Sistema de Partilha de Disponibilidades e Necessidades

Sarah Simon Luz – 38116 | Ana Laura Ferro - 39872

Sistemas Operativos | 26/04/2020

Índice

Objetivo 2
Estrutura do Trabalho 3
Interface 4
Classes 5
Implementação da base de dados 8
Caso a aplivação cliente crashar como são guardados os ultimos dados? 9

OBJETIVO

Foi pedido para construir uma aplicação servidor (que terá os dados, em armazenamento persistente), uma aplicação cliente (com funcionalidades de consulta de necessidades, reporte de disponibilidade de um produto num local, e registo de uma necessidade do produto X), uma implementação de middleware mencionada nas aulas.

Estas aplicações têm de comunicar entre si para:

- Guardar produtos determinados como necessários pelos utilizadores;
- Localização dos produtos em lojas cajo sejam encontrados por outros utilizadores;
- Listagem dos produtos necessários, sem identificar quem o adicionou como uma necessidade por motivos de segurança;
- Notificação quando um dos produtos é encontrado e adicionado à base de dados.

ESTRUTURA DO TRABALHO

Cliente/Servidor:

- A aplicação cliente comunica com a aplicação servidor através da invocação de métodos remotos (RMI). Ao chamar os métodos, os dados passados como parametros estarão serializados para abstrair da plataforma. Os métodos remotos irão aceder à base de dados, para inserir ou obter a informação desejada.

- Cliente tem três funcionalidades principais: registar, consultar e reporte.

    - Registar: regista a existencia de um produto numa loja;
    - Consultar: ver quais produtos sao necessidades;
    - Reporte: receber notificação da existencia do produto desejado.

Servidor/Base de Dados:

- Para ligar à base de dados a aplicação Servidor inicializa um objecto da classe AcessoBD que implementa a interface AcessoBDInterface e java.io.Serializable.

- Quando um objecto da classe AcessoBD é inicializado é feita uma conexão a base de dados através da classe PostgresConectar.

- A classe PostgresConectar, antes de poder fazer a conexão tem de obter as informações basicas da base de dados: host, nome da bade de dados, user da base de dados, password. Esses dados estão guardados num ficheiro chamado "bd.properties", que é acedido através da classe ValoresBD.

INTERFACE

AcessoBDInterface extends java.rmi.Remote

- Métodos:
  - **void adUtilizador**(String idCliente) throws java.rmi.RemoteException:
    - Adiciona à base de dados um utilizador novo,verificando antes se já existe um utilizador registado com o idCliente passado.

  - **void adLocalProduto**(PedidoDeRegisto p) throws java.rmi.RemoteException:
    - Adiciona à base de dados a loja onde o produto pode ser encontrado.

  - **ArrayList<PedidoDeConsulta> consulta**() throws java.rmi.RemoteException:
    - Consulta os items necessitados por outros utilizadores sem divulgar a identidade dos mesmos.

  - **int adNecessidades**(String id, String produto) throws java.rmi.RemoteException:
    - Adiciona à base de dados um produto que é considerado necessario pelo utilizador, guardando o id do mesmo. Devolve um id unico que representa a necessidade.

  - **ArrayList<PedidoDeConsulta> verificarExistencia**(String produto) throws java.rmi.RemoteException:
    - Verifica se o produto passado existe na table dos produtos localizados. se for encontrado em mais do que uma loja devolve um arraylist com objectos de PedidoDeConsulta com o nome da loja.

  - **ArrayList<PedidoDeReporte> notificarStock**(String idCliente) throws java.rmi.RemoteException:
    - Usado pela classe Notificacao, para determinar se os produtos adicionados pelo cliente como necessidades já foram encontrados.

CLASSES

AcessoBD extends UnicastRemoteObject implements AcessoBDInterface,java.io.Serializable:

- Classe principal para interagir com a base de dados.

