# Pagination with Spring Framework 4.x

## Introduction

In a web application, Pagination, or Page Navigation, is a technique often used to display a large set of data. For a large hundred or thousand or even millions of records, it would cause large performance problem to return all of that data in a single page. Pagination is used to present data to user one page at a time, typically 25 to 100 records per page. Moreover, keyword search, advance search option and column sorting are features often needed together with Pagination.

This article will focus on an implementation of Pagination with Spring Framework that will provide full features of pagination, search option and column sorting.

## Technologies

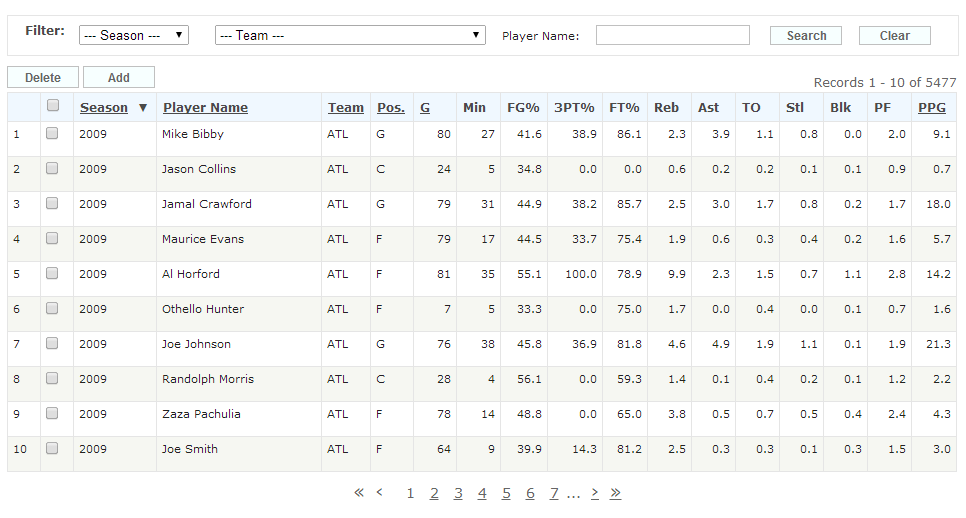
The following technologies are needed:

* Pagination Framework for Spring
* Spring Framework 4.0.1.RELEASE
* Hibernate as JPA provider
* JSP 2.2 and JSTL 1.2 (JEE 6)
* JDK 1.7

## Approach

This article will use a top-down approach, starting with user interface (View), working the way down to Controller, then to Service and Repository (Model / DAO) layer.

Suppose our Project Manager has gathered user requirements for a Basketball (NBA) Statistic System and an application prototype produced:



Top of the page will have a few search options for users to search through statistics. On the middle left there are 2 action buttons for users to delete or add statistics. On the middle right side total number of basketball players and the corresponding page number will be shown. The basketball players are displayed to user one page at a time, 10 players per page. Columns can be sorted by Season, Player Name, etc. And finally, pagination component is rendered for user to change page. The left most part is for go to the first page; follow by link to go to previous page; follow by numeric items to go to a specific page; follow by next page and last page link.

In order to implement pagination, we will need to create the following files:

|  |  |  |
| --- | --- | --- |
| **Object** | **Extends** | **Description** |
| BoSeasonStat | BoPaginationResultRow | Business Object for storing search options |
| BoSeasonStatSearchParam | BoPaginationParam | Business Object for storing one row of data |
| SeasonStatController | PaginationControllerAbstract | Controller to interpret user input and set data into model |
| SeasonStatService | PaginationServiceAbstract | Business Logic is implemented and raw data from JPA is mapped to Business Object |
| SeasonStatDao | PaginationDaoDbEntityManagerAbstract | Database query |
| seasonstat.jsp | http://pagination/pagination-spring3.tld | User Interface presentation |
| PgNbaPlayer  PgNbaSeason  PgNbaTeam | Nil | Database mapping object |

## User Interface (JSP)

To create a JSP page for Basketball Statistics System, we will use Pagination Framework for Spring. Pagination Framework contains a predefined JSP Tag Library for generating our pagination page. We will also need spring tag library and spring form tag library, together with standard JSTL core library.

