Cache Service

Summary

In the e-government framework, the Cache Service selects EhCache to guide. If the object costs much, has been frequently used while few changed, the following advantages can be obtained using cache.

- Since it is not required to fetch the frequently accessed data from database every time, overhead reduces.
- Since object is not used every time, memory can be used effectively.

Description

We'll explain basic setting for using a cache and basic method of use.

Bootstrap Source

We'll explain the method of creating the Cache Manager to use a cache through sample.

```
//Use class path to read the setting file and to create Cache Manager.
URLurl = getClass().getResource("/ehcache-default.xml");
manager = new CacheManager(url);
```

As above, the file contents of /ehcache-default.xml read through getResource are as follows:

Configuration

```
<ehcache>
<diskStore path="user.dir/second"/>
<cache name="sampleMem"
maxElementsInMemory="3"
eternal="false"
timeToIdleSeconds="360"
timeToLiveSeconds="1000"
overflowToDisk="false"
diskPersistent="false"
memoryStoreEvictionPolicy="LRU">
</cache>
</ehcache>
```

Basic Usage Source

The following is the sample of writing, reading and deleting by obtaining the Cache in the Cache Manager defined above.

```
// Obtain cache with cache Name
Cache cache = manager.getCache("sampleMem");

// 1.Enter data in Cache
cache.put(new Element("key1", "value1"));

// 2.Read data entered from Cache
Element value = cache.get("key1");

// 3. Delete data from Cache
cache.remove("key1");
```

- manager.getCache: Obtain relevant Cache using Cache Manager
- cache.put: Enter data in Cache(processed if entering other value for the same key value)
- cache.get: read data from Cache
- cache.remove: read data from Cache.

Cache Algorithm

Cache designates the storage size to manage essential data only since it uses memory by default. If it exceeds the size, data is deleted starting from unnecessary data. Determination on whether it is necessary data is based on algorithm. Supported algorithms are 3 types including LRU,FIFO,LFU. Setting and methods for these are as shown below.

LRU Algorithm

An algorithm to leave the recent use and it is set as follows.

Cofiguration

```
<cache name="sampleMem"
maxElementsInMemory="3"
               ...
memoryStoreEvictionPolicy="LRU">
```

From above setting, the maximum number of entry in memory(maxElementsInMemory) is 3 and the algorithm is set to LRU. Following are the source checking the results for the setting using this setting.

Sample Source

```
    Cache cache = manager.getCache("sampleMem");
    ...
cache.get("key2");
cache.get("key2");
cache.get("key1");
cache.get("key1");
cache.get("key3");

    // 4. Put New element in cache.
cache.put(new Element("key4", "value4"));

    // 5. get key2 but can't key2
assertNull("Can't get key2",cache.get("key2"));
```

FIFO Algorithm

An algorithm to remove the element entered first and is set as follows.

Configuration

```
<cache name="sampleMemFIFO"
maxElementsInMemory="3"
               ...
memoryStoreEvictionPolicy="FIFO">
```

From above setting, the maximum number of entries in storage (maxElementsInMemory) is 3 and set to FIFO. Following is the source to check the results for the setting using this setting.

Sample Source

```
    Cache cache = manager.getCache("sampleMemFIFO");
cache.put(new Element("key1", "value1"));
cache.put(new Element("key2", "value2"));
cache.put(new Element("key3", "value3"));
    ...
cache.get("key2");
cache.get("key2");
cache.get("key1");
cache.get("key1");
cache.get("key3");

    // 4. Put New element in cache.
cache.put(new Element("key4", "value4"));

    // 5. get key1 but can't key1
assertNull("Can't get key1",cache.get("key1"));
```

LFU Algorithm

An algorithm to remove the element less used, and is set as follows.

Configuration

```
<cache name="sampleMemLFU"
maxElementsInMemory="3"
               ...
memoryStoreEvictionPolicy="LFU">
```

From above setting, the maximum number of entries in memory(maxElementsInMemory) is 3 and the algorithm is set to LFU. Following is the source to check the results for the setting using this setting.

Sample Source

```
    Cache cache = manager.getCache("sampleMemLFU");
cache.put(new Element("key1", "value1"));
cache.put(new Element("key2", "value2"));
cache.put(new Element("key3", "value3"));
    ...
cache.get("key2");
cache.get("key2");
cache.get("key1");
cache.get("key1");
cache.get("key3");

    // 4. Put New element in cache.
cache.put(new Element("key4", "value4"));

    // 5. get key2 but can't key3
assertNull("Can't get key3",cache.get("key3"));
```

Cache Size & Device

Set the size of information to manage in Cache and storage device.

