Package websocket

```
import "tic-tac-toe/websocket"
```


Overview

Das Paket 'websocket' dient der Implementierung von Websockets für das Spiel Tic Tac Toe. Die Implementierung basiert auf dem Repository https://github.com/gorilla/websocket.

API-Anfragen ¶

API-Anfragen sendet immer ein Client/Browser/Frontend. Diese benötigen folgendes Format und stellen einen Spielzug dar:

```
{
    "player": 1,
    "col": 2,
    "row": 0,
    "reset": false
}
```

API-Antworten

API-Antworten sendet immer der Server bzw. das Backend. Diese sind im folgenden Format und stellen den Spielstand dar:

```
{
    "finished": false,
    "winner": 0,
    "nextPlayer": 1,
    "field": ["X  ", "O  ", "   "]
}
```

Index

func (c *Client) writePump()
type Hub
func NewHub() *Hub
func (h *Hub) Run()

Package files

client.go doc.go handler.go hub.go

Constants

Deinition von Konstanten.

```
const (
    // Zeit, die für das Schreiben einer Nachricht an die Gegenstelle zur Verfügung
steht.
    writeWait = 10 * time.Second

    // Zeit, die zum Lesen der nächsten Pong-Nachricht von der Gegenstelle erlaubt
ist.
    pongWait = 60 * time.Second

    // Sendet Pings an die Gegenstelle mit dieser Zeitspanne. Muss kleiner sein als
pongWait.
    pingPeriod = (pongWait * 9) / 10

    // Maximal zulässige Nachrichtengröße von der Gegenstelle.
    maxMessageSize = 512
)
```

Variables

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

Definition der Lese- und Schreib-Puffer-Größe.

```
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
}
```

func **Serve**

```
func Serve(hub *Hub, w http.ResponseWriter, r *http.Request)
```

Serve verarbeitet Websocket-Anfragen von der Gegenstelle.

func **handleMessage**

```
func handleMessage(msg []byte, res *game.TicTacToe) []byte
```

handleMessage verarbeitet die Anfrage/Nachricht eines Clients und gibt eine Antwort zurück.

Die Nachricht ist der vom Client durchgeführte Spielzug. handleMessage ruft die Spiellogik auf und gibt den neuen Spielstand zurück.

func **welcome**

```
func welcome(ttt *game.TicTacToe) []byte
```

welcome formatiert den aktuellen Spielstand als JSON type []byte.

type **Client**

Client ist ein Verbinder zwischen der Websocket-Verbindung und dem Hub.

```
type Client struct {
    hub *Hub

    // Die Websocket-Verbindung.
    conn *websocket.Conn

    // Gepufferter Kanal für ausgehende Nachrichten.
    send chan []byte
}
```

func (*Client) readPump

```
func (c *Client) readPump()
```

readPump pumpt Nachrichten von der Websocket-Verbindung zum Hub.

Die Anwendung führt readPump, je Verbindung, in einer Go-Routine aus. Die Anwendung stellt sicher, dass es höchstens einen Leser auf einer Verbindung gibt, indem sie alle Lesevorgänge aus dieser Go-Routine ausführt.

func (*Client) writePump

```
func (c *Client) writePump()
```

writePump pumpt Nachrichten vom Hub an die Websocket-Verbindung.

Für jede Verbindung wird eine Go-Routine gestartet, die writePump ausführt. Die Anwendung stellt sicher, dass es höchstens einen Schreiber für eine Verbindung gibt, indem alle Schreibvorgänge von dieser Go-Routine aus ausgeführt werden.

type **Hub**

Hub verwaltet die Menge der aktiven Clients und sendet Nachrichten an die Clients.

```
type Hub struct {
    // Registrierte Clients.
    clients map[*Client]bool

    // Eingehende Nachrichten von den Clients.
    broadcast chan []byte

    // Registrieren von Anfragen der Clients.
    register chan *Client

    // Aufheben der Registrierung von Anfragen der Clients.
    unregister chan *Client
}
```

func NewHub

```
func NewHub() *Hub
```

NewHub ist der Konstruktor zur Initialisierung eines neuen Hubs.

func (*Hub) Run

```
func (h *Hub) Run()
```

Run startet den Hub, um die Anfragen (register, unregister, message) der Clients zu verwalten.

Die Anwendung startet eine Go-Routine für die Funktion Run.