Sistemas Distribuídos

Relatório 2ª Meta

Miguel Alexandre da Mota Mendes Andrade Silva 501031238
Rafael José Mendes Marmelo 501031259
Richard Eduardo Nuno 501031269

Índice

1. Horas dispendidas

Meta 1: 72 horas
Meta 2: 23 horas

2. Introdução

A aplicação Connect4 foi elaborada no âmbito da disciplina de Sistemas Distribuídos. Pretende ser um jogo de quatro em linha num ambiente distribuído. Para tal, contamos com uma aplicação do lado do cliente e outra do lado do servidor. Para além de jogar quatro em linha, existe a possibilidade de conversar com o adversário enquanto joga. O utilizador poderá também conversar numa sala de conversação. Nessa sala de conversação todos os participantes que nela se encontram recebem as mensagens enviadas, aproximando-se portanto do conceito de *chat room*. Para além de se poder conversar com outras pessoas, é na sala de conversação que o utilizador poderá iniciar um novo jogo com outro participante e consultar a listagem das pontuações das pessoas que se encontram ligadas na altura. A aplicação suporta a possibilidade de se estar a jogar em vários jogos em simultâneo desde que seja com participantes diferentes. Nada impede no entanto que os jogadores comecem um novo jogo depois de ter terminado outro, não tendo de abrir uma nova janela de jogo para o efeito. Para o utilizador poder participar terá de estar autenticado, existindo para o caso de utilizadores novos a opção de registo na janela de autenticação.

3. Arquitectura da aplicação

Este projecto foge ligeiramente ao enunciado do projecto, usando o motor de base de dados PostgreSQL em alternativa aos ficheiros, de forma a garantir a persistência dos dados.

3.1. Estrutura

A arquitectura da aplicação pode ser consultada através dos diagramas de classes que se encontram em anexo. O fluxo da informação vai ser explicitado através de diagramas de sequência UML que se encontram em http://student.dei.uc.pt/~richard/sdist. O *setup* inicial de um cliente encontra-se descrito em ClientSetup. O fluxo da *thread* que vai receber as mensagens do servidor está em ClientReaderThread. O *setup* inicial do servidor primário encontra-se descrito em PrimaryServerSetup. Do mesmo modo o funcionamento de um servidor em *watch dog mode* encontra-se em WatchDogServer. A *thread* do servidor principal que responde aos *watch dogs* tem o seu funcionamento descrito em AliveSenderThread. As *threads* de resposta aos pedidos dos clientes têm o seu funcionamente delineado em ResponseThread. As *threads* que recebem as mensagens do cliente estão em ConnectionThreads. O fluxo genérico de acesso à Base de Dados encontra-se em DataBaseAcess.

3.2. Configurações

De forma a ser usado um ficheiro de configuração de forma simples, recorreu-se à ferramenta YAML. Esta permite armazenar dados serializáveis num ficheiro de texto, de uma forma legível e editável pelo utilizador. Neste contexto limitou-se à definição de simples valores dentro de uma *Hashtable*. De seguida apresenta-se a descrição de cada entrada no ficheiro de configuração e respectivos valores por defeito. Este ficheiro chama-se *config.yml* e pode ser encontrado na raiz da aplicação.

nome	**valor por defeito**	**descrição**
databaseIp	127.0.0.1	endereço IP da base de dados
databaseName	connect4	nome da base de dados
databaseUsername	postgres	username da base dados
databasePassword	postgres	password do username definido em *databaseUsername*
serverIp	127.0.0.1	endereço IP do servidor
multicastGroupIP	234.234.234.234	endereço IP do grupo de multicast (usado pra enviar mensagens dentro da *lounge*)
serverTcpPort	12345	porta TCP do servidor
serverUdpPort	12345	porta UDP do servidor
multicastSocketPort	6666	porta Multicast do servidor
serverRegistryPort	7000	porta do *rmiregistry*
serverRegistryNaming	server	nome do servidor a usar ao fazer *rebind* no *rmiregistry*
aliveTimer	1000	intervalo de tempo em que cada *watchdog* deve periodicamente fazer *ping* ao servidor primário
receiveTimeout	1500	intervalo de tempo máximo em que cada *watchdog* espera antes de tentar passar a servidor primário
responseThreadsNumber	5	número de *threads* de resposta
aliveMsgQuestion	ALIVE?	mensagem usada nos *watchdogs* para averiguar se existe servidor primário
aliveMsgAnwser	ALIVE!	mensagem usava pelo servidor primário para indicar aos *watchdogs* que se encontra vivo
clientNumReconnects	5	número de tentativas que cada cliente tenta efectuar antes de
maxLoungeLength	256	tamanho máximo das mensagens a serem trocadas na *lounge*

3.3. Logging

Com vista a ser mantido um *log* do lado do servidor, não só na consola mas também de forma persisitente, foi usada a API de Logging do Java. Esta encontra-se configurável através do ficheiro *logging.properties* que pode ser encontrado na raiz da aplicação. De seguida apresenta-se a descrição das entradas usadas e respectivo valores por defeito.

nome	**valor por defeito**	**descrição**
handlers	java.util.logging.ConsoleHandler,	criação de dois

	java.util.logging.FileHandler	*handlers* (um para *logging* na consola e outro para um ficheiro de texto)
.level	ALL	define o nível de detalhe para o *root logger*
java.util.logging.ConsoleHandler.level	INFO	define o nível de detalhe de *logging* para a consola
java.util.logging.ConsoleHandler.formatter	connect4.config.LogFormatter	define a classe responsável pela formatação do *logging* para a consola
java.util.logging.FileHandler.level	ALL	define o nível de detalhe de *logging* para o ficheiro de texto
java.util.logging.FileHandler.pattern	connect4.log	define o *pattern* a usar no nome do ficheiro de textp
java.util.logging.FileHandler.formatter	connect4.config.LogFormatter	define a classe responsável pela formatação do *logging* para o ficheiro de texto

