Go: how to create a TCP client and server

Go, also known as Golang, is an open-source programming language developed by Google. Its simplicity, efficiency and ease of use make it perfect for building network applications such as clients and servers. In this article, well explore how to create a TCP client and server in Go.

What is TCP?

Transmission Control Protocol (TCP) is a connection-oriented, reliable communication protocol used to transfer data over a network. Being a reliable protocol, TCP ensures the delivery of data without errors and in correct order.

Create the TCP Server

Lets start by creating the TCP server that will listen for incoming connections and handle client requests. Here is the code for the TCP server in Go:

```
package main

import (
        "fmt"
        "net"
)

func handleConnection(conn net.Conn) {
        defer conn.Close()
```

```
        // Qui puoi gestire la connessione con il
client.
        // Ad esempio, puoi leggere i dati inviati
dal client o rispondere alle sue richieste.

        fmt.Println("Nuova connessione accettata:",
conn.RemoteAddr().String())

        // Esempio di lettura dei dati inviati dal
client
        buffer := make([]byte, 1024)
        n, err := conn.Read(buffer)
        if err != nil {
                fmt.Println("Errore durante la
lettura dei dati:", err)
                return
        }
        fmt.Printf("Dati ricevuti: %s\n", buffer[:n])
}

func main() {
        // Avvia il server sulla porta 8080
        listener, err := net.Listen("tcp", ":8080")
        if err != nil {
                fmt.Println("Errore durante l'avvio
del server:", err)
                return
        }
        defer listener.Close()

        fmt.Println("Server in ascolto su",
listener.Addr())
```

```
        for {
                // Accetta una nuova connessione
                conn, err := listener.Accept()
                if err != nil {
                        fmt.Println("Errore durante
l'accettazione della connessione:", err)
                        return
                }

                // Gestisci la connessione in modo
asincrono
                go handleConnection(conn)
        }
}
```

The server creates a socket listening on port 8080 using `net.Listen` . When a client connects, the connection is accepted using `listener.Accept()` , and the server will handle the connection asynchronously using the `go handleConnection(conn)` goroutine.

The `handleConnection` function is responsible for managing the connection with the client. Currently, it just prints data received from the client, but you can customize it to respond to client requests based on your needs.

Create the TCP Client

Now that we have our server up and running, lets create a TCP client that connects to the server and sends data. Here is the code for the TCP client in Go:

```
package main
```

```
import (
        "fmt"
        "net"
)

func main() {
        // Connetti il client al server
        conn, err := net.Dial("tcp",
"localhost:8080")
        if err != nil {
                fmt.Println("Errore durante la
connessione al server:", err)
                return
        }
        defer conn.Close()

        // Dati da inviare al server
        data := "Ciao, server! Sono un client."

        // Invia i dati al server
        _, err = conn.Write([]byte(data))
        if err != nil {
                fmt.Println("Errore durante l'invio
dei dati:", err)
                return
        }

        fmt.Println("Dati inviati al server:", data)
}
```

The client connects to the server using `net.Dial` and specifying the address and port of the server to connect to. After that, it sends the data to

the server using `conn.Write([]byte(data))` .

Running Server and Client

To run the server, save the server code in a file called `server.go` and the client code in a file called `client.go` . Next, open two terminals and navigate to both folders containing the respective client and server files.

In the first terminal, run the server:

```
go run server.go
```

In the second terminal, run the client:

```
go run client.go
```

You should see server output indicating receiving data from the client and client output indicating sending data to the server.