3.4 Pagination

Bon pour relecture

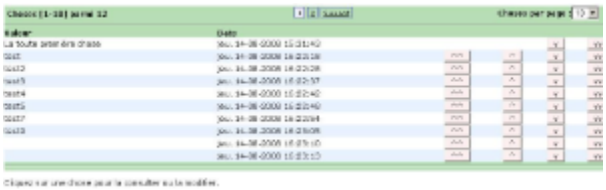
Il n'est pas rare dans les applications web de vouloir afficher des listes de valeurs. Parce que l'espace visible sur l'écran du client est limité, on souhaite souvent afficher les résultats sur plusieurs pages ; c'est ce qu'on appelle la pagination. La **première difficulté** de la pagination est donc de **n'afficher qu'une partie des résultats d'un ensemble plus important**, et de proposer une navigation visuelle entre les pages.La **deuxième difficulté** consiste, pour les accès aux bases de données, à **ne récupérer que les valeurs de la base qui doivent être affichées** ; cela est indispensable lorsque le nombre de résultats est énorme, assez gros en tout cas pour saturer la mémoire des processus. Nous montrons donc dans un premier temps comment on peut paginer des données sans se soucier de leur récupération. La dernière partie montre comment écrire des paginateurs qui ne récupèrent de la base de données que les données à afficher.

Sommaire :

Le tag *paginator*

Contrairement à ce que l'on peut trouver dans *esup-commons* V1 les tags **<e:** développés spécifiquement pour *esup-commons* ont été supprimés au profit de l'utilisation de librairies standard.
Seul le tag paginator a été conservé.

La pagination des données (e:paginator)



Exemple d'utilisation de la balise e:paginator.

```
<h:panelGroup>
  <e:paginator id="thingPaginator"
               paginator="#{thingsController.paginator}"
               itemsName="#{msgs['THINGS']}"
               onchange="javascript:{simulateLinkClick('thingsForm:submitPageSize');}" />
  <h:commandButton id="submitPageSize"
                   action="#{thingsController.paginator.forceReload}" />
</h:panelGroup>
```

Définition des attributs :

- **id** : Permet de définir l'identifiant du paginator.
- **paginator** : Attribut obligatoire, une expression EL qui pointe le paginateur (objet implémentant l'interface Paginator).
- **itemsName** : Indique le nom des items affichés. Correspond à 'Choses' dans l'image ci-dessus.
- **styleClass** : Permet de définir la classe CSS. Par défaut, la classe CSS est positionnée par le *bean* **tagsConfigurator**.
- **visibleBlocks** : Le paginateur est composé de 3 blocs : **itemsNumbers**, **navigation**, **itemsPerPage**. Cet attribut permet de définir quels blocks doivent être visibles. Il faut indiquer le nom des blocs séparés par une virgule sans espace (L'ordre d'écriture des blocs est important). Par défaut, les 3 blocs sont visibles.
- **onchange** : Le block **itemsPerPage** contient une liste déroulante permettant de définir la taille des pages. Cette attribut surcharge l'attribut **onChange** de cette liste déroulante. Dans l'exemple ci-dessus, à chaque changement de la taille de page, on simule un clic du bouton **submitPageSize** qui recharge le paginateur.

La balise **e:paginator** ne gère pas l'affichage des données. Elle s'occupe de la gestion des pages (voir le chapitre utilisation d'un paginateur dans une page JSF).

Configuration dynamique des balises

Le fichier **/properties/tags/tags.xml** doit déclarer un *bean* nommé **tagsConfigurator**, qui doit implémenter l'interface **TagsConfigurator**.

Les valeurs par défaut de ce *bean* suivent les recommandations de http://www.ja-sig.org/wiki/display/UPC/JSR-168+PLT.C+CSS+Style+Definitions+section. On se réfèrera au fichier d'exemple **/properties/tags/tags-example.xml** pour plus de détails.

Écriture d'un paginateur simple

Nous supposons qu'une page de l'application doit afficher les choses (**Thing**) du service (**Department**) courant de la page.

Notre paginateur (**ThingPaginator**) étend la classe abstraite **ListPaginator<Thing>**, qui implémente elle-même l'interface **Paginator<Thing>** :

```
public class ThingPaginator extends ListPaginator<Thing> {
  ...
}
```

Son constructeur positionne l'attribut **domainService** (le service métier) qui sera utilisé ultérieurement pour récupérer les choses (**Thing**) de la base de données :

```
public ThingPaginator(final DomainService domainService) {
  super(null, 0);
  this.domainService = domainService;
}
```

Un autre attribut **department** est utilisé pour mémoriser le service pour lequel le paginateur doit récupérer les choses. Cet attribut est positionné par le *setter* correspondant :

```
public ThingPaginator setDepartment(final Department department) {
  this.department = department;
  return this;
}
```

Le paginateur implémente enfin la méthode de récupération des données proprement dite :

```
protected List<Thing> getData() {
  return this.domainService.getThings(department);
}
```

Le paginateur ainsi écrit peut être utilisé par un contrôleur.

