WebSocket Documentation

Release v0.1b

Jamillo Santos

Nov 08, 2017

User Documentation

The WebSocket is a golang implementation of websockets (RFC6455) on top of the fasthttp library.

The main documentation is divided in two sections:

CHAPTER 1

Getting started

This section will take you through the installation and your first server implementation.

1.1 Installation

To start using the library you must fetch it from the github using th `go get` command:

```
$ go get github.com/jamillosantos/websocket
```

1.2 Usage

This section will show how the WebSocket library will be integrated to your existing fasthttp server.

Starting with a basic integration, the modifications on the code will be added in steps until the server is done.

1.2.1 1. A simple fasthttp server

This is a simple code that will serve a message when the client access the root endpoint (/).

```
1  package main
2
3  import (
4      "github.com/valyala/fasthttp"
5      "fmt"
6  )
7
8  func main() {
9      server := &fasthttp.Server{}
10     server.Handler = func(ctx *fasthttp.RequestCtx) {
11         switch string(ctx.URI().Path()) {
```

```
        case "/":
            fmt.Fprint(ctx, "This is the root of the server")
        default:
            fmt.Fprint(ctx, "404 Not Found")
            ctx.SetStatusCode(fasthttp.StatusNotFound)
        }
    }

    server.ListenAndServe(":8080")
}
```

1.2.2 2. Setting the WebSocket library

With a few line line added, you will get your application responding to websocket connections.

```
package main

import (
    "github.com/jamillosantos/websocket"
    "github.com/valyala/fasthttp"
    "fmt"
)

func main() {
    server := &fasthttp.Server{}
    manager := websocket.NewListeableManager()
    manager.OnConnect = func(conn websocket.Connection) error {
        log.Println("Incoming client ", conn.Conn().RemoteAddr())
        return nil
    }
    manager.OnMessage = func(conn websocket.Connection, opcode websocket.MessageType,
↪payload []byte) error {
        log.Println("OnMessage", opcode, payload)
        return nil
    }
    manager.OnClose = func(conn websocket.Connection) error {
        log.Println("see ya", conn.Conn().RemoteAddr())
        return nil
    }
    upgrader := websocket.NewUpgrader(manager)
    server.Handler = func(ctx *fasthttp.RequestCtx) {
        switch string(ctx.URI().Path()) {
        case "/":
            fmt.Fprint(ctx, "This is the root of the server")
        case "/ws":
            upgrader.Upgrade(ctx)
        default:
            fmt.Fprint(ctx, "404 Not Found")
            ctx.SetStatusCode(fasthttp.StatusNotFound)
        }
    }

    server.ListenAndServe(":8080")
}
```

CHAPTER 2

Connection Session

Sometimes we need to attach some information to a connection that just started. In order to provide this functionality the `Connection interface` provides a `Context()` and `SetContext()` methods.

2.1 Example

```
package main

import (
        "github.com/jamillosantos/websocket"
        "github.com/valyala/fasthttp"
        "fmt"
        "log"
)

type ConnCtx struct {
        name string
}

func main() {
        server := &fasthttp.Server{}
        manager := websocket.NewListeableManager()
        manager.OnConnect = func(conn websocket.Connection) error {
                log.Println("Incoming client ", conn.Conn().RemoteAddr())
                conn.SetContext(&ConnCtx{
                        name: "John Doe",
                })
                return nil
        }
        manager.OnMessage = func(conn websocket.Connection, opcode websocket.
↪MessageType, payload []byte) error {
                ctx := conn.Context().(*ConnCtx)
                log.Println("message from", ctx.name, opcode, payload)
```

```
                return nil
        }
        manager.OnClose = func(conn websocket.Connection) error {
                log.Println("see ya", conn.Conn().RemoteAddr())
                return nil
        }
        upgrader := websocket.NewUpgrader(manager)
        server.Handler = func(ctx *fasthttp.RequestCtx) {
                switch string(ctx.URI().Path()) {
                case "/":
                        fmt.Fprint(ctx, "This is the root of the server")
                case "/ws":
                        upgrader.Upgrade(ctx)
                default:
                        fmt.Fprint(ctx, "404 Not Found")
                        ctx.SetStatusCode(fasthttp.StatusNotFound)
                }
        }

        server.ListenAndServe(":8080")
}
```

CHAPTER 3

How it works

This section will explain how the connection will work.

3.1 1. The client requests the connection

The websocket connection MUST start as a normal normal HTTP request. The browser will call the given endpoint with a set of special headers asking for a websocket connection.

It happens when the client instantiates a `new WebSocket` object passing the endpoint of our server:

```
1         var socket = new WebSocket("/ws");
2     ..
```

3.2 2. The server upgrades the connection

Once the server receives the connection, it will respond upgrading the connection.

CHAPTER 4

websocket.Manager

CHAPTER 5

websocket.Upgrader

CHAPTER 6

Indices and tables

- genindex
- modindex
- search