来自{client_address}的数据: {data.decode('utf-8')}")

    # 将接收到的数据发送回客户端
    client_socket.send(data)
    print(f"已将数据发送回{client_address}")

    # 关闭客户端套接字
    client_socket.close()
    print(f"客户端{client_address}已断开连接")

if __name__ == "__main__":
    main()
```

- 启动服务

```
$ python3 echoserver.py  
服务器已启动，等待客户端连接...
```

 AI助手

- 用nc命令发送tcp数据

```
$ echo "Hello server" | nc 127.0.0.1 12346  
Hello server
```

 AI助手

- TCP服务器看到的内容

```
服务器已启动，等待客户端连接...  
客户端('127.0.0.1', 59214)已连接  
收到来自('127.0.0.1', 59214)的数据: Hello server  
  
已将数据发送回('127.0.0.1', 59214)  
客户端('127.0.0.1', 59214)已断开连接
```

 AI助手

TCP服务一切正常

实现Websocket客户端

因为python的运行环境是现成的，所以我又用它写了一个websocket客户端 wsclient.py 来发送数据，想运行的话得安装几个包

```
python3 -m pip install websocket
python3 -m pip install websocket-client
```

[AI助手](#)

真实的情况下我是用websocket客户端连接nginx，然后将nginx转发到websockify监听端口，然后websockify将数据转成tcp连接应用服务，但是这个过程太复杂了，不利于说明问题，所以后面的内容我省掉nginx，假装它不存在，直接用websocket客户端连接websockify监听端口

```
# -*- coding: utf-8 -*-
# 运行websocket服务器

import websocket

def on_message(ws, message):
    print(f"Received message: {message}")

def on_error(ws, error):
    print(f"Error: {error}")

def on_close(ws, close_status_code, close_msg):
    print(f"Connection closed with status code {close_status_code}")

def on_open(ws):
    print("Connection opened")
    # 在连接建立后发送一条消息
    ws.send("abcd")

if __name__ == "__main__":
    # WebSocket 服务器地址
    ws_url = "ws://10.10.49.172:8765"
```

[AI助手](#)

```
# 创建 WebSocket 连接
ws = websocket.WebSocketApp(ws_url,
                             on_message=on_message,
                             on_error=on_error,
                             on_close=on_close)

# 设置连接建立时的回调函数
ws.on_open = on_open

# 启动 WebSocket 连接
ws.run_forever()
```

为了websocket客户端的测试，我写了一个websocket服务器 wsserver.py 来接收数据，不过这不是一开始就测试了，而是在后面发现转换TCP失败查找原因，按流程分段测试时补充的脚本内容

```
import asyncio
import websockets

async def echo(websocket, path):
    try:
        async for message in websocket:
            print(f"Received message: {message}")
            await websocket.send(message)
    except websockets.exceptions.ConnectionClosed:
        print("WebSocket connection closed")

async def main():
    server = await websockets.serve(echo, "0.0.0.0", 12346)
```

 AI助手

```
print("WebSocket server is running...")
await server.wait_closed()

if __name__ == "__main__":
    asyncio.run(main())
```

测试结果websocket客户端发送的数据，websocket服务器可以正常收到。

开始测试websocketify功能

- 启动tcp服务echoserver，监听12346端口

```
$ python3 echoserver.py
服务器已启动，等待客户端连接...
```

 AI助手

- 启动tcp服务websocketify，监听8765端口，转换到12346端口

```
$ websocketify 8765 127.0.0.1:12346
WebSocket server settings:
- Listen on :8765
- No SSL/TLS support (no cert file)
- proxying from :8765 to 127.0.0.1:12346
```

 AI助手

- 运行websocket客户端发送的数据


```
$ python3 ./wsclient.py
Connection opened
Connection closed with status code 1003
```

 AI助手

报错了!!!

再看echoserver输出, 虽然连接成功, 但是收到的数据为空

```
客户端('127.0.0.1', 55032)已连接
收到来自('127.0.0.1', 55032)的数据:
已将数据发送回('127.0.0.1', 55032)
客户端('127.0.0.1', 55032)已断开连接
```

然后看下websockify输出, 显示转发连接12346端口成功了, 但是没有后续输出

