Cúram 8.2

Cúram JMX Developer Guide

Note

Before using this information and the product it supports, read the information in Notices on page 19

Edition

Contents

Chapter 1 Developing with Cúram JMX

Use the Cúram Java™ Management Extensions infrastructure to simplify the instrumentation of code and the collection of application operational data by using the JMX standard. Cúram JMX facilitates the creation of custom MBeans and their registration in the correct MBean server corresponding to the runtime environment.

1.1 Overview

The purpose of this guide is to describe how Cúram JMX can be extended with custom MBeans. This guide is intended for application developers interested in providing custom operational data via Cúram JMX.

1.2 What is Cúram JMX

Cúram Java Management Extensions (JMX) is an infrastructure that simplifies the instrumentation of code and the collection of application operational data using the JMX standard. Cúram JMX facilitates the creation of custom MBeans and their registration in the correct MBean server corresponding to the runtime environment.

Using Cúram JMX to Expose Application Statistics

In order to collect and expose custom application statistics an MBean needs to be created, the application code instrumented to provide the statistics and the JMX infrastructure configuration modified to initialize the newly created MBean.

Developing the Custom MBean

Cúram JMX supports only Open MBeans. An Open MBean is an MBean where the types of attributes and of operation parameters and return values are built using a small set of predefined Java ® classes. A multidimensional array of any one of these classes or their corresponding primitive types is also allowed.

These acceptable Java data types are listed below.

- java.lang.Void
- java.lang.Boolean
- java.lang.Character
- java.lang.Byte
- java.lang.Short
- java.lang.Integer
- java.lang.Long

- java.lang.Float
- java.lang.Double
- java.lang.String
- java.math.BigDecimal
- java.math.BigInteger
- java.util.Date
- javax.management.ObjectName
- javax.management.openmbean.CompositeData
- javax.management.openmbean.TabularData

The Interface

This example shows the definition of an interface for an MBean that returns some statistics in a tabular format and supports the reset of its statistics. It is not compulsory to declare the `reset` method. Declare it only if the MBean can or is allowed to reset its statistics.

When an administrative request is made to reset all JMX statistics the JMX infrastructure inspects the MBean definition and if it finds the `reset` operation it invokes it.

```
import javax.management.openmbean.OpenDataException;
import javax.management.openmbean.TabularData;

public interface MyStatsMBean {
  /**
   * MBean attribute holding the statistics.
   */
  TabularData getStats() throws OpenDataException;
  /**
   * This method is invoked by the JMX infrastructure when
   * a request is made to reset the JMX statistics.
   */
  void reset();
}
```

Figure 1: A custom MBean interface

End the name of the interface in StatsMBean It is important for all MBeans that export statistics to have an interface class name that ends in `StatsMBean`.

The Implementation

There are several options for creating your own MBean. We provide a set of abstract classes that can be subclassed to create MBeans for different types of work.

The table below provides information on each type and when it could be used.

Table 1: MBean abstract classes

MBean Abstract Class	Usage
curam.util.jmx.CuramMBeanAbstract	This is the super class of all Cúram MBeans. Use this class when full control is needed and any of the other abstract classes are not sufficient.

MBean Abstract Class	Usage
curam.util.jmx.mbean.GenericTabularStats	Generic MBean used for exposing tabular statistics. Use this class to implement a simple MBean that exposes a single set of generic, predefined invocation statistics.
curam.util.jmx.mbean.GenericNameValueStats	Generic MBean used for exposing a list of name-value items. Use this class to implement a simple MBean that exposes a set of statistics that are naturally organized as name-value pairs.
curam.util.jmx.mbean.GenericKeyedPoolStats	Generic MBean used for exposing usage statistics for keyed pools. A keyed pool is a pool that can cache multiple items for the same key.

Using CuramMBeanAbstract

Create an implementation of your interface that inherits from `curam.util.jmx.CuramMBeanAbstract`.

