UNIVERSIDADE
FEDERAL DO CEARÁ

DAVI BEZERRA YADA
EDUARDO RUI BRANDÃO
GEORGE RICARDO

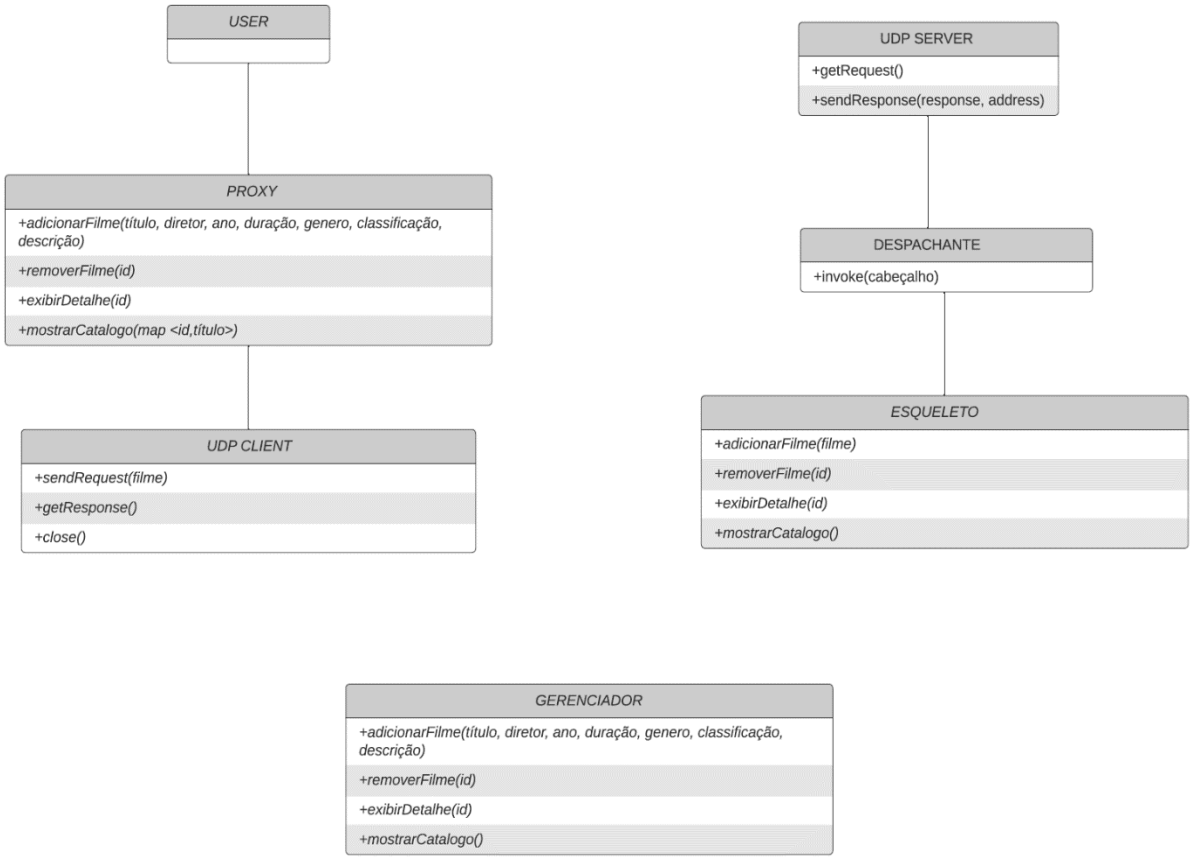
TRABALHO FINAL
SISTEMAS DISTRIBUÍDOS

QUIXADÁ-CE
2023

APRESENTAÇÃO DO PROJETO

Este trabalho tem como objetivo a implementação de uma aplicação no estilo Cliente/Servidor, cujo serviço fornecido é o de gerenciamento de catálogo de filmes utilizando banco de dados PostgreSQL. A conexão entre cliente e servidor é feita através de Socket UDP. O servidor está implementado na linguagem Java e o cliente em Python. Para que a comunicação entre cliente e servidor aconteça foi utilizada uma representação externa de dados em JSON para que a troca de mensagens fosse possível. O cliente tem as opções de **adicionar**, **remover**, **exibir detalhes** de um filme, e **mostrar o catálogo** completo. O servidor é responsável por receber as mensagens e fazer as operações condizentes. O servidor também é capaz de identificar se a requisição do cliente é duplicada ou não, guardando sempre a última mensagem recebida e fazendo a comparação, para que caso a mensagem seja duplicada, não seja necessário fazer o reprocessamento, apenas o reenvio da última requisição. Caso a mensagem não seja duplicada, o processamento da mensagem acontece normalmente. No lado do cliente, as operações são feitas pelo proxy que possui o método doOperation que faz todo o tratamento necessário para serialização da mensagem a ser enviada, e desserialização da resposta recebida.

