Estácio *Por que não paralelizar*

Nome: Michel Angelo da Silva Portugal .

Matrícula: 202208826174
Campus: Duque de Caxias
Disciplina: RPG0018 - Por que não paralelizar/ Sem 2/2023.

Objetivos 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.

Link do Repositório:

https://github.com/miaspe/PNP

Análise e Conclusão:

P: Como funcionam as classes Socket e ServerSocket?

R: São respectivamente para conectar ao servidor e pra responder a solicitação de conexão (ServerSocket) pra entrada e saída de dados de um servidor usando pra isso o protocolo e uma porta (normalmente TCP-IP)

P: Qual a importância das portas para a conexão com servidores?

R: Garante a unicidade(exclusividade) da conexão pra cada usuário e o principal que é evitar conflitos entre aplicativos diversos.

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

R: como a própria pergunta já respondeu para entrada(input) e saída(output) de um objeto que no caso provavelmente será um Dado (informação) desejada. A serialização é um método de organizar o Dado requerido , para exemplificar vejamos que se faz necessário “montar” a informação na outra ponta depois de transmiti-la , tomemos a palavra (SERIAL) que será dividida em (SE)(RI)(AL) temos três pacotes e eles serão transmitidos na internet passando por vários Roteadores ao longo do caminho antes de chegar até a máquina que solicitou, como os roteadores tem prioridades diferentes acabam causando atrasos na rede e podem fazer chegar primeiro o pacote 1(SE) depois o pacote 3(AL) e por fim o ultimo pacote 2(RI) se os pacotes não fossem marcados em série o computador montaria a palavra a medida que os pacotes chegassem gerando a palavra SEALRI, o que seria incorreto, mas mas o protocolo e a serialização fazem a informação ser montada corretamente

resultando em SERIAL corretamente; Claro que é um exemplo simplório mas ajuda a entender a importância da serialização.

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

R: graças a mecanismos internos da JPA tais como transações, contextos de persistência, caches, desconexão e outros todos dentro dos princípios ACID.

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

R:As Threads  permitem Programação Concorrente assim enquanto não receba uma resposta a aplicação não precisa parar, pode continuar um processamento em paralelo.

P: Para que serve o método invokeLater, da classe SwingUtilities?

R: Auxilia a continuar executando a interface gráfica para manter a responsividade da aplicação

P: Como os objetos são enviados e recebidos pelo Socket Java?

R: Usando o processo de serialização/ desserialização

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

R: alguns dos potenciais impecílios do processamento síncrono em sockets Java são:

- Bloqueio da Thread:

Operações síncronas em sockets podem bloquear a thread de execução até que a operação seja concluída. Isso significa que uma thread que está aguardando uma resposta de uma operação de leitura ou escrita não pode realizar outras tarefas enquanto está bloqueada.

- Desempenho Limitado:

Em ambientes com alto volume de conexões simultâneas, o processamento síncrono pode limitar o desempenho, uma vez que cada conexão bloqueia uma thread até que a operação seja concluída. Isso pode levar a um aumento no consumo de recursos e uma menor capacidade de manipular um grande número de clientes concorrentes.

- Requisitos de Recursos:

A alocação de uma thread por conexão pode exigir uma quantidade significativa de recursos do sistema. Em cenários onde há um grande número de conexões, isso pode levar a um consumo excessivo de memória e CPU para gerenciar todas as threads.

- Complexidade do Código:

O código de manipulação de sockets síncronos pode se tornar mais complexo quando é necessário lidar com várias conexões simultâneas. É necessário gerenciar manualmente o escalonamento de threads, o que pode tornar o código mais propenso a erros e difícil de manter.

-Escalabilidade Limitada:

A escalabilidade de aplicações que usam processamento síncrono em sockets pode ser limitada, especialmente quando há um grande número de clientes concorrentes. O gerenciamento de múltiplas threads pode se tornar um gargalo, prejudicando a capacidade da aplicação de escalar eficientemente.

-Timeouts:

Operações síncronas em sockets podem ficar bloqueadas indefinidamente, especialmente em situações de rede instável. A implementação de timeouts pode ser necessária para lidar com cenários em que uma conexão ou operação demora mais do que o esperado.

-Dificuldades com Comunicação Assíncrona:

Em aplicações onde a comunicação assíncrona é preferida, o modelo síncrono pode tornar-se um obstáculo. Em ambientes modernos, onde a eficiência e a capacidade de manipular simultaneamente várias operações são importantes, o processamento síncrono pode não ser a melhor escolha.

Link do Repositório:

https://github.com/miaspe/PNP