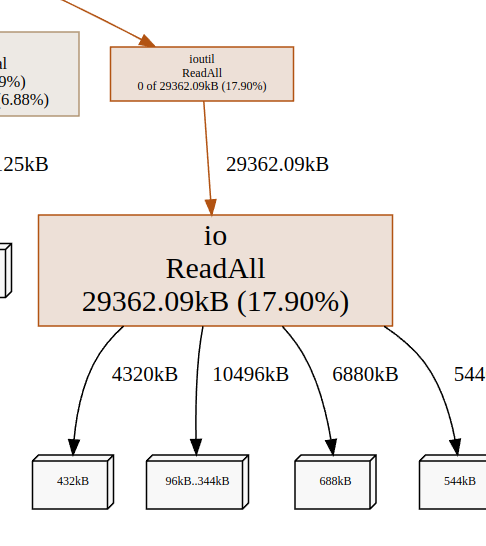
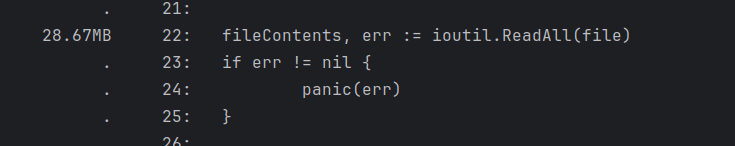
1.

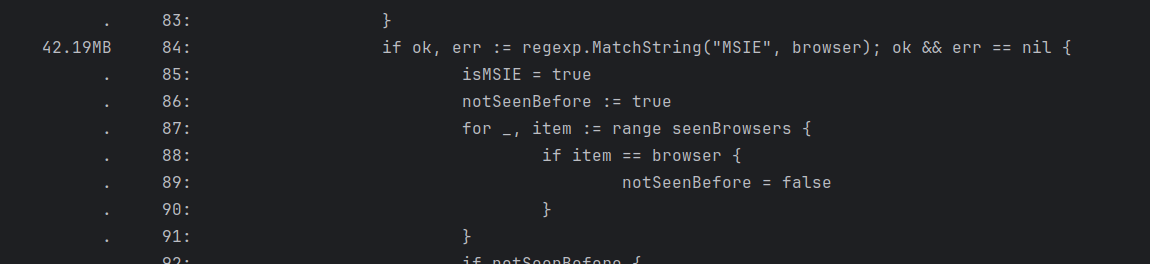
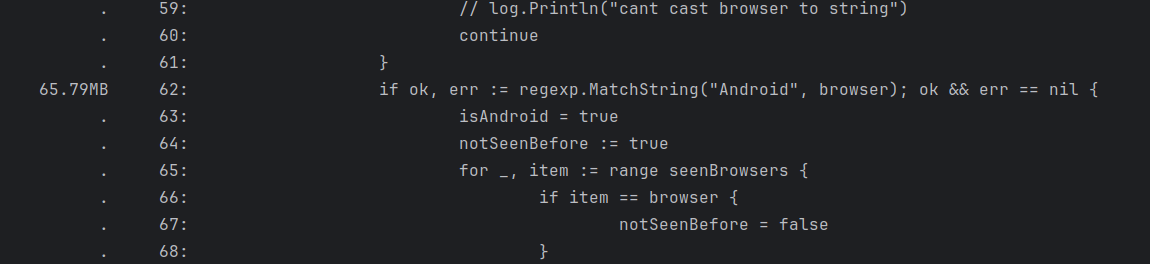
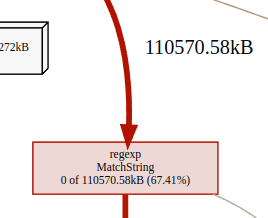


scanner := bufio.NewScanner(file)  
  
for scanner.Scan() {  
 line := scanner.Text()  
 user := make(map[string]interface{})  
 // fmt.Printf("%v %v\n", err, line)  
 err := json.Unmarshal([]byte(line), &user)  
 if err != nil {  
 panic(err)  
 }  
 users = append(users, user)  
}  
  
if err := scanner.Err(); err != nil {  
 panic(err)  
}



Заменил на

2.



