Struts Paging

This tutorial explain two ways to page through a list in struts. The first way works with normal struts tags and and the second with the really good display tag library.

General

Author:
Sascha Wolski
Sebastian Hennebrueder
http://www.laliluna.de/tutorials.html – Tutorials for Struts, EJB, xdoclet and eclipse.

Date:
Updated: December 2007
First edition: March 2005

Software:
Eclipse 3.x
Display Tag Library
Common-Lang 2.0
Struts 1.3 but can be used with newer Struts versions as well

PDF download: http://www.laliluna.de/download/struts-paging-en.pdf
Source download: http://www.laliluna.de/download/struts-paging.zip

Requirments

First off all we require the display tag library (http://www.displaytag.org) and for the display tag library you need the commons-lang and common-collections library.

Paging in Struts

Paging lists in struts costs a lot of efforts and there is no standard way to do this. While writing the example application for this tutorial, I found many problems on paging lists with the standard tag libraries of struts. I think a good alternative way to page through lists in struts is the display tag library, I will also explain in this tutorial. But first we do a paging of a list with the standard tag libraries.

Create a new Struts Project

Create a new struts project *PagingExample* and add a package *de.laliluna.tutorial.paging*.

Object Class Customer

Create a new java class *Customer* in the package *de.laliluna.tutorial.object*.

Add four properties, *id* of type int, *name*, *city* and *phone* of type String.

Provide a getter and setter method for each property.

Add a constructor which allow you to initialize the properties.

The following source code shows the class Customer:

```
/**
 * Object Class Customer
 */
public class Customer {

      //properties
      private int id;
      private String name;
```

```
	private String city;
	private String phone;

	//constructors
	public Customer(){}
	public Customer(int id, String name, String city, String phone){
		this.id = id;
		this.name = name;
		this.city = city;
		this.phone = phone;
	}

	//getter and setter methods
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getPhone() {
		return phone;
	}
	public void setPhone(String phone) {
		this.phone = phone;
	}
}
```

New Class TestList

Create a new java class *TestList* in the package *de.laliluna.tutorial.object*.

This class provides methods for paging a list.

Add three properties *list* of type ArrayList, *offset* and *length* of type int.

Provide a getter and setter methods for each property.

Add a constructor to initialize the properties and execute a method which fills the property list with a list of dummy data.

The following source code shows the class TestList:

```
/**
 * Class TestList
 */
public class TestList {

	//properties
	ArrayList list;
	int offset;
	int length;
```

```
	//constructors
	public TestList(int offset, int length){
		this.offset = offset;
		this.length = length;
		fillList();
	}

	//fill the list with dummy data
	public void fillList(){
		list = new ArrayList();
		list.add(new Customer(1, "Marie", "New York", "061/123456"));
		list.add(new Customer(2, "Klaus", "Bouston", "071/654321"));
		list.add(new Customer(3, "Peter", "Detroit", "091/3432432"));
		list.add(new Customer(4, "Alex", "New York", "061/1324234"));
		list.add(new Customer(5, "Eve", "Houston", "051/885656"));
		list.add(new Customer(6, "Sebastian", "Seattle", "0811/8234240"));
		list.add(new Customer(7, "Marie", "Ohio", "0771/23235"));
		list.add(new Customer(8, "Gabriel", "Oklahoma", "011/9945461"));
		list.add(new Customer(9, "John", "California", "0871/8230013"));
		list.add(new Customer(10, "Dave", "Iowa", "0981/2320026"));
	}

	//get list by offset and length
	// returns a subset of the list, from offset to offset + length
	public ArrayList getListByOffsetAndLength(){

		ArrayList arrayList = new ArrayList();

		//calc the to value
		int to = this.offset + this.length;
		//if offset or lenght exceed set them to list.size
		if(this.offset > list.size()) this.offset = list.size();
		if(to > list.size()) to = list.size();
		//add the list items to the return collection
		for(int i=this.offset;i<to;i++){
			arrayList.add(list.get(i));
		}
		//return the collection
		return arrayList;
	}

	//get list site
	public int getListSize(){
		return list.size();
	}

	//get pages returns a list of Integer values, representing the pages
	public Collection getPages(){

		Collection collection = new ArrayList();
		//calc how many pages are there
		int pages = list.size() / this.length;
		if(list.size() % this.length != 0)
			pages = pages + 1;

		//fill an collection with all pages
		for(int i=1;i<=pages;i++){
			collection.add(new Integer(i));
		}

		return collection;
	}

	// getter and setter methods
```

```
	public ArrayList getList() {
		return list;
	}
	public void setList(ArrayList list) {
		this.list = list;
	}
	public int getLength() {
		return length;
	}
	public void setLength(int length) {
		this.length = length;
	}
	public int getOffset() {
		return offset;
	}
	public void setOffset(int offset) {
		this.offset = offset;
	}
}
```

Action Class

Create a new java class *ExampleAction* in the package *de.laliluna.tutorial.paging.action*, which extends the class *Action*.

Add a variable *maxEntriesPerPage* to specify how many entries are displayed per page.

Add a variable *page* which holds the current page of the list. This variable will be set by the request parameter *page.* You have to catch a *NumberFormatException*, because the exception can occur if the request parameter *page* is null.

Calculate the offset with the *maxEntriesPerPage* and the current *page* value.

Initialize the *TestList* class and set it in the request.

