Sakai Caching

Information

This documents the proper way to handle caching in Sakai. This will be updated as improvements are made to the Sakai caching system.

A list of all caches in your current Sakai installation is visible in the Admin only **Memory** tool (found in the **Administration Workspace** site). Click the Status button to see a listing of all caches and information about them.

Sakai includes about 40 caches (as of Sakai 10) which can be individually configured. Correct settings for caches in production depends heavily on your use of Sakai so it is generally important to adjust caches individually for large installations.

Sakai added support for JCache (JSR-107) compatible caching in version 10+, more info here:

~~SAK-38636~~ - Rewrite MemoryService APIs to allow pluggable caching (JSR-107) CLOSED

Sakai caching configuration

Sakai caches are all configured via the Sakai config (mostly via the properties files - sakai.properties, local.properties, etc.). The configuration works like so for new and legacy caching in Sakai 10 or 2.9:

```
# memory.{cachename}={key=value,key=value,...}
# The main keys are eternal, timeToLiveSeconds, timeToIdleSeconds, maxElementsInMemory
# See the admin memory tool for a complete list of caches in your environment
# For example - the setting below controls the caching of users:
# - stores users for an hour max
# - idle users are removed from the cache after 15 mins
# - a max of 20000 users will be stored in the cache
memory.org.sakaiproject.user.api.UserDirectoryService.callCache=timeToLiveSeconds=3600,timeToIdleSeconds=900,
maxElementsInMemory=20000
```

There are more details available in the default.sakai.properties file (especially related to the more experimental caches).

Distributed Caching

Distributed caching in Sakai requires setting up a distributed caching system and then configuring Sakai to use that system. See the specific supported implementations below.

- Terracotta
- Hazelcast (work in progress - SAK-39395 - Implement hazelcast caching IN PROGRESS)

Developing

Developing using the caching system in Sakai is fairly easy. You basically use a service API to create a cache and then use the Cache object to store and retrieve caching data.

Sakai MemoryService

Using the Sakai MemoryService service and Cache APIs (Java classes) are the recommended way to handle caching in Sakai. Direct use of the underlying caching mechanism is strongly discouraged and will certainly stop working in later versions of Sakai (10+)

A developer who wants to implement caching in a tool or extension can simply use the MemoryService to request/establish a cache by name. By convention, caches are named by the fully qualified classname of the service that uses the cache and the name or type or data in the cache at the end. For example, "org.sakaiproject.user.api.UserDirectoryService.callCache" is the cache of users by request (or by call) and the service is the UserDirectoryService (org.sakaiproject.user.api.UserDirectoryService).

The memoryService would be injected via spring or obtained from the CM (see Using the MemoryService)

```
Cache myCache = memoryService.getCache("org.sakaiproject.my.MyService.myCache");
```

One you have the Cache object, you can use it to add or remove items from the cache. The typical usage is to add an item to the cache when it is retrieved and remove it from the cache when it is updated. This code tries to find the item in the cache first and then if it is not found, gets it from the storage (database, etc.) instead and adds it to the cache.

```
// This code would be located in a method to fetch a MyObject by its unique id
String ref = myItemId; // this is the unique id for the objects my code is working with
MyObject myObj = myCache.get(ref);
if (myObj == null) {
    // fetch my object from where it is stored since we did not find it in the cache
    myObj = my_storage.getMyItem(ref);
        // ... do other stuff as needed
    if (myObj != null) {
                // object was found so store it in the cache
        // NOTE: you may want to store that the object does not exist as well
        myCache.put(ref, myObj);
    }
}
```

This code updates the item in the cache when the data changes. If there is no code like this then the items in the cache can become stale and updates won't be reflected in a timely manner.

```
MyObject myObj = my_storage.saveMyItem(item);
// This code would be located in a method which updates the MyObjects (after the object is updated)
String ref = myObj.getId(); // this is the unique id for the object that was updated
myCache.put(ref, myObj);
```

This code removes the item from the cache when it is removed from the persistent storage

```
String ref = myItemId; // this is the unique id for the object that was removed
my_storage.removeMyItem(ref);
// This code would be located in a method which removes the MyObjects (after the object is removed)
myCache.remove(ref);
```

Those are the basics of using the caching in Sakai. Sakai generally follows the java caching standard so cache techniques that work in most systems will work in Sakai.

Implementing support for a new Cache system

Sakai includes support for JCache (JSR-107) type caches in Sakai 10+. This is managed using 2 APIs that can be implemented without changing any of the existing use of cache code throughout the rest of the system. The 2 APIs that have to be implemented are org.sakaiproject.memory.api.MemoryService and org.sakaiproject.memory.api.Cache. The existing implementations are the legacy ones (BasicMemoryService and MemCache) and the new Ehcache ones (EhcacheMemoryService, EhcacheCache).

Once those APIs are implemented, the developer needs to adjust the BaseMemoryService (by adding the new caching implementation to the init() and destroy() methods) to allow the new implementation to be enabled via configuration options. The configuration options in Sakai core are shown below.

```
# Configure the cache manager implementation (which caching system Sakai uses to manage its caches)
# Options include: ehcache, legacy
# Default: ehcache
#memory.cachemanager=ehcache
```

Caching in Sakai 2.9 or older

Caching in 2.9 or older was often done directly using Ehcache directly (or sometimes using the Sakai Cache API). This is expressed as a Spring Bean Factory with the id **org.sakaiproject.memory.api.MemoryService.cacheManager**

1. Create a cache in your service (in the components.xml) like so:

```
<bean id="org.sakaiproject.user.api.UserDirectoryService.cache"
    class="org.springframework.cache.ehcache.EhCacheFactoryBean">
  <property name="cacheManager">
    <ref bean="org.sakaiproject.memory.api.MemoryService.cacheManager"/>
  </property>
  <property name="cacheName">
    <value>org.sakaiproject.user.api.UserDirectoryService</value>
  </property>
</bean>
```

2. Spring inject the cache into your service like so:

```
<bean id="org.sakaiproject.user.api.UserDirectoryService" ...
...
  <property name="cache">
    <ref bean="org.sakaiproject.user.api.UserDirectoryService.cache" />
  </property>
</bean>
```

3. Use the cache within your code:

```
cache.get(key).getObjectValue();
cache.put(new Element(key,value));
cache.remove(key);
```

- **NOTE** This cache is not cluster wide and exists on one server only. You have to handle cluster wide expiring yourself.