Cache Device

Device related setting is the setting related to movement to disk management of memory management cache, including whether to move to the disk management of memory management cache when the number of memory management object is exceeded, whether to save as file at flush calling. Setting for this is as shown below:

Configuration

```
<cache name="sampleDisk"
overflowToDisk="true"
diskPersistent="true"
         ...>
</cache>
```

- overflowToDisk: whether to move to Disk when it exceeds the memory management object (true,false)
- diskPersistent: whether to save as a file at the time of flushing (trun , false)

Following is the source to check the results for the setting using this setting.

Sample Source

```
Ehcache cache = manager.getCache("sampleDisk");

    // 1. Put a content in Cache.
cache.put(new Element("key1", "value1"));

    // 2. Get that item from Cache.
Element value = cache.get("key1");
assertEquals("value1", value.getValue().toString());

    // 3. flush.
cache.flush();
FiledataFile = new File(manager.getDiskStorePath()+ File.separator + "sampleDisk.data");
    // 4. Check save as a file
assertTrue("File exists", dataFile.exists());
```

From above sample, when flush is executed, it is saved as a file. Check whether to move the memory management object disk at the example of following Cache Size.

Cache Size

Size related setting includes the number of maximum object and the setting for number of maximum object to manage at disck. Following is the setting for this.

Configuration

```
<cache name="sampleDisk"
maxElementsInMemory="3"
maxElementsOnDisk="2"
overflowToDisk="true"
        ...>
</cache>
```

- maxElementsInMemory: the maximum number of objects managed in the memory
- maxElementsOnDisk: the maximum number of objects managed in the disk.

Following is the source to check the results for the setting using this setting.

Sample Source

```
    Cache cache = manager.getCache("sampleDisk");

    // 1. Put 3 contents in Cache.
cache.put(new Element("key1", "value1"));
cache.put(new Element("key2", "value2"));
cache.put(new Element("key3", "value3"));

    // 2. Put Fourth content in Cache.
cache.put(new Element("key4", "value4"));
assertEquals(3, cache.getMemoryStoreSize()); //3 managed at Memory
assertEquals(1, cache.getDiskStoreSize());    //1 go to Disk.

    // 3. Put 5~7 contens in Cache.
cache.put(new Element("key5", "value5"));
cache.put(new Element("key6", "value6"));
cache.put(new Element("key7", "value7"));

    // All moved regardless of Disk Max Size
assertEquals(3, cache.getMemoryStoreSize()); //3 managed at Memory
assertEquals(4, cache.getDiskStoreSize());    //2 moved to Disk and 4 managed.

    // Move those in the memory to disk
cache.flush();
    // Changed to Max Size of disk.
assertEquals(0, cache.getMemoryStoreSize()); //Nothing left at Memory.
assertEquals(2, cache.getDiskStoreSize());    //Only 2 remains at Disk as indicated in DiskMaxSize.
```

From above example, in case of cache.put, it is continuously handed over from memory to disk regardless of MaxSize of Disk, but if flushing is performed, it was confirmed that only maximum number of disk storage is left.

Distributed Cache

Ehcache supports RMI,JGROUP,JMS as methods to support Distributed Cache. We'll examine JMS support setting using JGROUP andActiveMQ among these, and how to use it. See EhcacheUserGuidefor details.

Using JGroups

Jgroups creates the group as a communication toolkit based on multicast and supports sending and receiving between group members. See Sitefor more details.