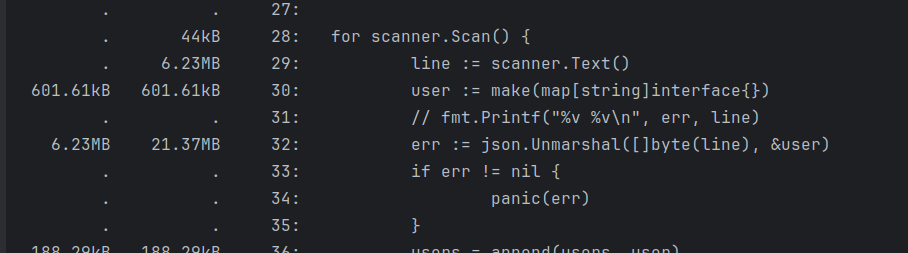
Заменил на

andr, err := regexp.Compile("Android")  
if err != nil {  
 panic(err)  
}  
mse, err := regexp.Compile("MSIE")  
if err != nil {  
 panic(err)  
}

if andr.MatchString(browser) {

if mse.MatchString(browser) {

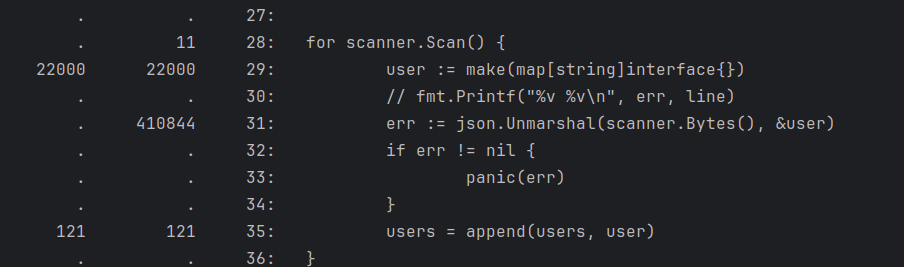
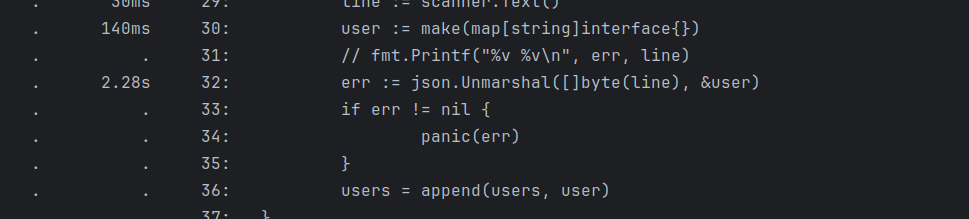
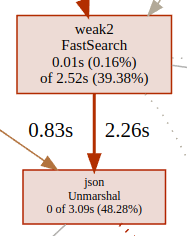
3.



Заменил на:

for scanner.Scan() {  
 user := make(map[string]interface{})  
 // fmt.Printf("%v %v\n", err, line)  
 err := json.Unmarshal(scanner.Bytes(), &user)  
 if err != nil {  
 panic(err)  
 }  
 users = append(users, user)  
}

4.

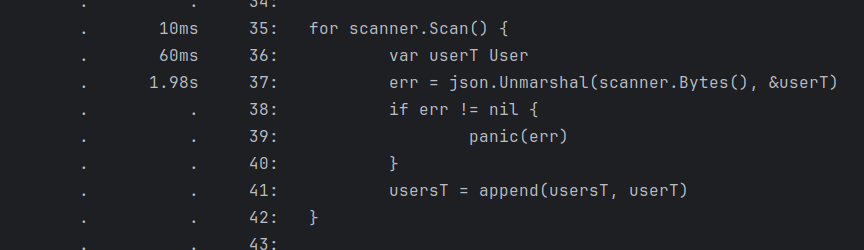
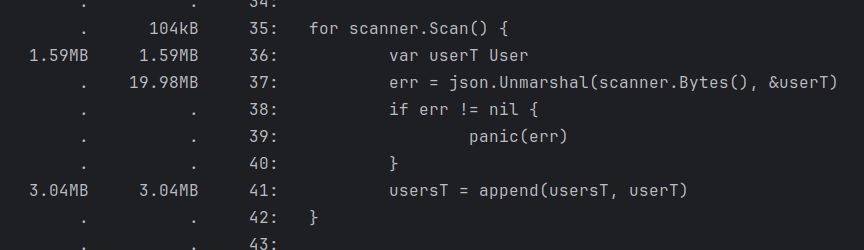
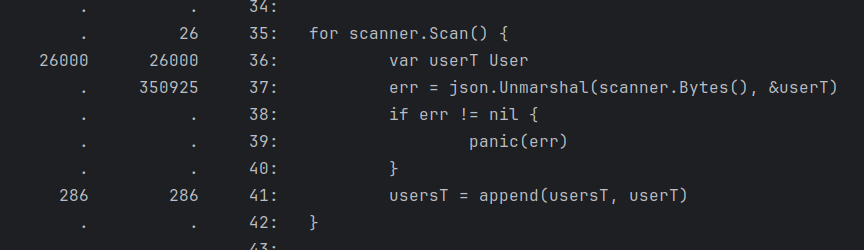