Cliente

- Métodos:
  - **Cliente**(AcessoBDInterface bd, String idCliente):
    - É o construtor, inicia o ficheiro para guardar dados caso haja um crash da aplicação Cliente.
  - **reFicheiro**():
    - Re-inicia o BufferedWritter.
  - **menu**(AcessoBDInterface bd, Cliente cl):
    - Verifica se o Cliente crashou, caso tenha crashado imprime na consola os ultimos valores escritos pelo Cliente.
    - Pergunta ao Cliente que operação com a base de dados deseja fazer.
  - **registar**(AcessoBDInterface bd, Cliente cl):
    - Regista na base de dados a localização de um produto.
  - **consultar**(AcessoBDInterface bd):
    - Pergunta à base de dados os produtos considerados necessidades.
  - **reporte**(AcessoBDInterface bd, Cliente cl):
    - Utilizador reporta que produto necessita.
    - Primeiro o método averiguada na bd se existe a localizacao desse produto desejado.
    - Caso existir devolve a ou as lojas onde o utilizador pode encontrar os produtos.
    - Caso contrário, é devolvido o id da necessidade e inicializada um objecto da classe notificacao, que irá notificar quando for encontrado o produto.

Notificacao extends Thread implements Runnable:

- Utilizado para avisar ao cliente quando um produto foi econtrado numa loja, para o fazer usa threads.

- Métodos:

    - **run**():
        - Enquanto não for encontrado o produto a thread continua a correr.
        - Se o produto for encontrado, escreve na consola do Cliente uma mensagem de alerta a notificar que o produto x foi encontrado na loja y.

    - **terminar**():
        - "Mata" a execução da thread.

Pedido:

- Classe abstrata que ajuda na representação das classes: PedidoDeRegisto, PedidoDeConsulta, PedidoDeReporte.

PedidoDeConsulta extends Pedido implements java.io.Serializable:

- Guardar um produto ou loja. Usado no método consultar() da classe Cliente para guardar os produtos procurados num ArrayList e no método reporte da classe Cliente, para guardar as lojas que possivelmente têm o produto procurado num ArrayList.

PedidoDeRegisto extends Pedido implements java.io.Serializable:

- Usado para guardar a localização do produto e o produto.

PedidoDeReporte extends Pedido implements java.io.Serializable:

- Classe para guardar os pedidos de reporte para quando a localizacao de um produto é adicionado à bd.

PostgresConectar:

- Conecta a aplicação servidor.

- Métodos:

    - **PostgresConectar**() throws Exception:
        - Construtor sem parametros, vai buscar os dados com a classe ValoresBD e o método getProperties().

    - **PostgresConectar**(String host, String db, String user, String pw) throws Exception:
        - Construtor

    - **connect**() throws Exception:
        - Conecta com o servidor.

    - **disconnect**():
        - Disconecta do servidor.

- **Servidor**
    - Ligação à base de dados.
    - Inicialização do rmi registry.

- **ValoresBD**:
    - A classe ValoresBD utiliza as funcionalidades do java.util.Properties para obter os dados.

    - Métodos:
        - **String getProperties**(String name):

IMPLEMENTAÇÃO DA BASE DE DADOS

- Utilizador:
    - Guardar o id do utilizadores.

id	integer

- Necessidades:
    - Guardar as necessidades dos utilizadores.

id	integer
produto	varchar(30)
idnecessidades	serial

- localproduto:
    - Guardar as lojas onde se pode encontrar o produto.

loja	varchar(30)
produto	varchar(30)

CASO A APLIVAÇÃO CLIENTE CRASHAR COMO SÃO GUARDADOS OS ULTIMOS DADOS?

Quando a aplicação Cliente é inicializada é criado um ficheiro input_+ id do cliente. Se já foi, criado verifica se o ficheiro está vazio.

Caso esteja vazio, quer dizer que não crashou.

Caso contrário, quer dizer que a aplicação crashou. Porquê?

Sempre que uma operação Registar ou Reporte é feita a operação é escrita no ficheiro e o produto e/ou loja.

Exemplo:

- Registar
  produto:
  agua
  loja:
  aldi

No ficheiro será escrito: registar, agua, aldi.

Este ficheiro é sempre re-inicializado quando a operação termina com sucesso e fica vazio. Se crashar o ficheiro terá os ultimos dados escritos pelo cliente na consola dentro do ficheiro.