See The Interface. To make it easier further on, derive the name of this class from the name of the implemented MBean by removing the `MBean` suffix. This super class provides the MBean with access to the application configuration via the execution context and it facilitates the handling of changes in application configuration data that might be of interest to the MBean.

```
package com.mytest;

import java.util.logging.Level;
import java.util.logging.Logger;

import javax.management.openmbean.
                       CompositeDataSupport;
import javax.management.openmbean.
                       CompositeType;
import javax.management.openmbean.
                       OpenDataException;
import javax.management.openmbean.OpenType;
import javax.management.openmbean.SimpleType;
import javax.management.openmbean.TabularData;
import javax.management.openmbean.
                       TabularDataSupport;
import javax.management.openmbean.TabularType;

import curam.util.jmx.CuramMBeanAbstract;

public class MyStats extends CuramMBeanAbstract
      implements MyStatsMBean {
private static final Logger log = Logger
     .getLogger(MyStats.class.getName());

private static final OpenType[] kItemTypes
                        = new OpenType[] {
  SimpleType.STRING,
  SimpleType.LONG,
};

private static final String[] kItemNames
                = new String[] {
```

```
     "Item",
     "Execution time(ms)"};

  private static final String[] kItemDescriptions
                 = new String[] {
     "The name of the item",
     "The execution time in milliseconds"};

  private static TabularType stTabularType;

  private static CompositeType stRowType;

  private static MyStats instance;

  static {
    try {
      stRowType = new CompositeType(
          "MyStatsType", "My statistics",
           kItemNames, kItemDescriptions, kItemTypes);
      stTabularType = new TabularType(
          "MyStats", "My statistics",
          stRowType, new String[] { kItemNames[0]});
    } catch (Exception e) {
      log.log(Level.SEVERE,
           "Failed to create the open types.", e);
    }
  }

  public MyStats() {
    super();
    instance = this;
  }

  /* (non-Javadoc)
   * @see com.mytest.MyStatsMBean#getStats()
   */
  public TabularData getStats()
        throws OpenDataException {
    if (stRowType == null
             || stTabularType == null) {
      return null;
    }
    TabularDataSupport sup = new TabularDataSupport(
             stTabularType);

    // sample stats
    Object[] values = new Object[2];
    // ...
    // get the values
    // ...
    CompositeDataSupport cd =
                          new CompositeDataSupport(
      stRowType, kItemNames, values);
    sup.put(cd);
    return sup;
  }

  /**
```

```
 * This method is invoked from instrumented code
 * to update the statistics for
 * <code>item</code>.
 * @param item the item to update statistics for
 * @param executionTime the execution time
 */
public static void updateStats(
           String item, long executionTime) {
  if(instance != null) {
    instance._updateStats(item, executionTime);
  }
}

/**
 * This method is invoked by the JMX infrastructure when
 * a request is made to reset the JMX statistics.
 * @see com.mytest.MyStatsMBean#reset()
 */
public void reset() {
 // ...
 // reset the statistics
 //...
}

private void _updateStats(String item, long executionTime) {
  // ...
  // update the items average execution time
  // ...
}
}
```

Figure 2: A custom MBean implementation

More complex MBeans that require dynamic configuration parameters or support per user data collection can override or utilize the provided protected methods in `curam.util.jmx.CuramMBeanAbstract`.

Using GenericTabularStats

This abstract class can be used to develop an MBean for exposing a single set of tabular statistics.

The statistics names will be as follows:

- **Target -** the monitored target (for example a URL or a method name)
- **Invocations -** the number of invocations made to the monitored target
- **Elapsed time(ms) -** the average elapsed time in milliseconds for an invocation of the monitored target
- **Std deviation elapsed time(ms) -** the standard deviation of the elapsed time in milliseconds
- **Min elapsed time(ms) -** the minimum elapsed time in milliseconds
- **Max elapsed time(ms) -** the maximum elapsed time in milliseconds
- **Errors -** the number of times the invocation failed

Use this class in the following manner:

