Porque não paralelizar

Rogério Benedito Geraldo – Matricula 202110027921

Campus Nova Ourinhos
Porque não paralelizar - 3° Semestre Letivo
Link do Github: https://github.com/Rogeriobg/por-que-n-o-paralelizar.git

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.

Análise e conclusão - Procedimento 1

Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket são usadas para comunicação de rede em Java.

Socket: Estabelece uma conexão de cliente com um servidor, permitindo enviar e receber dados.

ServerSocket: Escuta solicitações de conexão de clientes e cria um Socket para cada conexão aceita.

Qual a importância das portas para a conexão com servidores?

As portas são fundamentais para a comunicação entre clientes e servidores, pois identificam serviços específicos em uma máquina. Cada porta permite que dados sejam direcionados ao aplicativo correto, garantindo que múltiplos serviços possam operar simultaneamente no mesmo servidor. Sem o uso de portas, seria impossível distinguir entre diferentes tipos de tráfego de rede, resultando em uma comunicação desorganizada e ineficiente.

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

As classes ObjectInputStream e ObjectOutputStream em Java são utilizadas para ler e escrever objetos em streams, facilitando a transmissão de dados complexos entre diferentes partes de um programa ou através de redes. ObjectInputStream desserializa objetos, convertendo o fluxo de dados de volta em objetos Java, enquanto ObjectOutputStream serializa objetos, convertendo-os em um formato binário que pode ser transmitido ou armazenado.

Os objetos devem ser serializáveis para garantir que seu estado completo possa ser convertido em um formato que possa ser facilmente transmitido ou armazenado e posteriormente reconstruído. Isso é realizado implementando a interface Serializable, que permite

a conversão de objetos em uma sequência de bytes, preservando o estado do objeto para futura recuperação.

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

Mesmo utilizando classes de entidades JPA no cliente, o isolamento do acesso ao banco de dados é garantido por meio da arquitetura em camadas e da utilização de controles de transação e persistência no lado do servidor. O cliente manipula entidades JPA que são meras representações dos dados, enquanto o servidor controla o acesso direto ao banco de dados, gerencia transações e aplica regras de negócio. Isso significa que todas as operações de leitura, escrita, atualização e exclusão no banco de dados são mediadas pelo servidor, preservando a integridade, consistência e segurança dos dados.

Análise e conclusão - Procedimento 2

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

Threads podem ser utilizadas para tratamento assíncrono das respostas do servidor ao permitir a execução paralela de tarefas. Quando uma resposta é recebida, uma nova thread pode ser iniciada para processar essa resposta independentemente do fluxo principal do programa. Isso evita que a interface do usuário ou outras

operações sejam bloqueadas enquanto a resposta é tratada. Por exemplo, ao receber dados de um servidor, uma thread pode atualizar a interface ou armazenar dados sem interromper a execução contínua do aplicativo, melhorando a responsividade e eficiência do sistema.

Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities serve para garantir que o código fornecido seja executado na Event Dispatch Thread (EDT) do Swing. Isso é crucial para a manipulação segura de componentes da interface gráfica, uma vez que todas as atualizações e interações com a UI devem ocorrer na EDT. Utilizar invokeLater permite que tarefas sejam agendadas para execução assíncrona na EDT, evitando problemas de concorrência e garantindo que a interface do usuário permaneça responsiva. É especialmente útil para atualizar a UI a partir de threads que não são a EDT, garantindo a integridade e consistência das operações gráficas.

Como os objetos são enviados e recebidos pelo Socket Java?

Em Java, objetos são enviados e recebidos por meio de Socket utilizando streams de entrada e saída, especificamente ObjectOutputStream e ObjectInputStream. No lado do cliente, um ObjectOutputStream é criado a partir do OutputStream do Socket para enviar objetos serializáveis. No servidor, um ObjectInputStream é criado a partir do InputStream do Socket para ler esses objetos. O processo inverso ocorre para a recepção: o cliente cria um ObjectInputStream para receber objetos, e o servidor usa um ObjectOutputStream para enviá-los. Este mecanismo de serialização e desserialização permite a troca de objetos complexos entre cliente e servidor através da rede.

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

No uso de Socket em Java, o comportamento assíncrono e síncrono apresenta diferenças significativas em relação ao bloqueio do processamento.

Síncrono: Neste modelo, métodos como read e write bloqueiam a execução até completarem a operação de I/O. Isso pode levar a um bloqueio completo do thread de execução, resultando em uma interface de usuário não responsiva e possíveis atrasos na aplicação quando aguardando dados.

Assíncrono: Utilizando threads ou a API java.nio, o I/O não bloqueia o fluxo principal do programa. Threads paralelas ou callbacks tratam as operações de rede, permitindo que o processamento continue enquanto se aguarda a conclusão das operações de I/O. Isso melhora a responsividade e eficiência, especialmente em aplicações com múltiplas conexões ou interfaces de usuário interativas.