Universidade do Minho

Mestrado em Engenharia Informática

Paradigmas de Sistemas Distribuídos - 2014/2015

Banana Chat

9 de Janeiro de 2015

Duarte Duarte PG27715
Fábio Gomes PG27752
Hélder Gonçalves PG28505

Introdução

A computação distribuída ou sistema distribuído, é uma referência à computação paralela e descentralizada, realizada por dois ou mais computadores conectados através de uma rede, cujo objetivo é concluir uma tarefa em comum.

Durante o semestre existiu a oportunidade de entrar em contacto com várias técnicas e tecnologias de computação em sistemas distribuídos. Estas técnicas são muito importantes para a realização dos desafios propostos por este trabalho.

Um sistema de conversação pode ser considerado um sistema distribuído, em que existem um ou vários utilizadores/máquinas a enviar mensagens e outros a receber.

Para a realização deste trabalho é necessário ter em conta vários paradigmas de comunicação. O paradigma cliente/servidor é bastante importante num sistema distribuído. Além deste paradigma, são utilizadas várias tecnologias de *middleware* de forma a automatizar/facilitar todo o processo de comunicação. A utilização de programação baseada em actores é outro ponto essencial para atingir o objectivo pretendido.

Como se pode constatar, serão usadas uma mistura de várias técnicas de programação.

Servidor Principal - *Server*

Para que tudo isto funcione, o servidor tem que ser a primeira aplicação a ser iniciada. É ela que arranca o *Quasar*, a arquitectura *REST* e o *HandlerCurl* para administração. Para esta aplicação são reservadas as seguintes portas:

- 8080 - Cliente de administração via *browser* (HTML apenas para listagens)
- 12343 - Cliente de administração
- 12345 - Cliente normal de *chat*
- 12346 - Cliente de subscrição

Handler de Administração - *HandlerCurl*

Esta classe é responsável, tal como as outras de prefixo *Handler*, pela interação entre o utilizador (agora chamado de administrador) e o servidor *Quasar*. Apenas envia pedidos e recebe informações sobre esses mesmos pedidos.

São guardadas duas variáveis principais. A *quasar* que é uma referencia para o Quasar e um *HashMap* de *logins*. Quando esta *thread* se inicia, já temos esse tal *handler* pronto para comunicações. A porta para se ligar ao servidor é 12343. Enquanto que o administrador não envie informação que pretende sair, este *handler* fará todas as tarefas de administração que venham a pedido. São elas identificadas por índices.

1 - Listar Utilizadores

Faz-se um pedido via *listUsers()* ao quasar, iteramos os resultados e mostramos no ecrã.

2 - Criar Utilizadores

O administrador vai introduzir o nome de utilizador e a palavra-passe que pretende, em caso de já existir um com esse nome, uma mensagem de erro é indicada. Se tudo estiver bem, é recebida uma mensagem positiva e indicamos ao administrador que o novo registo foi efetuado

3 - Remover Utilizadores

Pergunta-se qual o nome do utilizador que ele pretende remover, é feito o pedido pela socket, depois mediante a resposta recebida informa-se o administrador.

4 - Listar Salas

É enviado um pedido *listRooms()*, recebida a lista, iteramos e é mostrado.

5 - Criar Salas

Recebido o nome pretendido para a nova sala e enviado o pedido, é recebida a confirmação da criação ou a informação que não foi possível.

6 - Remover Salas

O administrador indica qual o nome da sala a remover e de seguida saberá se tal aconteceu.

7 - Sair do Cliente de Administração

Fechada a *socket*, o ciclo é interrompido e o programa terminado.

Actores - *Quasar* e *HandlerQuasar*

Esta são as classes mais importantes no que diz respeito à comunicação. Na classe *HandlerQuasar* é iniciado o *Quasar* na porta que lhe for atribuída. Na classe *Quasar* são processadas todas as mensagens, quer sejam para enviar para outros utilizadores quer sejam comandos que despoletam acontecimentos. Esta classe inicia um *Acceptor* na porta que lhe for passada para aceitar novos utilizadores e inicia ainda o *Publisher*.

Msg

Esta subclasse é o que representa a mensagem, constituída por três atributos, um que é o tipo (será abordado mais à frente), outro a mensagem quando esta possui algum texto necessário e também o responsável pelo envio.

