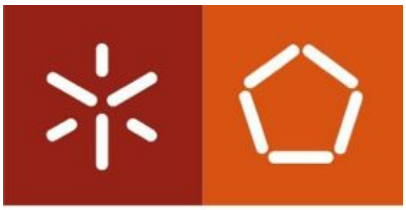

Universidade do Minho

Escola de Engenharia

Departamento de Informática

Mestrado Integrado Engenharia Informática

Paradigmas de Sistemas Distribuídos
2015/2016

“Chat Server”

Elementos do Grupo:

- João Lopes A61077
- Ricardo Varzim A51814

Introdução

O projecto a desenvolver tem como objectivo o desenvolvimento de um sistema que possua as funcionalidade de um servidor de chat.

Todo o sistema é implementado em JAVA utilizando ainda outros paradigmas para várias partes integrantes do mesmo. Desta forma, o sistema é constituído por um servidor que tem de ser capaz de dar resposta aos vários pedidos que lhe são enviados. Tais pedidos podem provir de três tipos diferentes de clientes.

O servidor tem de ser capaz de suportar uma grande quantidade de utilizadores ligados em simultâneo e para tal são utilizados actores através do recurso ao quasar. Uma vez que são utilizados actores, o servidor é portanto orientado à mensagem, onde são trocados pedidos e informação entre diferentes actores. O servidor deve ser constituído por várias salas de chat onde cada sala possui um conjunto de utilizadores que interagem com os restantes utilizadores da mesma sala.

Um primeiro tipo de cliente dá-se o nome de "end-users" que são os clientes que interagem com o sistema por forma a o utilizar para as suas funcionalidades mais básicas que incluem, para além do envio de mensagens, a consulta de salas que estão disponíveis e escolha de uma para interagir e ainda consultar os utilizadores presentes em cada sala. Este tipo de cliente deve ainda suportar o envio de mensagens recorrendo ao Telnet.

Deve existir ainda um Administrador que é considerado um cliente e que deve interagir com o servidor a partir de comunicação ZeroMQ. O principal papel deste cliente é a manutenção do servidor no que diz respeito à criação e remoção de salas bem como espectar os utilizadores de cada sala.

Por ultimo, existe ainda um terceiro cliente, Consola de Notificações, cujo objectivo é a subscrição a eventos como a criação ou remoção de salas sendo notificado sempre que algo a que tenha subscrito aconteça no servidor. Esta subscrição a eventos deve ser também efetuada através da comunicação ZeroMQ.

No seguimento deste relatório iremos explicar a forma como as várias partes integrantes do sistema foram implementadas bem como uma explicação resumida da forma como os vários actores interagem entre si.

Estruturação

Ambiente de Trabalho

O desenvolvimento da aplicação foi totalmente desenvolvida no IDE NetBeans 8.0.2. Em caso de execução no mesmo IDE pode ser necessário fazer os seguintes passos:

- Versão do JAVA actualizada;
- No projecto com actores (Trab1_PSD_Servidor) fazer o download das dependências clicando na pasta "Dependencies" e verificar que no ficheiro build.gradle, no "run", jvmArgs '-Djava.library.path=/usr/local/lib' está correto o caminho.
- Para os outros dois projetos basta incluir o jar do zmq na pasta Libraries e nas propriedades de cada um, no separador "run -> VM Options" colocar: -Djava.library.path=/usr/local/lib caso esse seja o caminho correto.

Servidor

Como referido anteriormente, o servidor é constituído por actores que comunicam entre si.

Existem três actores principais que são iniciados aquando o arranque do servidor: *Acceptor*, *RoomManager* e *Login*.

É da responsabilidade do actor *Acceptor* ficar à espera de novas ligações de end-users. Para tal toma partido de *FiberSocketChannel*s. Sempre que um cliente se liga ao socket, a partir da porta 12345 definida por nós, é criado um novo actor de nome *User* que representa o cliente do lado do servidor. Cada actor User vai então receber pedidos do seu cliente e por cada pedido reconhecido é enviada uma mensagem a um outro actor que seja o mais apropriado para iniciar o processo de dar resposta ao cliente.

O actor *Login* possui a informação referente aos cliente que possuem conta criada no sistema. Sempre que um cliente tenta utilizar o sistema tem primeiro de criar conta sendo passado o nome de utilizador e password do actor User para o actor Login onde vai ser armazenado. Depois de possuir conta no sistema, tem também de fazer login para o utilizar. Neste caso, cada actor User possui uma variável que indica se o mesmo tem ou não login efetuado. Caso não tenha login efetuado, é enviada uma mensagem de pedido de login do actor User para o actor Login sendo enviada de volta uma mensagem

para o actor User a confirmar login efetuado caso o actor Login possua informação que o cliente tem conta criada no sistema.