|  |
| --- |
| <%@taglib prefix="spring" uri="http://www.springframework.org/tags"%>  <%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>  <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  <%@taglib prefix="pg" uri="http://pagination/pagination-spring3.tld" %> |

Next, we will need html form tag for search option submission. We will use spring form tag to create our form. With this form tag, we will expose a business object pparam (short for page parameters) using attribute modelAttribute. The action attribute is set to the page url, which is defined in another business object paginationResult. Business objects (BO) pparam and paginationResult formed the 2 (and only 2) POJO objects that we will use to present our pagination data.

|  |
| --- |
| <form:form id="pgform" method="post" modelAttribute="pparam" action="${pageLink}">  .  .  .  </form:form> |

Next, use Pagination Framework tag library to create our pagination search and data content. Pagination tag library has tag-file name “pagination” (<pg:pagination>) to help generate the pagination content. pg:pagination has 6 attributes: pparam, paginationResult, searchContent, controlButton, columnsContent and contentAfterLastRow. For pparam and paginationResult, I will explain them in more details in Business Object section below. Now let’s first look at how we use <pg:pagination> tag:

|  |
| --- |
| <pg:pagination pparam="${pparam}" paginationResult="${paginationResult}">  <jsp:attribute name="searchContent">  <table class="searchtable">  <tr>  <td class="caption" style="width:50px;">Filter:</td>  <td style="width:120px;">  <form:select path="season" cssStyle="width:110px;">  <form:option value="" label="--- Season ---"/>  <form:option value="2009" label="2009"/>  <form:option value="2008" label="2008"/>  <form:option value="2007" label="2007"/>  <form:option value="2006" label="2006"/>  <form:option value="2005" label="2005"/>  <form:option value="2004" label="2004"/>  <form:option value="2003" label="2003"/>  <form:option value="2002" label="2002"/>  <form:option value="2001" label="2001"/>  <form:option value="2000" label="2000"/>  </form:select>  </td>  <td style="width:200px;">  <form:select path="teamName">  <form:option value="" label="--- Team ---"/>  <form:options items="${teamList}" />  </form:select>  </td>  <td style="width:80px;">Player Name:</td>  <td style="width:160px;"><form:input path="playerName" cssStyle="width:150px;"/></td>  <td style="width:75px;"><span class="button"><form:button id="searchButton" name="buttonAction" value="searchButton" class="button">Search</form:button></span></td>  <td style="width:75px;"><span class="button"><form:button id="clearButton" name="buttonAction" value="clearButton" class="button">Clear</form:button></span></td>  </tr>  </table>  </jsp:attribute>  <jsp:attribute name="controlButton">  <div style="padding-top:10px;">  <span class="button"><form:button id="deleteButton" name="buttonAction" value="deleteButton" class="button">Delete</form:button></span>  <span class="button"><form:button id="addButton" name="buttonAction" value="addButton" class="button">Add</form:button></span>  </div>  </jsp:attribute>  <jsp:attribute name="columnsContent">  <td class="cell"><span><c:out value="${bo.season}"/></span></td>  <td class="cell"><span style="white-space:nowrap;"><c:out value="${bo.playerName}"/></span></td>  <td class="cell"><span style="white-space:nowrap;" title="<c:out value="${bo.teamName}"/>"><c:out value="${bo.teamAlias}"/></span></td>  <td class="cell"><span><c:out value="${bo.position}"/></span></td>  <td class="cell" style="text-align:right;"><span><c:out value="${bo.gamePlay}"/></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.minutePerGame}" pattern="###"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.fgPercent}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.tpPercent}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.ftPercent}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.reboundPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.assistPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.turnoverPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.stealPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.blockPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.pfPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.pointPerGame}" pattern="##0.0"></fmt:formatNumber></span></td> </jsp:attribute>  </pg:pagination> |

searchContent, controlButton, columnsContent and contentAfterLastRow are all jsp fragments.

searchContent contains the search options; form objects (input, radiobutton, button, etc.) use spring form tag library with attribute “path” set to variable in pparam (remember we set modelAttribute=”pparam” in our form). For example, <form:input path="season"> will reference pparam.season variable.

controlButton contains 2 buttons for delete and add action.

columnsContent defines how we present the search result, that is each row of data (each player), to our users. Here an object instance named “bo” is binded to this jsp fragment and can be used to load data from our BoSeasonStat object (see below).

contentAfterLastRow should defines an area just below the last row of data in current page. We didn’t need it in this example.

