Chapter 10. Handling Events on the Client

10.1. Introduction

In this lab, you will gain hands-on experience working with event handling in a GemFire client cache. In addition, you will gain experience writing a continuous query, which will give you a chance to blend OQL concepts with event processing concepts.

Concepts you will gain experience with:

- Writing & registering a cache listener
- Registering interest in events related to certain keys
- Writing a continuous query event handler
- Submitting a continuous query

Estimated completion time: 40 minutes

10.2. 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*)).

10.3. Instructions

This lab is broken into two key sections. The first will involve writing and registering a cache listener. The second section will focus on using a continuous query.

10.3.1. Using a CacheListener

In this section, you will be writing and registering a cache listener attached to your client cache. To understand this part of the lab, let's outline the components that will be involved.

This solution is made up of two components:

- One worker client that retrieves and adds data in its cache
- One consumer client, which receives events from the cacheserver

The consumer client registers interest in some or all keys in the data region. The worker client updates its cache, and the updates are automatically forwarded to the cacheserver. The server forwards to the consumer only those events in which the consumer has registered interest.

The consumer client has an asynchronous listener that reports local cache activity to standard out, so you can see what is happening. Note that while the ClientWorker could receive similar events, it hasn't registered interest or configured a CacheListener so events will only be sent to the ClientConsumer.

10.3.1.1. Implementing a CacheListener

First, you will implement a CacheListener that will log all create events. Perform the following steps.

1. (`TODO-01`) Open the `LoggingCacheListener` class in the `com.gopivotal.bookshop.buslogic` package (under `src/main/java`). Notice that this class extends the `CacheListenerAdapter` class, which means that it doesn't have to implement all the methods defined by the `CacheListener` interface. This way, you only have to implement (override) the methods you are interested in for the events you want to log.

   Locate the `init()` method and implement the functionality to extract a property named 'filename' to a String. Then, use this value to call `initializeLogger()` to set up the logger to point to this file.

2. (`TODO-02`) Next, locate the afterCreate() method. Implement this method to log an info type message using the logger class attribute. In the log message, include both the key and new value. Optionally, implement the other methods defined by the CacheListener to log additional events of interest (ex when an entry is updated or deleted).

   Tip

   Refer to the JavaDocs for the `CacheListener` interface as well as the `EntryEvent` classes to determine the correct methods to invoke.

3. (`TODO-03`) Next, open the `clientConsumerCache.xml` file and modify the pool definition to ensure that subscriptions are enabled.
4. (`TODO-04`) Finally, locate the BookMaster region definition and modify the region configuration to register this `LoggingCacheListener` class as a cache listener. Be sure to also specify the parameter to represent

the 'filename' property.

10.3.1.2. Registering Interest in ClientConsumer

Registering interest is simply a matter of obtaining a reference the appropriate region and invoking `registerInterest()` with the desired key or keys.

(`TODO-05`) Open the `ClientConsumer` class (in the `com.gopivotal.bookshop.buslogic` package) and locate the line marked with the `TODO` item. Make the appropriate call to the `BookMaster` region instance to register interest in entries with key `999`.

10.3.1.3. Running ClientWorker and ClientConsumer

In order to test the behavior you just implemented, it will be necessary to run both the ClientConsumer and ClientWorker in the following way.

1. First, make sure you have started the server side processes using the `startServer.sh` script (`startServer.bat` for Windows) in the `server-bootstrap/scripts` lab folder.
2. Run the ClientConsumer by either locating the class in the Package Explorer or in an editor tab. Right mouse click on the file or in the editor and select Run As -> Java Application. You will see output to the console indicating when the application has started and is ready for the other application to run.
3. Next, run the ClientWorker by either locating the class in the Package Explorer or in an editor tab. Right mouse click on the file or in the editor and select Run As -> Java Application. This application will run to a certain point. You should see the following in the console output.

   ```
   Connecting to the distributed system and creating the cache.
   Note the other client's region listener in response to these gets.
   Press Enter to continue.
   ```

   Place your cursor in the console and press Enter. At this point, the program will continue with inserting and then destroying an entry with key 999.
4. Switch back to the console for ClientConsumer. Refer to the illustration below to see how this is done.

   On the ClientConsumer console output, you should see entries that indicate that an entry was inserted per the logger. The exact output may differ from the above but should be based on how you wrote the logging message in your LoggingCacheListener.

10.3.2. Using a Continuous Query

In this next section, you will be combining your understanding of event processing with the prior experience gained with OQL style queries. This will involve implementing a CQListener to handle Continuous Query events and writing the necessary code to register the listener for a specific query.

1. (`TODO-06`) Open the `SimpleCQListener` class and implement the code of the `onEvent()` method. Write code to print out the various values of the `CqEvent` object that is passed in.
2. Next, open the CQClient class and locate the registerCq() method. Perform the following steps to set up and register the continuous query.
   a. (`TODO-07`) Use the pool instance to get a QueryService instance
   b. (`TODO-08`) Use the QueryService to create a CqAttributes instance, registering the SimpleCQListener class created in the prior step.
   c. (`TODO-09`) Write a query to trigger an event when a BookOrder is created having a totalPrice greater than $100.
   d. (`TODO-10`) Using the CqAttributes you created and the query you wrote, create a new CqQuery and then execute it. If you decide to execute with initial results, capture the results and iterate over them, printing out the orders.
3. Finally, test out your implementation
   a. Start by running the `CQClient` class. Right mouse click on the file or in the editor and select `Run As -> Java Application`. This application will run to a certain point. You should see the following in the console output.

      ```
      Made new CQ Service
      Press enter to end
      ```

   b. Next, locate the `DataProducer` class and run using a similar approach as the prior step. You will see the following output.

      ```
      Press enter to populate an order over $100
      ```

      Place your cursor in the console and hit `Enter`. This will cause the `DataProducer` to insert an order for $100, which should trigger the `CQListener`. As a result, you should see the console switch back to the CQClient app and display whatever output you defined when implementing the `onEvent()` method of the `SimpleCQListener` class.

c. Use the technique you used above in Running ClientWorker and ClientConsumer to switch back to the `DataProducer` program that is still running. Place your cursor in the console area again and hit `Enter`. This will insert a BookOrder for a total price less that $100. Therefore, you should see no additional output from the `SimpleCQListener`.

d. At this point, the `DataProducer` app has terminated. Switch back to the console for the `CQClient`. First verify no additional output was generated. Then, place your cursor in the console area and hit `Enter` to cause this application to end.

Congratulations! You have completed this lab.