Diagrama UML representando o esquema da aplicação



Cliente

Cabecalho.py

Esta classe faz a implementação da mensagem que é trocada entre cliente e servidor, é responsável por empacotar as informações do objeto a ser enviado, do tipo de mensagem (requisição ou resposta), o método escolhido, os argumentos utilizados em cada método e o requestId da mensagem.

Construtor (__init__):
O construtor inicia os atributos da classe (messageType, object reference, methodId, arguments, requestId) com os valores fornecidos como parâmetros durante a criação de uma instância da classe.

Método to_json:
Converte os atributos da instância da classe para um dicionário chamado headerData e, em seguida, usa a função dumps da biblioteca JSON para converter esse dicionário em uma string JSON. A string JSON resultante é retornada.

Método from_json:
Este é um método de classe, indicado pelo decorador @classmethod, que cria uma instância da classe a partir de uma string JSON fornecida como argumento. A função loads da biblioteca JSON é utilizada para converter a string JSON em um dicionário chamado headerData, e em seguida, uma instância da classe Cabeçalho é criada utilizando os valores do dicionário como argumentos. O get("requestId", 0) é usado para fornecer um valor padrão de 0 para o requestId caso ele não esteja presente no JSON.

Filme.py

Implementação da classe Filme que criará o objeto a ser manipulado na troca de mensagens entre cliente e servidor. Aqui são definidos os atributos do filme. Em síntese, a classe filme é usada para representar informações sobre um filme e fornece métodos para converter essas informações em uma string JSON.

Construtor(__init__):
O construtor inicia as propriedades da classe (id, título, diretor, ano, período, gênero, classificação, descrição) com valores passados como parâmetros quando a instância da classe é instanciada. Como o ID é auto incrementado no banco de dados, ele não precisa ser atribuído, então é definido como None.

Método to_json:
Converte as propriedades da instância da classe em um dicionário chamado filme_data e, em seguida, usa a função dump da biblioteca JSON para converter esse dicionário em uma string JSON. A string JSON resultante é retornada.

Método from_json:
Especificado pelo decorador @classmethod que cria uma instância da classe a partir da string JSON fornecida como argumento. A função de carregamento do módulo JSON é usada para converter a string JSON em um dicionário, e uma instância da classe filme é criada usando a sintaxe cls(**dictionary) para passar os valores do dicionário para a classe como argumentos nomeados construtor.

Proxy.py

Esta classe é a que de fato faz as operações do lado do cliente, ela possui as informações de conexão do cliente (host e porta), possui também os métodos que fornecerão os serviços para o cliente. Quando o usuário envia uma requisição, o proxy é responsável por fazer a manipulação dos dados e retornar a resposta para o cliente. Ele recebe a resposta serializada do servidor, realiza a desserialização, e devolve o resultado para o usuário. Estas operações são feitas no **doOperation**. O proxy também possui um **timeout** que estabelece um contador de tentativas de conexão com o servidor, caso esse limite seja atingido, ele encerra a conexão do cliente.

Construtor(__init__):
Inicializa um objeto proxy com um cliente UDP (UDPClient), um contador que rastreia tentativas de comunicação (count), o número máximo de tentativas (maxCount) e um identificador de solicitação (requestId).

Método adicionarFilme:
Adicione um filme chamando o método doOperation com os parâmetros apropriados. Processa a resposta do servidor e imprime a mensagem correspondente.

Método RemoverFilme:
Remove um filme chamando o método doOperation com os parâmetros apropriados. Trata as respostas do servidor e imprime mensagens apropriadas.

Método exibirDetalhe:
Exibe detalhes de um filme chamando o método doOperation com os parâmetros adequados. Trata as respostas do servidor e imprime detalhes do filme ou mensagens de erro.

Método mostrarCatalogo:
Exibe o catálogo de filmes chamando o método doOperation com os parâmetros adequados. Trata as respostas do servidor e imprime o catálogo ou mensagens de catálogo vazio.

Método de timeout:

Aqui é feita comparação do contador com o maxCount, caso seja menor, é exibida uma mensagem de tempo excedido (timeout), e incrementa + 1 ao valor do contador. Caso o contador ultrapasse o valor de maxCount, é exibida a mensagem que o servidor está indisponível e a conexão do cliente é encerrada.