LineReader

Esta subclasse processa a mensagem e verifica se esta se trata de um comando ou de uma mensagem normal, caso seja um comando envia uma *Msg* para quem a tiver de tratar, caso contrario, sendo uma mensagem normal, envia uma *Msg* com o tipo *DATA* para o *userRef* (a referência deste actor).

User

Esta subclasse é responsável por manter a informação correspondente ao utilizador dessa instancia bem como proceder nos diferentes tipos de *Msg* que recebe. Pode ser uma mensagem de *LOGIN*, de

dados (isto é uma mensagem normal de comunicação) entre outras menos utilizadas tais como *LEAVE* e *EOF*.

Room

Esta subclasse é responsável por manter as referências de todos os *Users* que estão por ela referenciados, isto é que estão a interagir nessa sala de *chat*. Quando esta classe recebe uma mensagem de *ENTER* adiciona o *User* referenciado na *Msg* à sua *hash* de utilizadores que se encontram na sala.

Uma mensagem de *LEAVE* faz com que o utilizador deixe a sala em que está e se junte a uma nova sala que vem na *Msg* e por fim uma *Msg* que tenha o tipo *LINE* envia para todos os utilizadores referenciados pela sala a mensagem que recebe.

Acceptor

Esta subclasse é responsável por admitir novos utilizadores, se um utilizador se ligar ao *chat* será aceite aqui. Depois de um utilizador se ligar o *Acceptor* cria um actor do tipo *User* que será responsável por esta nova ligação e envia uma *Msg* com o tipo *ENTER* para a *mainroom* onde será autenticado.

Tipo das *Msg*

1. DATA - mensagem composta por texto já processado;
2. EOF - mensagem que termina uma ligação visto que só é atribuído este tipo se o utilizador fechar a conexão;
3. ENTER - mensagem que ordena a entrada numa nova sala;
4. LEAVE - mensagem que ordena a saída da sala em que se encontra;
5. LINE - mensagem que contém texto ainda não processado;
6. LOGON - mensagem que contém as credenciais de *login* para verificar se o utilizador se encontra autenticado;
7. PM - mensagem composta por uma mensagem normal e com o *username* do utilizador que a enviou (mensagem privada, apenas enviada para o destinatário mencionado);
8. HELP - mensagem que apenas envia todos os comandos disponiveis;
9. SUPDATE - mensagem que liga ou desliga as notificações do *publisher* (funciona como um *toggle*).

Publisher/Subscriber - Publisher e Subscriber

É certa a existência de utilizadores que pretendem estar a par de determinado tipo de eventos. Mas, nem todos os utilizadores são obrigados a receber informação de todos os eventos que ocorrem no servidor. Pode ser bastante maçador para um utilizador comum. Assim, é dada ao utilizador a hipótese de receber notificações dos eventos que pretende. A este utilizador vamos dar o nome de *subscriber*. Para um *subscriber* receber notificações de novos eventos é necessário que exista alguém responsável por avisar que determinado evento ocorreu. O responsável por esta tarefa tem o nome de *publisher*.

No servidor de *chat* implementado existe um *publisher* que publica quatro tipos de eventos:

- Publica o *login* de utilizadores no sistema;
- Publica o *logout* de utilizadores no sistema;
- Publica a criação de novas salas de conversação;
- Publica a desaverbação de salas de conversação.

O *publisher* é inicializado uma única vez. A porta definida para este tipo de comunicação é a 12346 via *TCP*, onde se liga uma *socket ZMQ*.

Assim, quando um dos eventos descritos acima ocorre o *publisher* só tem de enviar uma mensagem para o *socket* já definido. Agora o *ZMQ* será responsável por enviar esta mensagem a todos os *subscribers* que estejam à escuta.

Subscriber

Baseado nas implementações *ZMQ*, sabemos que esta classe tem uma *socket* do tipo *ZMQ.SUB*. Indicada a porta no construtor, é feita a ligação da tal *socket* via *tcp* para o *localhost*. Baseado na lista de canais recebida, é feito o *subscribe* a cada um desses canais.

Além do *doRun()* que será executado aquando da iniciação desta *thread*, há 2 métodos:

- *subscribeNewChannels*

  Recebe uma lista de canais novos e faz *subscribe* a todos eles juntando aos que tem.

- *stopSubscriber*

  É feito o *disconnect* na porta usual, 12346, e fechada a *socket*.

