6.1. Gestion du cache

Objectif

La gestion du cache a pour objectif d'améliorer les performances au sein de l'application sans contraintes sur le développement.

Ainsi, on peut stocker en mémoire certains appels méthodes dans une map de données.

Dans l'application, on utilise 2 façons d'utiliser le cache :

- les annotations Spring
- l'utilisation ad hoc du cache via ehcache

Ces 2 méthodes se basent sur un stockage en mémoire défini par un bean cacheManager.

Déclaration Spring du cacheManager

Définition du bean cacheManager

Voici la définition du bean cacheManager :

```
<bean id="cacheManager"
    scope="singleton"
    class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
    <description>
        This bean is used to configure EhCache.
    </description>
    <property name="configLocation" value="classpath:/properties/cache/ehcache.xml" />
</bean>
```

La seule chose à lui définir est l'emplacement de la configuration définissant l'ensemble des caches.

Définition du ehcache

Le fichier ehcache.xml définit l'ensemble des caches utilisé dans l'application.

On définit le cache de la manière suivante :

```
<cache name="org.esupportail.commons.services.ldap.CachingLdapServiceImpl"
    maxElementsInMemory="100" eternal="false" timeToIdleSeconds="300"
    timeToLiveSeconds="600" overflowToDisk="true" />
```

On déclare alors :

- name : le nom du cache (qui peut être le nom de la classe ciblée)
- maxElementsInMemory : le nombre maximum d'éléments cachable en mémoire
- eternal : définit si le cache se purge ou non
- timeToIdleSecond : le nombre maximum de secondes d'un élément durant lequel il peut exister dans la mémoire cache sans être consulté.
- timeToLiveSeconds : le nombre maximum de secondes d'un élément durant lequel il peut exister dans le cache indépendamment de leur utilisation.
- overflowToDisk : définit si lorsqu'on atteint le maxElementsInMemory, on écrit les données sur le disque

De plus, on définit l'espace de stockage disque dans le cas de gestion de cache avec overflowToDisk à true :

```
<diskStore path="/tmp/opiR1/cache" />
```

Enfin, on va définir un cache par défaut dans lequel on stocke les données non définis par les précédents cache :

```
<defaultCache maxElementsInMemory="10000" eternal="false"
    timeToIdleSeconds="120" timeToLiveSeconds="120" overflowToDisk="true"
    diskPersistent="false" diskExpiryThreadIntervalSeconds="120"
    memoryStoreEvictionPolicy="LRU" />
```

Gestion du cache par annotation Spring

Fonctionnement

Le cache est géré par annotation. Il suffit donc d'annoter une méthode pour quelle soit gérée par le cache.

Exemples :

Mise en cache du traitement.

```
@Cacheable(modelId = CacheModelConst.ACCESS_RIGHT_DEFAULT)
```

```
public Traitement getTraitement(final Integer id) {
        if (log.isDebugEnabled()) {
                log.debug("entering getTraitement( " + id + " )");
        }
        return daoService.getTraitement(id);
}
```

On vide le cache lors d'un ajout d'un traitement.

```
@CacheFlush(modelId = CacheModelConst.ACCESS_RIGHT_DEFAULT)
```

```
public void addTraitement(final Traitement traitement) {
        if (log.isDebugEnabled()) {
                log.debug("entering addTraitement( " + traitement + " )");
        }
        daoService.addTraitement(traitement);
        //flush the other instance
        executeFlushCache(CacheModelConst.ACCESS_RIGHT_DEFAULT, "addTraitement");
}
```

Déclaration Spring

Fichier properties/cache/cache.xml

Le bean cacheManager déclare le fichier ehcache.xml qui décrit chaque cache.

```
<bean id="cacheProvider"
        scope="singleton"
        class="org.springmodules.cache.provider.ehcache.EhCacheFacade">
        <property name="cacheManager">
                <ref local="cacheManager" />
        </property>
</bean>
```

Ces beans ci-dessous permettent d'utiliser les annotations pour la gestion du cache. L'attribut cachingModels définit un libellé pour le modelId (utilisé dans l'annotation) et le nom du cache dans ehcache.xml

```
<bean id="cachingAttributeSource"
        class="org.springmodules.cache.annotations.AnnotationCachingAttributeSource">
</bean>
<bean id="cachingInterceptor"
        class="org.springmodules.cache.interceptor.caching.MetadataCachingInterceptor">
        <property name="cacheProviderFacade" ref="cacheProvider" />
        <property name="cachingAttributeSource" ref="cachingAttributeSource" />
        <property name="cachingModels" >
                <props>
                        <prop key="accessRight.default">
                                cacheName=fr.univ.rennes1.cri.opiR1.domain.beans.parameters.accessRight.default
                        </prop>

                        <prop key="nomenclature">
                                cacheName=fr.univ.rennes1.cri.opiR1.domain.beans.parameters.nomenclature
                        </prop>
                        <prop key="references">
                                cacheName=fr.univ.rennes1.cri.opiR1.domain.beans.parameters.references
                        </prop>
                        <prop key="geographieApogee">
                                cacheName=gouv.education.apogee.commun.transverse.dto.geographie
                        </prop>
                        ...
                </props>
        </property>
</bean>
<bean id="cachingAttributeSourceAdvisor"
        class="org.springmodules.cache.interceptor.caching.CachingAttributeSourceAdvisor">
        <constructor-arg ref="cachingInterceptor" />
</bean>
```

