EhCache - optimisation des performances de Nuxeo

1. Qu'est-ce qu'EhCache ?

EhCache est une solution **OpenSource** permettant une gestion poussé du cache d'une application. Cette solution est basé sur la norme **JSR-107**. Cette conform au standard permet à cette solution d'être intégré à différentes applications. Cela permet également de grandes interactions entre différents services (JMX, RMI, etc...).

Le principe de base d'EhCache est de permettre la mise en cache d'Objets Java dans la mémoire vive ainsi que sur le disque dur de la machine. Le but étant de limité la monter en RAM des applications. Un Objet peu utiliser se doit donc d'être mis en cache sur le disque dur et non en RAM. Pour gérer la façon dont EhCa va basculer les objets de la mémoire vive au disque dur il y a 3 modes différents :

- **LRU** : Least Recently Used, le concept de ce mode est transférer sur le disque dur les éléments utilisées le moins récemment. La notion temporel est très important dans ce mode. Cependant ce mode demande beaucoup de calcule pour être correctement gérer.
- **LFU** : Less Frequently Used, ce mode conserve un historique des accès, ce qui lui permet de transférer les éléments peu utiliser sur le disque dur.
- **FIFO** : First In First Out, le mode FIFO est une optimisation du mode LRU. Cela fonctionne sous forme de pile, les derniers éléments de la pile sont transfé sur le disque dur. Cette approche est plus simple que le LRU mais moins précise, cas on tien moins compte de l'utilisation réel des éléments.

Il est également possible d'attribuer une durée de vie à un cache afin qu'il soit invalidé à intervalle régulier.

Ces algorithmes complexes permettent à EhCache d'être performant. D'autre optimisation sont effectués pour avoir une bonne gestion des ressources CPU (ge des multi coeurs, etc...). Il est également possible de faire des **clusters** de EhCache via un serveur **Terracotta**. Le rapprochement de **JBoss** avec **Terracotta** n promet des futures implémentations **NoSQL** pour Hibernate basé sur les mécanisme d'EhCache.

JBoss avait sur Seam 2 fait une grande avancés avec leur composants JSF **s:cache**, permettant de mettre en cache le rendu JSF d'une région donné, le tout a une grande élégance.

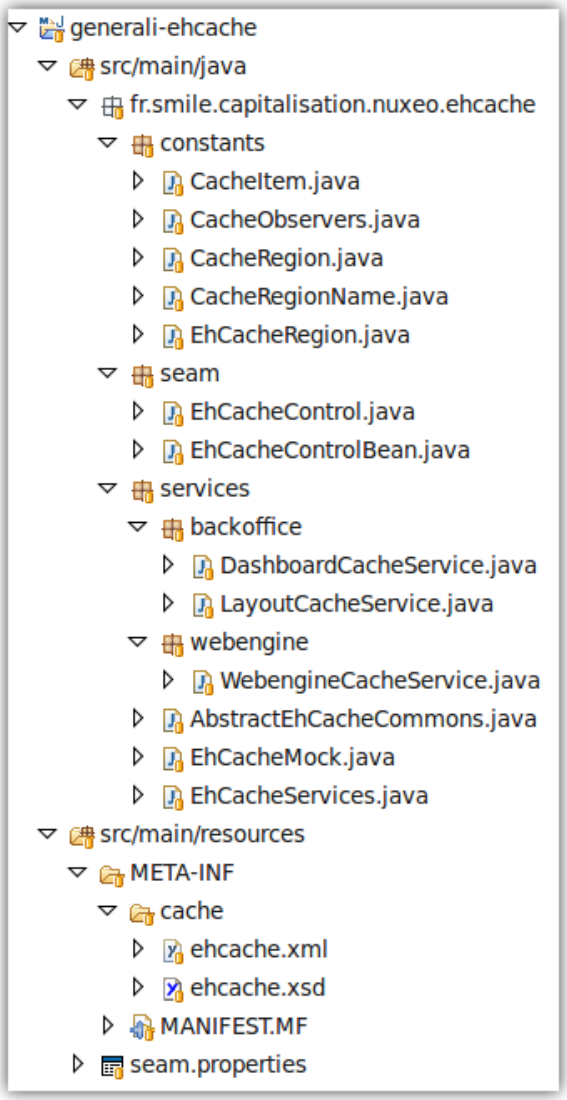
Un point important pour comprendre le fonctionnement d'EhCache est le fait qu'il fonctionne sur le principe de régions. On a en réalité plusieurs caches différen appelés régions. Ces régions peuvent avoir des configurations différentes, permettant ainsi de gérer de façon plus fine le cache. Si on devrait faire une analogie avec un "objet java", EhCache serait un équivalent à une Map<String,Map<String,Object>> .

