链滴

基于 golang websocket 的聊天室 demo

作者：xhaoxiong

原文链接：https://ld246.com/article/1547221179606

来源网站：链滴

测试聊天室demo

具体代码来自github.com

main.go（入口函数）

```
package main

import (
  "log"
 "net/http")

func serveHome(w http.ResponseWriter, r *http.Request) {
  if r.Method != "GET" {
  http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
  return
  }
  http.ServeFile(w, r, "home.html")
}

func main() {
 hub := newHub()
  //开启一个调度器
  go hub.run()
  //demo页面
  http.HandleFunc("/", serveHome)
  //websocket 握手
  http.HandleFunc("/ws", func(w http.ResponseWriter, r *http.Request) {
  serveWs(hub, w, r)
 }) err := http.ListenAndServe(":9017", nil)
  if err != nil {
  log.Fatal("ListenAndServe: ", err)
 }}
```

hub.go（调度代码，通过map对client的维护，以及消息的广播 ）

```
package main

import "log"

type Hub struct {
  //用于维护用户的map
  clients map[*Client]bool

  //用于广播给用户的chan
  broadcast chan []byte

  //用于用户订阅的chan
  register chan *Client

  //用于用户取消订阅的chan
  unregister chan *Client
}
```

```
//实例化一个调度器
func newHub() *Hub {
  return &Hub{
 broadcast:  make(chan []byte),
  register:   make(chan *Client),
  unregister: make(chan *Client),
  clients:    make(map[*Client]bool),
  }
}

//开始运行调度器
func (h *Hub) run() {

  for {
  select {
  case client := <-h.register:
        log.Printf("客户端 %d: 订阅\n", client.id)
 h.clients[client] = true
  case client := <-h.unregister:
        log.Printf("客户端 %d: 取消订阅\n", client.id)
  if _, ok := h.clients[client]; ok {
  delete(h.clients, client)
  close(client.send)
 }  case message := <-h.broadcast:
        log.Println("通过调度器把数据塞给client的chan，然后client.writePump广播消息")
  for client := range h.clients {
  select {
  case client.send <- message:
            log.Printf("将消息发送给客户端%d\n", client.id)
  default:
            close(client.send)
  delete(h.clients, client)
 } } } }}
```