## Business Object (BO)

### pparam

Page Parameter (pparam) is a business-object representing the search options. We declare this object in com.github.paginationspring.example.spring4.bo.BoSeasonStatSearchParam. This object extends com.github.paginationspring.bo.BoPaginationParam (POJO) and added 5 fields from our search options:

|  |
| --- |
| public class BoSeasonStatSearchParam extends BoPaginationParam {  private String season;  private String playerName;  private String teamName; |

pparam will keep value of search option in this pojo and use it in our pagination framework.

### paginationResult

Pagination Result is a business-object representing results of our search. It contains one page of data (as seen by user on his current page), and it also contains some settings like record-per-page, default sorting order, etc. Each row of data (in our case, each player) is represented by declaring object com.github.paginationspring.example.spring4.bo.BoSeasonStat. This object extends com.github.paginationspring.bo.BoPaginationResultRow (POJO) and added fields for display our search results:

|  |
| --- |
| public class BoSeasonStat extends BoPaginationResultRow<Integer> {  private int nbaSeasonId;  private String playerName;  private String height;  private String weight;  private String teamAlias;  private String teamName;  private String position;  private String season;  private int gamePlay;  private double minutePerGame;  private double pointPerGame;  private double reboundPerGame;  private double assistPerGame;  private double stealPerGame;  private double blockPerGame;  private double turnoverPerGame;  private double pfPerGame;  private double fgPercent;  private double ftPercent;  private double tpPercent;  @Override  public Integer getPk() {  return messageId;  } |

Notice the type inference of <Integer>, this Integer represents the object type of the unique key of this row of data (i.e. nbaSeasonId).

## Controller

Controller will define settings of our pagination page, map requests URLs into a handler method and pass business logic to service layer. First we define our controller with @Controller annotation and extend a controller base PaginationControllerAbstract<BoSeasonStatSearchParam> as shown here:

|  |
| --- |
| @Controller  public class SeasonStatController extends PaginationControllerAbstract<BoSeasonStatSearchParam> { |

In our constructor, we will define some settings for our pagination results.

|  |
| --- |
| public SeasonStatController() {  setOptionDisplayCheckbox(true);  setOptionDisplaySerialNo(true);  setOptionWidth(950);  setDefaultRecordPerPage(10);  setDefaultSortName("Season");  setDefaultSortAscDesc("d");  setPageLink("/org/pagination/example/spring4/bball-season-stat.do"); // url  } |

|  |  |
| --- | --- |
| setOptionDisplayCheckbox | whether or not the left side checkbox should be displayed |
| setOptionDisplaySerialNo | whether or not the left side serial no. should be displayed |
| setOptionWidth | the width of our search results, in pixels |
| setDefaultRecordPerPage | default record per page |
| setDefaultSortName | default sorting column |
| setDefaultSortAscDesc | default sorting order, value “a” is ascending and “d” is decending. |
| setPageLink | url of your pagination page |
| setRewriteUrl | to give you nicer, SEO (Search Engine Optimization) friendly, urls. For example, by setting setRewriteUrl to true, URL mailclient/inbox.do?sortAscDesc=a&sortName=From becomes mailclient/sort/From/a/inbox.do |

Next, we will need to map request URLs into a handler method. We do this by using the @RequestMapping, @ModelAttribute and @RequestParam annotations:

|  |
| --- |
| @RequestMapping(value="/bball-season-stat.do", method = {RequestMethod.GET, RequestMethod.POST})  public String defineJsp(@ModelAttribute(PPARAM) BoSeasonStatSearchParam pparam, @RequestParam(value=BUTTON\_ACTION, required=false) String buttonAction, Model model) throws Exception {  Map<String, Object> map = assignModel(pparam, buttonAction);  model.addAllAttributes(map);  model.addAttribute("teamList", seasonStatService.retrieveTeamAliass());  if ( "deleteButton".equalsIgnoreCase(buttonAction) && !ArrayUtils.isEmpty(pparam.getSelectedIds()) ) {  // deleteButton button is pressed  for ( String pk : pparam.getSelectedIds() ) {  log.info("selected id="+pk);  }  }  return "/org/pagination/example/spring4/view/seasonstat";  } |

In @RequestMapping, you will notice URL1, URL2, URL 3 and URL 4 variables. There constant variables predefined some url patterns to be used when we set option setRewriteUrl to true. If setRewriteUrl is set to false, simply use @RequestMapping(value=”/inbox.do”…).

