Chapter 7. Custom Partitioned Regions

7.1. Introduction

In this lab, you will gain hands-on experience with creating a custom partition resolver.

Concepts you will gain experience with:

- Creating a custom partition scheme in GemFire
- Implementing a custom partition resolver

Estimated completion time: 30 minutes

7.2. 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*)).

7.3. Detailed Instructions

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

7.3.1. Enabling Co-located regions

Recall that in order to ensure related data ends up on the same member for partitioned regions, one key step is causing the buckets to be aligned between related regions.

(`TODO-01`): Open the `serverCache.xml` file and add the appropriate configuration to the `BookOrder` region to ensure bucket assignments are aligned with the `Customer` region.

7.3.2. Custom Partitioning

To create a Custom Partitioning scheme for GemFire, we need to implement the `PartitionResolver` interface. We want to partition on `customerId`, which is a common field shared between `Customer` objects and `BookOrder` objects. We will need to make this an attribute of the key as well as the `orderId`. We have been using `Integer` representing just the `orderId` as the key up until this point, so we will need to wrap both the `orderId` and `customerId` in a `OrderKey` class.

1. Open the OrderKey class
    a. (`TODO-02a`) Notice that it contains the key (orderNumber) and the customerNumber, which will be used for partitioning.
    b. (`TODO-02b`) Notice that the `hashCode()` and `equals()` methods are based on the `orderNumber` part of the key and not the `customerNumber`. The addition of the `customerNumber` is for the `PartitionResolver` to use, not to impact the 'equality' of entries.
2. Create the PartitionResolver
    a. (`TODO-03`) Open the class `CustomerPartitionResolver` and implement the `getRoutingObject()` method.
    b. Using the `EntryOperation`, return the part of the key that will be used for ensuring that `BookOrder` objects related to a given `Customer` land in the same bucket.
    c. Notice the `close()` and `getName()` methods. `The close()` method is a callback method required by the `CacheCallback` interface. In this case, there's nothing to do when the cache closes. Similarly, the `getName()` method is specified by the PartitionResolver interface and offers a way to attach a name to the resolver.
3. (`TODO-04`) Modify partion.xml
    a. Open `xml/serverCache.xml` file
    b. Notice that for the sake of this lab, we've reduced the number of buckets to `5`. That means any entries will land in one of these 5 buckets (0-4) depending on the key value used.
    c. Add the appropriate attributes to the `BookOrder` Region to have `com.gopivotal.bookshop.domain.CustomerPartitionResolver` declared as the Partition Resolver.
    d. Start the locator
    e. Start server1 and verify the configuration. If the first server fails to start, it's likely due to incorrectly specifying the partition resolver. In addition, you'll need to make sure that the partition resolver is on the classpath.

Tip

As with the prior lab, you will need to explicitly include the option: `--classpath=../target/classes/` to the server start commands. This is not only so the special function for listing partitioned region buckets is

available but also because you are registering a custom PartitionResolver that must be on the server's classpath.

f. Once server 1 starts properly, start servers 2 and 3.

4. Run the System test

a. (`TODO-05`) Open the `DataLoader` class under `com.gopivotal.bookshop.buslogic` package and observe the `populateCustomers()` method and the `populateBookOrders()` method. Notice that for `cust1` with customer number `5598`, there is a corresponding book order (`17600`) that has been created as evidenced by `key1` in the `populateBookOrders()` method. Notice a similar case for `cust3`.

b. (`TODO-06`) Run `DataLoader`. This will load `Customer` objects and `BookOrder` objects into their respective regions. If your PartitionResolver has been correctly implemented, `Customer` objects and related `BookOrder` objects should end up on the same member because they will each use the same bucket number for storage.

c. (`TODO-07`) Verify the distribution by using the `PRBFunctionExecutor` (under the `com.gopivotal.training.prb` package) to list the bucket distribution.

d. After exectution completes, look at the console output and observe two things:

i. The bucket numbers for the `Customer` region and `BookOrder` regions are on the same member. If they are not, go back and double check your `serverCache.xml` configuration and ensure that you've correctly specified co-location.

ii. That the data appears be aligned as evidenced by the fact that there is one `Customer` entry in each bucket and that there appears to be one `BookOrder` entry for bucket `3` and `4`. Refer to the table below to see how the keys align to buckets. If you DON'T see the described alignment, go back and double check your `PartitionResolver` implementation.

Table 7.1. Keys aligning with bucket numbers (assuming total buckets = 5)

Key (customerNumber)	Bucket Number
5598	3
5542	2
6024	4

e. Return to gfsh (re-connect to the locator if necessary) and stop all servers and the locator using the **shutdown** command.

Congratulations!! You have completed this lab.