Nome do Campus: Polo Mondubim

Nome do Curso: Desenvolvimento Full Stack

Nome da Disciplina: Por que não paralelizar?

Número da Turma: 9001

Semestre Letivo: 2023.3

Integrante da Prática: Samuel Mota Araujo

Repositório GIT: https://github.com/samuelmotapf

Introdução

Este relatório descreve o desenvolvimento da missão prática, que aborda a criação de servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA (Java Persistence API). A prática tem como objetivos a criação de servidores Java baseados em Sockets, a implementação de clientes síncronos e assíncronos para esses servidores, o uso de Threads para implementação de processos paralelos, e o acesso ao banco de dados via JPA.

Objetivos da Prática

Os objetivos desta missão prática são:

1. Criar servidores Java com base em Sockets.

2. Criar clientes síncronos para servidores com base em Sockets.

3. Criar clientes assíncronos para servidores com base em Sockets.

4. Utilizar Threads para implementação de processos paralelos.

Ao final do exercício, espera-se que o aluno tenha criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de ter utilizado os recursos nativos do Java para implementação de clientes síncronos e assíncronos.

Materiais Necessários

1. SQL Server, com o banco de dados gerado em prática anterior (loja).
2. JDK e IDE NetBeans.
3. Navegador para Internet, como o Chrome.
4. Banco de dados SQL Server com o Management Studio.

Desenvolvimento da Prática

Procedimento 1: Criando o Servidor e Cliente de Teste

1. *Projeto do Servidor - CadastroServer:*

- Implementou-se o protocolo onde o cliente conecta, envia login e senha, o servidor valida as credenciais, e, se válidas, entra em um ciclo de resposta.

- O cliente envia a letra 'L', e o servidor responde com o conjunto de produtos.

2. *Camada de Persistência em CadastroServer:*

- Criar o pacote `model` para implementação das entidades.

- Utilizar a opção "New..Entity Classes from Database" para adicionar todas as tabelas do banco de dados SQL Server.

- Adicionar a biblioteca Eclipse Link (JPA 2.1) e o arquivo jar do conector JDBC para o SQL Server.

3. *Camada de Controle em CadastroServer:*

- Criar o pacote `controller` para implementação dos controladores.

- Utilizar a opção "New..JPA Controller Classes from Entity Classes".

- Na classe `UsuarioJpaController`, adicionar o método `findUsuario` para retornar o usuário a partir de um login e senha.

- Adicionar os atributos `ctrl` e `ctrlUsu` dos tipos `ProdutoJpaController` e `UsuarioJpaController`, respectivamente.

- Adicionar o atributo `s1` para receber o Socket.

- Definir um construtor recebendo os controladores e o Socket.

- Implementar o método `run` para a Thread, encapsulando os canais de entrada e saída do Socket em objetos `ObjectOutputStream` e `ObjectInputStream`.

- Obter login e senha, verificar o login com `ctrlUsu` e iniciar o loop de resposta.

4. *Classe de Execução em CadastroServer:*

- Instanciar objetos do tipo `EntityManagerFactory`, `ctrl`, `ctrlUsu`, e um objeto do tipo `ServerSocket` escutando a porta 4321.

- Em um loop infinito, obter a requisição de conexão do cliente, instanciar uma Thread e iniciar a execução.

5. *Projeto do Cliente - CadastroClient:*

- Instanciar um Socket apontando para localhost, na porta 4321.

- Encapsular os canais de entrada e saída do Socket em objetos `ObjectOutputStream` e `ObjectInputStream`.

- Escrever login e senha na saída, enviar o comando 'L', receber a coleção de entidades e apresentar o nome de cada entidade.

6. *Configuração do Projeto do Cliente:*

- Copiar o pacote `model` do projeto do servidor para o cliente.

- Adicionar a biblioteca Eclipse Link (JPA 2.1).

Resultados Esperados

1. Código organizado na construção do projeto Java com acesso a banco de dados usando Sockets, Threads e JPA.

2. Exploração das funcionalidades oferecidas pelo NetBeans para melhoria da produtividade.

3. Demonstração das habilidades básicas no uso prático de Threads em ambientes cliente e servidor.

Análise e Conclusão

1. *Classes Socket e ServerSocket:*

- As classes `Socket` e `ServerSocket` em Java são utilizadas para comunicação entre cliente e servidor. O `Socket` é o ponto final em uma conexão, enquanto o `ServerSocket` é responsável por aguardar por pedidos de conexão.

2. *Importância das Portas para a Conexão:*

- As portas são fundamentais para a conexão com servidores, pois permitem que diferentes serviços rodando no mesmo servidor sejam acessados de maneira única.

3. *Classes de Entrada e Saída ObjectInputStream/ObjectOutputStream:*

- As classes `ObjectInputStream` e `ObjectOutputStream` são usadas para serializar e desserializar objetos em Java. Os objetos transmitidos devem ser serializáveis para que possam ser convertidos em fluxos de bytes para transmissão e reconstruídos no destino.

4. *Isolamento do Acesso ao Banco de Dados:*

- Mesmo utilizando as classes de entidades JPA no cliente, o isolamento do acesso ao banco de dados é garantido pela implementação do servidor. O cliente envia solicitações, e o servidor, que possui acesso ao banco de dados, as executa e retorna os resultados ao cliente.

Relatório Discente de Acompanhamento

O relatório discente de acompanhamento deve ser adicionado ao arquivo no formato PDF, contendo informações como a Logo da Universidade, nome do Campus, nome do Curso, nome da Disciplina, número da Turma, semestre letivo, nome dos integrantes da Prática, e o endereço do repositório GIT onde o projeto está armazenado.

Conclusão

Este relatório documenta o desenvolvimento da missão prática "Por que não paralelizar", destacando as etapas realizadas, os resultados obtidos e a análise das principais funcionalidades e conceitos abordados. O código desenvolvido atende aos requisitos estabelecidos, demonstrando o entendimento e aplicação dos conceitos de Sockets, Threads e JPA na construção de sistemas distribuídos.