Заменил сохранение данных в мапу на структуру

type User struct {  
 Browsers []string `json:"browsers"`  
 Email string `json:"email"`  
 Name string `json:"name"`  
}

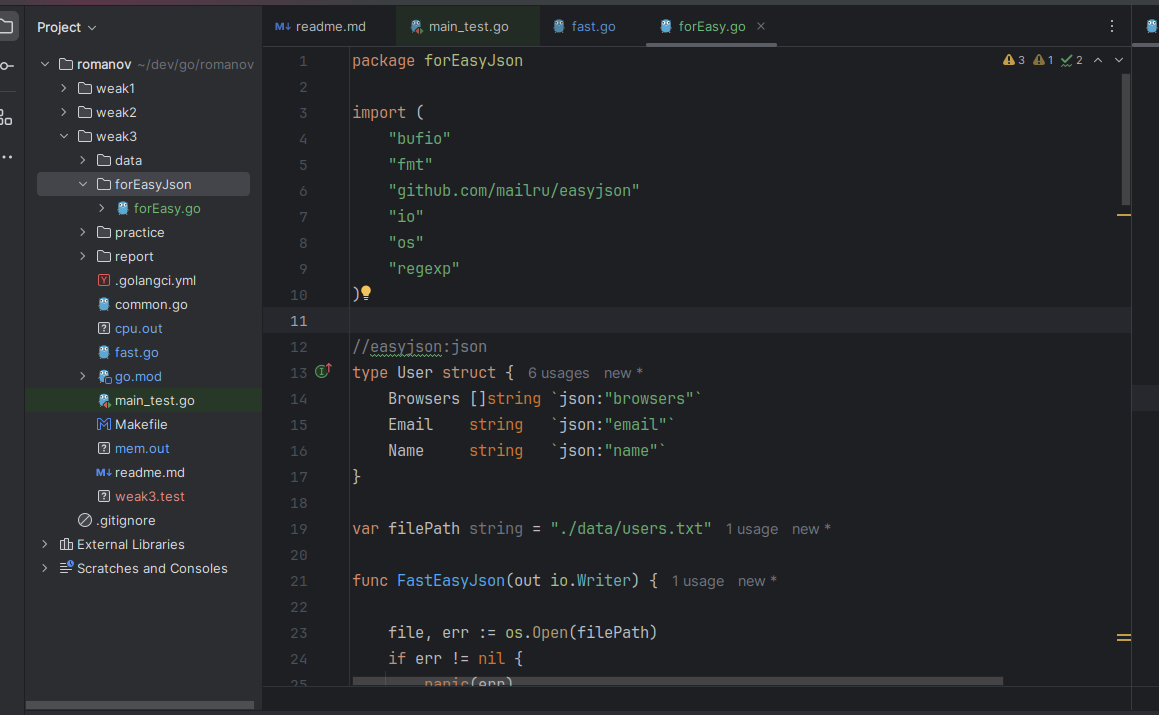
for scanner.Scan() {  
 var userT User  
 err = json.Unmarshal(scanner.Bytes(), &userT)  
 if err != nil {  
 panic(err)  
 }  
 usersT = append(usersT, userT)  
}

5.