RoomManager tem a responsabilidade de lidar com todos os pedidos que se relacionem com as salas de chat do servidor. Este possui o conjunto de salas presentes no sistema a partir da relação nome sala e referência para a mesma (HashMap<String, ActorRef>). Sempre que é criada ou removida uma sala ou um cliente tenta entrar numa certa sala ou até listar as salas e clientes em cada sala do sistema, é a partir da comunicação com este actor que é obtida uma resposta.

Frequentemente é necessário passar um conjunto de informação entre os vários actores. Para tal existe o método "*send()*" que envia um objecto. Na implementação escolhida pelo grupo, esse objecto é sempre uma *Message* (classe) que por sua vez tem um tipo que é definido numa variável do tipo *enum* e um outro objecto. O objecto que pertence à classe *Message* vai variar consoante o tipo de pedido e informação que esse pedido requer. Para tal, foram criadas um conjunto de classes consideradas mensagens que se encontram no mesmo package da classe *Message* que contém a informação que é necessária passar entre clientes, consoante o pedido efetuado.
A título de exemplo considere-se a classe:

```
//Tipo de mensagem utilizada quando um utilizador quer fazer login ou criar conta
public static class MessageTypeAutentication {
    public final String nome;
    public final String pass;
    public final ActorRef utilizador;

    public MessageTypeAutentication(String nome, String pass, ActorRef utilizador) {
      this.nome = nome;
      this.pass = pass;
      this.utilizador = utilizador;
    }
}
```

A classe acima vai ser o objecto da classe *Message* que vai conter a informação relevante para criar conta e/ou fazer login no sistema.

Todo o tipo de pedidos dos clientes para o sistema, seguem este padrão de troca de mensagens consoante o pedido efetuado.

Ainda no servidor existe um outro actor cujo objectivo é fazer a comunicação entre o cliente Administrador e o servidor. O Administrador comunica com o servidor a partir de ligações ZeroMQ. Como o papel do Administrador é o de fazer um pedido e esperar por uma resposta a esse pedido, então o padrão de comunicação apropriado é o de REQ-REP sendo o REPLY o que se encontra do lado do servidor. Este actor fica à escuta na porta 12346 e sempre que recebe uma mensagem verifica o tipo da mesma e envia

para o actor *RoomManager* uma mensagem em conformidade com o pedido do Administrador. Espera depois por uma resposta do actor RoomManager e envia essa resposta por reply para o Administrador.

Para ser possível enviar informação para o cliente de consola de notificações, que utiliza comunicação do tipo PUB-SUB, sendo o PUBLISHER o que se encontra do lado do servidor, no arranque do servidor é criado um ZMQ.Socket do tipo PUB e é utilizado esse socket em partes especificas do código dos vários actores (por exemplo, logo após à criação de uma sala) para enviar informação para a consola de notificações.

Cliente (end-user)

A aplicação cliente é constituída basicamente por duas Threads, uma para controlar o input do utilizador e enviar o mesmo para o servidor, a outra escutar e listar as mensagens provenientes do Servidor. A implementação é bastante simples, usamos uma Socket que inicialmente tenta a conexão ao servidor (caso falhe termina a ligação). De seguida, criamos as duas Threads de Comunicação (Entrada e Saída) que recebem a referência à Socket previamente conectada ao Servidor.

A Thread de Entrada é usada para receber as mensagens, sendo que a mesma bloqueia enquanto espera uma mensagem do servidor. Após receber a mensagem, trata a mesma e redireciona o resultado para a linha de comandos.

A Thread de Saída é usada para escutar o input proveniente do utilizador. Assim que uma mensagem for introduzida, após o tratamento da mesma é enviado para o servidor o resultado da mesma.

O facto de dividirmos o cliente em duas Threads tem como intuito tornar independentes os processos de leitura e escrita para o servidor. Assim evitamos que a aplicação cliente se bloqueie numa das operações enquanto a outra está pendente.

A lógica e interpretação dos comandos são deixados do lado do servidor, sendo que da parte do cliente apenas é tratado o caso de saída da aplicação. Assim sendo o cliente terá de respeitar os parâmetros dos comandos para que o servidor os interprete.
Em particular os comandos terão de terminar com um espaço e letra maiúscula (indiferente para os argumentos).

