Testing and Debugging Reactor Applications

Exercise Objectives

In this exercise, you will:

- Implement testing in a Reactor Application
- Hook up a debugger to a Reactor, and step through the code

Note: the test given here requires that the example project have the natural language toolkit included in the project; if not, modify the test

---

Setting Up `SentimentAnalysisTest.java`

Add these imports:

```
import java.util.Collections;
import com.google.common.collect.ImmutableSet;
```

The test framework can be summarized as:

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

      ... // Insert test code here: Part 1

    } finally {
      clear();
    }
  }
}
```

Part 1: Setting Up Testing

Add this test code in the `try` block:

```
ApplicationManager appManager = deployApplication(SentimentAnalysisApp.class);

// Starts the Flow
FlowManager flowManager = appManager.startFlow("analysis");

// Write messages to the Stream and Flow (Part 2)
try {
  ...
}

// Start Procedure and verify (Parts 3 and 4)
ProcedureManager procedureManager = appManager.startProcedure("sentiment-query");
try {
  ...
}

TimeUnit.SECONDS.sleep(1);
```

Part 2: Writing Messages to the Stream

```
// Write messages to the Stream and Flow
try {
  StreamWriter streamWriter = appManager.getStreamWriter("sentence");
  streamWriter.send("i love the movie");
  streamWriter.send("i hate the movie");
  streamWriter.send("i am neutral towards the movie");
  streamWriter.send("i am happy that I got this working.");

  // Wait for the last Flowlet to process all tokens
  RuntimeMetrics countMetrics =
    RuntimeStats.getFlowletMetrics("SentimentAnalysisApp", "analysis", "update");
  countMetrics.waitForProcessed(4, 15, TimeUnit.SECONDS);
} finally {
  flowManager.stop();
}
```

Part 3: Start Procedure and Verify

```
// Start Procedure and verify
ProcedureManager procedureManager = appManager.startProcedure("sentiment-query");
try {
  String response = procedureManager.getClient().query("aggregates",
    Collections.<String, String>emptyMap());

  // Verify the aggregates
  Map<String, Long> result = new Gson().fromJson(response, new
    TypeToken<Map<String, Long>>(){}.getType());
  Assert.assertEquals(2, result.get("positive").intValue());
  Assert.assertEquals(1, result.get("negative").intValue());
  Assert.assertEquals(1, result.get("neutral").intValue());
```

Part 4: Start Procedure and Verify

```
  // Verify retrieval of sentiments
  response = procedureManager.getClient().query("sentiments",
    ImmutableMap.of("sentiment", "positive"));
  result = new Gson().fromJson(response, new TypeToken<Map<String, Long>>(){}.getType());
  Assert.assertEquals(ImmutableSet.of("i love the movie",
    "i am happy that I got this working."), result.keySet());

  response = procedureManager.getClient().query("sentiments",
    ImmutableMap.of("sentiment", "negative"));
  result = new Gson().fromJson(response, new TypeToken<Map<String, Long>>(){}.getType());
  Assert.assertEquals(ImmutableSet.of("i hate the movie"), result.keySet());

  response = procedureManager.getClient().query("sentiments",
    ImmutableMap.of("sentiment", "neutral"));
  result = new Gson().fromJson(response, new TypeToken<Map<String, Long>>(){}.getType());
  Assert.assertEquals(ImmutableSet.of("i am neutral towards the movie"), result.keySet());
} finally {
  procedureManager.stop();
}
```

Include NLTK Before Building Package

- For these tests to work, you will need to complete the SentimentAnalysis Application to include the NLTK (natural language toolkit) as described in an earlier example
- Stop the existing Flows and Procedures, and with Reactor running, build the package and watch as the tests are run
- Build using `mvn clean package`
- To build without tests, use `mvn clean package -DskipTests`

Debugging: Setup

- Open the `pom.xml` in IntelliJ
- From the IntelliJ toolbar, select `Run->Edit Configurations`
- Click `+` and choose `Remote Configuration`
- Create a debug configuration by entering a name; for example, *Continuuity*
- Set the host name; for example, `localhost`
- Set the debugging port as `5005` in the Port field
- Save the configuration using the OK button

Debugging: Running

Start Reactor in debugging mode; if already running, use the `restart` option:

```
$ ./bin/reactor.sh restart --enable-debug
```

- To run the debugger, select `Run->Debug->Continuuity`
- Set a breakpoint in a code block
- For example, just after the definition of the variable `text` in the `process` method in the `Normalization` Flowlet
- Start the Flow, send in a sentence and control will stop at the breakpoint
- Check the value of `text` and see that it matches what you sent

---

Exercise Summary

You should now be able to:

- Implement testing in a Reactor Application
- Hook up a debugger to a Reactor, and step through the code

Exercise Completed

Chapter Index