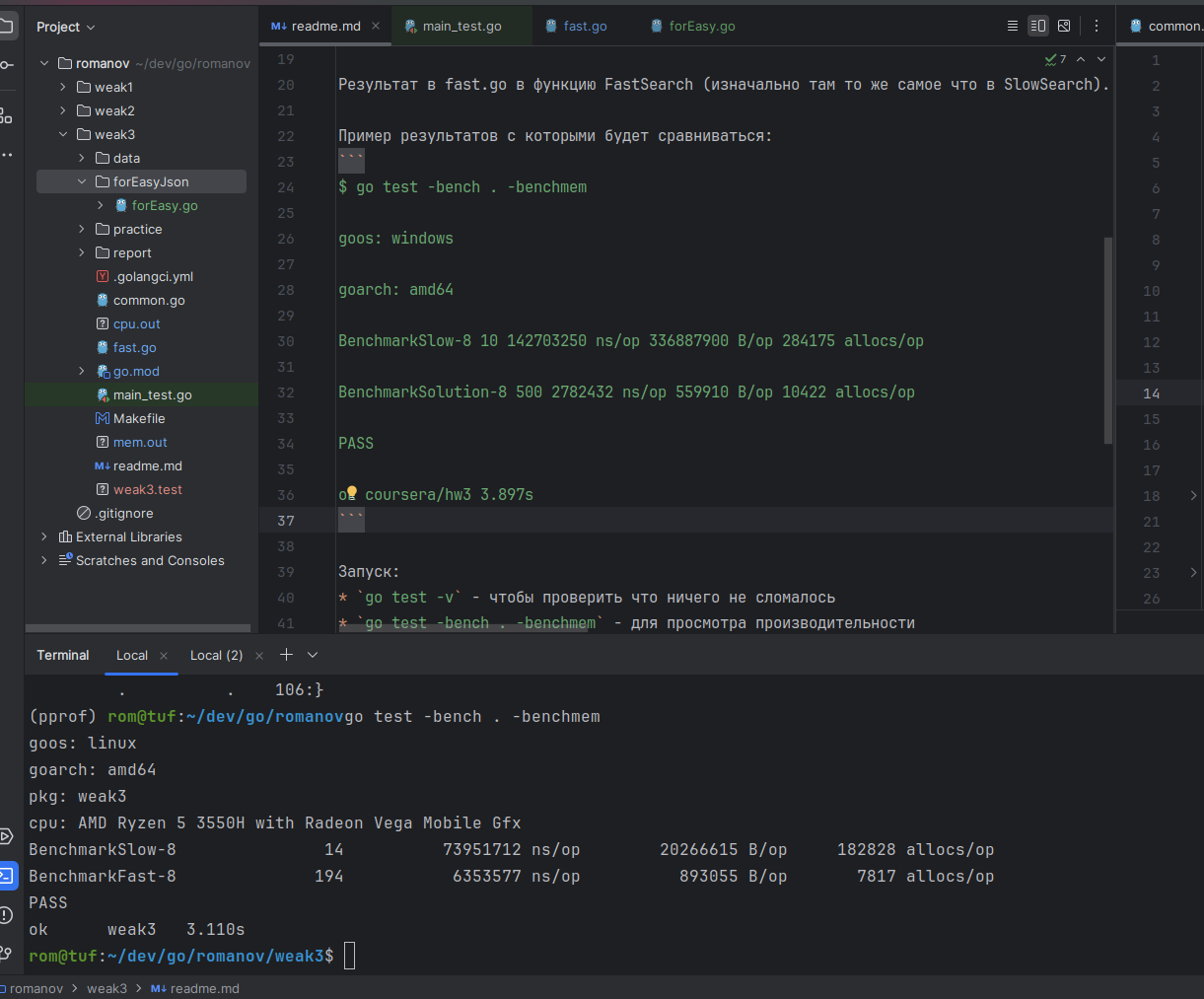


На этом этапе буду подключать easyjson, т.к. самым неоптимизированным местом сейчас является Unmarshal

Вынес основной код в отдельный пакет + использовал easyjson



В итоге имею следующие результаты:



Мой код вызывает меньше аллокаций памяти на одну операцию