Relazione Wordle 3.0 - Tommaso Mangiavacchi

Indice

Architettura e scelte di implementazione

Il progetto è articolato in 7 classi, 2 file json e 1 file txt fornito con il progetto.

- WodleServerMain.java
- WordleServer.java
- SecretWordTask.java
- ServerTask.java
- WordleClientMain.java
- NotificationTask.java
- UserData.java
- config.json
- data.json.json
- words.txt

WodleServerMain

Si tratta di una piccola classe main, server side, che amministra l'avvio iniziale del server e i successivi riavvii.
Crea un Task chiamato "WordleServer", e lo manda in esecuzione in un Thread.

Successivamente stampa sulla console la stringa **"Press 1 to reboot"** e si blocca in attesa che l'amministratore del server digiti il suddetto comando qual'ora intenda riavviare il server.
In caso di riavvio, viene chiamato un metodo *.interrupt()* sul thread WordleServer e si attende finche non termina l'esecuzione.
Nel caso in cui l'amministratore inserisca un comando errato(non "1"), si stampa un messaggio di errore e il ciclo si ripete.

WordleServer

Assieme al task "ServerTask" rappresenta il core delle operazioni effettuate server-side, tra le varie operazioni le più importanti sono:

- estrazione dei dati di configurazione memorizzati in config.json
- definizione di una **ConcurrentHashMap** in cui verranno memorizzate coppie <Username, UserData>, in modo atomico
- estrazione dei dati degli utenti memorizzati in data.js e creazione di oggetti UserData contententi i dati, tali oggetti verranno inseriti nella ConcurrenthashMap

- definizione di un **ServerSocket** che accetti le connessioni con i vari client
- esecuzione di un **CachedThreadPool** per fornire il servizio ai vari client
- esecuzione di un **ScheduledExecutorService** che esegue il task "SecretWordTask" in modo periodico, con periodo definito nel file di configurazione
- prima della terminazione, salva i dati in data.json e rimuove le coppie <Username, UserData> dalla ConcurrentHashMap**.**

SecretWordTask

Si tratta di un Task mandato in esecuzione sullo ScheduledExecutorService nella classe WordleServer.
Genera, in modo casuale, una parola presente nel vocabolario words.txt.

ServerTask

Si occupa dell'interazione tra client e server.
La classe è costituita dai seguenti metodi:

- **register():** chiede all'utente username e password, se esiste già un utente con lo stesso username, manda un messaggio di errore.
- **login():** chiede all'utente username e password, se non esiste un utento con tale username oppure esiste ma la passoword non coincide, da errore, altrimenti annuncia esito positivo e permette al client di impartire i comandi sottostanti, al server.
- **logout():** esegue il logout.
- **playWORDLE():** se non è stata ancora estratta la nuova SW e il client prova a giocare ancora, il server da errore, altrimenti da esito positivo.
- **sendWord():** permette al client di indovinare la parola segreta. Se il client ha già utilizzato la parola oppure non appartiene al vocabolario, non vengono decrementati i tentativi a disposizione.
- **sendMeStatistics():** mostra le statistiche dell'utente aggiornata all'ultima parita
- **share():** il task spedisce un datagramma, contenente l'esito dell'ultima partita, indirizzato ad un gruppo multicast.
- **showMeSharing():** quando l'utente seleziona questo comando, il server manda un messaggio speciale al client e il client legge le notifiche dalla apposita

struttura dati costruita per memorizzare i messaggi recepiti

La comunicazione avviene tramite l'ausilio degli oggetti Scanner, per leggere lo stream di Input associato al socket e PrintWriter per scrivere nello stream di Output anch'esso associato al medesimo socket.
Il Task mostra ciclicamente un menu dei comandi al client, se il client immette un comando non presente nel menu, il server risponde con il messaggio **Bad input, try again** .