client.go（client信息的发送与接收）

```
package main

import (
  "bytes"
 "log" "net/http" "time"
 "github.com/gorilla/websocket")

var iii int

const (
  //写入数据允许的等待时间
  writeWait = 10 * time.Second

  //pong的周期
  pongWait = 60 * time.Second

  //ping的周期 （需要小于pong的周期）
```

```
  pingPeriod = (pongWait * 9) / 10

  //消息最大的字节数
  maxMessageSize = 512
)

var (
 newline = []byte{'\n'}
 space = []byte{' '}
)

var upgrader = websocket.Upgrader{
 ReadBufferSize:  1024,
  WriteBufferSize: 1024,
  //允许跨域
  CheckOrigin: func(r *http.Request) bool {
  return true
  },
}

// Client is a middleman between the websocket connection and the hub.
type Client struct {
  //客户端的调度器
  hub *Hub

  //每个client实例化一个链接
  conn *websocket.Conn

  //从调度器中的向客户端发送数据
  send chan []byte

  //每个客户端的唯一标识
  id int
}

//从链接中进行读取
func (c *Client) readPump() {
  defer func() {
 c.hub.unregister <- c
 c.conn.Close()
 }()  log.Println("设置读取信息的最大值")
 c.conn.SetReadLimit(maxMessageSize)
 c.conn.SetReadDeadline(time.Now().Add(pongWait))
 c.conn.SetPongHandler(func(string) error { c.conn.SetReadDeadline(time.Now().Add(pongWai
)); return nil })

  for {
  log.Println("读取连接中的信息")
 _, message, err := c.conn.ReadMessage()
  if err != nil {
  if websocket.IsUnexpectedCloseError(err, websocket.CloseGoingAway, websocket.CloseAbno
malClosure) {
  log.Printf("error: %v", err)
 }  return
```

```
 }
 log.Println("去除消息中的空格和换行")
 message = bytes.TrimSpace(bytes.Replace(message, newline, space, -1))
 c.hub.broadcast <- message
 }}

//从chan中获取信息进行写入
func (c *Client) writePump() {
 ticker := time.NewTicker(pingPeriod)
 defer func() {
 ticker.Stop()
 c.conn.Close()
 }()  for {
 select {

 case message, ok := <-c.send:
        log.Println("writePump 接收到调度器send message")
 c.conn.SetWriteDeadline(time.Now().Add(writeWait))
 if !ok {
 c.conn.WriteMessage(websocket.CloseMessage, []byte{})
 return
 }
 log.Println("指定下一条发送消息的类型")
 w, err := c.conn.NextWriter(websocket.TextMessage)
 if err != nil {
 return
 }
 w.Write(message)

 // Add queued chat messages to the current websocket message.
 n := len(c.send)
 for i := 0; i < n; i++ {
 w.Write(newline)
 w.Write(<-c.send)
 }
 if err := w.Close(); err != nil {
 return
 }
 case <-ticker.C:
        c.conn.SetWriteDeadline(time.Now().Add(writeWait))
 log.Println("接收到定时器的心跳ping-pong")
 if err := c.conn.WriteMessage(websocket.PingMessage, nil); err != nil {
 return
 }
 } }}

//每新增一个连接初始化一个客户端进行维护
func serveWs(hub *Hub, w http.ResponseWriter, r *http.Request) {

 conn, err := upgrader.Upgrade(w, r, nil)
 if err != nil {
 log.Println(err)
 return
 }
```

```
 //用于查看用户（测试的假数据）
 iii++
 log.Println("初始化一个新的客户端")
client := &Client{hub: hub, conn: conn, send: make(chan []byte, 256), id: iii}
client.hub.register <- client

 //客户端写入
 log.Println("开启client.writePump")
 go client.writePump()

 log.Println("开启client.readPump")
 //客户端读取
 go client.readPump()
}

<html>
<head>
    <title>Chat Example<title>
    <script>
  window.onload = function () {
 var conn;
var msg = document.getElementById("msg");
var log = document.getElementById("log");

function appendLog(item) {
 var doScroll = log.scrollTop \> log.scrollHeight \- log.clientHeight \- 1;
 log.appendChild(item);
if (doScroll) {
 log.scrollTop = log.scrollHeight \- log.clientHeight;
 }
}
 document.getElementById("form").onsubmit = function () {
 if (!conn) {
 return false;
 }
 if (!msg.value) {
 return false;
 }
 conn.send(msg.value);
 msg.value = "";
return false;  };
if (window\["WebSocket"\]) {
 conn = new WebSocket("ws://123.207.1.120:9017/ws");
 conn.onclose = function (evt) {
 var item = document.createElement("div");
 item.innerHTML = "Connection closed.";
 appendLog(item);
 };
 conn.onmessage = function (evt) {
 var messages = evt.data.split('\\n');
for (var i = 0; i < messages.length; i++) {
 var item = document.createElement("div");
 item.innerText = messages\[i\];
 appendLog(item);
```

```
  }
 };
  } else {
  var item = document.createElement("div");
  item.innerHTML = "Your browser does not support WebSockets.";
  appendLog(item);
  }
};
    </script>
<style type="text/css">
html {
  overflow: hidden;
}

body {
  overflow: hidden;
  padding: 0;
  margin: 0;
  width: 100%;
  height: 100%;
  background: gray;
}

#log {
  background: white;
  margin: 0;
  padding: 0.5em 0.5em 0.5em 0.5em;
  position: absolute;
  top: 0.5em;
  left: 0.5em;
  right: 0.5em;
  bottom: 3em;
  overflow: auto;
}

#form {
  padding: 0 0.5em 0 0.5em;
  margin: 0;
  position: absolute;
  bottom: 1em;
  left: 0px;
  width: 100%;
  overflow: hidden;
}
</style>
<body>
  <div id="log"></div>
  <form id="form">
  <input type="submit" value="Send"/>
  <input type="text" id="msg" size="64">
  </form>
</body>

</html>
```

原文链接：基于 golang websocket 的聊天室 demo