In our method declaration, the return type String is logical view name and in our case, the jsp file name. Method argument pparam is annotated with @ModelAttribute to bind request parameters to bean properties from our form object (remember we have <form:form modelAttribute=”pparam”>). Method argument buttonAction is annotated with @RequestParam to access button request parameter searchButton, clearButton, deleteButton and addButton. Finally, method argument model is used to expose paginationResult object to web view.

Inside this method, we will call a parent method assignModel from our Pagination Framework, set the returned objects into our model arguments and result the jsp url. The assignModel method will call a service bean, which we will be injecting into our controller. The service bean implements our business logic in our pagination, which we will explain in the next section.

|  |
| --- |
| Map<String, Object> map = assignModel(pparam, buttonAction);  model.addAllAttributes(map);  .  .  .  return "/org/pagination/example/spring4/view/seasonstat"; |

At last, inject our service bean with @Autowired annotation and call a parent method setPaginationService:

|  |
| --- |
| @Autowired  public void setPaginationService(SeasonStatService seasonStatService) {  super.setPaginationService(seasonStatService);  this.seasonStatService=seasonStatService;  } |

## Service

Service bean serve 2 purposes: 1) map database raw data into our business object and 2) define column properties for result data.

First, we define our service bean with @Service annotation and extend a service base PaginationServiceAbstract as shown here:

|  |
| --- |
| @Service  public class SeasonStatService extends PaginationServiceAbstract<BoSeasonStatSearchParam, BoSeasonStat, PgNbaSeason> { |

Type inference of <BoSeasonStatSearchParam, BoSeasonStat, PgNbaSeason> represent our page parameter, business object and data entity object respectively. Data Entiry PgNbaSeason is generated with Hibernate / JBoss Tools from database structure.

To map database raw data into our business object, override assignDataToBo method:

|  |
| --- |
| @Override  protected BoSeasonStat assignDataToBo(PgNbaSeason entity) throws Exception {  BoSeasonStat bo = new BoSeasonStat();  bo.setNbaSeasonId(entity.getNbaSeasonId());  bo.setPlayerName (entity.getPgNbaPlayer().getFirstName()+" "+entity.getPgNbaPlayer().getLastName());  bo.setHeight (entity.getPgNbaPlayer().getHeightFeet()+"-"+entity.getPgNbaPlayer().getHeightInches());  bo.setWeight (String.valueOf(entity.getPgNbaPlayer().getWeight()));  bo.setTeamName (entity.getPgNbaTeam().getLocation()+" "+entity.getPgNbaTeam().getTeamName());  bo.setTeamAlias (entity.getPgNbaTeam().getTeamAlias());  bo.setPosition (entity.getPgNbaPlayer().getPosition());  bo.setSeason (entity.getSeason ());  bo.setGamePlay (entity.getGamePlay ());  if ( entity.getGamePlay() != 0 ) {  bo.setMinutePerGame ((double) entity.getMinute ()/(double) entity.getGamePlay());  bo.setPointPerGame ((double) entity.getPoint ()/(double) entity.getGamePlay());  bo.setReboundPerGame ((double) entity.getRebound ()/(double) entity.getGamePlay());  bo.setAssistPerGame ((double) entity.getAssist ()/(double) entity.getGamePlay());  bo.setStealPerGame ((double) entity.getSteal ()/(double) entity.getGamePlay());  bo.setBlockPerGame ((double) entity.getBlock ()/(double) entity.getGamePlay());  bo.setTurnoverPerGame ((double) entity.getTurnover ()/(double) entity.getGamePlay());  bo.setPfPerGame ((double) entity.getPf ()/(double) entity.getGamePlay());  }  if ( entity.getFgAttempt() != 0 ) {  bo.setFgPercent (((double) entity.getFgMade() / (double) entity.getFgAttempt())\*(double) 100);  }  if ( entity.getTpAttempt() != 0 ) {  bo.setTpPercent(((double) entity.getTpMade() / (double) entity.getTpAttempt())\*(double) 100);  }  if ( entity.getFtAttempt() != 0 ) {  bo.setFtPercent(((double) entity.getFtMade() / (double) entity.getFtAttempt())\*(double) 100);  }  return bo;  } |