le JAR d'EhCache ne pèse que 260Ko, ce n'est donc pas une librairie trop gourmande. Il faut cependant savoir que **EhCache nécessiste** une dépendance vers **SLF4J**.

homepage d'EhCache : http://ehcache.org/

2. Le module EhCache pour Nuxeo

Il faut savoir que **Nuxeo n'utilise pas nativement EhCache**. Il existe bien une dépendance vers cette API dans Nuxeo mais elle n'est pas configuré pour fonctio correctement avec Seam. Le problème vient du fait que le fichier **components.xml** de nuxeo **ne contient pas le namespace** spécifique à la **gestion du cache p Seam**. Il faut donc impérativement surcharger ce fichier pour que cela fonctionne. De ce fait il n'est pas actuellement possible d'avoir un bundle isolé contenant l'ensemble des sources et configuration nécessaire à son fonctionnement.

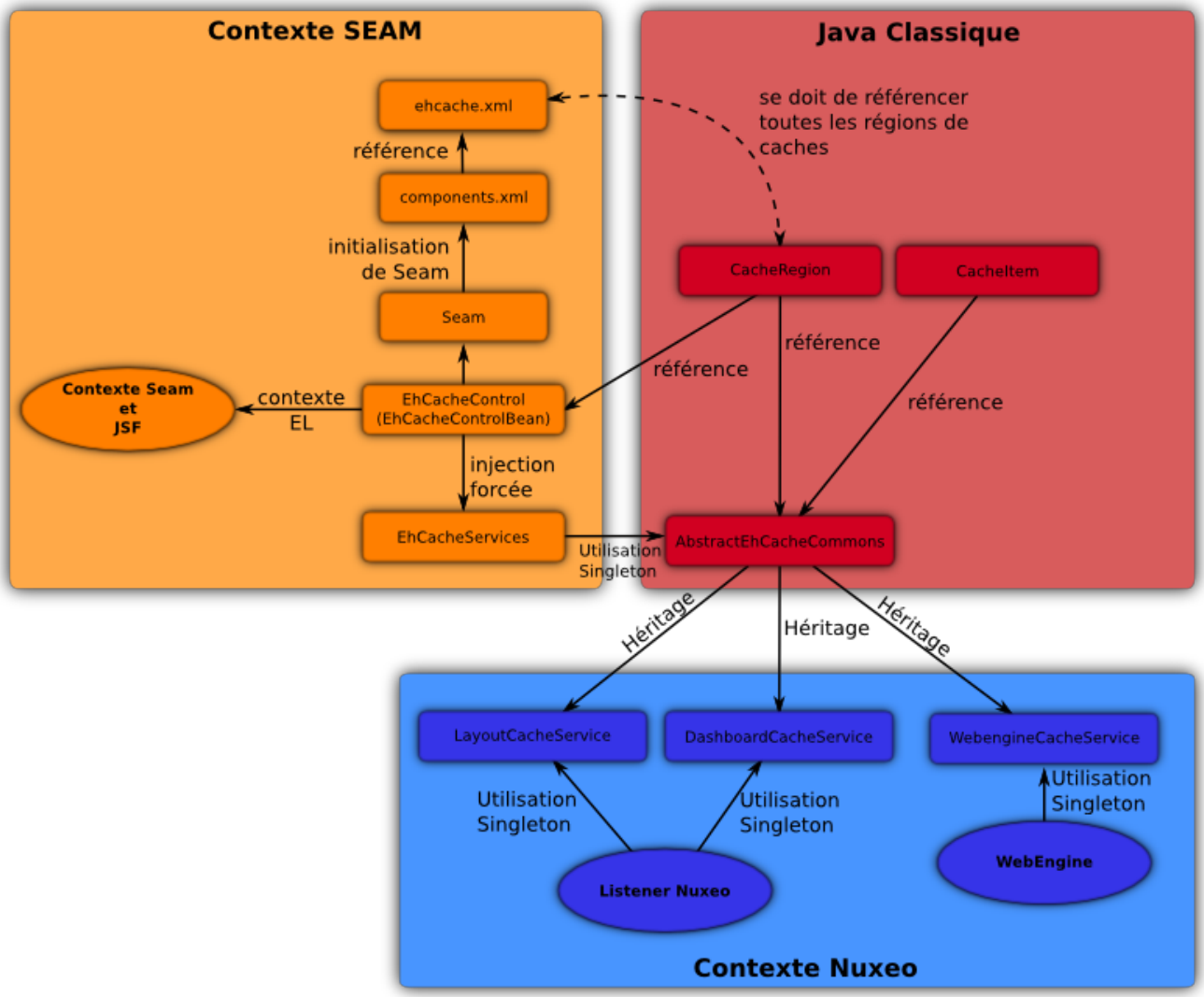


Le module est réellement constitué de 3 grandes parties :

- constants : qui contient l'ensemble des énumérations et constantes nécessaires.
- seam : qui contient le bean Seam contrôlant le cache
- services : qui contient des singletons qui permettent d'utiliser le cache aussi bien dans le backoffice, webengine, ou dans des listeners Nuxeo.

> ***il est à noter que les listeners Nuxeo ne sont pas forcément liées au contexte Seam***

2.1. Fonctionnement :



Pour fonctionner la mécanique de cache utilise différentes couches applicatives :