Método doOperation:

É o método que realiza as operações. Cria uma mensagem de cabeçalho com todas as informações necessárias, serializa essa mensagem e encaminha para o servidor. Após isso, aguarda a resposta do servidor, caso não obtenha a resposta dentro do tempo estabelecido, é gerado um timeout, e então tenta novamente receber a mensagem do servidor, que nesse ponto terá reenviado a requisição, a partir disso, o método zera o contador de timeout, incrementa o requestId, pois a mensagem foi recebida, e envia a mensagem já retirando o lixo, ou seja, apenas a mensagem de fato.

UDPClient.py

Esta classe é responsável por criar o socket UDP do cliente, de estabelecer um tempo de resposta, receber e enviar mensagens. Em resumo, a classe implementa a lógica de comunicação UDP, permitindo que um cliente envie solicitações para um host e porta específicos, receba respostas e feche a conexão quando necessário. O uso de timeouts é incorporado para lidar com situações em que a resposta do servidor não é recebida dentro de um determinado período.

Construtor (__init__):
Inicializa os atributos da classe, como o host, a porta e o objeto de socket UDP. O socket é configurado para utilizar o protocolo IPv4 (AF_INET) e o tipo de socket datagrama (SOCK_DGRAM). Um timeout de 2 segundos é configurado para operações de recebimento.

Método sendRequest:
Este método envia uma solicitação para um determinado host e porta. O objeto a ser enviado é convertido para bytes usando a codificação UTF-8 antes de ser enviado.

Método getResponse:
Tentativa de recebimento de uma resposta do servidor. Utiliza a função recvfrom para receber até 1024 bytes de dados e o endereço do remetente. Caso ocorra um timeout (exceção timeout), retorna None. Caso contrário, decodifica os dados recebidos usando UTF-8 e os retorna como uma string.

Método close:
Encerra a conexão e fecha o socket do cliente.

User.py

Interface principal do cliente, na qual o usuário faz a interação com o serviço em questão. O cliente faz as operações juntamente ao proxy onde é feita uma instância da classe dentro da interface do usuário. Nesta classe é criado um menu interativo para que o usuário possa escolher entre as opções disponíveis. De acordo com a escolha, o proxy referente é acionado e realiza a operação correspondente.

Funções de teste (testaNum e testInput):
Tem por objetivo testar os inputs do usuário, para impedir que ele preencha os campos de forma incorreta colocando valores inconsistentes nas variáveis, evitando assim quebra de código ou bugs.

Loop principal:
Loop onde o usuário pode escolher entre as opções disponíveis para que sejam realizadas as operações correspondentes. As operações que podem ser realizadas são: adicionar um filme, remover um filme, exibir detalhes de um filme, mostrar o catálogo completo, limpar tela e sair do menu. Ao escolher a opção, adicionar um filme, por exemplo, o usuário informa os atributos necessários (título, diretor, ano, duração, gênero, classificação e descrição), é criado então um objeto Filme com esses atributos, e então o proxy se encarrega de encaminhar esse objeto. Caso o cliente escolha por sair, a conexão (socket) com o cliente é finalizada e o loop se encerra.

Servidor

UDPServer.java

Essa classe define os métodos de conexão do lado servidor. Ela será responsável por implementar os sockets de conexão, receber e tratar as requisições e enviar os serviços requisitados.

Método getRequest:

Método responsável por receber os bytes enviados pelo lado cliente através de um socket UDP. Esse método recebe uma mensagem Cabecalho e de acordo com o seu requestId é feito o devido tratamento. O servidor, antes de chamar os outros métodos para tratar a mensagem, verifica se o campo requestId da mensagem Cabecalho já foi recebido como requisição e enviado como resposta, como uma forma de tratar mensagens duplicadas. Se for o caso, ele possui um buffer que armazena a última resposta enviada, junto do endereço (IP e PORTA) do cliente e de lá pega a última resposta e faz o reenvio para o cliente, sem submeter o resto da mensagem para processamento.

Método sendResponse:

Esse método recebe como parâmetro um objeto cabeçalho e um DatagramPacket, em que o primeiro é uma mensagem a ser enviada para o cliente e o segundo é um pacote envolvendo as informações de endereço do cliente (IP e PORTA). O primeiro parâmetro é transformado em uma string JSON através do método toJson() e depois é feito um novo DatagramPacket com as informações do que foi passado por parâmetro, mas incluindo também a string JSON a ser enviada (os dados de resposta). Após a montagem do pacote, é realizado o envio através do método send().

Método isduplicatedMessage:

Esse método recebe como parâmetro os dados de endereço (IP e PORTA), o requestId e o methodId da última requisição que chegou ao servidor. Com esses argumentos, o método verifica se a mensagem já foi respondida pelo servidor e se necessita apenas de um reenvio ou se é uma nova mensagem que deve ser processada e enviada para o cliente.