To define column properties, override assignColumnsDefinition method:

|  |
| --- |
| @Override  public void assignColumnsDefinition(List<BoPaginationColumn> columns)  throws Exception {  log.debug("setting columns def.");  BoPaginationColumn col = null;    col = new BoPaginationColumn();  col.setColumnName("Season");  col.setOrderColumns("a.season, c.teamAlias, b.lastName, b.firstName");  col.setOrderDirections("desc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Player Name");  col.setOrderColumns("b.lastName, b.firstName, a.season");  col.setOrderDirections("asc, asc, asc");  col.setWidth(150);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Team");  col.setOrderColumns("c.teamAlias, a.season, b.lastName, b.firstName");  col.setOrderDirections("asc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Pos.");  col.setOrderColumns("b.position, a.season, c.teamAlias, b.lastName, b.firstName");  col.setOrderDirections("asc, asc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("G");  col.setOrderColumns("a.gamePlay, a.season, c.teamAlias, b.lastName, b.firstName");  col.setOrderDirections("asc, asc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Min");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("FG%");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("3PT%");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("FT%");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Reb");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Ast");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("TO");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Stl");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("Blk");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("PF");  col.setWidth(30);  columns.add(col);  col = new BoPaginationColumn();  col.setColumnName("PPG");  col.setOrderColumns("(a.point/a.gamePlay), a.season, c.teamAlias, b.lastName, b.firstName");  col.setOrderDirections("desc, asc, asc, asc, asc");  col.setWidth(30);  columns.add(col);  } |

We create a new BoPaginationColumn bean for each column. BoPaginationColumn has 4 variables: columnName, orderColumns, orderDirections and width. The columnName is the name of column to be displayed to user. The orderColumns is a query string to be added to our JPA query “order by” clause. The orderDirections has value “asc” or “desc” for ascending or descending shorting of the column. For sorting across multiple fields, use comma to separate between required fields (as shown in “From” column above). The width variable defines the column width in pixels.

At last, inject our DAO bean with @Autowired annotation and call a parent method setPaginationDao:

|  |
| --- |
| @Autowired  public void setPaginationDao(SeasonStatDao seasonStatDao) {  super.setPaginationDao(seasonStatDao);  this.seasonStatDao = seasonStatDao;  } |

## Data Access Layer (DAO)

DAO bean load the raw data from database using JPA entity manager. First, we define our DAO with @Repository annotation and extends PaginationDaoDbEntityManagerAbstract<PgNbaSeason, BoSeasonStatSearchParam> from Pagination Framework:

|  |
| --- |
| @Repository  public class SeasonStatDao extends PaginationDaoDbEntityManagerAbstract<PgNbaSeason, BoSeasonStatSearchParam> { |

PgNbaSeason is the database entity bean and BoSeasonStatSearchParam is the incoming page parameters.

First, we will setup query to load data from database:

|  |
| --- |
| private static final String EJBQL = "select distinct a, b, c " +  "from PgNbaSeason a " +  "join a.pgNbaPlayer b " +  "join a.pgNbaTeam c " +  "where a.gamePlay > 0 ";    private static final String[] RESTRICTIONS = {  "a.season = :season",  "lower(concat(concat(b.firstName,' '),b.lastName)) like :playerNameCustomized",  "c.teamAlias=:teamName"  };  public SeasonStatDao() {  this.ejbql = EJBQL;  this.restrictions = Arrays.asList(RESTRICTIONS);  } |

Let’s look at our query. We select distinct PgNbaSeason (alias a) in order to pass this object to assignDataToBo method in Service bean. In order for the query to be able to sort by “Player Name” field (see assignColumnsDefinition in Service bean), we need to select pgNbaPlayer (alias b) in our query. Moreover, we will define pgNbaTeam (alias c) and in where clause we restrict our query for gamePlay larger than 0.

