Paginação de registros sob demanda.

Recupere grande quantidade de registros consumindo menos recursos paginando eles por demanda.



**Cenário**

Considere uma aplicação que recupera grande volume de dados e pretende exibi-los para o usuário em uma tabela, trata-se – nesse caso – de um sistema de listagem de clientes que exibe todos os clientes em um *datatable* onde cada linha representa um cliente. Obviamente, para não tornar a listagem muito longa, as linhas são exibidas por partes que chamamos de páginas e o tamanho delas é determinado pelo desenvolvedor. Sendo assim, vamos analisar a forma como o conteúdo dessa tabela é gerenciado, de que maneira as páginas dessa tabela são exibidas e como carregar o conteúdo para elas, para isso temos duas possibilidades.

A primeira possibilidade consiste no cenário em que todos os clientes cadastrados na base são carregados e colocados em memória. Facilmente encontramos aplicações que fazem seleção em todos os registros da base de dados (sejam Clientes, Usuários entre outros) e inserem esses objetos em uma coleção para serem visualizados pelo usuário final. Muitas vezes alguma ação do usuário, por exemplo, qualquer filtro aplicado ou algum critério de ordenação no *datatable* faz com que os registros sejam todos carregados novamente.

O segundo cenário que vamos apresentar nesse artigo é aquele que é recuperado da base de dados apenas os registros que estão sendo visualizados pelo usuário, por exemplo, suponhamos que existam 4 mil clientes na base, com essa abordagem vamos selecionar apenas a parte que compõe a primeira página do *datatable,* ignorando o restante dos registros que serão carregados a medida que alguma ação do *datatable* for realizada (mudar a página, aplicar filtro, buscar, ordenar entre outras).

Utilizando JSF 2.2, Primefaces 5, Wildfly, JPA e EJB 3 fizemos uma aplicação para mostrar uma forma melhor de paginar um grande volume de dados, tendo em vista a performance, assim como a redução no tráfego de dados. Vamos comparar uma implementação que obtém os resultados através das duas formas mencionadas.

Introdução

Geralmente, as aplicações *web* que densevolvemos necessitam exibir uma grande quantidade de registros em uma tabela, a paginação desses registros é uma forma de interação com o usuário que permite dividir o conteúdo em várias páginas, evitando que as páginas fiquem desnecessariamente longas. Existem vários artigos e discussões na *web* bem relevantes sobre implementar ou não paginação no conteúdo da nossa aplicação e, caso seja implementado, qual deve ser o tamanho das páginas. Veja por exemplo a postagem [Avoid the Pains of Pagination](http://uxmovement.com/navigation/avoid-the-pains-of-pagination/) e os artigos: [The Impact of Paging vs. Scrolling on Reading Online Text e Passages](http://usabilitynews.org/the-impact-of-paging-vs-scrolling-on-reading-online-text-passages/) [Paging vs. Scrolling: Looking for the Best Way to Present Search Results](http://usabilitynews.org/paging-vs-scrolling-looking-for-the-best-way-to-present-search-results/).

Uma vez que vamos implementar paginação para o conteúdo exibido em nossa aplicação, há uma porção de coisas que devemos nos atentar. Principalmente, a forma como o conteúdo é recuperado da base de dados e colocado na memória para que fique disponível ao usuário final.

A solução comumente adotada para essa tarefa é a de recuperar uma única vez o conteúdo da base de dados e depois adiciona-lo em uma coleção para ser utilizado pelo usuário e, a partir dessa coleção, pode-se paginar os dados em uma tabela. Note que, independente do volume de dados, no momento que o usuário solicita – por exemplo – os clientes cadastrados no sistema, a aplicação obtém tudo e coloca isso disponível na memória. Ou seja, a paginação pode ser feita com o que foi recuperado da base de dados, mas esse conteúdo na verdade já está totalmente disponível para o cliente (embora não esteja visível), pois a aplicação o colocou em memória. Então, dizemos que a paginação é feita em memória.

Para resolver esse problema, utilizamos o que chamamos de paginação por demanda, que consiste em obter da base de dados apenas o conteúdo no qual o usuário possa interagir, ou seja apenas a página que é mostrada. Assim, é reduzido tanto a quantidade memória quanto o trafego de dados que é utilizado pela aplicação.

A seguir, veremos ambas as implementações em funcionamento, tanto a paginação em memória quanto a paginação por demanda e faremos um estudo de caso que consiste em uma comparação da quantidade de memória e tempo utilizado no momento que consultamos os registros cadastrados na base.

Descrição do problema

Implementação de prova de conceito

Ambiente de desenvolvimento

Paginação em memória

Paginação por demanda

Estudo de caso

Memória

Tempo

Conclusão

Referências

* <http://uxmovement.com/navigation/avoid-the-pains-of-pagination/>
* <http://usabilitynews.org/the-impact-of-paging-vs-scrolling-on-reading-online-text-passages/>
* <http://usabilitynews.org/paging-vs-scrolling-looking-for-the-best-way-to-present-search-results/>

Autores

|  |  |
| --- | --- |
| C:\DevMedia\Revistas\Java Magazine\_Edicoes\jm 105\pablo.png |  |
| **Pablo Bruno de Moura Nóbrega** <http://pablonobrega.wordpress.com> | **Joel Xavier Rocha**  <http://joelxr.github.io> |
| [pablonobrega2004@gmail.com](mailto:pablonobrega2004@gmail.com) | [joelxr@gmail.com](mailto:joelxr@gmail.com) |
| Líder de Projeto Java, certificado OCJP e OCWCD, Graduado em Ciências da Computação pela Universidade de Fortaleza – UNIFOR, Mestre em Computação pela Universidade Estadual do Ceará – UECE, cursa MBA em Gerenciamento de Projetos na UNIFOR, trabalha na Secretaria Municipal de Finanças de Fortaleza – SEFIN e desenvolve sistemas há cerca de oito anos. | Analista de Sistemas Java, certificado OCJP, bacharel em Engenharia da Computação pelo Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE). Trabalha na Secretária de Finanças do Município de Fortaleza no Ceará (SEFIN) e desenvolve e mantem sistemas há mais de cinco anos. |