Chapter 13. Writing Server Side Functions

13.1. Introduction

In this lab, you will gain hands-on experience with writing and registering a GemFire function. A test client will invoke the function on several GemFire servers to extract customer details for a given city.

Concepts you will gain experience with:

- Developing a GemFire function
- Register functions using XML configuration
- Execute the function on multiple servers
- Invoke the function from a client application

Estimated completion time: 30 minutes

13.2. 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*)).

13.3. 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.

13.3.1. Develop the Function

Function execution allows you to move function behavior to the application member hosting the data. In this exercise, you develop a function to perform a sum a specified field of all entries for a given region. This will offer an opportunity to explore one of the use cases functions can be used for in GemFire. For simplicity, this 'generic' summing function, we'll assume that the type used for summing is `java.lang.Float`.

To perform this task, open the `server-functions` project in the STS.

1. Open the `GenericSumFunction` class in the `com.gopivotal.bookshop.buslogic` package. Notice that this class extends `FunctionAdapter` class and also implements the `Declarable` interface.
2. You will implement the execute() function using the following steps.
    a. (`TODO-01`) Enforce that the `FunctionContext` is an instance of `RegionFunctionContext`. It's important that this be done as this function is only designed to be run via the `onRegion(..)` method calls. You might throw a `FunctionException` if this prerequisite isn't met. Assuming the `FunctionContext` is an instance of `RegionFunctionContext`, cast up to an instance of `RegionFunctionContext`.
    b. (`TODO-02`) Use the FunctionContext to get the argument passed from the client. This will represent the field on the `PdxInstance` for which the summing operation will be performed.
    c. (`TODO-03`) Use the `PartitionRegionHelper` class to get all the local region data. We'll later add additional configuraiton to ensure that the local data obtained is strictly primary region data. Also initialize an instance of `BigDecimal` as this will be used to hold the sum amount that will be returned by the function.
    d. (`TODO-04`) Iterate over the local data, which should be a collection of `PdxInstance` objects. We'll later add configuration to enable PDX Serialization and to enforce that data read on the server side remains serialized.
    e. (`TODO-05`) Use the argument provided to de-serialize (extract) the field that will be used for summing. Initially, extract an `Object`. In a successive step, make sure the field is an instance of `Float` as you will be making that assumption. If it an instance of `Float`, add this value to the current sum amount.
    f. (`TODO-06`) Once all entries have been processed, send the final sum back to the caller. Since you will be only sending one result for this execution, make sure that you signal that you are completed sending results.
3. (`TODO-07`) Add an overridden method that will enforce that only primary buckets be considered when processing local data.

> **Tip**
>
> If in doubt, go back and review the slides around the Function API. It will help if you annotate the method with `@Override` so that you can be sure you are correctly overriding the method.

13.3.2. Register the Function and Start Servers

You'll need to register the function you just wrote with the server cache configuration.

1. (`TODO-08`) Next, open the `serverCache.xml`file and add the necessary configuration to enable PDX Serialization using the `ReflectionBasedAutoSerializer`.
2. (`TODO-09`) Add the appropriate configuration to register the function
3. (`TODO-10`) Start the locator and 2 servers using similar approach as in the prior few labs. In particular, you will need to use the `--classpath=../target/classes` option. Next, execute `OrderLoader` class found in the `com.gopivotal.bookshop.buslogic` package in STS. This will load the Order Region with data.

13.3.3. Calling the Function

Finally, you will test the function using the `SummingFunctionTest` JUnit test found in `src/test/java`.

1. In the next lab exercise, you will focus on the writing the code to execute the function and process the results. For now, you will just worry about executing the function and verifying the results.
2. (`TODO-11`) Go ahead and run the program from the STS. You see that the test passes.
3. Stop the servers and locator.

Congratulations!! You have completed this lab.