WordleClientMain

All'avvio, legge il file di config.json e si configura in base ad esso, quindi imposta un timout al socket, definisce l'indirizzo del server al quale si connetterà, definirà un timeout, ecc….
Si connette al server, e dopo l'instaurazione della comunicazione, predispone un thread che rimane in ascolto di notifiche in un gruppo multicast.
Le notifiche saranno memorizzate in un **ArrayList<String>** e non saranno recuperabili dopo la terminazione del processo client.
Al termine del processo saranno chiusi socket, NotificationTask e i vari stream.

NotificationTask

Il costruttore prende come parametro, oltre al timeout e alla struttura dati per la memorizzazione, l'indirizzo e la porta del gruppo multicast in cui rimarrà in ascolto.
Crea un **MulticastSocket** e si unisce al gruppo multicast, poi rimane in ascolto.
Alla termine del processo, esce dal gruppo multicast e chiude il **MulticastSocket**.

Usedata

Classe costituita da metodi getter e setter per modificare e salvare i dati dell'utente e i suoi progressi.
Dopo ogni vincita/sconfitta, si sostituisce l'oggetto Userdata presente nella ConcurrentHashMap con uno aggiornato all'ultima partita.

Config.json

```
{
    "server_hostname":   "localhost",
    "server_port":   10000,
```

```
    "timeout":   2000,
    "multicastAddress":   "239.255.255.250",
    "multicastPort":   45588,
    "secretWordRate":   20,
    "datafile":   "data.json",
    "wordfile":   "words.txt"
}
```

Strutture dati Server-Side

ConcurrentHashMap<String, UserData> hashMap

> Struttura dati concorrente in cui verranno salvate coppie (Username,Userdata).
> Un oggetto *UserData* è composto da variabili in cui vengono allocati i dati sull'utente.

```
public class UserData {
    private final String password;
    private int gamesWon=0;
    private int lastStreak=0;
    private int maxStreak=0;
    private int playedMatches=0;
    private double guessDistribution=0;
    //metodi setter
    //metodi getter
    .
    .
    .
    .
}
```

ArrayList<String> hints

> Contiente i suggerimenti dell'ultima partita.
> Esempio: il client seleziona

ArrayList<String> guessedWords:

> Contiene le guessed words inviate nell'ultima partita.

> l’opzione sendWord(), manda “frogflower” e il server genera il suggerimento confrontanto la guessedWord con la SecretWord.
> L’array è usato anche dal metodo Share().

> Utile perché, insieme a “hints”, permette al giocatore di vedere lo stato della partita(GW precedenti con i rispettivi suggerimenti),

ArrayList<String> words

> Array in cui vengono caricato il vocabolario presente nel file words.txt.
> È stato scelto di caricare la parole in una struttura dati attiva nel server in modo da rendere lo scorrimento e accesso, molto più veloce rispetto ad un file testuale.

Data.json

> File in cui sono salvati i dati degli utenti, utilizzato dal server per ricostruire lo stato del sistema prima del riavvio.
> Esempio:

```
[
  {
    "username": "filippo",
    "password": "pluto29$",
    "maxStreak": 4,
    "lastStreak": 3,
    "playedMatches": 15,
    "gamesWon": 6,
    "guessDistribution": 6.0
  },
  ........
]
```