Restrictions define our search options. There are 2 kinds of restrictions binding: 1) binding directly to variable in pparam (BoSeasonStatSearchParam) and 2) binding via customization.

The first restriction a.season=:season means that PgNbaSeason.season are binding directly to variable “season” in pparam. Other restrictions are defined in binding in customizeRestrictions method:

|  |
| --- |
| @Override  public void customizeRestrictions(Map<String, Object> queryParameters,  BoSeasonStatSearchParam pparam) throws Exception {  if ( StringUtils.isEmpty(pparam.getPlayerName()) ) {  queryParameters.put("playerNameCustomized", null);  } else {  queryParameters.put("playerNameCustomized", "%"+StringUtils.trim(pparam.getPlayerName().toLowerCase())+"%");  }  } |

In this customizeRestrictions method, playerNameCustomized is customized and then put into a map queryParameters. Our Pagination Framework will use this map to bind variables into our JPA query.

Finally, we need to inject entityManager via @PersistenceContext annotation:

|  |
| --- |
| @PersistenceContext  public void setEntityManager(EntityManager entityManager) {  super.setEntityManager(entityManager);  } |

In this example, we used JPA Entity Manager (PaginationDaoDbEntityManagerAbstract) for persistence management. There are 2 more options available in our Pagination Framework, one is PaginationDaoDbHibernateTemplateAbstract which uses spring hibernate template and the other is PaginationDaoLuceneAbstract which uses lucene index as datasource.

## Spring Framework Setup

To setup spring framework, we setup spring MVC in web.xml as we normally would:

|  |
| --- |
| <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/applicationContext.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>2</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/org/pagination/example/spring4/\*</url-pattern> </servlet-mapping> |

In applicationContext.xml file, define the following for our annotated components:

|  |
| --- |
| <context:annotation-config />  <context:component-scan base-package="com.github.paginationspring.example.spring4" />  <mvc:annotation-driven /> |

You should also define EntityManagerFactory for our JPA.

In dispatcher-servlet.xml, define the following for view resolver:

|  |
| --- |
| <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="order" value="2" />  <property name="prefix" value="/WEB-INF/jsp/"/>  <property name="suffix" value=".jsp"/>  </bean> |

## Running our example

Download this zip file and extract the content into your computer. The content included full source code used in this example. Open command prompt and type run.bat (for window OS) or ./run.sh (for linux OS). Choose your application server and wait around 30 seconds for the server to startup. After server started, go to url <http://localhost:8080/pagination-example-spring4>

## Appendix A – AJAX pagination

Pagination Framework support AJAX via using jquery. Adding AJAX to pagination enable our application to update only part of our web page, without reloading the whole page. For example, navigating through page 1 to page 2 will only update the “results” part of the page. When we implement AJAX pagination, objects in Service, Dao, Entity and Bo are exactly the same. However, your JSP and Controller will be different from traditional pagination.

### Controller for AJAX

You should declare your controller like traditional pagination:

|  |
| --- |
| @Controller  public class SeasonStatAjaxController extends PaginationControllerAbstract<BoSeasonStatSearchParam> { |

In our constructor, we will define some settings for our pagination results.

|  |
| --- |
| public SeasonStatAjaxController() {  setOptionDisplayCheckbox(true);  setOptionDisplaySerialNo(true);  setOptionWidth(950);  setDefaultRecordPerPage(10);  setDefaultSortName("Season");  setDefaultSortAscDesc("d");  setPageLink("/org/pagination/example/spring4/bball-season-stat2\_ajax.do");  setAjax(true);  } |

Notice that we are setting ajax option to true and our page link is setting to an ajax page fragment (instead of a whole jsp page).