Les paginateurs ne s'appuient pas forcément sur une base de données ; on peut ainsi imaginer un paginateur qui traitera les fichiers trouvés dans un répertoire donné. Dans ces cas, le service métier n'est pas toujours nécessaire.

Exercice : Écrire un paginateur simple Afficher l'énoncé

Utilisation d'un paginateur

Dans le code Java

Un paginateur sera typiquement un attribut d'un contrôleur. Nous prenons ici pour exemple le contrôleur des « choses », qui les affiche de manière paginée :

```
private ThingPaginator paginator;
```

La méthode **reset()** est appelée automatiquement par la méthode **afterPropertiesSet()** de la classe **AbstractDomainAwareController**. Dans cette méthode, on initialise le paginateur et on charge ses premières données (avec un service vide, la liste des choses récupérées sera vide) :

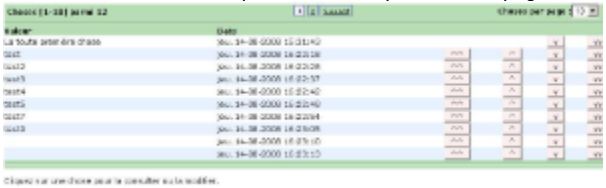
```
paginator = new ThingPaginator(getDomainService());
paginator.loadData();
```

à chaque fois que le service de la page change, il suffit d'en informer le paginateur et de recharger ses données (la méthode **reloadData()** provient de l'interface **Paginator**) :

```
paginator.setDepartment(department).reloadData();
```

Dans une page JSF

Nous montrons dans cette partie comment présenter un paginateur, pour obtenir un affichage de ce genre :



On commence par englober le tout d'un formulaire, nécessaire pour faire fonctionner les boutons de navigation :

```
<h:form id="administratorsForm">
```

On commence ensuite une table pour parcourir les entrées du paginateur (on suppose ici que le paginateur est un attribut **paginator** d'un contrôleur **contr oller**) :

```
<h:dataTable rendered="#{not empty controller.paginator.visibleItems}"
             id="data"
             rowIndexVar="variable"
             value="#{controller.paginator.visibleItems}"
             var="thing"
             border="0" style="width:100%"
             cellspacing="0" cellpadding="0">
```

L'attribut **rendered** fait que cette table ne sera affichée que lorsque le paginateur a des éléments visibles. A chaque tour de boucle, la variable **thing** parcourt la liste **controller.getPaginator().getVisibleItems()**. On ajoute ensuite à la table une entête, sous forme d'un *facet* **<f:facet name="header">**. C'est ce *facet* qui contiendra :

- Les numéros des éléments affichés,
- Les liens vers les pages proches de la page courante
- Une boite de dialogue déroulante permettant de changer le nombre d'éléments affichés par page.

Ces 3 points sont gérés par la balise **e:paginator** (cf. utilisation du tagLib e:paginator)

On parcourt ensuite des colonnes dans lesquelles on affiche ce que l'on veut, par exemple :

```
<t:column>
   <h:text value="#{thing.value}" />
</t:column>
```

Exercice : Afficher un paginateur sur une page JSF Afficher l'énoncé

⚠ Il est avantageux de partir d'un exemple existant pour afficher un paginateur, on pourra par exemple se référer au fichier **/webapp/stylesheets /administrators.jsp** du projet *esup-blank*.

Écriture d'un paginateur Hibernate

L'intérêt d'un paginateur *Hibernate* est de ne récupérer de la base de données que les éléments qui doivent être affichés.Cela se fait en étendant la classe **AbstractHibernatePaginator<E>**, qui possède la méthode abstraite suivante :

```
String getQueryString();
```

Cette méthode doit retourner la requête *HQL* (*Hibernate Query Language*) qui correspond à la récupération de l'ensemble des éléments, sans tenir compte de ceux qui seront affichés ou non (la classe s'occupe ensuite de limiter les éléments récupérés).

Lorsque cette requête *HQL* est invariable, il est possible d'étendre la classe **FixedAbstractHibernatePaginator<E>**, qui initialise sa requête *HQL* par son constructeur.

Un exemple de requête *HQL* fixe est **"FROM Thing"**, qui récupère simplement toutes les instances de la classe **Thing** dans la base de données.

Exercice : Écrire un paginateur Hibernate Afficher l'énoncé