Chapter 8. Cache Management

8.1. Introduction

In this lab, you will learn to configure cache expiration and cache eviction.

Concepts you will gain experience with:

- Configuring cache expiration
- Configuring cache eviction

Estimated completion time: 30 minutes

8.2. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)).

8.3. Detailed Instructions

Instructions for this lab are divided into specific sections. Each section describes the steps to perform specific tasks.

8.3.1. Configuring Cache Expiration

Expiration ensures that your data is current and pertinent by removing old and unused entries from the GemFire. It can also give you some control over Region size. When you try to access an expired entry, GemFire looks to another source for a fresh update. If the data does not exist in another member, the cache loader you have installed on the data region is called.

This exercise uses idle time expiration, configured through the XML file to destroy expired entries. The program creates four entries and then waits beyond the expiration time. During the wait time, the program updates one entry and accesses another. When the expiration time elapses, the third and fourth entries (which are not accessed and not modified) expire. The cache listener installed on the region reports all changes to the entries.

8.3.1.1. Configure the XML

Configure and add the corresponding region attributes in `xml/serveCache.xml` to:

1. (`TODO-01`) Configure the `Customer` region with entry idle time expiration. Set the timeout to `10 seconds`, and `destroy` expiration action.
2. (`TODO-02`) Configure the `ServerCacheListener` cache listener to report changes to the cached data.

8.3.1.2. Configure the Application

Open the `SampleDataExpiration` class skeleton within `com.gopivotal.bookshop.buslogic` package, and add the code for the following behavior:

1. (`TODO-03`) Add the code in the `getExpirationTime()` method to return the entry idle timeout setting from the region attributes (`serverCache.xml` file).

   Tip

   You can access these configured values by invoking the appropriate `get()` method on the region.

2. (`TODO-04`) Add the code in the `updateCustomer()` method to update the customer details of the key passed in. Make changes to any information, except the key value and the customer ID. A good option is to add a new address.
3. (`TODO-05`) Review the `retrieveAndPrintRegionData()` method and observe it's purpose is to iterate over the entries and print out the details of each.
4. (`TODO-06`) Add the code in the `run()` method, in the following sequence:
   a. Retrieve and populate the cache with the customer's data.
   b. Show the contents of the cache (calling the `retrieveAndPrintRegionData()` method).
   c. Get the idle timeout value, and use `Thread.sleep()` method to suspend the thread execution for (timeout - 1) seconds (remember the `sleep()` method assumes time is in milliseconds).
   d. Perform a `get()` on the Customer region to get one customer, and update the details of the second customer by calling `updateCustomer()`.
   e. Use the `Thread.sleep()` method to sleep past expiration timeout.
   f. Show the contents of the cache past expiration timeout.

8.3.1.3. Executing the Application

Execute the application in STS or command prompt window, and observe that after the expiration timeout, the cache contains only data that are either accessed or modified. In this case, we are running in an embedded model for simplicity. Notice that because you retrieved entry 5540 and updated entry 5541, they remain in the cache after expiration.

```
Received afterCreate event for entry: 5540, Customer [customerNumber=5540, firstName=Lula, lastName=Wax]
Received afterCreate event for entry: 5541, Customer [customerNumber=5541, firstName=Tom, lastName=Mcginns]
Received afterCreate event for entry: 5542, Customer [customerNumber=5542, firstName=Peter, lastName=Fernendez]
```

```
    Received afterCreate event for entry: 5543, Customer [customerNumber=5543, firstName=Jenny, lastName=Tsai]
************ Server 1 : Loaded customers data ***************

    5540 => Customer [customerNumber=5540, firstName=Lula, lastName=Wax]
    5541 => Customer [customerNumber=5541, firstName=Tom, lastName=Mcginns]
    5542 => Customer [customerNumber=5542, firstName=Peter, lastName=Fernendez]
    5543 => Customer [customerNumber=5543, firstName=Jenny, lastName=Tsai]
Before the idle time expiration, access and update one entry each...

Accessed customer, Lula  data !!
    Received afterUpdate event for entry: 5541, Customer [customerNumber=5541, firstName=Tom, lastName=Mcginns]
************ Updated customer data ***************

    Received afterDestroy event for entry: 5543
    Received afterDestroy event for entry: 5542
After the expiration timeout, the cache contains:

    5540 => Customer [customerNumber=5540, firstName=Lula, lastName=Wax]
    5541 => Customer [customerNumber=5541, firstName=Tom, lastName=Mcginns]
Closing the cache and disconnecting.
```

8.3.2. Configuring Cache Eviction

You can use eviction to control how much heap your data regions use. Eviction controls your data region size by removing least recently used (LRU) entries to make way for new data. It kicks in when your data region reaches a specified size or entry count.

In this exercise, the data region is configured to keep the entry count at `4` or below by destroying LRU entries. A cache listener installed on the region reports the changes to the region entries.

8.3.2.1. Configuring the XML

(`TODO-07`) Configure and add the corresponding region attributes in the `serverCache.xml` to destroy entries when the region reaches entry count of `4` . You should put this just after the cache listener entry in the file.

8.3.2.2. Running the Application

Open the `SimpleDataEviction` class within the `com.gopivotal.bookshop.buslogic` package. Observe that the code is written to first insert 4 Customer entries, followed by a fifth entry.

Run this program and observe the output. You should see something like the following:

```
    Received afterCreate event for entry: 5540, Customer [customerNumber=5540, firstName=Lula, lastName=Wax]
    Received afterCreate event for entry: 5541, Customer [customerNumber=5541, firstName=Tom, lastName=Mcginns]
    Received afterCreate event for entry: 5542, Customer [customerNumber=5542, firstName=Peter, lastName=Fernendez]
    Received afterCreate event for entry: 5543, Customer [customerNumber=5543, firstName=Jenny, lastName=Tsai]
************ Loaded customers data ***************
    5540 => Customer [customerNumber=5540, firstName=Lula, lastName=Wax]
    5541 => Customer [customerNumber=5541, firstName=Tom, lastName=Mcginns]
    5542 => Customer [customerNumber=5542, firstName=Peter, lastName=Fernendez]
    5543 => Customer [customerNumber=5543, firstName=Jenny, lastName=Tsai]
************ Loaded customers data ***************
    Received afterCreate event for entry: 5545, Customer [customerNumber=5545, firstName=Fifth, lastName=Customer]
    Received afterDestroy event for entry: 5540
************ Inserted one more customer data ***************
    5541 => Customer [customerNumber=5541, firstName=Tom, lastName=Mcginns]
    5542 => Customer [customerNumber=5542, firstName=Peter, lastName=Fernendez]
    5543 => Customer [customerNumber=5543, firstName=Jenny, lastName=Tsai]
    5545 => Customer [customerNumber=5545, firstName=Fifth, lastName=Customer]
```

Notice that in order to insert the fifth entry, it had to evict one of the other entries. The method it used was to locate the least recently used (the first entry) and evict it. Observer the above output and note that when entry `5545` was created that entry `5540` was destroyed.

Congratulations!! You have completed this lab.