Mestrado Integrado em Engenharia Informática

Laboratórios de Informática III

Trabalho Java

Daniel Vieira A73974
Nadine Oliveira A75614
Duarte Freitas A63129

12 de Junho de 2018

Conteúdo

Capítulo 1

Introdução

Este trabalho está inserido na unidade curricular Laboratórios de Informática III, que tem como objetivo aplicar os diversos conhecimentos adquiridos até agora, nomeadamente na modulação e encapsulamento de dados. Nesta fase do projeto, após a implementação do mesmo na linguagem C, é proposto que este seja implementado na linguagem java. Para tal, foi necessário implementar uma classe onde foi estrturado um parser, capaz de processar vários ficheiros XML, que contêm várias informações presentes no Stack Overflow. Após estar efetuado o devido armazenamento da informação util retirada destes ficheiros, foi então iniciada o processo de respostas das interrogações propostas.

Capítulo 2

Parsing do XML e Estruturas

Para dar inicio ao projeto, foi necessário implementar um parser dos snapshots fornecidos. Para tal, recorreu-se à classe *SAXParser*. Escolheu-se a *SAXParser*, em detrimento da classe *DOMParser*, visto que o *DOMParser*, carrega todo o conteúdo dos ficheiros xml para memória enquanto o *SAXParser* processar o ficheiro nodo a nodo, usando apenas uma infima parte de memória. Dado o tamnho dos ficheiros a processar, o *SAXParser* foi sem dúvida a escolha a ser feita, para aém disto, efetua o parsing bastante mais rápidamente.
Dentro dos snapshots fornecidos, apenas quatro deles foram processados, pois bastava para recolher toda a informação necessária para responder às interrogações propostas, foram então processados os seguintes ficheiros: **Posts.xml**, **Users.xml**, **Votes.xml**, e ainda *Tags.xml*
Ao longo do parsing de cada ficheiro, foi sendo armazenada a informação obtida nas devidas estruturas. Para tal foram criadas quatro estruturas que se considerou necessárias, não sou para armazenar a informação, mas também para permitir obter um bom desempenho das queries a desenvolver.
Estruturas implementadas:

- **Users:**
  Para armazenar a informação dos users, foi criado um *hashMap*, mapeando por id de user um objeto *User*.

- **Posts:**
  Nos posts, optou-se por dividir a informação processada em perguntas e respostas, para tal foram criados dois *treeMaps*, mapeando por data um *hashMap* de posts mapeados por id de post. Assim, para cada chave do *treeMap* de posts, vai ter associado um *hashMap*, que contém todos os posts realizados naquele mês. Foi optado por criar um período mensal e não diário, porque concluiu-se que não iria trazer grande vantagem, e optou-se então, por seguir a mesma linha de pensamento que foi tida no projeto anterior desenvolvido em C. Optou-se ainda por usar um *TreeMap* em detrimento de um *HashMap* para os posts, devido ao facto de grande parte das queries implicarem ordenação cronológica, e o *TreeMap* permite, que à medida que os dados são inseridos, a ordenação seja mantida por chave, ou seja por data no caso. Com o *HashMap*, já não era possível obter esta vantagem, durante a insersação, este não mantém qualquer tipo de ordenação.

- **Tags:**
  Para as tags, foi apenas criado uma *hashMap*, mapeando por nome de tag o seu id.

Capítulo 3

Classes

Na implementação deste projeto, foram criadas as seguintes classes:

3.1 Users

Na classe *Users*, foram identificadas todas as variaveis relativas à informação dos users. Estas variáveis foram devidamente encapsuladas, bem como criados os *getters* e *setters* das mesmas, permitindo assim o acesso ao conteúdo das mesmas. A informação armazenada para os users foi: o id, nome, descrição (AboutMe), reputação e ainda o número de posts criados.

3.2 Posts

Na classe *Posts*, foram identificadas todas as variaveis relativas à informação dos posts, bem como os devidos *getters* e *setters* e construtores necessários. A informação armazenada para os posts foi: o id, tipo de post, data de criação, titulo, id da pergunta (parentId), as tags, o id do dono do post, a classificação (score), o número de comentários, o número de respostas e ainda o número de votos. Optou-se por calcular o número de votos de cada post à medida que era feito o parsing do ficheiro *Votes.xml*, evitando assim a criação de uma estrutura para guardar esta informação, e a necessidade de a processar mais tarde.

3.3 Struct

Na classe *Struct*, encontra-se declaradas todas as estruturas onde se guarda a informação recolhida. Todas as estruturas se encontram devidamente encapsuladas (private) e foram criados ainda os *getters* e *setters*, que permitem a aceder e manipular o seu conteúdo por métodos externos à classe.

