RPG0018 - Por que não paralelizar

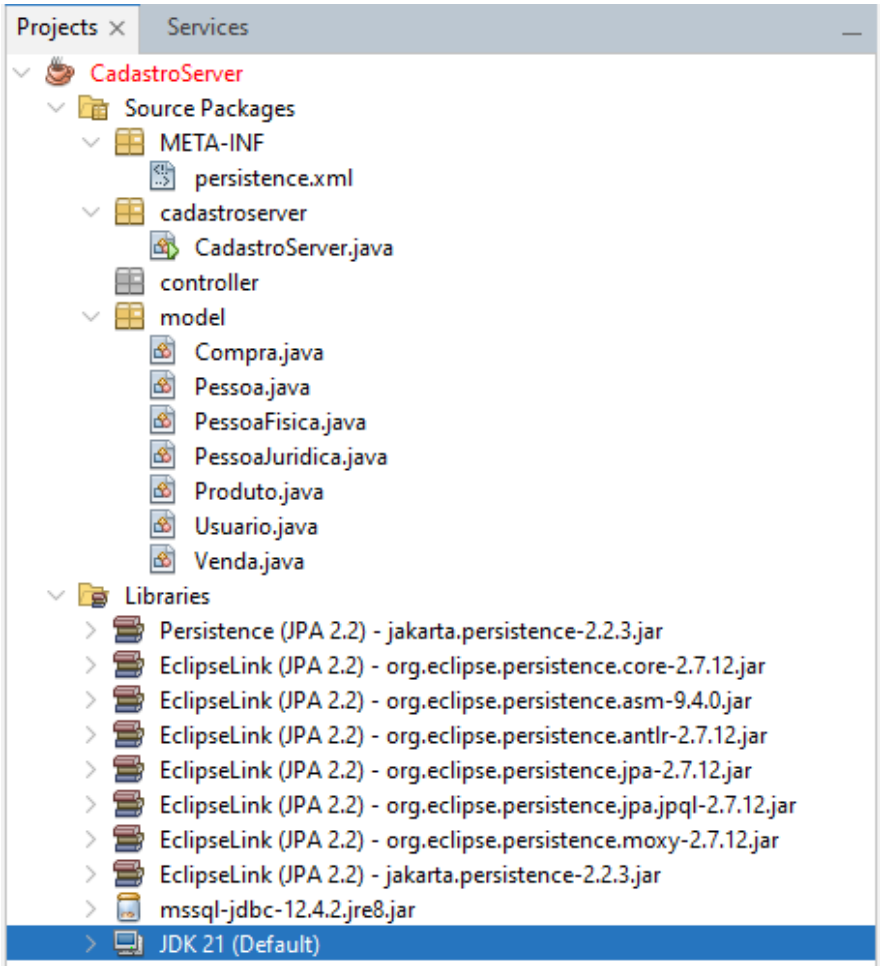
Ricardo Alves dos Santos Junior - 202208681158

Universidade Estácio de Sá

Nível 5: Por que não paralelizar?– 2023.3 -Terceiro Semestre

Repositório no GitHub: https://github.com/RicardoASJunior/CadastroServer

Criação do Projeto:



Objetivo da Prática

- Criar servidores Java com base em Sockets.

- Criar clientes síncronos para servidores com base em Sockets.
- Criar clientes assíncronos para servidores com base em Sockets.
- Utilizar Threads para implementação de processos paralelos.
- No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.

1º Procedimento | Criando o Servidor e Cliente de Teste

a) Como funcionam as classes Socket e ServerSocket?

R: As classes Socket e ServerSocket são usadas para estabelecer e gerenciar comunicações entre clientes e servidores usando o protocolo TCP. A classe Socket representa um ponto de conexão entre dois dispositivos na rede, permitindo enviar e receber dados. A classe ServerSocket é usada pelo lado do servidor para escutar e aceitar conexões de clientes em uma determinada porta.

b) Qual a importância das portas para a conexão com servidores?

