Chapter 9. Server Side Event Handling

9.1. Introduction

In an earlier lab, you became familiar with client side even handling. In this lab, you will learn to configure events on the server side of a GemFire distributed system.

Concepts you will gain experience with:

- Creating and registering a CacheLoader
- Creating and registering a CacheWriter

Estimated completion time: 30 minutes

9.2. 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*)).

9.3. Detailed Instructions

Instructions for this lab are divided into specific sections. Each section describes the steps to perform specific tasks. Before beginning any of these tasks.

9.3.1. Preparation

In performing the steps of this lab, you will take a bit of a Test Driven Development approach. You will first run the series of tests, find all (or most) fail and then set about creating solutions to cause them to pass.

1. To begin, navigate to the `server-events` project in the STS. Locate the `ServerEventsTests` class in `src/test/java` under the `com.gopivotal.bookshop.tests` package. Open it up and take a look at the various tests.
2. Next start the locator and one server. The easiest way to do this is to open a terminal or command window and change directories to this lab folder (`server-events`). Then issue the following command.

```
gfsh run --file=serverStart.gf
```

> **Note**
>
> If gfsh fails to run, it's likely because you don't have your environment variables set. Recall that these were defined in one of the first labs in the `server-bootstrap/scripts` folder. Locate that file and either run `setEnv.bat` for Windows or **source gf.config** for Mac/Linux. Once done, go back and re-try the above gfsh command.

3. Finally, run the tests (right mouse click on `ServerEventsTest -> Run As - JUnit Test`). You will see that 2 of the 3 tests currently fail.

9.3.2. Creating and Implementing a CacheLoader

The objective of the cache loader is to load data on cache misses on the GemFire server. `The load()` method is called when the `region.get()` operation can't find the value in the cache. The value returned from the cache loader is put into the cache, and returned to the `get()` operation.

1. The definition of the `BookMasterCacheLoader` class is provided in the `com.gopivotal.bookshop.buslogic` package. Open this class file and add the code in the `load()` method to return a new `BookMaster` instance. It can either be explicitly generated in this method or by a helper class that you create. Ordinarily this would be loaded from an external data source such as a JDBC data source.
2. Open the `serverCache.xml` file and add the necessary configuration to register the `BookMasterCacheLoader` with the `BookMaster` region.
3. Stop and re-start the server. It should now be running with the newly registered `BookMasterCacheLoader`.

> **Note**
>
> To stop the servers, you can use a similar command to the above start command.

```
gfsh run --file=serverStop.gf
```

4. Re-run the `ServerEventsTests` unit tests and note that the first test (`testCacheLoader()` now passes).

9.3.3. Creating and Implementing a CacheWriter

In this section, you will implement a `CacheWriter` that performs validation of new entries. If you recall, any new entries that are created will fire the `beforeCreate()` event method. One of the benefits of having this method called before the actual insert is that we can perform validation to ensure entries meet our desired expectation.

1. To begin, refer to your `ServerEventsTests` unit test. The last two tests are designed to assert the correct behavior of your `ValidatingCacheWriter` implementation.

   

   Note

   In reality, the `testValidatingCacheWriterSuccess()` method would pass even if there was no `CacheWriter` registered as you've already seen. The main purpose of this method is to ensure that a correct insert does NOT generate an error.

2. Now, open the `ValidatingCacheWriter` class in the `com.gopivotal.bookshop.buslogic` package. Notice that the `beforeCreate()` method is left open for you to implement the functionality. Also note that there is a method called `validateNewValue()` where the logic is performed to determine a valid entry. Take a moment to examine this code. You've had some experience already with querying from the client. This is an example of a query being performed on the server. As you can see, a prospective book is considered valid if no other entry exists having the same `itemNumber` value.
3. Now, return to the `beforeCreate()` method and implement the appropriate functionality to obtain the correct value, call the validate method and throw a `CacheWriterException` if the book is considered invalid.
4. Return to the `serverCache.xml` file and add the necessary configuration to register the `ValidatingCacheWriter` with the `BookMaster` region.
5. Save your work, stop and re-start the server.
6. Verify you've correctly implemented and registered the `ValidatingCacheWriter` by re-running the `ServerEventsTests` JUnit test. All tests should now pass.

Congratulations!! You have completed this lab.
