Getting Started With Quartz Scheduler Locality API

Terracotta Quartz Scheduler Where introduces an Enterprise feature that allows jobs and triggers to be run on specified Terracotta clients instead of randomly chosen ones. Quartz Scheduler Where is a locality API that can be used to direct jobs to nodes having enough resources or the relevant data for successfully executing those jobs. Also introduced with Quartz 2.0 is a more readable fluent-interface approach to creating and scheduling jobs and triggers.

This section shows you how to install, configure, and use the locality API.

Installation

To access the Quartz Scheduler Locality API in a standard installation, include quartz-terracotta-ee-2.0.0-SNAPSHOT.jar in your classpath. This jar is found under the `${TERRACOTTA_HOME}/quartz` directory.

For DSO installation, see this.

Configuration

1. Set the Enterprise Terracotta JobStore in `quartz.properties`:

   ```
   org.quartz.jobStore.class = org.terracotta.quartz.EnterpriseTerracottaJobStore
   ```

2. Create or choose an implementation of `org.quartz.spi.InstanceIdGenerator` to generate node IDs to be used in the locality configuration. You can choose from one of the provided classes:
   - org.quartz.simpl.HostnameInstanceIdGenerator – Returns the hostname as the instanceId.
   - org.quartz.simpl.SimpleInstanceIdGenerator – Returns a mix of hostname and timestamp (to be sure we have a unique name), default!
   - org.quartz.simpl.SystemPropertyInstanceIdGenerator – Returns the value of the `org.quartz.scheduler.instanceId` system property of the JVM. Available with Quartz 2.0 or higher.
     The generator class must be specified in `quartz.properties`:

     ```
     org.quartz.jobStore.class = org.terracotta.quartz.EnterpriseTerracottaJobStore
     ```

3. Configure the trigger and nodes (Terracotta clients) in `quartzLocality.properties`. For example:

   ```
   # Set up node groups that can be referenced from application code:
   org.quartz.locality.nodeGroup.slowJobs = node0, node3
   org.quartz.locality.nodeGroup.group1 = node1, node2
   org.quartz.locality.nodeGroup.allNodes = node0, node1, node2, node3

   # Set up trigger groups whose triggers fire only on nodes in the specified node groups:
   org.quartz.locality.nodeGroup.slowJobs.triggerGroups = slowTriggers
   org.quartz.locality.nodeGroup.group1.triggerGroups = fastTriggers
   ```

   Any job with a trigger in the "slowTriggers" group will fire only on nodes in the group "slowJobs", while jobs with triggers in "fastTriggers" will fire only on nodes in the group "group1".
4. `quartzLocality.properties` must be on the classpath, the same as `quartz.properties`.

Use in Application Code

The following example is a code snippet that uses Quartz Scheduler Where to create a locality-aware trigger and a locality-aware job.

```
import static org.quartz.JobBuilder.newJob;
import static org.quartz.TriggerBuilder.newTrigger;
import static org.quartz.locality.LocalityTriggerBuilder.localTrigger;
import static org.quartz.locality.NodeSpecBuilder.node;
import static org.quartz.locality.constraint.NodeGroupConstraint.partOfNodeGroup;

import org.quartz.JobDetail;
import org.quartz.locality.LocalityTrigger;
// Other required imports ...

// Using the new fluent interface, create a locality-aware job that can be run on any node from nodeGroup
"group1" that runs a Linux OS:
LocalityJobDetail jobDetail =
         localJob(
             newJob(myJob.class)
                 .withIdentity("myJob")
                 .storeDurably(true)
                 .build())
             .where(
                 node()
                     .is(partOfNodeGroup("group1")
                     .is(OsConstraint.LINUX)))
             .build();

// Create a trigger for myJob using the new fluent interface:
Trigger trigger =
         localTrigger(newTrigger()
             .forJob("myJob")
             .withIdentity("myTrigger")
             .withSchedule(simpleSchedule()
                    .withIntervalInSeconds(10)
                    .withRepeatCount(2))
             .build();

// Create a second job:
JobDetail jobDetail2 =
         localJob(
             newJob(myJob2.class)
                 .withIdentity("myJob2")
                 .storeDurably(true)
                 .build())
             .build();

// Create a locality-aware trigger for myJob2 using the new fluent interface:
LocalityTrigger trigger2 =
         localTrigger(newTrigger()
             .forJob("myJob2")
             .withIdentity("myTrigger2")
             .where(
                 node()
                     .is(partOfNodeGroup("allNodes")))  // fire on any node in allNodes that ...
                     .has(atLeastAvailable(100, MemoryConstraint.Unit.MB))  // has at least 100MB in free
memory available.
             .build();
```

This example showed how memory and node-group constraints are used to route locality-aware triggers and jobs. trigger2, for example, is set to fire myJob2 on a node in a specific group ("allNodes") with a specified minimum amount of free memory. A constraint based on operating system (Linux, Microsoft Windows, Apple OSX, and Oracle Solaris) is also available.

Locality With the Standard Quartz Scheduler API

It is also possible to add locality to jobs and triggers created with the standard Quartz Scheduler API by assigning the triggers to a trigger group specified in `quartzLocality.properties`.

DSO Installation

DSO users must install tim-quartz-2.0-ee. First, add the TIM to your Terracotta configuration file (`tc-config.xml` by default):

```
...
<clients>
  ...
  <modules>
    <module name="tim-quartz-2.0-ee" />
    ...
  </modules>
  ...
</clients>
...
```

To install the TIMs declared in the Terracotta configuration file, use the following command (UNIX/Linux):

```
${TERRACOTTA_HOME}/bin/tim-get.sh install-for /path/to/tc-config.xml
```

Use `tim-get.bat` with Microsoft Windows.