PROJETO FRAMEWORK - CELEPAR

PAGINAÇÃO DE DADOS EM APLICAÇÕES JAVA PARA INTERNET

Janeiro – 2005

Sumário de Informações do Documento

Tipo do Documento: Definição		
Título do Documento: Controle de Exceções em Aplicações Java para Internet		
Estado do Documento: EB (Elaboração)		
Responsáveis: Cleverson Budel, Diego Pozzi, Fábio Sgoda, Filipe Lautert		
Palavras-Chaves: paginação, java, controle de fluxo, projetos web		
Resumo: Tratamento de paginação em aplicações java para web		
Número de páginas: 11		
Software utilizados:		
Versão	**Data**	**Mudanças**
1.0		

Sumário

1 PAGINAÇÃO DE DADOS EM APLICAÇÕES JAVA PARA INTERNET

1.1 Introdução

Visando diminuir o tempo de respostas as requisições feitas por usuários em sistemas web e não onerar o desempenho dos servidores de aplicação foi padronizada a forma de paginação de dados nas aplicações desenvolvidas em java.

1.2 Descrição da forma de paginação

A arquitetura dos sistemas está dividida em camadas, onde cada camada tem seus objetos com suas respectivas tarefas. Na camada de acesso a dados existem objetos do tipo DAO (Data Acess Object) que fazem a ponte entre o banco de dados e a camada de negócios da aplicação.

Nos objetos do tipo DAO deve-se implementar os métodos de busca paginada. É solicitado ao banco de dados que traga apenas a quantidade de registros que se deseja mostrar na tela.

A camada de negócios da aplicação, representada pelas classes do tipo Facade, é responsável em chamar os métodos das classes DAO. A coleção de registros retornada pelo método da classe DAO é repassada para as Facades que devem fazer alguns tratamentos, como: se for solicitada a primeira paginação além da busca pelos registros paginados é necessário que se busque o número da quantidade total de registros que a pesquisa retornaria se não estivesse paginada (select count), pois só através desta informação torna-se possível informar ao usuário a quantidade de páginas que podem ser buscadas.

As informações necessárias a paginação na camada de apresentação são encapsulados em um objeto chamado Pagina. Este objeto será passado para a página JSP que fará o devido tratamento para a apresentação.

1.3 Exemplo

A seguir, demonstramos através de um exemplo a forma de paginação que foi descrita acima.

1.3.1 Camada de Acesso a Dados

Como dito anteriormente faz parte da camada de acesso a dados as classes do tipo DAO. Aqui temos um exemplo de implementação desta classe para realizar busca de alunos de forma paginada.

```
package gov.pr.celepar.framework.dao.implementation;

import org.apache.log4j.Logger;
import java.util.*;
import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Query;
import gov.pr.celepar.framework.pojo.Aluno;
import gov.pr.celepar.framework.dao.AlunoDAO;
import gov.pr.celepar.framework.database.BaseHibernateDAO;
import gov.pr.celepar.framework.exception.ApplicationException;

/**
 * Classe Exemplo:
 * Responsável por manipular os dados da tabela de aluno.
 */
public class HibernateAlunoDAO extends BaseHibernateDAO implements AlunoDAO{

	private static Logger log = Logger.getLogger(HibernateAlunoDAO.class);
	private static Logger logAuditoria = Logger.getLogger("AUDITORIA");

	/**
	 * Recupera lista de alunos.
	 * @return Collection
	 * @throws ApplicationException, Exception
	 */
	public Collection listarAluno() throws ApplicationException, Exception{
		return listarAluno(null, null);
	}

	/**
	 * Recupera lista de alunos.
	 * Se for passado valores válidos para os parâmetros é realizada uma pesquisa
	 *  paginada, senão pesquisa todos os dados e retorna a coleção completa.
	 * @param qtdPagina - quantidade de objetos por página.
	 * @param numPagina - número da página a ser pesquisada.
	 * @return Collection
	 * @throws ApplicationException, Exception
	 */
	public Collection listarAluno(Integer qtdPagina, Integer numPagina) throws ApplicationException, Exception{
		Collection coll = new ArrayList();
		try {
			openSession();
			Query q = session.createQuery("from Aluno");
			if (qtdPagina != null && numPagina != null) {
				q.setMaxResults(qtdPagina.intValue());
				q.setFirstResult( (numPagina.intValue()-1) * qtdPagina.intValue());
			}
			coll = q.list();
			logAuditoria.info("LISTAGEM DE ALUNOS: Realizada pelo usuário Xxxxxx");
		} catch (HibernateException he) {
```

```
                    throw new ApplicationException("msgKey", he);
                } catch (Exception e) {
                    throw e;
                } finally {
                    try {
                        closeSession();
                    } catch (Exception e) {
                        log.error("Erro ao Fechar Conexao com o Hibernate: ", e);

                    }
                }
                return coll;
        }

        /**
         * Recupera quantidade total de alunos.
         * @return Collection
         * @throws ApplicationException, Exception
         */
        public Integer buscarQtdAlunos() throws ApplicationException, Exception {
                Integer qtd = null;
                try {
                    openSession();
                    Query q = session.createQuery("select count(*) from Aluno");
                    qtd = (Integer) q.uniqueResult();
                } catch (HibernateException he) {
                    throw new ApplicationException("msgKey", he);
                } catch (Exception e) {
                    throw e;
                } finally {
                    try {
                        closeSession();
                    } catch (Exception e) {
                        log.error("Erro ao Fechar Conexao com o Hibernate: ", e);
                    }
                }

                return qtd;
        }

        // ... Ourtros métodos referentes a Alunos
}
```

Observamos nesta classe que ela possue duas assinaturas de métodos para retornar a lista de alunos mas apenas uma implementação real do método, isso facilita uma possível alteração. Observe também que temos o método que retorna a quantidade total de registros.

