**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №4

# «Система асинхронного парсинга и отображения матчей по Dota 2 с использованием Goquery»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-35Б |  | Гапанюк Ю. Е. |
| Бердников Н.О. |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г

**Описание программы:** Программа предназначена для парсинга информации о матчах Dota 2 с веб-сайта и отображения этих данных через веб-интерфейс. Основные функции программы включают:

Парсинг матчей: Программа использует библиотеку Goquery для извлечения данных о матчах, таких как команды, игроки, счет, статус матча и дата. Парсинг осуществляется для нескольких матчей одновременно с использованием горутин для ускорения процесса.

Отображение матчей: Программа предоставляет веб-интерфейс для просмотра списка последних матчей, а также детализированную информацию по каждому матчу. Пользователь может загрузить больше матчей, а также просмотреть подробности конкретного матча, включая составы команд и результат.

Асинхронность: Для загрузки матчей используется механизм синхронизации (WaitGroup и мьютексы)

Веб-интерфейс: Для отображения информации о матчах и их деталях используются HTML-шаблоны, которые динамически заполняются данными, полученными с веб-сайта.

**Текст программы:**

**Main.go:**

package main

import (

"fmt"

"log"

"net/http"

"strconv"

"sync"

"text/template"

"github.com/PuerkitoBio/goquery"

)

type MatchStatus int

const (

Draw MatchStatus = iota // 0

FirstTeamWin // 1

SecondTeamWin // 2

)

type Match struct {

ID string

FirstTeam string

SecondTeam string

FirstTeamPlayers []string

SecondTeamPlayers []string

Score [2]int

MatchStatus MatchStatus

Date string

}

func NewMatch(id string, firstTeam string, secondTeam string, firstTeamPlayers []string, secondTeamPlayers []string, score [2]int, matchStatus MatchStatus, date string) \*Match {

return &Match{

ID: id,

FirstTeam: firstTeam,

SecondTeam: secondTeam,

FirstTeamPlayers: firstTeamPlayers,

SecondTeamPlayers: secondTeamPlayers,

Score: score,

MatchStatus: matchStatus,

Date: date,

}

}

var matches []Match

var mu sync.Mutex

func parseMatch(matchNumber int, in chan<- Match, wg \*sync.WaitGroup) {

defer wg.Done()

id := strconv.Itoa(matchNumber)

url := "https://www.cybersport.ru/matches/dota-2/" + id

res, err := http.Get(url)

if err != nil {

log.Printf("Ошибка при получении матча %s: %v", id, err)

return

}

defer res.Body.Close()

if res.StatusCode != 200 {

log.Printf("Status code error: %d %s", res.StatusCode, res.Status)

return

}

doc, err := goquery.NewDocumentFromReader(res.Body)

if err != nil {

log.Fatal(err)

}

var teams [2]string

doc.Find("div.participantTitle\_QqRL7").Each(func(i int, s \*goquery.Selection) {

res, \_ := s.Html()

teams[i] = res

})

var players [10]string

doc.Find("div.playerHeader\_Ul3yT span").Each(func(i int, s \*goquery.Selection) {

res, \_ := s.Html()

players[i] = res

})

var score [2]int

doc.Find("div.matchScore\_N3WUO span").Each(func(i int, s \*goquery.Selection) {

res, \_ := s.Html()

score[i], \_ = strconv.Atoi(res)

})

var matchStatus MatchStatus

if score[0] == score[1] {

matchStatus = Draw

} else if score[0] > score[1] {

matchStatus = FirstTeamWin

} else {

matchStatus = SecondTeamWin

}

var date string

doc.Find("div.matchTime\_ji1GK").Each(func(i int, s \*goquery.Selection) {

res, \_ := s.Html()

date = res

})

match := \*NewMatch(

id,

teams[0],

teams[1],

players[:5],

players[5:],

score,

matchStatus,

date,

)

in <- match

}

func matchHandler(w http.ResponseWriter, r \*http.Request) {

id := r.URL.Query().Get("id")