Publisher

Agora este é um *ZMQ.PUB* que contém métodos, chamados *publish*, que vão fazer envios das mensagens formatas relacionadas com o seu evento.

A sintaxe geral é “[evento] [data] nome_do_[cliente|sala]*”*.

Clientes

Cliente de Comunicação - *CommunicationClient*

É no programa cliente que os utilizadores vão ter acesso às funcionalidades típicas de um *chat*:

- Listar salas;
- Entrar e sair das salas;
- Enviar Mensagens para a sala de *chat* e de forma privada para um utilizador específico;
- Subscrever eventos;
- Listar os comandos disponíveis.

Para comunicar com o utilizador, via consola, e com o servidor, via *sockets*, é necessário utilizar *streams* de leitura e de escrita para ambos os casos.

O cliente é iniciado e feita a ligação ao servidor na porta 12345, as *streams* para o *standard I/O* de forma a que possa introduzir as credenciais de acesso. Feita a entrada da combinação *username/password*, é escrita para a tal *socket*. Depois o Cliente fica à espera do resultado do *login*. Em caso afirmativo são iniciadas duas *thread MsgReader* para fazer de ponte entre o I/O do Utilizador para a *socket* que liga ao Servidor, para assim trocarem mensagens. Esta classe de *threads* só termina quando o valor *exit* for alterado.

Existem os seguintes comandos possíveis para o utlizador:

- /jr - Mudar de sala;
- /mr - Mudar para a sala principal (*mainroom*);
- /pm - Enviar uma mensagem privada para um utilizador específico;
- /li - *Login;*
- /lo - *Logout;*
- /lr - Listar as salas disponíveis.

Cliente de Administração - *AdminClient*

Este cliente tem um papel muito importante para a gestão de todo o sistema. Ele é responsável pelo registo e desaverbação de utilizadores e salas, assim como a listagem dos utilizadores e salas existentes.
A aplicação só permite que esteja ligado um cliente administrador de cada vez. No caso de já existir um cliente administrador conectado e outro se tentar ligar, este vai ficar à espera até o outro fechar a conexão.

O cliente é bastante simples. Começa por se tentar conectar ao servidor que está à escuta, neste caso é a porta 12343. Quando conectado são criados quatro *buffers* de comunicação, dois de escrita e outros dois de recepção. Tem um de recepção e escrita para o ambiente do utilizador, normalmente o terminal, e outros dois para a *socket*. Assim que prontos os *buffers* de comunicação, o utilizador entra num ciclo até ordem de saída, definida pelo utilizador. Neste ciclo, o utilizador pode executar qualquer um dos comandos que estão disponibilizados. Cada operação retorna um resultado que é posteriormente transmitido ao utilizador em questão.

Cliente de Subscrição - *SubscribeClient*

Como pedido, uma implementação de protocolo *publisher/subscriber* foi adicionada. Utilizando o ZMQ, um conjunto de *subscribers* que ligam-se a um *publisher.*

Como se trata de um *subscriber*, o contexto do *ZMQ* é *ZMQ.SUB*. O Utilizador depois pode escolher entre as subscrições disponíveis qual (ou quais) quer subscrever. Existem:

- *Login* de utilizadores
- *Logout* de utilizadores
- Criação de salas
- Remoção de salas

Escolhidos os tipos de subscrição, é hora de fazer a subscrição ao *ZMQ*, para tal tem que se fazer *subscribe* dessas mensagens. Por fim fica bloqueado num ciclo enviando para o *output* as mensagens recebidas.

Conclusão

Há uma interface de administração utilizando *REST*. É possível fazer os mesmos pedidos usando comandos *curl* com os vários métodos de *request*, mas esta não tem uma interface própria. É sugerido que se utilize o cliente, pois é essa a sua principal função, facilitar a apresentação ao administrador dos vários tipos de acções.

Foram diversas as dificuldades que tivemos na implementação deste *chatserver*. A primeira grande dificuldade encontrada foi na junção de todos os *middlewares* utilizados. Pelo facto do *Maven* nunca ter sido usado por nós, houve dificuldades nas suas configurações iniciais, de modo a pôr a aplicação a funcionar.

Assim tivemos um projecto bastante interessante, onde pudemos por em prática todas as abordagem lecionadas nas aulas.