Ces beans, ci-dessous, permettent d'utiliser les annotations pour vider les caches. Les “models” sont définis dans un fichier de properties. Un model peut représenter plusieurs cache. (fonctionnement spécifique à ehcache) Ceci permet de vider plusieurs caches en même temps.

```
<bean id="flushingAttributeSource"
 >
</bean>
<bean id="flushingInterceptor"
        class="org.springmodules.cache.interceptor.flush.MetadataFlushingInterceptor">
        <property name="cacheProviderFacade" ref="cacheProvider" />
        <property name="flushingAttributeSource" ref="flushingAttributeSource" />
        <property name="flushingModels" ref="cachePropertyConfigurer"/>
</bean>
<bean id="flushingAttributeSourceAdvisor"
        class="org.springmodules.cache.interceptor.flush.FlushingAttributeSourceAdvisor">
        <constructor-arg ref="flushingInterceptor" />
</bean>
      <bean id="cachePropertyConfigurer"
        class="org.springframework.beans.factory.config.PropertiesFactoryBean">
        <property name="location" value="classpath:/properties/cache.properties" />
</bean>
```

Ce bean permet de définir les beans dans lesquelles nous pourrons utiliser les annotations. Dans OPI ces beans correspondent à la couche métier de l'application.

```
<bean id="autoproxy"
        class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator">
        <property name="beanNames">
                <list>
                        <idref bean="domainService" />
                        <idref bean="parameterService" />
                        <idref bean="domainApoService" />
                        <idref bean="businessCacheService" />
                </list>
        </property>
        <property name="interceptorNames">
                <list>
                        <value>cachingInterceptor</value>
                        <value>flushingInterceptor</value>
                </list>
        </property>
</bean>
```

Fichier properties/cache.properties

Ce fichier contient la description des models pour le flush (pour vider les caches). Un model peut regrouper plusieurs cacheName ce qui permet de vider plusieurs cache lors du passage dans une méthode.

- Exemple :

```
Le model user vide le cache des utilisateurs et celui des pièces manquantes.
user=cacheNames=fr.univ.rennes1.cri.opiR1.domain.beans.user.User,fr.univ.rennes1.cri.opiR1.domain.beans.user.candidature.MissingPiece
```

Utilisation

Définition des models

Les models utilisés pour le cache et pour le flush sont identiques et sont déclarés dans une classe static : fr.univ.rennes1.cri.opiR1.web.utils.CacheModelConst

- Exemple dans la classe CacheModelConst

  /**
- The name of model for the domain access right cache.
- (Profile, Treatment,...)
  */
  public static final String ACCESS_RIGHT_DEFAULT = "accessRight.default";* Exemple d'utilisation dans une annotation : cf ci-dessus dans la partie Fonctionnement

Gestion du cache avec ehcache

L'utilisation ad hoc du cache revient à initialiser, charger et lire "manuellement" les éléments via le cacheManager.

Initialisation du cache

Voici un exemple de code permettant l'instanciation du cache :

```
/**
 * The cache object for the VET.
 */
private Cache cacheVet;

// on consulte dans le cacheManager si l'entrée du cache existe déjà
if (!cacheManager.cacheExists("org.esupportail.apogee.domain.dto.enseignement.VersionEtapeDTO")) {
    cacheManager.addCache("org.esupportail.apogee.domain.dto.enseignement.VersionEtapeDTO");
}
cacheVet = cacheManager.getCache("org.esupportail.apogee.domain.dto.enseignement.VersionEtapeDTO");
```

La classe Cache fait partie de la librairie net.sf.ehcache.

Chargement du cache

Une fois le cache instanciation faite, on initialise le cache avec les données qu'on souhaite stocker :

```
/**
 * Initialisation du cache VET.
 */
public void initCacheVet() {
    // on vide le contenu du cache
    cacheVet.flush();
    // on prépare l'élément à stocker qui peut avoir n'importe quel format
    Map<String, List<VersionEtapeDTO>> mapOfVet = new HashMap<String, List<VersionEtapeDTO>>();
    ...
    // mise en cache des éléments de la map, la clé étant la clé dans le cache, l'élément stocké étant la liste
    for (Map.Entry<String, List<VersionEtapeDTO>> vetEntry : mapOfVet.entrySet()) {
        cacheVet.put(new Element(vetEntry.getKey(), vetEntry.getValue()));
    }
}
```

La classe Element fait partie de la librairie net.sf.ehcache.

Lecture du cache

Lorsqu'on veut récupérer un des éléments stocké dans le cache, on procède comme suit :

```
public VersionEtapeDTO getVersionEtape(final String codEtp, final Integer codVrsVet) {
    // dans le cas ou le cache ne contient pas la clé
    // c'est que soit l'élément n'est pas dans le cache, soit le cache a été flushé
    // dans tous les cas, on réinitialise le cache
    if (cacheVet.get(codEtp) == null) {
        initCacheVet();
    }
    if (cacheVet.get(codEtp) != null) {
        Element element = cacheVet.get(codEtp);
        VersionEtapeDTO result = null;
        // on récupère l'élément stocké dans le cache
        List<VersionEtapeDTO> v = (List<VersionEtapeDTO>) element.getObjectValue();
        for (VersionEtapeDTO vet : v) {
            if (codVrsVet.equals(vet.getCodVrsVet())) {
                result = vet;
                break;
            }
        }
        return result;
    }
    return null;
}
```