How to Write DUnit/Integration tests using geode's JUnit Rules

This articles describes how to write geode Dunit tests using Junit rules.

Use ClusterStartupRule

This rule would ease the steps to start up locator/server in specific VMs with specified gemfire properties. The rule will take care of tearing down and cleaning up the VMs and file systems after tests are done so you don't need to worry about closing anything.

Below is a simple Example of using this rule with default properties:

```
@Category(DistributedTest.class)
public class ExampleTest {

  @Rule
  public ClusterStartupRule cluster = new ClusterStartupRule(); //See Note 1

  @Test
  public void simpleUsage() throws Exception {
    // start up a default locator in vm0
    MemberVM locator = cluster.startLocatorVM(0); //See Note 2

    // start up a default server in vm1, joining the locator
    MemberVM server = cluster.startServerVM(1, locator.getPort()); // see Note 3

    // you can use these attributes of the locator/server
    locator.getName(); // by default this would locator-0
    locator.getPort(); // these are the random ports by default.
    locator.getJmxPort();
    locator.getHttpPort();
    locator.getWorkingDir(); // this should be dunit/vm0 unless the rule is constructed using temp folder.

    // you can also do code invocation inside the vms
    locator.invoke(()->{
      // access the locator started in this VM
      InternalLocator internalLocator = ClusterStartupRule.getLocator();
      // operations and assertions here
    });

    server.invoke(()->{
      // access the cache and server in this VM
      InternalCache cache = ClusterStartupRule.getCache();
      CacheServer cacheServer = LocatorServerStartupRule.serverStarter.getServer();
      // operations and assertions here.
    });
  }
}
```

1. This is the most common way to create the rule. There are two more variations of it

   ```
   // this will use a temporary folder for the working dir of the locator/server created by this rule.
   // use this if you want to examine the content of the workingdir of the server/locator and do not
   // want it to be contaminated with dunit test launcher's own logs.
   public ClusterStartupRule cluster = new ClusterStartupRule().withTempWorkingDir();

   // Or
   // this will have the server/locators logs go into the log file instead of go into the console.
    public ClusterStartupRule cluster = new ClusterStartupRule().withLogFile();

   //or both.
   ```

2. When you start up a locator/server, you can also pass in a properties object that would configure the server/locator

```
Properties locatorProps = new Properties();
locatorProps.setProperty(ConfigurationProperties.GROUPS, "group1,group2");
locatorProps.setProperty(ConfigurationProperties.HTTP_SERVICE_PORT, "8080");
locatorProps.setProperty(ConfigurationProperties.ANY_PROPERTIES, value); // set any gemfire properties
that are in ConfigurationProperties
MemberVM locator = cluster.startLocatorVM(0, locatorProps);

Properties serverProps = new Properties();// set any gemfire properties that are in
ConfigurationProperties
locatorProps.setProperty(ConfigurationProperties.ANY_PROPERTIES, value);
MemberVM server = cluster.startServer(1, serverProps, locator.getPort);
```

Use LocatorStarterRule

This rule starts up a locator in the current JUnit VM instead of in a dunit vm. It's useful in integration tests. Here is a simple example of it:

```
@Rule
public LocatorStarterRule locatorRule = new LocatorStarterRule() // simplest way to create the rule, you can
call one or more of the following to configure the rule
    .withProperty(ConfigurationProperties.ANY_PROPERITES, value) // configure the locator with a single property
    .withProperties(properties) // configure the locator with a property object
    .withJMXManager() // start the locator with JMX manager, this is the default behavior for locator, so even
if you don't call this, a jmxManager will be started for you
    .withConnectionToLocator() // connect this locator with other locators.
    .withSecurityManager(SimpleTestSecurityManager.class) // a convenient way to start the locator with a
security manager, same effect as a .withProperty call.
    .withAutoStart(); // this will start the locator before executing any test code.

@Test
public void test() throws Exception {
   // if the locator is started, then we can use it to get these attributes:
   locatorRule.getName(); // by default this would be locator, if not configured by the properties
   locatorRule.getPort(); // these are the random ports by default if not configured by the properties
   locatorRule.getJmxPort();
   locatorRule.getHttpPort();
   InternalLocaotr locator = locatorRule.getLocator();

   // if the locator is not auto started, you can start the locator by calling
   locatorRule.startLocator();

   // operations and assertions here.
}
```