Return the forward *example*.

```
/**
 * Action Class ExampleAction
 */
public class ExampleAction extends Action {

	public ActionForward execute(
		ActionMapping mapping,
		ActionForm form,
		HttpServletRequest request,
		HttpServletResponse response) {

		int maxEntriesPerPage = 3;

		int page = 1;
		try{
			page = Integer.parseInt(request.getParameter("page"));
		}catch(NumberFormatException e){}

		int offset = maxEntriesPerPage * (page - 1);

		request.setAttribute("testList", new TestList(offset,
maxEntriesPerPage));

		return mapping.findForward("example");
	}

}
```

Configure the struts-config.xml

Open the *struts-config.xml* and configure the action mapping.

The folowing source code shows the content of the struts-config.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.3//EN" "http://struts.apache.org/dtds/struts-config_1_3.dtd">
<struts-config>
      <action-mappings >
      <action path="/example"
type="de.laliluna.tutorial.paging.action.ExampleAction" >
         <forward name="example" path="/example.jsp" />
      </action>
      </action-mappings>
      <message-resources
parameter="de.laliluna.tutorial.paging.ApplicationResources" />
</struts-config>
```

The output (JSP File)

Create a new JSP file *example.jsp* in the folder */WebRoot* of your project.

Open the file and add the following content.

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"%>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html"%>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<%@ taglib uri="http://displaytag.sf.net" prefix="display" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html>
  <head>
    <html:base />
    <title>example.jsp</title>
  </head>

  <body>

      ... example code ...

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

Inside the <body> element insert the following examples.

Example 1

In the first example we use a normal *logic:iterate* tag to display the list of dummy data. The attribute property of the *logic:iterate* tag calls the getter Method *getListByOffsetAndLength()* and returns a list of items for the current page. For example if the page is 1 it returns the items 0 to 2. Below the list we display the pages which are existing. The attribute property of the second logic:iterate element calls the getter method *getPages()* in the *TestList* class and returns a list of pages. Inside the *logic:iterate* element a *html:link* will be generated. The attribute *paramId* sets a name for a parameter which is added to the link and the attribute *paramName* specifies the value for this parameter.

```
<h3>Simple paging with struts tags. </h3>

<tabl e border="1">
<tr>
```

```
      <td><b>ID</b></td>
      <td><b>Name</b></td>
      <td><b>City</b></td>
      <td><b>Phone</b></td>
</tr>
<%-- Iteration over the list --%>
<logic:iterate name="testList" property="listByOffsetAndLength" id="var">
<tr>
      <td><bean:write name="var" property="id" /></td>
      <td><bean:write name="var" property="name" /></td>
      <td><bean:write name="var" property="city" /></td>
      <td><bean:write name="var" property="phone" /></td>
</tr>
</logic:iterate>
<tr>
      <td colspan="4">
      <logic:iterate name="testList" property="pages" id="varpage">
            <html:link action="/example" paramName="varpage" paramId="page">
                  <bean:write name="varpage" />
            </html:link>
      </logic:iterate>
      </td>
</tr>
</table>
```

The first example shows you a way to page a list with standard tag libraries in struts. In my opinion, it is very difficulty and costs a lot of time.

Example 2

Now I will show you, how you can use the display tag library to do the same but more elegant.

Step 1

First off all, download the display tag library (http://www.displaytag.org)

Download the commons-lang-2.0.jar (http://jakarta.apache.org/commons/lang/)

Extract the archives.

Copy the *display tag-1.0.jar* of the display tag archive to your lib folder */WEB-INF/lib/*

Copy the *displaytag-11.tld* of the display tag archive to your *WEB*-INF folder

Copy the *commons-lang-2.0.jar* of the commons-lang archive to your lib folder */WEB-INF/lib/*

Be sure you have delete the old version of commons-lang.

That's all.

Note: If you have some problems, read the original install guide of the display tag library: http://displaytag.sourceforge.net/install.html

Step 2

On the top of the JSP file *example.jsp* add the reference to the tag library

```
<%@ taglib uri="http://displaytag.sf.net" prefix="display" %>
```

Step 3 Using the tag library

In example 2 we use the *display:table* element to output the list. The attribute *name* of the *display:table* element specify the bean which holds the list. The attribute *property* specify list of all items to display.

To use the paging functionality you have to set the attribute *pagesize*, in our case 3 item per page

should be displayed. Thats all, very nice and easy !!

The attribute *requestURI* specifies the URI that is used for the links to change a page. We have to specify this, because by default it refers to the JSP file (*example.jsp*) and we fill and set the list in the action class.

```
<h3>Simple use of display tag library for paging with struts. </h3>
      <display:table name="testList.list"
                           pagesize="3"
                           requestURI="/example.do" />
```

Example 3

The last example shows a advanced usage of the *display:table* element for paging lists. If you want to specify customer column names or display not all properties you have to define the columns inside the body of the *display:table* element. Furthermore you can specify if a column can be sorted or some other functionality. The attribute *property* of the *display:column* element specify the property you want to display. The *title* attribute let you set a custom title for each column and the *sortable* attribute specify if the column is sortable.

```
<h3>Adavanced use of display tag library for paging with struts. </h3>

      <display:table name="testList.list"
                           pagesize="3"
                           requestURI="/example.do" >
            <display:column property="id" title="ID" sortable="true" />
            <display:column property="name" title="Name" sortable="true" />
            <display:column property="city" title="City" sortable="true" />
            <display:column property="phone" title="Phone" sortable="true" />
      </display:table>
```

Detailed information about the elements you will find on the display tag homepage.
http://displaytag.sourceforge.net/tagreference-displaytag-12.html

That's all you can test the project now. Call the project with the link below.

http://localhost:8080/PagingExample/example.do