3.4 TCDExample

Esta é a classe "principal", pois é nesta classe que vão ser inicializadas as estruturas e implementadas as soluções às interrogações propostas. Para tal, foi criada uma instancia da classe *Struct*, onde estão declaradas as estruturas a utilizar. Foi também criada uma instancia da classe *Parser*, que vai ser usada no método *load*, permitindo assim realizar o parsing dos ficheiros xml e o armazenamento da informação

nas respetivas estruturas.
Como classes auxilires, foram criadas mais duas, *AuxiliaPosts* e *AuxiliaUsers*, onde vão estar declarados alguns métodos auxiliares, usados na classe *TCDExample*. A criação destas classes, teve como objetivo, proporcionar um código mais legível e estruturado na implentação das queries.

Capítulo 4

Implementação das Interrogações

A Classe *TCDExample* contem todos os métodos responsáveis para a resolucao dos diversas queries que nos foram pedidas, bem como do método *load* responsável pelo carregamento dos dados apresentados nos snapshots dados pelos docentes.Estes métodos responsabilizam-se pelos dados necessários para obter os resultados pretendidos, sendo que, tomam partido das APIs de cada um dos objetos que compõem esta classe.

4.1 Interrogacoes propostas

Neste capitulo vai ser discutido com mais detalhe, a implementação da resolução às interrogações propostas.

4.1.1 init

- Método que instancia todas as classes necessarias para a conclusao do trabalho. Destas *Classes* fazem parte :

MyLog

Struct - Class que agrupa todas as estruturas;

Parser - Class que usa o StAX para fazer o parser dos ficheiros;

AuxiliaPosts - Class com os metodos auxiliares das querys relativos aos Posts;

AuxiliaUsers - Class com os metodos auxiliares das querys relativos aos Users;

4.1.2 load

- Método que carrega os ficheiros pretendidos usando métodos auxiliares que se encontram na Classe *Parser*.

4.1.3 Query 1 *infoFromPost*

Dado um id do Post retorna o título desse Post bem como o Nome do Autor.

-Para resolver esta interrogação percorremos os *Map* perguntas e respostas e caso encontremos um id igual retiramos as informacoes que precisamos(no caso de ser pergunta). Caso o post seja uma res-

posta,retiramos o id do post corresponde a resposta(*idParent*) e invocamos novamente a funçao mas com o novo ID como parametro.

4.1.4 Query 2 *topMostActive*

Retorna o Top N utilizadores com maior número de posts de sempre.

Criamos um *TreeMap* (*tree*) que vai ter como chave o CountPosts e como value a lista com os Users. Começamos por percorrer o TreeMap dos Users e coloca los na nova tree ficando assim ordenados pelo CountPost. De seguida percorremos esta tree e vamos passando os IDs para um ArrayList enquanto esta nao atingir o tamanho recebido como parametro(N).

4.1.5 Query 3 *totalposts*

-Dado um intervalo de tempo arbitrário,devolve o número total de posts. Visto que separamos os Posts em perguntas e respostas temos que percorrrer os dois Treemap . Utilizando o metodo subMap ficamos so com o intervalo de tempo desejado no entanto temos de verificar novamente a data pois dentro desse mes pode haver posts com dias inferiores ou superiores aos desejados. Apos os ciclos ficamos com os contadores de cada um e retornamos o Pair com os respetivos valores (numero perguntas, numero respostas).

4.1.6 Query 4 *questionsWithTag*

-Dado um intervalo de tempo arbitrário,devolve todas as perguntas que contêm a tag dada. Utilizando o metodo SubMap sobre a nossa TreeMap dos PostsPerguntas iremos ficar com o intervalo desejado. Ao percorrermos agora o TreeMap temos fazer a comparação das datas e logo a seguir verificar se a tag desejada faz parte das tags de cada pergunta. Se sim guardamos no nosso TreeMap auxiliar a data e os posts dessas respetivas datas. Por fim passamos os IDs do post para uma Lista e invertemos a lista pois estes Posts estavam organazidos cronologicamente.

4.1.7 Query 5 *getUserInfo*

No inicio criamos um *TreeMap* e um *ArrayList* a que demos o nome auxpost e lista,respetivamente. Para obtermos a short bio apenas precisamos de aceder ao HashMap dos Users e atraves do metodo get facilmente obtemos esse User. Agora percorremos os Posts das respostas e com o metodo descendingMap garantimos que vamos apanhar os ultimos posts. Guardamos alguns no nosso TreeMap e passamos para as perguntas onde vamos fazer exatamente o mesmo. Por fim fazemos novamente o descendingMap para obter os mais recentes e passamos para uma lista ate esta atingir o tamanho desejado. Agora retornamos um Pair com a informaçao do User(short bio) e os posts mais recentes.