R: As portas são números que identificam os serviços ou processos que estão sendo executados em um dispositivo na rede. As portas permitem que os dispositivos diferenciem entre os diferentes tipos de tráfego e direcionem as requisições e respostas para os destinos corretos. As portas são associadas aos protocolos de transporte, como TCP ou UDP, e são indicadas nos cabeçalhos desses protocolos. As portas são padronizadas em todos os dispositivos conectados à rede, e algumas portas são reservadas para certos protocolos. Por exemplo, a porta 80 é usada para o protocolo HTTP, que é usado para transferir páginas web.

c) Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?

R: As classes de entrada e saída ObjectInputStream e ObjectOutputStream são usadas para serializar e deserializar dados e objetos primitivos que foram escritos usando ObjectOutputStream. Essas classes permitem armazenar e recuperar grafos de objetos em um arquivo ou em um fluxo de rede. As classes implementam as interfaces ObjectInput e ObjectOutput, que são subinterfaces de DataInput e DataOutput. Isso significa que todos os métodos de entrada e saída de dados primitivos abordados em Data Streams também são implementados em object streams.

d) Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?

R: Mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados porque as entidades JPA são gerenciadas por um contexto de persistência, que é uma coleção de objetos que representam o estado dos dados no banco de dados. O contexto de persistência pode ser propagado com a transação JTA entre as instâncias de EntityManager em uma determinada unidade de persistência, ou pode ser isolado para cada instância de EntityManager, criando um novo contexto de

persistência. O isolamento do contexto de persistência permite que a aplicação acesse um contexto de persistência que não é afetado por outras operações concorrentes no banco de dados.

2º Procedimento | Servidor Completo e Cliente Assíncrono

a) Como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor?

R: As Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor, de forma que o cliente não fique bloqueado esperando a resposta do servidor. Para isso, o cliente pode criar uma Thread separada para ler os dados do Socket, enquanto a Thread principal continua executando outras tarefas. A Thread de leitura pode notificar a Thread principal quando receber uma resposta do servidor, usando mecanismos de sincronização, como wait e notify, ou callbacks.

b) Para que serve o método invokeLater, da classe SwingUtilities?

R: O método invokeLater, da classe SwingUtilities, serve para executar uma tarefa na Thread do Event Dispatch, que é a Thread responsável por atualizar a interface gráfica do Swing. Esse método é útil quando se quer atualizar algum componente gráfico a partir de outra Thread, pois o Swing não é thread-safe, ou seja, não garante a consistência dos dados se eles forem acessados por mais de uma Thread. O método invokeLater recebe como parâmetro um objeto Runnable, que representa a tarefa a ser executada, e coloca essa tarefa na fila de eventos do Swing, para ser executada assim que possível.

c) Como os objetos são enviados e recebidos pelo Socket Java?

R: Os objetos são enviados e recebidos pelo Socket Java usando as classes ObjectInputStream e ObjectOutputStream, que permitem serializar e deserializar objetos que implementam a interface Serializable. Essas classes são decoradores das classes InputStream e OutputStream, que são obtidas a partir do Socket. Para enviar um objeto pelo Socket, basta criar um ObjectOutputStream a partir do OutputStream do Socket, e chamar o método writeObject, passando o objeto como parâmetro. Para receber um objeto pelo Socket, basta criar um ObjectInputStream a partir do InputStream do Socket, e chamar o método readObject, que retorna um objeto do tipo Object, que deve ser convertido para o tipo desejado.

d) Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento.

R: A utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java tem implicações no bloqueio do processamento. Se o cliente usar um comportamento síncrono, ele ficará bloqueado esperando a resposta do servidor a cada requisição que enviar, o que pode reduzir o desempenho e a interatividade do cliente. Se o cliente usar um comportamento assíncrono, ele poderá enviar várias requisições sem esperar a resposta do servidor, e continuar executando outras tarefas, o que pode aumentar o desempenho e a interatividade do cliente. No entanto, o comportamento assíncrono requer o uso de Threads e mecanismos de sincronização, o que pode aumentar a complexidade e os riscos de erros de programação.

Conclusão

O trabalho não foi concluído devido a organização e o tempo de entrega. Foram criadas as classes de acordo com as tabelas do banco sql, porém não consegui implemantar os controladores.