```
rrentTime)
    go client.read()
    go client.write()
    // 用户连接事件
    clientManager.Register <- client
}
```

接收到 WebSocket 请求以后，通过 WsPage 函数将协议升级为 WebSocket 协议，并创建一个 client 对象，然后通过两个 goroutine 协程 read 和 write 来处理这个 client 的 socket 消息。

```
// internal/websocket/client.go
// read 函数
for {
    _, message, err := c.Socket.ReadMessage()
    if err != nil {
        return
    }
    // 处理消息
    handlerMsg(c, message)
}
```

read 函数中有一个 for 循环，一直在读消息，读到之后就转到 handlerMsg 中处理来自客户端的消息。

```
// internal/websocket/client.go
// write 函数
for {
    select {
    case <-c.closeSignal:
        g.Log().Infof(mctx, "websocket client quit, user:%+v", c.User)
        return
    case message, ok := <-c.Send:
        if !ok {
            // 发送数据错误 关闭连接
```

```
            g.Log().Warningf(mctx, "client write message, user:%+v", c.User)
            return
        }
        _ = c.Socket.WriteJSON(message)
    }
}

```

write 函数中，通过 for select + channel 的组合来建立了一个消息管道，等待要发送的消息，如果要发送消息，就往这个管道中发送就可以了，这个管道就是 c.Send, 是一个 chan 的指针类型。

前端创建 WebSocket 实例，并连接

前端创建 WebSocket 实例，连接的地址后面带上 token，做认证用，token 是登录时获取的 jwt，里面有部分用户身份信息。连接之后，收发数据使用的是 WebSocket 协议。无论是 HTTP 还是 WebSocket，都是基于 tcp 来传输数据的，这一点大家要清楚。

注意在建立 WebSocket 连接之前，客户端会先发送一个 HTTP 请求到服务器，请求升级到 WebSocket 协议。这个请求会包含一些特殊的头部字段，例如 Upgrade 和 Sec-WebSocke .start()
    go clientManager.ping()
    g.Log().Debug(mctx, "start websocket..")
}

```

```
// internal/websocket/client_manager.go
// 管道处理程序
func (manager *ClientManager) start() {
    for {
        select {
        case conn := <-manager.Register:
            // 建立连接事件
            manager.EventRegister(conn)
        case login := <-manager.Login:
            // 用户登录
            manager.EventLogin(login)
        case conn := <-manager.Unregister:
            // 断开连接事件
```
```

```
            manager.EventUnregister(conn)
        case message := <-manager.Broadcast:
            // 全部客户端广播事件
            clients := manager.GetClients()
            for conn := range clients {
                conn.SendMsg(message)
            }
        // 其他广播事件...
}

```

在这个 start 函数中，会注册监听很多事件，比如 Register，Unregister，就会影响到 client 的增加与减少，这些用户数据就是通过 ClientManager 中的 Clients 获取的。

```
// internal/websocket/client_manager.go
// GetClients 获取所有客户端
func (manager *ClientManager) GetClients() (clients map[*Client]bool) {
    clients = make(map[*Client]bool)
    manager.ClientsRange(func(client *Client, value bool) (result bool) {
        clients[client] = value
        return true
    })
    return
}

```

更多关于 clients 的操作都可以在 client_manager.go 中找到。

另外，除了这些，WebSocket 还可以向用户发送消息，就像上面代码中提到的广播事件。