Use ServerStarterRule

This rule starts up a server in the current JUnit VM instead of in a dunit vm. It's useful in integration tests. Here is a simple example of it:

```
@Rule
public ServerStarterRule serverRule = new ServerStarterRule() // simplest way to create the rule, you can call
one or more of the following to configure the rule
    .withProperty(ConfigurationProperties.ANY_PROPERITES, value) // configure the locator with a single property
    .withProperties(properties) // configure the locator with a property object
    .withJMXManager() // start the server with JMX manager
    .withConnectionToLocator() // connect this server with another locator.
    .withSecurityManager(SimpleTestSecurityManager.class) // a convenient way to start the locator with a
security manager, same effect as a .withProperty call.
        .withEmbeddedLocator() // start an embedded locator on this server
        .withPDXPersistent() // with pdx persistent
        .withRestService() // start the rest service on this server
        .withAutoStart(); // this will start the server before executing any test code.
    .withRegion(REGION_SHORTCUT, regionName);// this will create the region before executing any test code

@Test
public void test() throws Exception {
   // if the server is started, then we can use it to get these attributes:
   serverRule.getName(); // by default this would be server, if not configured by the properties
   serverRule.getPort(); // these are the random ports by default if not configured by the properties
   serverRule.getJmxPort();
   serverRule.getHttpPort();
   InternalCache cache = serverRule.getCache();

   // if the locator is not auto started, you can start the locator by calling
   serverRule.startServer();

   // operations and assertions here.
}
```

Use GfshCommandRule

This is the rule that's useful if you would like to test out some gfsh commands and verify the output. To use this, you will need to have a jmxManager running (either a locator or a server with JmxManager started).

Here is some example of how you would want to to use the rules:

This example auto starts a locator and then use the gfshRule to connect to the locator in the @Before

```
@Category(IntegrationTest.class)
public class ExampleTest {
  @Rule
  public LocatorStarterRule locator = new LocatorStarterRule().withAutoStart();

  @Rule
  public GfshCommandRule gfshRule = new GfshCommandRule();

  @Before
  public void before() throws Exception {
    gfshRule.connectAndVerify(locator);
  }

  @Test
  public void simpleUsage() throws Exception {
    // gfshRule already connect, ready to execute some command and verify output.
    gfshRule.executeAndAssertThat("list members")
                .statusIsOK()
                .containsOutput("blah,blah");
  }
}
```

This example starts a server with JMXManager and then use the gfshRule to connect to the jmxManager of the server

```
@Category(IntegrationTest.class)
public class ExampleTest {
  @Rule
  public ServerStarterRule serverRule = new ServerStarterRule().withJMXManager().withRegion(RegionShortcut.
REPLICATE, "testRegion");

  @Rule
  public GfshCommandRule gfshRule = new GfshCommandRule();

  @Before
  public void before() throws Exception {
    gfshRule.connectAndVerify(serverRule.getJmxPort(), GfshCommandRule.PortType.jmxManger);
  }

  @Test
  public void simpleUsage() throws Exception {
    // gfshRule already connect, ready to execute some command
    String result = gfshRule.execute("list members");
        // examine the result and do some assertions
  }
}
```

In the previous two examples, the GfshShellConnectionRule is created with empty parameter, so you need to manually call the connect() method to connect to a jmx manager in the @Before or in the body of the test. If you want to auto connect to a give jmx manager, you need to create the rule with a PortProvider and PortType. Here is an example of it:

```
@Category(IntegrationTest.class)
public class ExampleTest {
  public LocatorStarterRule locator = new LocatorStarterRule().withAutoStart();
  public GfshCommandRule gfshRule = new GfshCommandRule(locator::getJmxPort, GfshShellConnectionRule.PortType.
jmxManger);

  @Rule
  public RuleChain ruleChain = RuleChain.outerRule(locator).around(gfshRule);

  @Test
  public void simpleUsage() throws Exception {
    // gfshRule already connect, ready to execute some command
    gfshRule.executeAndAssertThat("list members").statusIsOK();
  }
}
```

Related articles

- Releasing Apache Geode
- Running a Geode cluster in docker
- Adding REST Cluster Management Operations
- Creating a persistent cache service
- Publishing Geode Metrics to External Monitoring Systems