Instead one RequestMapping methods, for AJAX implementation, we need to define two RequestMapping, one for the whole page (defineJsp) and the other for page fragment (defineAjaxJsp). In defineJsp method, call assignModel with parameter “loadResult” set to false to disable the actual loading of data. In defineAjaxJsp, call assignModel like we normally would.

|  |
| --- |
| @RequestMapping(value="/bball-season-stat2.do", method = {RequestMethod.GET, RequestMethod.POST})  public String defineJsp(@ModelAttribute(PPARAM) BoSeasonStatSearchParam pparam, Model model) throws Exception {  log.debug("pparam.resultIndex="+pparam.getResultIndex());  Map<String, Object> map = assignModel(pparam, null, false);  model.addAllAttributes(map);  model.addAttribute("teamList", seasonStatService.retrieveTeamAliass());  return "/org/pagination/example/spring4/view/seasonstat2";  }  @RequestMapping(value="/bball-season-stat2\_ajax.do", method = {RequestMethod.GET, RequestMethod.POST})  public String defineAjaxJsp(@ModelAttribute(PPARAM) BoSeasonStatSearchParam pparam, @RequestParam(value=BUTTON\_ACTION, required=false) String buttonAction, Model model) throws Exception {  log.debug("pparam.resultIndex="+pparam.getResultIndex());  Map<String, Object> map = assignModel(pparam, buttonAction);  model.addAllAttributes(map);  if ( "deleteButton".equalsIgnoreCase(buttonAction) && !ArrayUtils.isEmpty(pparam.getSelectedIds()) ) {  // Delete button is pressed  log.debug("Delete button is pressed");  if ( pparam.getSelectedIds() != null ) {  for ( String pk : pparam.getSelectedIds() ) {  log.debug("delete selected id="+pk);  }  }  }  return "/org/pagination/example/spring4/view/seasonstat2\_ajax"; // jsp  } |

### User Interface (JSP) for AJAX

For AJAX pagination, we will need 2 JSPs, one for the whole page (seasonstat2.jsp) and the other for page fragment (seasonstat2\_ajax.jsp).

For seasonstat2.jsp, it is the same as in traditional pagination except that we don’t need to define jsp attribute “columnsContent”. We also need to add jquery to our page header:

|  |
| --- |
| <script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js" type="text/javascript"></script> |

For seasonstat2\_ajax.jsp, use pagination tag library pg:pagination\_ajax and added the jsp attribute “columnsContent” there.

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  <%@taglib prefix="spring" uri="http://www.springframework.org/tags"%>  <%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>  <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  <%@taglib prefix="pg" uri="http://pagination/pagination-spring3.tld" %>  <%@page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <pg:pagination\_ajax pparam="${pparam}" paginationResult="${paginationResult}">  <jsp:attribute name="columnsContent">  <td class="cell"><span><c:out value="${bo.season}"/></span></td>  <td class="cell"><span style="white-space:nowrap;"><c:out value="${bo.playerName}"/></span></td>  <td class="cell"><span style="white-space:nowrap;" title="<c:out value="${bo.teamName}"/>"><c:out value="${bo.teamAlias}"/></span></td>  <td class="cell"><span><c:out value="${bo.position}"/></span></td>  <td class="cell" style="text-align:right;"><span><c:out value="${bo.gamePlay}"/></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.minutePerGame}" pattern="###"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.fgPercent}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.tpPercent}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.ftPercent}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.reboundPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.assistPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.turnoverPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.stealPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.blockPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.pfPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  <td class="cell" style="text-align:right;"><span><fmt:formatNumber type="number" value="${bo.pointPerGame}" pattern="##0.0"></fmt:formatNumber></span></td>  </jsp:attribute>  </pg:pagination\_ajax> |

### Running AJAX application:

After you start the server, go to url: <http://localhost:8080/pagination-example-spring4/org/pagination/example/spring4/bball-season-stat2.do>

You can try navigating through page 1, 2… etc or search for a result. As you can see, the pagination actions will not reload the whole page.

## Appendix B – Search engine friendly URL

For SEO, we always want to make our URL search engine robot friendly. Make the following changes to our Controller:

|  |
| --- |
| public SeasonStatController() {  .  .  .  setRewriteUrl(true);  } |
| @RequestMapping(value={  URL1+"/bball-season-stat.do"  , URL2+"/bball-season-stat.do"  , URL3+"/bball-season-stat.do"  , URL4+"/bball-season-stat.do"  }, method = {RequestMethod.GET, RequestMethod.POST}) |

That’s it! Run the example and you will see the pagination URL is shortened and more friendly.