if id == "" {

http.Redirect(w, r, "/", http.StatusSeeOther)

return

}

// Ищем матч по ID

var match Match

mu.Lock()

for \_, m := range matches {

if m.ID == id {

match = m

break

}

}

mu.Unlock()

if match.ID == "" {

http.Error(w, "Матч не найден", http.StatusNotFound)

return

}

tmpl := `

<!DOCTYPE html>

<html>

<head>

<title>Матч {{.FirstTeam}} vs {{.SecondTeam}}</title>

</head>

<body>

<h1>{{.FirstTeam}} vs {{.SecondTeam}}</h1>

<h2>ID {{.ID}} </h2>

<p>Дата: {{.Date}}</p>

<p>Счет: {{index .Score 0}} - {{index .Score 1}}</p>

<h2>Игроки</h2>

<h3>{{.FirstTeam}}</h3>

<ul>

{{if .FirstTeamPlayers}}

{{range .FirstTeamPlayers}}

<li>{{.}}</li>

{{end}}

{{else}}

<li>Нет данных об игроках</li>

{{end}}

</ul>

<h3>{{.SecondTeam}}</h3>

<ul>

{{if .SecondTeamPlayers}}

{{range .SecondTeamPlayers}}

<li>{{.}}</li>

{{end}}

{{else}}

<li>Нет данных об игроках</li>

{{end}}

</ul>

<a href="/">Назад к списку матчей</a>

</body>

</html>

`

// Парсинг шаблона и проверка на ошибки

t, err := template.New("matchDetail").Parse(tmpl)

if err != nil {

http.Error(w, "Ошибка при рендеринге страницы", http.StatusInternalServerError)

return

}

// Выполнение шаблона и вывод данных

err = t.Execute(w, match)

if err != nil {

http.Error(w, "Ошибка при выводе данных на страницу", http.StatusInternalServerError)

return

}

}

func loadHandler(w http.ResponseWriter, r \*http.Request) {

loadMatches(5)

http.Redirect(w, r, "/", http.StatusSeeOther)

}

func mainHandler(w http.ResponseWriter, r \*http.Request) {

tmpl := `

<!DOCTYPE html>

<html>

<head>

<title>Список матчей</title>

</head>

<body>

<h1>Последние матчи</h1>

<ul>

{{range .Matches}}

<li><a href="/match/?id={{.ID}}">{{.FirstTeam}} vs {{.SecondTeam}} ({{.Date}})</a></li>

{{else}}

<li>Нет матчей</li>

{{end}}

</ul>

<form action="/load/" method="get">

<button type="submit">Показать еще</button>

</form>

</body>

</html>

`

// Передача данных в шаблон

t, \_ := template.New("matches").Parse(tmpl)

data := struct {

Matches []Match

}{

Matches: matches,

}

t.Execute(w, data)

}

func main() {

http.HandleFunc("/", mainHandler)

http.HandleFunc("/match/", matchHandler)

http.HandleFunc("/load/", loadHandler)

fmt.Println("Starting server at :8080")

loadMatches(5)

err := http.ListenAndServe(":8080", nil)

if err != nil {

panic(err)

}

}

func loadMatches(count int) {

in := make(chan Match)

lastID := 10074000

if len(matches) != 0 {

lastID, \_ = strconv.Atoi(matches[len(matches)-1].ID)

}

var wg sync.WaitGroup

for n := lastID; n >= lastID-count; n-- {

wg.Add(1)

go parseMatch(n, in, &wg)

}

go func() {

wg.Wait()

close(in)

}()

for match := range in {

mu.Lock()

matches = append(matches, match)

mu.Unlock()

}

}

**Результат вывода:**

Страница просмотра всех матчей /  
****

Страница детального просмотра матча /match/{id}

****

**Использованные источники:**

1. [Электронный ресурс] Курс ПиКЯП (Парадигмы и конструкции языков программирования). URL:<https://github.com/ugapanyuk/courses_content/wiki/COURSE_PCPL_MAIN>