Este tipo de cliente pode ainda ser utilizado via Telnet com localhost e porta 12345.

Administrador

O Administrador utiliza comunicação via ZeroMQ do tipo REQ-REP sendo o REQUEST o que se encontra do lado do Administrador.

Assim que é iniciado é apresentada uma API com um conjunto de opções para escolha dos vários pedidos que são suportados pelo serviço.

Consola de Notificações

A Consola de Notificações utiliza, tal como o Administrador, comunicação via ZeroMQ mas do tipo PUB-SUB sendo o SUBSCRIBER o que se encontra do lado da consola.

Assim que é iniciado é apresentada uma API com um conjunto de opções para escolha das várias subscrições disponíveis que pode efetuar. Sempre que escolhe uma nova subscrição a mesma fica associada às subscrições anteriores, passando assim a receber notificações da nova subscrição e das anteriores.

Utiliza uma thread para poder estar à escuta no socket para não bloquear a API uma vez que é possível subscrever a um novo evento a qualquer momento.

Comunicação

Servidor/Cliente(end-users)

A comunicação dos clientes (end-users) com o servidor dá-se a partir do reconhecimento de pedidos/comandos juntamente com a possibilidade de mais informação consoante o pedido.

Desta forma, o servidor reconhece os seguintes comandos vindos dos end-users:

- *CREATE_ACCOUNT user pass* (cria uma conta no sistema);
- *LOGIN user pass* (faz login de uma conta que se encontre no sistema);
- *LIST_ROOMS* (lista o nome das salas disponíveis);
- *LIST_MY_ROOM_USERS* (que lista os utilizadores ativos na room a que pertence);
- *PRIVATE_MESSAGE userDestino message* (envia uma mensagem privada para o utilizador com nome “userDestino”);
- *CHANGE_ROOM newRoom* (sai da room a que pertence e vai para a “newRoom”);

- *LOGOUT* (faz logout);
- *EXIT* (sai da aplicação).

- Para enviar uma mensagem normal a todos os clientes da room a que pertence basta escrever a mensagem sem nenhum dos comandos acima referidos.

NOTA: Para que os comandos acima sejam reconhecidos é necessário colocar um espaço em branco no fim da totalidade do mesmo caso contrário vão ser reconhecidos como mensagens.

Administrador

A comunicação com o Administrador é feita através de uma API cujas possibilidades disponíveis/implementadas são:

- Listar as salas que existem;
- Criar uma sala com um determinado nome;
- Listar os utilizadores de uma determinada sala;
- Remover uma determinada sala.

Consoante a opção escolhida é feita uma ligação ZeroMQ com o servidor para dar resposta ao pedido.

Consola de Notificações

A Consola de Notificações disponibiliza uma API ao utilizador cujas opções disponíveis/implementadas são:

- Subscrição à criação de salas;
- Subscrição à remoção de salas;
- Subscrição à entrada de utilizadores em qualquer sala;
- Subscrição à saída de utilizadores de qualquer sala.

Como referido anteriormente, as opções disponíveis são acumulativas, sempre que é feita uma nova subscrição passa a receber informação dessa subscrição e das subscrições anteriormente efetuadas.

Conclusão

Neste trabalho colocamos em prática uma grande parte dos conhecimentos adquiridos até então nesta disciplina. De destacar as seguintes mais valias em termos de conhecimento adquirido como a utilização de actores na gestão de pedidos orientados à mensagem, padrões Publisher - Subscriber e Request - Reply do ambiente ZeroMQ.

Deparámo-nos com algumas dificuldades no decorrer deste trabalho, de salientar a implementação de ZeroMQ que tem um processo de instalação demorado e com muitos pormenores.

Ainda referente à utilização de ZeroMQ deparamo-nos com uma grande dificuldade em utilizar o mesmo para a ligação do Administrador com os actores do servidor devido a warnings contínuos de threads que estavam a bloquear o CPU que acontecia sempre que colocávamos um método "*recv"* dentro de um ciclo "*while(true)"*. A solução passou por criar uma thread especifica para lidar com esse tipo de ciclos nos métodos "*recv*".

Em relação à utilização de actores não houve dificuldades, após a compreensão da forma como as mensagens devem ser trocadas e captadas entre actores, a implementação passou por ser praticamente repetitiva.

Este trabalho foi importante para consolidar o conhecimento adquirido, sendo que a nosso ver é indispensável uma abordagem mais prática destas ferramentas para ter uma noção mais adequada do seu potencial e das suas dificuldades de utilização.