```
$ websockify 8765 127.0.0.1:12346
WebSocket server settings:
  • Listen on :8765
  • No SSL/TLS support (no cert file)
  • proxying from :8765 to 127.0.0.1:12346
10.2.48.36 - - [13/Dec/2023 20:03:11] 10.2.48.36: Plain non-SSL (ws://) WebSocket connection
10.2.48.36 - - [13/Dec/2023 20:03:11] connecting to: 127.0.0.1:12346
```

定位问题吧, 期间不断调整nginx配置, 查看nginx和websockify日志, 调整wsclient写法, 调整echoserver写法, 分步测试wsclient和echoserver功能都没发现问题, 具体测试细节暂且不表, 后来我发现可以修改websockify启动参数输出更详细的日志, 测试过程如下

再次启动websockify

启动websockify再次使用wsclient测试，输出了额外的信息

```
$ websockify 8765 127.0.0.1:12346 --log-file /tmp/ws.log -v --traffic
WebSocket server settings:
  - Listen on :8765
  - No SSL/TLS support (no cert file)
  - proxying from :8765 to 127.0.0.1:12346
10.2.48.36: new handler Process
10.2.48.36 - - [13/Dec/2023 20:13:42] "GET / HTTP/1.1" 101 -
10.2.48.36 - - [13/Dec/2023 20:13:42] 10.2.48.36: Plain non-SSL (ws://) WebSocket connection
10.2.48.36 - - [13/Dec/2023 20:13:42] connecting to: 127.0.0.1:12346

Traffic Legend:
  } - Client receive
  }. - Client receive partial
  { - Target receive

  > - Target send
  >. - Target send partial
  < - Client send
  <. - Client send partial
}.10.2.48.36 - - [13/Dec/2023 20:13:42] 127.0.0.1:12346: Client closed connection
10.2.48.36 - - [13/Dec/2023 20:13:42] 127.0.0.1:12346: Closed target
```

 AI助手

顺着结尾的两条日志 `Client closed connection`，我找到了这一篇问题

[websockify - Client disconnects immediately after connection #365](#) 结论说是websockify不支持发送文本了

Hmm... Are you trying to send text over that socket? We dropped support for text frames in the big cleanup.

Could you print the code and reason from the close in your close handler?

我一看我的wsclient.py可不就是发送的文本嘛，看来有救了，赶紧改成了发送bytes `ws.send(b'\x01\x02\x03\x04')`，但错误依旧，尝试各种发送函数 无果，我严重怀疑我的换个python的websockets包有问题，所以我用html写了一个wsclient.html：

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>WebSocket Binary Data Example</title>
</head>
<body>
  <script>
    // 替换为你的 WebSocket 服务器地址
    const socket = new WebSocket("ws://10.10.49.172:8765");

    // 监听连接打开事件
    socket.addEventListener("open", (event) => {
      console.log("WebSocket connection opened.");

      // 构造二进制数据，这里使用一个简单的示例
      const binaryData = new Uint8Array([49, 50, 51, 52]);

      // 发送二进制数据
      socket.send(binaryData);

      console.log("Binary data sent successfully.");
    });

    // 监听接收消息事件
    socket.addEventListener("message", (event) => {
      console.log("Received message:", event.data);
    });
```

 AI助手

```
// 监听连接关闭事件
socket.addEventListener("close", (event) => {
  console.log("WebSocket connection closed.");
});

// 监听错误事件
socket.addEventListener("error", (event) => {
  console.error("WebSocket error:", event);
});
</script>
</body>
</html>
```

这次终于正常了，websocketify输出变成了由应用服务主动断开

```
$ websocketify 8765 127.0.0.1:12346 --log-file /tmp/ws.log -v --traffic
```

```
WebSocket server settings:
```

- Listen on :8765
- No SSL/TLS support (no cert file)
- proxying from :8765 to 127.0.0.1:12346

```
10.2.48.36: new handler Process
```

```
10.2.48.36 - - [13/Dec/2023 20:24:29] "GET / HTTP/1.1" 101 -
```

```
10.2.48.36 - - [13/Dec/2023 20:24:29] 10.2.48.36: Plain non-SSL (ws://) WebSocket connection
```

```
10.2.48.36 - - [13/Dec/2023 20:24:29] connecting to: 127.0.0.1:12346
```

```
Traffic Legend:
```

- } - Client receive
- }. - Client receive partial
- { - Target receive
- > - Target send
- >. - Target send partial

