Anterior • Trilha I • Próximo ? comentar

Lição 7 - Um chat em Java

- **Objetivo(s):** Ensinar o aluno a implementar um chat em Java usando sockets e threads.
- **Direitos autorais e licença:** Veja notas de direitos autorais e licença no final da lição.

Conteúdo

7.1 - Requisitos do *chat*

Um exercício interessante para praticarmos o uso de *sockets* é a criação de um *chat*. Se considerarmos que sempre existem somente duas pessoas conversando, poderíamos implementar o cliente e o servidor de maneira idêntica uma vez que eles desempenhariam o mesmo papel. Contudo, normalmente um *chat* envolve várias pessoas conversando simultaneamente. Dessa forma, o servidor assume o papel de receber a mensagem de um cliente e repassar para os demais.

O cliente deve desempenhar duas funções ao mesmo tempo: enviar suas próprias mensagens e receber as mensagens enviadas pelo servidor. Dessa forma, devemos utilizar múltiplas *threads*, pois caso contrário o envio e recebimento de mensagens não poderiam ser feitos simultaneamente.

7.2 - Implementando o Cliente

O cliente fará a conexão com o servidor usando *sockets* TCP. Portanto, ele precisará saber o *host* e a porta em que o servidor está executando:

```
public class Cliente {
	private String host;
	private int porta;

	public Cliente(String host, int porta) {
		setHost(host);
		setPorta(porta);
	}

	public String getHost() {
		return host;
	}

	public void setHost(String host) {
		this.host = host;
	}

	public int getPorta() {
		return porta;
	}

	public void setPorta(int porta) {
		if (porta <= 0) {
			throw new IllegalArgumentException("Porta deve ser um número positivo");
		}

		this.porta = porta;
	}
}
```

Conforme foi dito, o cliente deverá possuir duas *threads*: uma para envio de mensagens digitadas pelo usuário e outra para recebimento de mensagens enviadas pelo servidor.

O envio de mensagens pode ser feito pela *thread* principal, ou seja, pode ser implementado na classe cliente como o método abaixo:

```
public void enviaMensagens() {
	try {
		// le msgs do teclado e manda pro servidor
		Scanner teclado = new Scanner(System.in);
		PrintStream stream = new PrintStream(socket.getOutputStream());

		while (teclado.hasNextLine()) {
			stream.println(teclado.nextLine());
		}
	} catch (IOException e) {
		System.out.println("Erro de Entrada/Saída");
	}
}
```

O método acima lê dados da *stream* referente ao teclado e envia, usando a *stream* de saída do *socket*. A criação do objeto "stream" usando a classe java.io.PrintStream foi para facilitar o envio dos dados, ou seja, usando essa classe não é necessária a conversão de/para um *array* de *bytes*.

Antes de continuar a implementação da classe Cliente temos que implementar a *thread* responsável por receber as mensagens enviadas pelo servidor. Isso será feito com a classe abaixo:

```
import java.io.InputStream;
import java.util.Scanner;


public class RecebedorMsgCliente extends Thread {
	private InputStream entrada;
```

```
	public RecebedorMsgCliente(InputStream entrada) {
		this.entrada = entrada;
	}

	public synchronized void start() {
		super.start();
	}

	public void run() {
		// recebe msgs do servidor e imprime na tela
		Scanner s = new Scanner(entrada);

		while (s.hasNextLine()) {
			System.out.println(s.nextLine());
		}
	}

}
```

Baixe o código-fonte acima neste link: http://wiki.marceloakira.com/pub/GrupoJava/UmChatEmJava/RecebedorMsgCliente.java

Note que implementamos essa classe como uma subclasse de *Thread*, e reescrevemos o método *start* para permitir a inicialização da *thread*. O comportamento do recebedor de mensagens foi implementado no método *run*. Este consiste em receber os dados do servidor, usando a *stream* de entrada fornecida na criação do objeto, e imprimi-los na tela.

Usando esta nova classe podemos terminar a implementação do cliente. A classe completa ficaria como abaixo. Implementamos um método *init* responsável por instanciar os objetos, iniciar a *thread* recebedora e fazer a leitura do teclado.

```
import java.io.IOException;
import java.io.PrintStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class Cliente {
	private String host;
	private int porta;

	private Socket socket;
	private RecebedorMsgCliente recebedor;

	public Cliente(String host, int porta) {
		setHost(host);
		setPorta(porta);
	}

	public String getHost() {
		return host;
	}

	public void setHost(String host) {
		this.host = host;
	}

	public int getPorta() {
		return porta;
	}

	public void setPorta(int porta) {
		if (porta <= 0) {
			throw new IllegalArgumentException("Porta deve ser um número positivo");
		}

		this.porta = porta;
	}

	public void init() {
		try {
			socket = new Socket(host, porta);

			recebedor = new RecebedorMsgCliente(socket.getInputStream());
			recebedor.start();

			System.out.println("Cliente iniciado!");

			enviaMensagens();
		} catch (UnknownHostException e) {
			System.out.println("Host desconhecido!");
		} catch (IOException e) {
			System.out.println("Erro de entrada/saída!");
		}
	}

	public void enviaMensagens() {
		try {
			// le msgs do teclado e manda pro servidor
			Scanner teclado = new Scanner(System.in);
			PrintStream stream = new PrintStream(socket.getOutputStream());

			while (teclado.hasNextLine()) {
				stream.println(teclado.nextLine());
			}
		} catch (IOException e) {
```

```
                        System.out.println("Erro de Entrada/Saída");
                }
        }

        public static void main(String[] args) {
                Cliente c = new Cliente("localhost", Servidor.PORTA);
                c.init();
        }

}
```