- Le contexte Seam : C'est la partie principale, c'est elle qui va gérer le cache ainsi que sa configuration. Seam utilie le **CacheProvider** pour la gestion du cache. Ce composant est un wrapper de cache qui va utiliser EhCache avec sa configuration (ehcache.xml). Le composant EhCacheControl permet un controle simplifier du cache, avec des **factory seam** pour récupérer le nom des différentes régions dans les view JSF.
- Java classique : L'une des particularités de Nuxeo est le fait qu'il n'est pas toujours dans le contexte Seam. De ce fait il est nécessaire d'avoir des objets le pattern **Singleton** afin de pouvoir récupérer notre cache dans Nuxeo.
- Le Contexte Nuxeo : On a 3 implémentation spécifiques de la classe abstraite **AbstractEhCacheCommons** qui nous permet dans le contexte Nuxeo de gérer notre cache. Ils sont conçus pour être utilisés:
  - soit dans des listeners Nuxeo (pour la partie backoffice)
  - soit dans les classes Java contrôlant Webengine.

Les différentes implémentations sont directement liées aux régions définit dans le fichier de configuration **ehcache.xml**. Il est préférable pour avoir un code clai d'avoir une implémentation par région de cache. **Toutes les implémentations doivent être des Singletons**.

Le fait que Nuxeo peut être hors contexte Seam nous oblige à avoir un service qui va récupérer de force le composant Seam. Cette pratique est à éviter mais da ce cas précis est nécessaire. La classe EhCacheServices va recrée un contexte Seam à son initialisation afin de récupérer ce composant. Cette classe est également un Singleton afin qu'elle ne s'initialise qu'une seule fois au cours de la vie de l'application, limitant ainsi cette injection forcée.