Método sendLastResponse:

Esse método recebe como parâmetro o endereço do cliente (IP e PORTA). Cria uma mensagem com uma string informando que é uma resposta reenviada, cria um DatagramPacket inserindo essa mensagem e as informações da última mensagem enviada e após a criação desses dados, realiza o envio através do método send.

Despachante.java

Essa classe é responsável por implementar o método Invoke. A classe despachante é um intermédio entre as operações e o envio das mensagens, pois ela que controla o que deve ser feito com o Cabecalho recebido pelo cliente.

Método Invoke:

Esse método recebe uma string JSON que é o próprio cabeçalho enviado pelo cliente. Com essa mensagem, ele terá acesso aos seus atributos e irá ler o campo methodID. Com esse campo ela irá saber qual método chamar para tratar os argumentos do Cabecalho. Irá chamar a classe esqueleto para tratar os argumentos, receber a resposta e retornar um novo Cabecalho para o UDPServer.

Esqueleto.java

Essa classe possui as assinaturas idênticas às da classe que realmente implementa as operações finais com os objetos. É responsável por receber como parâmetro os argumentos do Cabecalho e a partir dele montar uma mensagem para cada operação que pode ser feita no serviço final.

Método adicionarFilme:

Esse método recebe uma string JSON com os argumentos do Cabecalho e cria um objeto Filme com o método fromJson. Após isso é chamado o método adicionarFilme da classe Database (que faz a implementação de fato do serviço) e então o objeto é passado.

Método removerFilme:

Esse método recebe uma string JSON com os argumentos do Cabecalho que contém o ID do filme a ser removido. Após isso o método removerFilme da classe Database é invocado e o ID é passado como parâmetro para remoção do filme do banco de dados.

Método exibirDetalhe:

Esse método recebe uma string JSON com os argumentos do Cabecalho que contém o ID do filme a ser exibido. A invocação do método exibirDetalhe é feita junto a classe Database, e então o filme com o ID especificado é retornado com os detalhes do filme.

Método mostrarCatalogo:

Esse método não recebe nenhum parâmetro, ele apenas é invocado dentro do Database que retorna o catálogo com ID e título dos filmes cadastrados no banco de dados.

Database.java

Essa classe é responsável por estabelecer a conexão com o banco de dados e implementar os métodos que de fato irão processar e fornecer os serviços para o cliente. Ela terá seus métodos chamados pelo esqueleto de acordo com a requisição.

Método adicionarFilme:

Esse método recebe um objeto Filme como argumento e faz o cadastramento desse filme no banco de dados utilizando os atributos que foram passados como parâmetros. Faz um query com as informações do Filme, e retorna o resultado (sucesso ou falha).

Método removerFilme:

Esse método recebe um ID como argumento e usa esse ID para fazer uma busca no banco de dados pelo filme especificado. Caso exista esse filme, ele é removido do banco, caso não exista, é retornado uma mensagem que o filme não está cadastrado.

Método exibirDetalhe:

Esse método recebe um ID como argumento e utiliza esse ID para buscar o filme no banco de dados. Caso o filme exista no banco, o método retorna as informações desse filme, caso não exista é retornado uma mensagem que o filme não está cadastrado.

Método mostrarCatalogo:

Esse método não possui parâmetros, ele apenas é invocado e faz um query no banco de dados que retorna todos os filmes que estão cadastrados, retornando um catálogo com ID e título dos filmes que estão no banco de dados. Caso o catálogo esteja vazio, é exibida uma mensagem do tipo.

Filme.java

Essa classe é responsável por fazer a implementação do objeto Filme do lado do servidor. Aqui o objeto Filme é criado com todos os atributos necessários. Possui métodos para fazer a serialização dos argumentos e também a desserialização. Os getters são responsáveis por fazer o retorno dos atributos do filme.

Método toJson:

É responsável por criar um novo JSONObject com os atributos do filme.

Método fromJson:

Faz a desserialização do JSONObject para uma string, após isso cria um novo objeto Filme com os atributos que foram desserializados.

Cabecalho.java

Essa classe faz a representação no lado do servidor da mensagem que será trocada entre cliente e servidor. A mensagem possui os campos de: tipo de mensagem, objeto que irá ser referenciado, o método utilizado e também os argumentos. Os getters são responsáveis por retornar os atributos da mensagem.

Método toJson:

Cria um novo JSONObject para representar a mensagem que irá ser serializada.

Método fromJson:

Desserializa o JSONObject e cria uma nova mensagem com os atributos obtidos.