4.1.8 Query 6 *mostVotedAnswer*

-Dado um intervalo de tempo devolve os IDs das N respostas com mais votos, decrescente do número de votos sendo que o numero de votos é obtido pela diferenca entre Up Votes e Down Votes

Utilizando o metodo SubMap sobre as respostas ficamos com o intervalo desejado. Agora percorremos o TreeMap das respostas obtido e vamos guardar num novo TreeMap os posts de acordo com o numero de votos. Por fim guardamos para a lista os IDs do posts ate essa mesma lista atingir o tamanho recebido como parametro.

4.1.9 Query 7 *mostAnsweredQuestions*

Com recurso ao método *SubMap* da classe *TreeMap*, foi possivel obter um *NavigableMap* que contém a informação relativa às perguntas, apenas do perído desejado. Depois foi guardado num *TreeMap* mapeado por número de respostas, todos os posts desse período. Por fim obteve-se o top N desse *treemMap*, que como estava mapeado por número de respostas, já se encontrava ordenado pelo que era pedido, retornando a lista comos ids dos posts com mais respostas.

4.1.10 Query 8 *containsWord*

Começou-se por percorrer o *treeMap* de todas as perguntas guardadas. Para cada pergunta encontrada é testado se, o titulo respetivo, contem a *word* fornecida como argumento. Caso contenha, o post é guardado num *List* de posts.

4.1.11 Query 9 * bothParticipated

Criamos no inicio 3 *TreeMaps* a que demos o nome de *auxpostP,auxpostR,both* e 1 *ArrayList* a que demos o nome de listafinal. Utilizando as funções auxiliares *guardaPerguntasOwnerID* e *guardaRespostasOwnerID* conseguimos guardar em *auxpostP* e *auxPostR* as perguntas e as respostas entre os id's,respetivamente. De seguida criamos 2 arrayList que vai ficar com o post das perguntas que o id1 e id2 têm em comum e é guardado num *TreeMap*.Fazemos o mesmo procedimento para as Respostas e guardamos num *List* os ids das N perguntas em que ambos participam, retornando essa lista.

4.1.12 Query 10 * betterAnswer

Percorreu-se o *HashMap* de respostas, e sempre que se encontra-se um post em que o seu *parentId* era igual ao id passado como argumento, recolhe-se toda a informação necessária para efetuar o calculo e é aplicada a formula descrita no enunciado. Esse resultado é guardado numa variável local, bem como o post respetivo, e à medida que as respostas são percorridas os calculos são efetuados e é guardado sempre aquele com um resultado superior. Por fim é devolvido o id da resposta com melhor pontuação.

4.1.13 Query 11 *mostUsedBestRep*

Com recurso ao método *SubMap* da classe *TreeMap*, foi possivel obter um *NavigableMap* que contém a informação apenas do perído desejado. De seguida foi guardado num *TreeMap*, todos os users que postaram perguntas nesse período, mapeado por reputação, tendo como valor uma lista de users. Este *TreeMap* foi ordenado e limitado ao top N de users referido no enunciado. Após se ter o top N de users, foi percorrido novamente o *NavigableMap* para o período estipulado. Para todos os posts presentes nesse período, verificou-se se pertenciam a um user presente no *TreeMap* de users calculado anteriormente. Caso pertence-se, as tags desse post eram guardadas num *HashMap* de tags e o seu contador incrementado. Por fim eese *HashMap* é ordenado pelo contador e é guardado numa lista os ids das tags respetivas.

Capítulo 5

Conclusão

Acabado o trabalho,cabe-nos fazer uma retrospetiva de todo o trabalho feito. Com a realização deste projeto usando JAVA,o grupo ficou a conhecer melhor esta linguagem orientada aos Objetos, sem descurar os principios de encapsulamento de dados, que garantem a segurança e o bom funcionamento do programa.

Fizemos também com que uma possivel modificação de uma Classe não danificasse o comportamento do programa, garantido um nivel de abstracao em todas as Classes.
Isto é,desde que as assinaturas dos métodos de cada classe se mantenham as mesmas,alterações às estruturas de dados da mesma apenas requerem pequenas correções nesses mesmos métodos,sendo que a reutilização do código é garantida em toda a API.
Achamos também mais fácil a realização do código das interrogações visto que,além de ser muito mais alto nível este paradigma, existem muitas classes já implementadas que usamos a nosso proveito.
Em suma,todo este projeto foi realizado de modo a responder a todos os problemas propostos no enunciado. Foi respeitado o encapsulamento de modo a proteger os dados, como é apanágio em projetos de grande dimensão como este.

Como trabalho futuro, consideramos que há ainda muitas otimizações que podem ser feitas, nomeadamente com recurso à classe *Stream* presente em java 8, permitindo assim, obter uns melhores tempos de execução das queries implementadas.