3. Comment utiliser EhCache :

Le fonctionnement du module peut sembler complexe, mais son utilisation est d'une facilité enfantine.

3.1. Enums VS String :

Le cache utilise des chaînes de caractères pour les différentes clés ou régions, cependant il faut au maximum éviter l'utilisation massive de l'objet String. Leurs est assez conséquent et il devient rapidement très complexe de "refactorer". Dans le meilleur des cas les développeurs utiliseront des constantes, mais ça laiss toujours la possibilité de redéfinir à chaque fois la clé au moment de l'utilisation du cache. Il en va de même pour toutes les méthodes prenant un objet String co clé.

Pour cela il faut utiliser une énumération implémentant une interface. Cela permet de généraliser le code et permettre la création de nouvelles énumération. Le s'en retrouve simplifié et bien plus claire.

1. La première chose à faire est de crée l'interface

```
1 public interface MyKey {
2     String key();
3 }
```

2. On peut en suite crée notre énumération

```
 1 public enum MyKeyItem implements MyKey{
 2     foo,
 3     bar,
 4     foobar;
 5
 6     @Override
 7     public String key() {
 8         return this.name();
 9     }
10 }
```

3. Il faut modifier notre méthode qui utilise l'objet String

```
1 public Object myMethod(final String myKey){
2     return service.getObject(myKey);
3 }
4
5 public Object myMethod(final MyKey myKey){
6     return service.getObject(myKey.key());
7 }
```

4. Ce qui nous permet à présent d'utiliser notre méthode avec n'importe quel énumération implémentant l'interface MyKey

```
1 dao.myMethod(MyKeyItem.foo);
```

Ce principe est utilisé à différents endroit dans le module EhCache (au niveau des régions de cache et des éléments). On retrouve ce concepts également dans module Webengine (récupération des templates).

3.2. Utilisation avec JSF :

C'est là où on voit l'un des intérêt à utiliser Seam avec JSF à l'instar de Spring. JBoss a créé des composants de haut niveau pour faciliter le travail des développeurs. Le composant **s:cache** en est un parfait exemple.

```
 1 <f:subview xmlns:h="http://java.sun.com/jsf/html"
 2   xmlns:f="http://java.sun.com/jsf/core"
 3   xmlns:s="http://jboss.com/products/seam/taglib"
 4   id="theme_commons_logo_block">
 5
 6     <h:form>
 7         <s:cache region="#{regionLayout}" key="layout_logo_cache" >
 8             <h:commandLink action="#{navigationContext.goHome()}" id="theme_commons_logo_link">
 9                 <h:graphicImage id="theme_commons_logo_img" value="/themes/img/logo_top.png"/>
10             </h:commandLink>
11         </s:cache>
12     </h:form>
13
14 </f:subview>
```

Comme on le vois il suffit d'encadrer notre bloc de code JSF dans le composant **s:cache** pour qu'il soit mis en cache. Ce composant prend comme paramètre d options :

- **region :** Comme on l'a vu précédemment c'est les régions définit dans le fichier ehcache.xml, les factory contenu dans le bean EhCacheControlBean permettent de transmettre le nom de la région directement via le **contexte EL**
- **key :** Comme pour une Map, EhCache associe un objet avec une clé, ce paramètre permet de donné le nom de la clé sous le quel on souhaite enregistr notre fragment HTML en cache.

3.3. Vider le cache :

Pour vider le cache on a plusieurs façon de réaliser cela:

- Soit on ajoute une action avec un bouton dans l'interface qui fera appel à un bean Seam
- Soit on utilise un listener Nuxeo qui va se déclancher à la suite d'un évenement.