Baixe o código-fonte acima neste link: http://wiki.marceloakira.com/pub/GrupoJava/UmChatEmJava/Cliente.java

7.3 - Implementando o Servidor

Para facilitar a codificação, podemos padronizar a porta do servidor colocando-a como uma constante.

A classe "Servidor" deverá manter uma lista de todos os clientes conectados ao *chat* para poder enviar uma mensagem recebida para todos os clientes.

Dessa forma, a classe inicia como abaixo:

```
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class Servidor {
        public static final int PORTA = 12345;

        private List<Socket> clientes;

        public Servidor() {
                clientes = new ArrayList<Socket>();
        }
}
```

O servidor deverá possuir um método responsável por enviar uma dada mensagem a todos os clientes conectados no *chat*. Veja abaixo:

```
public void distribuiMensagem(String msg) {
        try {
                for (Socket cliente : clientes) {
                        PrintStream stream = new PrintStream(cliente.getOutputStream());
                        // Envia a mensagem juntamente com o nome do destinatário
                        stream.println(formataCliente(cliente) + ": " + msg);
                }
        } catch (IOException e) {
                System.out.println("Erro de entrada/saída!");
        }
}

private String formataCliente(Socket cliente) {
        return cliente.getInetAddress().getHostAddress() + ":" + cliente.getPort();
}
```

O método *formataCliente* formata no endereço do cliente o formato "IP:porta" para que o destinatário da mensagem seja incluído como cabeçalho da mesma.

De forma semelhante ao cliente, o servidor também desempenha duas funções simultâneas: envio e recebimento de mensagens. Portanto, também devemos implementar uma *thread* adicional para o recebimento das mensagens. Essa *thread* deverá receber a mensagem e repassá-la ao servidor através de uma chamada ao método *distribuiMensagem*. Fazemos isso implementando a classe abaixo:

```
import java.io.InputStream;
import java.util.Scanner;

public class RecebedorMsgServidor extends Thread {
        private InputStream entrada;
        private Servidor servidor;

        public RecebedorMsgServidor(InputStream entrada, Servidor servidor) {
                this.entrada = entrada;
                this.servidor = servidor;
        }

        public synchronized void start() {
                super.start();
        }

        public void run() {
                Scanner s = new Scanner(entrada);
                while (s.hasNextLine()) {
                        servidor.distribuiMensagem(s.nextLine());
                }
                s.close();
        }

}
```

Baixe o código-fonte acima neste link: http://wiki.marceloakira.com/pub/GrupoJava/UmChatEmJava/RecebedorMsgServidor.java

Com isso, finalizamos o servidor. Veja a classe abaixo:

```
package rede.chat;

import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class Servidor {
        public static final int PORTA = 12345;

        private List<Socket> clientes;

        public Servidor() {
                clientes = new ArrayList<Socket>();
        }

        public void init() {
                try {
                        ServerSocket socketRecepcao = new ServerSocket(PORTA);
                        System.out.println("Servidor esperando conexão...");

                        while (true) {
                                Socket cliente = socketRecepcao.accept();
                                System.out.println("Nova conexão com o cliente " + formataCliente(cliente));


                                // Adiciona o novo cliente à lista de clientes
                                clientes.add(cliente);

                                // Cria uma nova thread para receber mensagens deste cliente
                                RecebedorMsgServidor recebedor = new RecebedorMsgServidor(cliente.getInputStream(), this);
                                recebedor.start();
                        }
                } catch (IOException e) {
                        System.out.println("Erro de entrada/saída!");
                }
        }

        public void distribuiMensagem(String msg) {
                try {
                        for (Socket cliente : clientes) {
                                PrintStream stream = new PrintStream(cliente.getOutputStream());
                                // Envia a mensagem juntamente com o nome do destinatário
                                stream.println(formataCliente(cliente) + ": " + msg);
                        }
                } catch (IOException e) {
                        System.out.println("Erro de entrada/saída!");
                }
        }

        private String formataCliente(Socket cliente) {
                return cliente.getInetAddress().getHostAddress() + ":" + cliente.getPort();
        }

        public static void main(String[] args) {
                Servidor servidor = new Servidor();
                servidor.init();
        }
}
```

Baixe o código-fonte acima neste link: http://wiki.marceloakira.com/pub/GrupoJava/UmChatEmJava/RecebedorMsgServidor.java

Na implementação que fizemos todos os clientes conectados recebem uma mensagem enviada, inclusive o emissor da mensagem. Tente modificar a implementação apresentada para que o cliente que enviou a mensagem não a receba.

7.4 - Direitos autorais e licença

Anterior • Trilha I • Próximo ?

7.5 - Comentários

Adicionar