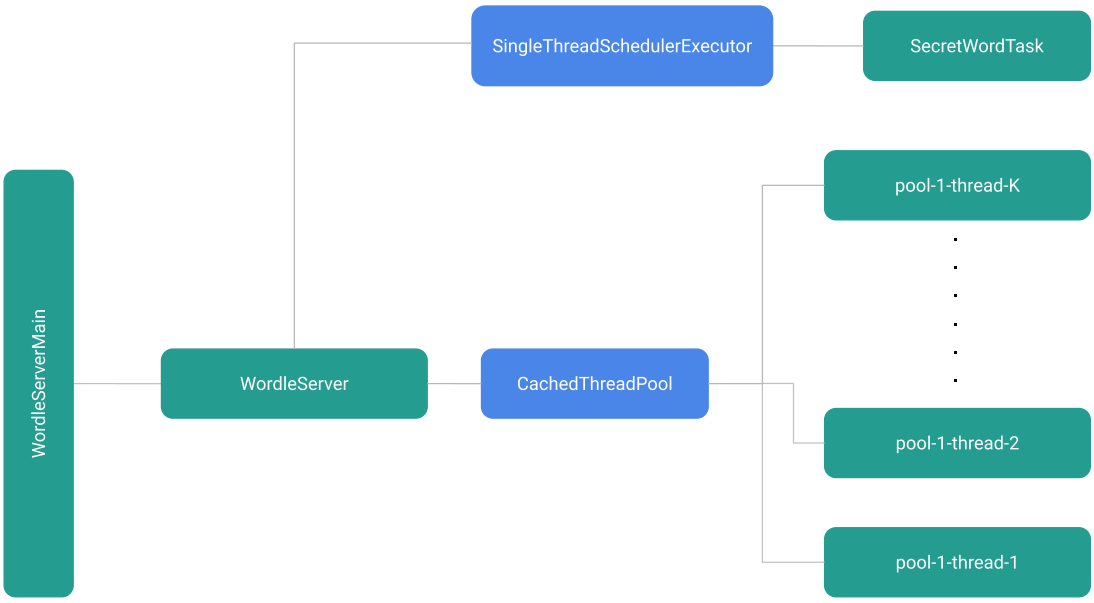
Strutture Dati Client-Side

ArrayList<String> notifications

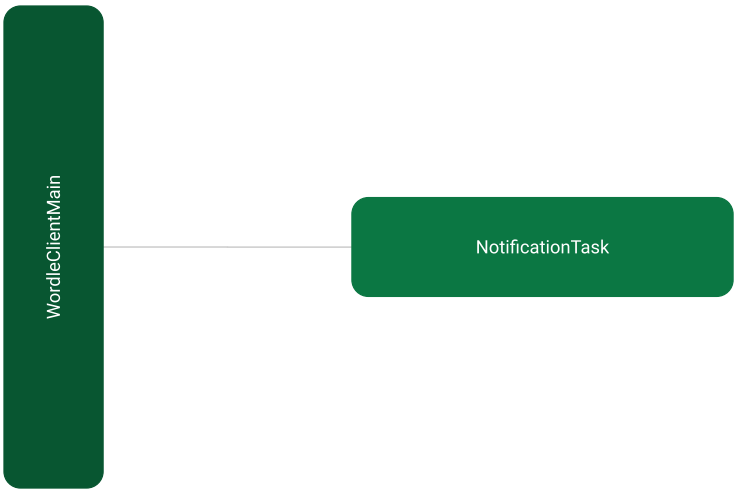
> Struttura dati definita per memorizzare le notifiche ricevute nel MulitcastSocket in ascolto nel thread NotificationTask.

Schema dei Thread

Server-side:



Client-side:



Sincronizzazione e strutture dati condivise

L’unica struttura dati condivisa che necessita di sincronizzazione è l’hashMap creata dal processo WordleServer e condivisa tra processi del ThreadPool, anch’esso creato in WordleServer.
Per rendere definire una classe Thread Safe si potevano prendere due vie:

- creare una hashMap e sincronizzarla con i monitor

Oppure:

- fare uso di una **ConcurrentHashMap** delle Concurrent Collections

Il progetto fa uso della seconda opzione, ovvero di una struttura dati con strategia di locking **fine-grained.**

Istruzioni per l’uso

Le classi WordleSeverMain e WordleClientMain non necessitano di nessun argomento passato da riga di comando.

L’unica libreria esterna utilizzata è GSON.

Per selezionare i comandi mostrati dal server non basta inserire altro che numeri mostrati nel menu.

Esempio: se è il nostro primo accesso e ci si vuole registrare, dovremo inserire “1” e premere invio, poi inserire il nome utente con cui vogliamo registrarci e premere invio, poi la password…ecc…

Qual’ora l’utente inserisca un comando non sintatticamente valido(quindi un numero non presente nel menu stampato su riga di comando), il server manderà una stringa di errore e riproverà chiedere l’input.

GitHub

https://github.com/Usnebd/Wordle