3.3.1. Bean Seam :

La rien de bien complexe, ça reste du Seam tout ce qu'il y a des plus classique. Cependant il est toute fois préférable d'utiliser les implémentations spécifiques p permettre un refactoring et un lecture du code simplifié.

```
1 @Name(Foobar.name)
2 @Scope(ScopeType.PAGE)
3 public class FoobarBean implements Foobar {
4
5     public void cleanCache(){
6         final DashboardCacheService service = DashboardCacheService.getInstance();
```

```
        service.deleteAll();
    }

    public void cleanCache(final String key){
        final DashboardCacheService service = DashboardCacheService.getInstance();
        service.delete(key);
    }
}
```

Du coups il devient possible d'appel ces methodes via le **context EL**

```
<h:form id="myform">
    <h:commandButton action="#{foobar.cleanCache()}" id="buttonCleanAll" value="Clean ALL" />

    <h:commandButton action="#{foobar.cleanCache('myKey')}" id="buttonCleanKey" value="Clean myKey" />
</h:form>
```

3.3.2. Listener Nuxeo :

Pour faire fonctionner le cache avec un listener Nuxeo il faut tout d'abord c'est un fichier XML de contribution :

```
<?xml version="1.0"?>
<component name="org.foo.bar.listeners">

    <extension target="org.nuxeo.ecm.core.event.EventServiceComponent" point="listener">

        <listener name="refreshPath"
                  async="false"
                  postCommit="false"
                  order="150"
                  class="org.foo.bar.listener.CacheListener">
            <event>documentCreated</event>
        </listener>
    </extension>

</component>
```

Comme pour tout fichier OSGI il faut bien penser à l'ajouter dans le **MANIFEST** :

```

Manifest-Version: 1.0
Bundle-ManifestVersion: 1
Bundle-Name: Generali Intranet Core
Bundle-SymbolicName: fr.generali.intranet.ehcache;singleton:=true
Bundle-Version: 1.0.0
Bundle-Vendor: Nuxeo
Nuxeo-Component: OSGI-INF/listener-contrib.xml
```

Une fois cela crée, on peut implémenter notre classe :

```
public class CacheListener extends AbstractListener {

    /** The event. */
    protected Event event;

    /** The doc ctx. */
    protected DocumentEventContext docCtx;

    protected void process(DocumentModel doc) throws ClientException {
        DashboardCacheService.getInstance().deleteAll();
    }

    @Override
    protected Set<String> getTypes() {
        Set<String> types = new HashSet<String>();
        types.add("BasicContent");
        return types;
    }
}
```

Ce listener se déclenchera à chaque création de nouveau document héritant du type documentaire "BasicContent" et invalidera le cache. Après il est possible d
jouer avec les listener afin de gérer son cache de façon optimale (http://doc.nuxeo.com/display/NXDOC/Events+and+Listeners)

3.4. Utilisation avec Webengine :

Dans Webengine rien de bien sorcier, on procède de la même façon en utilisant le service dédié à cela

