Testing Reactor Applications

Module Objectives

In this module, you will learn:

- Strategies for testing Reactor Applications
- How to test Flows
- How to test MapReduce jobs
- Adding unit tests in maven

Strategies in Testing Applications (1 of 2)

The Reactor comes with a convenient way to unit test your Applications

The base for these tests is **ReactorTestBase**, which is packaged separately from the API in its own artifact because it depends on the Reactor's runtime classes

Include it in your test dependencies in one of two ways:

- include all JAR files in the `lib` directory of the Continuuity Reactor SDK installation, or
- include the `continuuity-test` artifact in your Maven test dependencies (see the `pom.xml` file of the *WordCount* example)

Strategies in Testing Applications (2 of 2)

Notes:

- For building an application, you only need to include the Reactor API in your dependencies
- For testing, however, you need the Reactor run-time
- To build your test case, extend the `ReactorTestBase` class

Strategies in Testing Flows (1 of 4)

Let’s write a test case for the *WordCount* example:

```
public class WordCountTest extends ReactorTestBase {
  @Test
  public void testWordCount() throws Exception {
```

First, deploy the Application, then start the Flow and the Procedure:

```
// Deploy the Application
ApplicationManager appManager = deployApplication(WordCount.class);

// Start the Flow and the Procedure
FlowManager flowManager = appManager.startFlow("WordCounter");
ProcedureManager procManager = appManager.startProcedure("RetrieveCount");
```

Strategies in Testing Flows (2 of 4)

Now that the Flow is running, send some events to the Stream:

```
// Send a few events to the Stream
StreamWriter writer = appManager.getStreamWriter("wordStream");
writer.send("hello world");
writer.send("a wonderful world");
writer.send("the world says hello");
```

To wait for all events to be processed, get a metrics observer for the last Flowlet in the pipeline (the "associator") and wait for its processed count to either reach 3 or time out after 5 seconds:

```
// Wait for the events to be processed, or at most 5 seconds
RuntimeMetrics metrics = RuntimeStats.
  getFlowletMetrics("WordCount", "WordCounter", "associator");
metrics.waitForProcessed(3, 5, TimeUnit.SECONDS);
```

Strategies in Testing Flows (3 of 4)

Start verifying that the processing was correct by obtaining a client for the Procedure, and then submitting a query for the global statistics:

```
// Call the Procedure
ProcedureClient client = procManager.getClient();

// Query global statistics
String response = client.query("getStats", Collections.EMPTY_MAP);
```

If the query fails for any reason this method will throw an exception

In case of success, the response is a JSON string; deserialize the JSON string to verify the results:

```
Map<String, String> map = new Gson().fromJson(response, stringMapType);
Assert.assertEquals("9", map.get("totalWords"));
Assert.assertEquals("6", map.get("uniqueWords"));
Assert.assertEquals(((double)42)/9,
  (double)Double.valueOf(map.get("averageLength")), 0.001);
```

Strategies in Testing Flows (4 of 4)

Then ask for the statistics of one of the words in the test events

The verification is a little more complex, because the response is a nested map, and the value types in the top-level map are not uniform:

```
  // Verify some statistics for one of the words
  response = client.query("getCount", ImmutableMap.of("word","world"));
  Map<String, Object> omap = new Gson().fromJson(response, objectMapType);
  Assert.assertEquals("world", omap.get("word"));
  Assert.assertEquals(3.0, omap.get("count"));

  // The associations are a map within the map
  Map<String, Double> assocs = (Map<String, Double>) omap.get("assocs");
  Assert.assertEquals(2.0, (double)assocs.get("hello"), 0.000001);
  Assert.assertTrue(assocs.containsKey("hello"));
}
```

Strategies in Testing MapReduce Jobs

Similar to *Strategies in Testing Flows*

The `TrafficAnalyticsTest` class should extend from `ReactorTestBase`:

```
public class TrafficAnalyticsTest extends ReactorTestBase {
  @Test
  public void test() throws Exception {
```

The `TrafficAnalytics` application can be deployed using the `deployApplication` method from the `ReactorTestBase` class:

```
// Deploy an Application
ApplicationManager appManager = deployApplication(TrafficAnalyticsApp.class);
```

Strategies in Testing MapReduce Jobs

The MapReduce job reads from the `logEventTable` DataSet

The data in the `logEventTable` should be populated by running the `RequestCountFlow` and sending the data to the `logEventStream` Stream:

```
FlowManager flowManager = appManager.startFlow("RequestCountFlow");

// Send data to the Stream
sendData(appManager, now);

// Wait for the last Flowlet to process 3 events or at most 5 seconds
RuntimeMetrics metrics = RuntimeStats.
    getFlowletMetrics("TrafficAnalytics", "RequestCountFlow", "collector");
metrics.waitForProcessed(3, 5, TimeUnit.SECONDS);
```

Strategies in Testing MapReduce Jobs

Start the MapReduce job and wait for a maximum of 60 seconds:

```
// Start the MapReduce job.
MapReduceManager mrManager = appManager.startMapReduce("RequestCountMapReduce");
mrManager.waitForFinish(60, TimeUnit.SECONDS);
```

Verify that the MapReduce job was run correctly by obtaining a client for the Procedure, and then submitting a query for the counts:

```
ProcedureClient client = procedureManager.getClient();

// Verify the query.
String response = client.query("getCounts", Collections.<String, String>emptyMap());

// Deserialize the JSON string.
Map<Long, Integer> result = GSON.
    fromJson(response, new TypeToken<Map<Long, Integer>>(){}.getType());
Assert.assertEquals(2, result.size());
```

The assertion will verify that the correct result was received

Strategies in Testing

Notes

- Complete source code and tests can be found in the SDK's `TrafficAnalytics` (`examples/TrafficAnalytics/`).
- Many of the SDK examples come with unit tests
- The `pom.xml` files of the examples show how to include unit tests

---

Strategies in Testing: Adding Unit Tests With Maven

```
<dependencies>  <!-- From the WordCount example -->
    <dependency>
         <groupId>com.continuuity</groupId>
         <artifactId>continuuity-test</artifactId>
         <version>${continuuity.reactor.version}</version>
         <scope>test</scope>
         <exclusions>
              <exclusion>
                  <groupId>asm</groupId>
                  <artifactId>asm</artifactId>
              </exclusion>
         </exclusions>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    . . .
</dependencies>
```

Module Summary

You should now know:

- Strategies for testing Reactor Applications
- How to test a Flow
- How to test MapReduce jobs
- Adding unit tests in maven

Module Completed