1.3.2 Camada de Negócios

Na camada de negócios temos as classes do tipo Facade que contem todas as regras da aplicação. Essas classes chamam os métodos necessários das classes DAO. Abaixo temos um exemplo de uma Facade que realiza os passos necessárias para uma busca paginada de alunos.

```
package gov.pr.celepar.framework.facade;

import java.util.Collection;
import gov.pr.celepar.framework.dao.factory.DAOFactory;
import gov.pr.celepar.framework.exception.ApplicationException;
import gov.pr.celepar.framework.pojo.Aluno;
import gov.pr.celepar.framework.pojo.Endereco;
import gov.pr.celepar.framework.util.Pagina;

/**
 * Classe Exemplo:
 * Responsável por encapsular os serviços de matrícula e a sua toda regra de negócio.
 */
public class MatriculaFacade{

	/**
	 * Busca um objeto Aluno através de seu código.
	 * @param codAluno código do Aluno a ser localizado
	 * @return Aluno
	 * @throws ApplicationException, Exception
	 */
	public Aluno buscarAlunoPorPK(Integer codAluno) throws ApplicationException, Exception{
		DAOFactory hibernateFactory = DAOFactory.getDAOFactory(DAOFactory.HIBERNATE);
		return hibernateFactory.getAlunoDAO().buscarAlunoPorPK(codAluno);
	}

	/**
	 * Recupera lista de alunos.
	 * @return Collection
	 * @throws ApplicationException, Exception
	 */
	public Collection listarAluno() throws ApplicationException, Exception{
		DAOFactory hibernateFactory = DAOFactory.getDAOFactory(DAOFactory.HIBERNATE);
		return hibernateFactory.getAlunoDAO().listarAluno();
	}

	/**
	 * Lista paginada de alunos.
	 * @param pag - objeto de paginação contendo parametros para pesquisa.
	 * @return Pagina - encapsula resultados da pesquisa e parametros para paginação.
	 * @throws ApplicationException, Exception
	 */
	public Pagina listarAluno(Pagina pag) throws ApplicationException, Exception{
		DAOFactory hibernateFactory = DAOFactory.getDAOFactory(DAOFactory.HIBERNATE);
		if (pag.getPaginaAtual().intValue() == 1) {
			pag.setTotalRegistros(hibernateFactory.getAlunoDAO().buscarQtdAlunos());
		}
		pag.setRegistros(hibernateFactory.getAlunoDAO().listarAluno(pag.getQuantidade(), pag.getPaginaAtual()));

		return pag;
	}
}
```

Vemos neste exemplo que o método que realiza a listagem de alunos - listarAluno(Pagina pag)

– verifica se a página a ser pesquisada é a primeira, se for, é feita a busca pela quantidade total de registros para informar a quantidade de páginas que podem ser visualizada pelo usuário.

1.3.2 Camada de Apresentação

1.3.2.1 Action

Na camada de negócios temos as classes do tipo Action que tem a responsabilidade de gerenciar as requisições dos usuários. Essas classes gerenciam e alimentam os dados que serão passados para as JSPs, as quais farão a apresentação da tela.