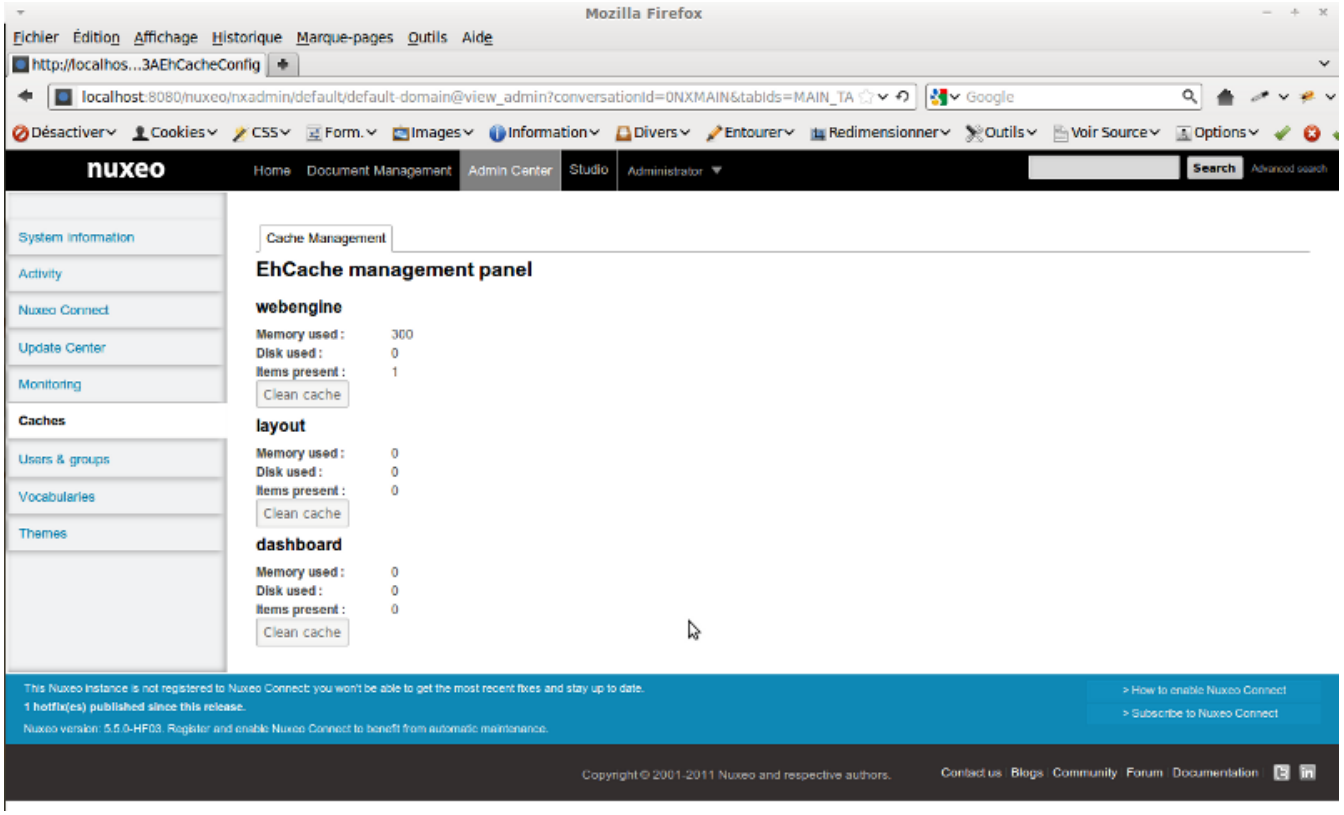
```
@Path("myRestService")
public class MyRestService {

```

```
4       @GET
5       @Produces("text/html; charset=UTF-8")
6       public Object getView(){
7           String myValue = grabData();
8           return getTemplate(EnumFreemarkerView.home).arg(FtlParams.myValue.name(), myValue);
9       }
10
11      public String grabData(){
12          final WebengineCacheService cache = WebengineCacheService.getInstance();
13          String result = cache.getObject(FrontCacheItem.myValue, String.class)
14
15          if(result==null){
16              cache.setObject(FrontCacheItem.myValue, "foobar");
17          }
18
19          return result;
20      }
21  }
```

3.4.1. Admin Center :

Pour permettre de vider le cache plus facilement, le module EhCache ajoute une vue dans l'admin center. Cette vue permet de visualiser l'utilisation des différer régions de caches (utilisation de la mémoire et du disque dur). Il est possible de vider les caches via des actions contenues dans cette vue.



moduleEhCache.png (65,81 ko) Patrick Guillerm, 04/04/2012 12:05

ehCacheFlow.png (121,82 ko) Patrick Guillerm, 05/04/2012 10:51

ehCacheAdminCenter.png (97,55 ko) Patrick Guillerm, 05/04/2012 16:15