 AI助手

```
< - Client send
<. - Client send partial
}>{<10.2.48.36 - - [13/Dec/2023 20:24:29] 127.0.0.1:12346: Target closed connection
10.2.48.36 - - [13/Dec/2023 20:24:29] 127.0.0.1:12346: Closed target
```

echoserver也看到了发送的内容【1234】：

```
$ python3 echoserver.py
服务器已启动，等待客户端连接...
客户端('127.0.0.1', 37438)已连接
收到来自('127.0.0.1', 37438)的数据: 1234
已将数据发送回('127.0.0.1', 37438)
客户端('127.0.0.1', 37438)已断开连接
```

 AI助手

单独实现一个js版本websocket客户端

刚刚用html里的js发送了websocket数据，需要浏览器的帮助，如果是在服务器上局限性很大，所以我调整了一下，改成了js版本websocket客户端，用node .js运行，wsclient.js内容如下：

```
const WebSocket = require('ws');

// 替换为你的 WebSocket 服务器地址
const socket = new WebSocket("ws://10.10.49.172:8765");

// 监听连接打开事件
socket.addEventListener("open", (event) => {
    console.log("WebSocket connection opened.");
});

// 构造二进制数据，这里使用一个简单的示例
```

 AI助手

```
const binaryData = new Uint8Array([49, 50, 51, 52]);

// 发送二进制数据
socket.send(binaryData);

console.log("Binary data sent successfully.");
});

// 监听接收消息事件
socket.addEventListener("message", (event) => {
  console.log("Received message:", event.data);
});

// 监听连接关闭事件
socket.addEventListener("close", (event) => {
  console.log("WebSocket connection closed.");
});

// 监听错误事件
socket.addEventListener("error", (event) => {
  console.error("WebSocket error:", event);
});
```

运行结果如下：

```
$ node wsclient.js
WebSocket connection opened.
Binary data sent successfully.
Received message: <Buffer 31 32 33 34>
WebSocket connection closed.
```

 AI助手

至此，websockify的测试就结束了，它可以满足我们的需求，将websocket请求转换成tcp请求

什么是VNC

前面在介绍websockify多次提到VNC，其实VNC (Virtual Network Computing) 是一种远程桌面协议，允许用户在一个计算机上通过网络远程控制另一个计算机的桌面。VNC 提供了一种在远程计算机上查看和操作桌面的方式，就像你坐在那台计算机前一样，以下是关于 VNC 的一些关键概念：

- Server：在 VNC 中，远程计算机上运行的应用程序称为 VNC 服务器。该服务器负责监听连接请求，并将计算机的桌面图像发送给连接的客户端。
- Viewer：连接到 VNC 服务器的应用程序称为 VNC 查看器。这是用户使用的远程桌面客户端，允许他们查看和操作远程计算机的桌面。
- Port：VNC 服务器通过一个特定的网络端口监听连接请求。通常，VNC 默认使用 5900 端口。如果有多个 VNC 服务器在同一台计算机上运行，它们可能使用不同的端口（5901、5902 等）。
- Security：VNC 提供了一些安全性选项，如密码保护和加密。这有助于确保在远程访问时保护计算机的安全。
- Authentication：VNC 服务器和查看器之间的连接通常需要身份验证。这可以是使用密码进行简单的身份验证，也可以是更复杂的加密和密钥交换过程。
- websockify：对于一些场景，特别是在 Web 浏览器中访问 VNC，你可能会使用工具如 `websockify` 将 VNC 的协议转换为 **WebSocket 协议**，以便在浏览器中实现 VNC 远程桌面访问。

总结

- websockify可以轻松实现从websocket请求向tcp请求的转换，使用很方便
- 转换路径 `wsclient --ws--> websockify --tcp--> tcp application --tcp--> websockify --ws--> wsclient`
- wss对应的是ws的加密版本，他们的关系是wss=ws+SSL/TSL，类似的https=http+SSL/TSL
- 无论是nginx还是websockify都可以配置服务器证书，将应用服务器从SSL/TSL中解脱出来

- VNC是一种远程桌面协议，允许用户在一个计算机上通过网络远程控制另一个计算机的桌面

==>> 反爬链接，请勿点击，原地爆炸，概不负责! <<==

迈出这一步确实不太容易，一旦行动了就会发现没有想象中的那么难，我们都没有预知未来的能力，无法判定目前的选择是对还是错，既然选择了就要努力走下去