Configuration

```
<cacheManagerPeerProviderFactory
class="net.sf.ehcache.distribution.jgroups.JGroupsCacheManagerPeerProviderFactory"
properties="connect=UDP(mcast_addr=224.10.10.10;mcast_port=5555;ip_ttl=32;
mcast_send_buf_size=150000;mcast_recv_buf_size=80000):
PING(timeout=2000;num_initial_members=6):
MERGE2(min_interval=5000;max_interval=10000):
                                            FD_SOCK:VERIFY_SUSPECT(timeout=1500):
pbcast.NAKACK(gc_lag=10;retransmit_timeout=3000):
UNICAST(timeout=5000):
pbcast.STABLE(desired_avg_gossip=20000):
                                            FRAG:
pbcast.GMS(join_timeout=5000;join_retry_timeout=2000;shun=false;print_local_addr=false)"
propertySeparator="::"/>
<cache name="cacheSync"
maxElementsInMemory="1000"
eternal="false"
timeToIdleSeconds="1000"
timeToLiveSeconds="1000"
overflowToDisk="false">
<cacheEventListenerFactory
class="net.sf.ehcache.distribution.jgroups.JGroupsCacheReplicatorFactory"
properties="replicateAsynchronously=false, replicatePuts=true,
                                                                                                                                          re
                                                                                                                                          re

</cache>
```

Following is the source to check the results for the setting using this setting.

Sample Source

```
    // Read the above setting file
```

```
URLurl = this.getClass().getResource("/ehcache-distributed-jgroups.xml");
    // Create 2 Cache Managers
    manager1 = new CacheManager(url);
    manager2 = new CacheManager(url);

for (inti = 0; i< 10 ; i++) {
manager1.getEhcache(CACHE_SYNC).put(new Element(new Integer(i), "value"));
    }
    // Require time for replication.
Thread.currentThread().sleep(100);
    // Replicated so that same cache information is entered in manager2.
assertTrue(manager1.getEhcache(CACHE_SYNC).getKeys().size() ==
manager2.getEhcache(CACHE_SYNC).getKeys().size() );
```

Using ActiveMQ

ActiveMQ is the open source providing JMS messaging. See Site for details

Configuration

```
<cacheManagerPeerProviderFactory
class="net.sf.ehcache.distribution.jms.JMSCacheManagerPeerProviderFactory"
properties="initialContextFactoryName=egovframework.rte.fdl.cache.distribute.TestActiveMQInitialCon
textFactory,
providerURL=tcp://localhost:61616,
replicationTopicConnectionFactoryBindingName=topicConnectionFactory,
getQueueConnectionFactoryBindingName=queueConnectionFactory,
replicationTopicBindingName=ehcache,
getQueueBindingName=ehcacheGetQueue"
propertySeparator=","
                />

<cache name="CacheAsync"
maxElementsInMemory="1000"
eternal="false"
timeToIdleSeconds="1000"
timeToLiveSeconds="1000"
overflowToDisk="false">

<cacheEventListenerFactory class="net.sf.ehcache.distribution.jms.JMSCacheReplicatorFactory"
properties="replicateAsynchronously=true,
replicatePuts=true,
replicateUpdates=true,
replicateUpdatesViaCopy=true,
replicateRemovals=true,
asynchronousReplicationIntervalMillis=1000"
propertySeparator=","/>
</cache>
```

Following is the source to check the results for the setting using this setting.

Sample Source

```
URLurl = this.getClass().getResource("/ehcache-distributed-activemq.xml");
    manager1 = new CacheManager(url);
    manager2 = new CacheManager(url);

cacheName = "CacheAsync";
for (inti = 0; i< 10 ; i++) {
manager1.getEhcache("CacheAsync").put(new Element(new Integer(i), "value"));
    }
    // certain amount of time is required for replication.
Thread.currentThread().sleep(1000);

assertTrue(manager1.getEhcache("CacheAsync").getKeys().size() ==
manager2.getEhcache("CacheAsync").getKeys().size() );
```

6. Spring Integration

Explain the basic cache service using the setting as well as setting for how to use Ehcache in Spring.

Configuration - Spring Application Context

```
<bean id="ehcache" class="org.springframework.cache.ehcache.EhCacheFactoryBean">
<property name="cacheManager">
<bean class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
<property name="configLocation" value="classpath:ehcache-default.xml"/>
</bean>
</property>
</bean>
```

With above setting, cache can be obtained not through Manager.   ehcache-default.xml defined in ConfigLocation is the same as the setting file described in Configuration of Cache Basic.

Sample Source

```
@Resource(name="ehcache")
EhcachegCache ;

    // Find cache with cache Name
Ehcache cache = gCache.getCacheManager().getCache("sampleMem");

cache.put(new Element("key1", "value1"));
Element value = cache.get("key1");
```

From above example, it is confirmed that Ehcache is brought using ehcache and cache information is read through getCacheManager(). After that, it can be confirmed that it is used in the same method as those explained above.

Reference

EhCache