4. Formato das mensagens da aplicação

Na aplicação existem duas packages que contêm as mensagens utilizadas, a package messages.client e messages.server. Todas as classes nelas contidas extendem uma classe genérica Message que é serializável de modo a poder ser transmitida. De uma forma geral, as mensagens contêm informação que permita identificar os intervenientes na comunicação e informação a ser alterado pelo receptor da mensagem. Quando o cliente tem informação a transmitir cria uma mensagem do tipo mensagem de cliente, o servidor ao receber esta mensagem trata a informação nela contida e normalmente cria uma mensagem do tipo mensagem de servidor. O processo é inverso quando é o servidor a tomar a iniciativa de enviar informação. Para mais informação acerca de cada mensagem existente nas duas packages poderão ser consultados os diagramas anexados ao documento.

5. Tratamento de falhas transitórias

As falhas transitórias da conexão entre o cliente e servidor são tratadas de formas semelhante tanto na versão de Sockets como na versão de RMI. Porém, existem também algumas características individuais a realçar.

No ficheiro de configuração está definida a entrada *clientNumReconnects*. Como indicado no ponto 3.1., tem como finalidade definir quantas tentativas deve o cliente efectuar de forma a tentar reestabelecer a conexão e por conseguinte enviar as mensagens desejadas.

Do lado do servidor existe também suporte em caso de falhas de conexção bem como oferece uma forma persistente de salvaguardar as mensagens ainda não processadas. Foi implementada uma *queue* tendo como base o motor de base de dados *PostgreSQL*. Quando uma mensagem chega ao servidor, esta é colocada na *queue*. Existe depois um número pré-determinado de *threads* de resposta (definido em *responseThreadsNumber*) responsáveis por esperarem que alguma mensagem entre na *queue* de forma a ser processada. Isto permite que, em caso de o servidor falhar imediatamente antes da mensagem ser processada, esta seja processada pelo *watchdogs* que passou entretanto a ser o servidor primário.

6. Implementação do mecanismo de replicação

O servidor deve registar-se de duas formas diferentes. Por um lado, sockets, deve tentar registar-se no porto pré-definido sob a entrada *serverTcpPort* no ficheiro de configuração. Por outro lado, RMI, deve substituir o binding para o nome pré-definido sob a entrada *serverRegistryNaming* no ficheiro de configuração com a referência do servidor. Dado que, em caso de esta referência já existir, e se for feito rebind de um nome já existente, a referência antiga ser ignorada, tem-se que ter o cuidado especial em tentar registar-se primeiro no porto de *sockets* e apenas depois fazer o *rebind* em RMI.

No caso de sockets, os *watchdogs* enviam para o servidor a mensagem definida sob *aliveMsgQuestion* e ficam à espera duma mensagem do servidor com o conteúdo definido em *aliveMsgAnswer*. Caso um *watchdog* não receba a resposta do servidor primário num certo intervalo de tempo (definido em *receiveTimeout*), compete pelo lugar como servidor primário.

Sempre que um servidor ganha o lugar de servidor primário, este tem a responsabilidade de ir buscar a informação dos jogos activos à base de dados e fazer *dispatch* de possíveis mensagens que ainda se encontrem na *queue*. À primeira vista este mecanismo de *dispatch* parece algo impossível. Um servidor primário que terminou agora de arrancar não possui os Sockets abertos com os clientes que usam sockets, nem tem a referência aos objectos remotos no caso dos clientes RMI. Posto isto, foi implementado um mecanismo do lado do cliente que tem como finalidade enviar ao servidor a informação do cliente. Caso seja um cliente de sockets tenta abrir um novo socket, caso seja um cliente RMI envia ao servidor a instância do objecto remoto.

7. Integração entre a versão de Sockets e a versão de RMI

Esta integração ocorreu de forma completamente transparente, dada a peculiar estrutura do lado do servidor (*queue* de mensagens e *threads* de resposta).

8. Manual do utilizador

O manual do utilizador encontra-se em http://student.dei.uc.pt/~richard/sdist.

9. Manual de instalação e configuração

O manual de instalação e configuração encontra-se em http://student.dei.uc.pt/~richard/sdist.

10. Descrição dos testes efectuados à aplicação

Além de existir um processo de teste contínuo à medida que se desenvolvia a aplicação, no fim desta estar concluída foram efectuados os seguintes testes:

- Em caso de falha do servidor a aplicação mantém o seu funcionamento (mecanismo de *watch dogs*).
- As mensagens são recebidas como são enviadas tanto em mensagens públicas (sala de chat inicial) como em mensagens privadas.
- O funcionamento entre clientes Socket foi testado. O funcionamento entre clientes RMI também foi testado. Estes testes foram feitos para toda a aplicação (mensagem chat publica, mensagem chat privada e jogo)
- A interoperabilidade entre clientes de Socket e RMI quando um cliente de RMI convida um cliente de Socket (e vice versa) também foi testada para toda a aplicação.
- A situação particular da criação de novos jogadores e começar a jogar com eles imediatamente há um problema. Para funcionar tem desligar os clientes e voltar a tentar fazer login com eles.