Abaixo temos um exemplo da classe Action que gerencia a paginação de alunos:

```
package gov.pr.celepar.framework.action;

import java.util.Collection;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import gov.pr.celepar.framework.form.MatriculaForm;
import gov.pr.celepar.framework.facade.MatriculaFacade;
import gov.pr.celepar.framework.pojo.Aluno;
import gov.pr.celepar.framework.util.Dominios;
import gov.pr.celepar.framework.util.Pagina;
import gov.pr.celepar.framework.exception.ApplicationException;

/**
 * Classe Exemplo:
 * Responsável por manipular as requisições do usuário.
 */
public class MatriculaAction extends BaseDispatchAction{

	/**
	 * Realiza o encaminhamento necessário para executar a listagem de alunos.
	 * @param mapping
	 * @param form
	 * @param request
	 * @param response
	 * @return ActionForward
	 * @throws ApplicationException
	 * @throws Exception
	 */
	public ActionForward listarAlunos(ActionMapping mapping, ActionForm form, HttpServletRequest request,
HttpServletResponse response) throws ApplicationException, Exception{
		MatriculaForm matriculaForm = (MatriculaForm)form;
```

```
                MatriculaFacade matriculaFacade= new MatriculaFacade();

                try {
                        Collection listAlunos= matriculaFacade.listarAluno();
                        request.setAttribute("listAlunos", listAlunos);
                        actionForward= mapping.findForward("proximaPagina");
                } catch (ApplicationException appEx){
                        actionForward= mapping.findForward("paginaAtual");
                        throw appEx;
                } catch (Exception e) {
                        actionForward= mapping.findForward("paginaAtual");
                        throw e;
                }

                return actionForward;
        }

        /**
         * Realiza o encaminhamento necessário para executar a listagem paginada de alunos.
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return ActionForward
         * @throws ApplicationException
         * @throws Exception
         */
        public ActionForward listarAlunosComPaginacao(ActionMapping mapping, ActionForm form, HttpServletRequest request,
HttpServletResponse response) throws ApplicationException, Exception{
                MatriculaForm matriculaForm= (MatriculaForm)form;
                MatriculaFacade matriculaFacade= new MatriculaFacade();

                String indicePagina= request.getParameter("indice") == null ? "1" : request.getParameter("indice");
                String totalRegistros= request.getParameter("totalRegistros") == null ? "20" : request.getParameter("totalRegistros");

                Pagina pagina = new Pagina(new Integer(Dominios.QTD_PAGINA), new Integer(indicePagina), new Integer
(totalRegistros));

                try {
                        pagina = matriculaFacade.listarAluno(pagina);

                        request.setAttribute("pagina", pagina);

                        actionForward= mapping.findForward("pgAlunos");
                } catch (ApplicationException appEx){
                        actionForward= mapping.findForward("pgAlunos");
                        throw appEx;
                } catch (Exception e) {
                        actionForward= mapping.findForward("pgAlunos");
                        throw e;
                }

                return actionForward;
        }
}
```

O método listarAlunosComPaginacao faz o gerenciamento da paginação. Observe que é criado um objeto do tipo Pagina que encapsula todos os dados necessários a paginação, como:

quantidade de registros que devem ser mostrados em cada página, página corrente, quantidade total de registros que podem ser exibidos e a coleção de dados a ser mostrada em cada página. Este objeto é utilizado também pela JSP que irá apresentar os dados, com isso evitamos de passar para vários parâmetros para JSP, passamos apenas o objeto Pagina.

1.3.2.1 JSP

Aqui está a JSP utilizada para mostrar os dados dos alunos. Utilizamos neste exemplo a taglib para geração de tabela de resultados disponível nos componentes do grupo framework. É aconselhado, mas não obrigatório, que façamos uso desta taglib.

```
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://celepar.pr.gov.br/taglibs/html-1.0" prefix="ch" %>
<%@ page import="gov.pr.celepar.framework.util.Pagina" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html locale="true">
  <head>
    <html:base />
    <title>Exemplo – Lista de Alunos</title>
    <link href="../../css/default.css" rel="stylesheet" type="text/css">
  </head>

  <body>
    <%@ include file="/pages/ctlr_mensagens.jsp" %>
    <html:form action="/matricula" method="post">
      <table border="0">
        <tr>
          <td><b>LISTA DE ALUNOS</b></td>
        </tr>
      </table>

       <c:url var='icon_exibir' value='/images/icon_exibir.png' />
       <c:url var='icon_alterar' value='/images/icon_alterar.png' />
       <c:url var='icon_excluir' value='/images/icon_excluir.png' />
       <c:url var='link_navegacao_grid'
value='/matricula.do?action=listarAlunosComPaginacao' />
       <c:url var='link_exibir_grid'
value='/matricula.do?action=exibirAluno&codigo=%1' />
       <c:url var='link_alterar_grid'
value='/matricula.do?action=alterarAluno&codigo=%1' />
       <c:url var='link_excluir_grid'
value='/matricula.do?action=excluirAluno&codigo=%1' />

       <ch:table classTable="tabela" classTitulo="titulo" classLinha1="linha1"
classLinha2="linha2">
           <ch:lista bean="${pagina}" atributoId="cdAluno" />
           <ch:action imagem="${icon_exibir}" link="${link_exibir_grid}"
label="Exibir" />
           <ch:campo atributo="nomeAluno" label="Nome Aluno" />
           <ch:campo atributo="cpfAluno" label="CPF" />

           <ch:action imagem="${icon_alterar}"  link="${link_alterar_grid}"
label="Alterar" />
```

```
            <ch:action imagem="${icon_excluir}" link="${link_excluir_grid}"
label="Excluir" />
            <ch:painel pagina="${link_navegacao_grid}" classe="painel"
atributoIndice="indice" />
        </ch:table>

    </html:form>
  </body>
</html:html>
```

Resultado gerado:

1.4 Conclusão

Muitas vezes não nos preocupamos com a paginação de dados o que nos causa inúmeros transtornos em questões de performance, gerenciamento e layout. Por isso é muito importante adotarmos um padrão para realização de paginação.

Qualquer controle de paginação onera o tempo de desenvolvimento do projeto, mas é necessária. A forma apresentada atende a realidade da empresa e visa, com a padronização, diminuir o tempo gasto com esse controle.