11

Opcional: Buscando dados sob demanda com LazyDataModel do Primefaces

Até então nossa paginação está sendo feita apenas em memória já que a lista de livros ainda está sendo carregada por completo. O ideal é que, para melhorar a performance, realizemos a paginação também no banco de dados. Assim carregamos os produtos somente que iremos ver naquele momento.

Para fazer isso o Primefaces possui um componente chamado `LazyDataModel`. Que é responsável por buscar uma quantidade (definida pelo `rows`) de instâncias de um modelo (no nosso caso `Livro`). Nossa missão agora é alterar um pouco nosso código e fazer uso desse magnífico componente :). Vamos lá!

1) Crie uma classe no pacote `br.com.caelum.livraria.modelo` com o nome `LivroDataModel` que herde da classe `LazyDataModel`. A classe deve ser tipada com *generics* com o tipo do nosso modelo (o Livro):

```
public class LivroDataModel extends LazyDataModel<Livro> {

}
```

2) Precisamos dizer ao `LazyDataModel` qual é o valor máximo de registros que possuímos de livros. Vamos fazer isso em seu construtor chamando o método `setRowCount()` passando a quantidade. Podemos usar o método `contaTodos` do nosso DAO genérico.

```
public LivroDataModel() {
    super.setRowCount(???);
}
```

Vamos precisar buscar a quantidade de livros que possuímos no banco e para isso vamos precisar instanciar nosso DAO genérico dentro da classe `LivroDataModel`. Além disso, crie um método `quantidadeDeElementos` dentro da classe `DAO` com a seguinte implementação:

```
// classe DAO
public int quantidadeDeElementos() {
        EntityManager em = new JPAUtil().getEntityManager();
        long result = (Long) em.createQuery("select count(n) from " + classe.getSimpleName() + "
                .getSingleResult();
        em.close();

        return (int) result;
    }
```

O construtor da classe *LivroDataModel* ficará:

```
public LivroDataModel() {
    super.setRowCount(dao.quantidadeDeElementos());
}
```

3) Precisamos também sobreescrever o método `load` que será responsável por buscar uma quantidade de livros no banco de dados e devolvê-la ao `dataTable` :

```
@Override
public List<Livro> load(int inicio, int quantidade, String campoOrdenacao, SortOrder sentidoOrde

}
```

Para buscar a lista paginada, a JPA nos ajuda através dos métodos `setFirstResult` e `setMaxResult` da interface `Query` . Porém já criamos um método que faz busca paginada para você :). Ele se chama `listaTodosPaginada` e está dentro do DAO genérico:

```
@Override
public List<Livro> load(int inicio, int quantidade, String campoOrdenacao, SortOrder sentidoOrde
    return dao.listaTodosPaginada(inicio, quantidade);
}
```

4) Crie um atributo do tipo `LivroDataModel` dentro da classe `LivroBean` . (**Não esqueça de criar o getter para ele**)

```
public class LivroBean implements Serializable {
    // outros atributos

    private LivroDataModel livroDataModel = new LivroDataModel();

    // outros atributos
    // getters setters
}
```

5) Por último, altere o valor do atributo `value` do componente `p:dataModel` para realizar o binding com o `LivroDataModel` . Além disso, coloque `lazy=true` :

```
<h:form id="formTabelaLivros">
    <p:dataTable value="#{livroBean.livroDataModel}" var="livro" id="tabelaLivros" paginator="tr
```

Reinicie o servidor, a paginação deve continuar funcionando. Porém dessa forma, buscamos os dados no banco conforme precisamos deles :) Mas ainda temos um problema: precisamos alimentar o filtro, ou seja, buscar os livros pelo título. Como fazer?

Podemos receber os valores que foram entrados pelo usuário nos filtros através do último parâmetro do método `load` da classe `LivroDataModel` :

```
@Override
public List<Livro> load(int inicio, int quantidade, String campoOrdenacao, SortOrder sentidoOrde
    String titulo = (String) filtros.get("titulo");
    return dao.listaTodosPaginada(inicio, quantidade);
}
```

Vamos alterar um pouco o método `listaTodosPaginada` para aplicar o filtro na busca:

```
// classe DAO
public List<T> listaTodosPaginada(int firstResult, int maxResults, String coluna, String valor)
    EntityManager em = new JPAUtil().getEntityManager();
    CriteriaQuery<T> query = em.getCriteriaBuilder().createQuery(classe);
    Root<T> root = query.from(classe);

    if(valor != null)
        query = query.where(em.getCriteriaBuilder().like(root.<String>get(coluna), valor + "%")

    List<T> lista = em.createQuery(query).setFirstResult(firstResult).setMaxResults(maxResults)

    em.close();
    return lista;
}
```

Estamos usamos a *Criteria API* da JPA 2.0 para implementar essa busca. Caso queira aprender mais sobre ela pode assistir nosso [treinamento de JPA (https://cursos.alura.com.br/course/jpa-avancado)](https://cursos.alura.com.br/course/jpa-avancado).

Por fim, no método `load` vamos passar os novos argumentos ao método `listaTodosPaginada` :

```
@Override
public List<Livro> load(int inicio, int quantidade, String campoOrdenacao, SortOrder sentidoOrde
    String titulo = (String) filtros.get("titulo");

    return dao.listaTodosPaginada(inicio, quantidade, "titulo", titulo);
}
```

Agora, após reiniciarmos o servidor podemos ver o filtro funcionando.