1. Create your MBean interface and class as described in The Interface
2. Make your MBean class a subclass of this class
3. Use the methods provided by this class to push statistics data to your MBean

Usage should be as follows where `MyGenericTabularStats` is the implementation of your MBean:

```
public class MyGenericTabularStats extends GenericTabularStats
implements
  MyGenericTabularStatsMBean {

  private static volatile MyGenericTabularStats instance;

  public MyGenericTabularStats() {

    super();
    instance = this;
  }

  public static void addStats(String target, long elapsedTime,
boolean error) {

    if (instance != null) {
      instance.addStatistics(target, elapsedTime, error);
    }
  }
}
```

Figure 3: Usage example

Using GenericNameValueStats

Use this class to implement a simple MBean that exposes a set of statistics that are naturally organized as name-value pairs.

Use this class in the following manner:

1. Create your MBean interface and class as described in The Interface
2. Make your MBean class a subclass of this class
3. Use the methods provided by this class to push statistics data to your MBean

Usage should be as follows where `MyGenericNameValueStats` is the implementation of your MBean:

```
 public class MyGenericNameValueStats extends
GenericNameValueStats implements
   MyGenericNameValueStatsMBean {

   private static volatile MyGenericNameValueStats instance;

   public MyGenericNameValueStats() {

     super();
     instance = this;
   }

   public static void addOrUpdateStats(String name, Object
value) {
      if (instance != null) {
        instance.addOrUpdateStatistics(name, value);
      }
    }
}
```

Figure 4: Usage example

Using GenericKeyedPoolStats

Generic MBean used for exposing usage statistics for keyed pools. A keyed pool is a pool that can cache multiple items for the same key.

The statistics names will be as follows:

- **Key** - the key
- **Active** - the average number of active/borrowed items in the pool for items with this key
- **Size** - the average number of items in the pool for this key

Use this class in the following manner:

1. Create your MBean interface and class as described in The Interface
2. Make your MBean class a subclass of this class
3. Use the methods provided by this class to push statistics data to your MBean

Usage should be as follows where `MyGenericKeyedPoolStats` is the implementation of your MBean:

```
 public class MyGenericKeyedPoolStats extends
GenericKeyedPoolStats implements
   MyGenericKeyedPoolStatsMBean {

   private static volatile MyGenericKeyedPoolStats instance;

   public MyGenericKeyedPoolStats() {

     super();
     instance = this;
   }

   public static void addStats(String key, long active, boolean
size) {

     if (instance != null) {
       instance.addStatistics(key, active, size);
     }
   }
 }
```

Figure 5: Usage example

Using curam.util.jmx.NumericalCounterStatisticsAggregator

This example shows how to use `curam.util.jmx.NumericalCounterStatisticsAggregator` and `curam.util.jmx.NumericalCounterStatistics` to calculate and make available various arithmetic values for a numerical counter (average, minimum, maximum and standard deviation).

```
import curam.util.jmx.NumericCounterStatisticsAggregator;
...
    /** Elapsed time statistics. */
    private NumericCounterStatisticsAggregator
                              elapsedTimeStats;

    /** Error counter. */
    private AtomicLong errors;

    /** Constructor. */
    MyClass() {
      super();
      errors = new AtomicLong(0);
      elapsedTimeStats =
           new NumericCounterStatisticsAggregator();
    }

    /**
     * Get the number of invocations.
     *
     * @return the number of invocations.
     */
    long getInvocations() {
      return this.elapsedTimeStats
                   .getNumberOfSamples();
    }

    /**
     * Get elapsed time statistics.
     *
     * @return elapsed time statistics.
     */
    NumericCounterStatistics getElapsedTimeStats() {
      return elapsedTimeStats.getAll();
    }

    /**
     * Get error counter.
     *
     * @return error counter.
     */
    long getErrors() {
      return errors.get();
    }

 /**
  * Add a statistics sample.
  *
  * @param elapsedTime the elapsed time.
  * @param error true if invocation ended in error.
  */
 void addStats(long elapsedTime, boolean error) {
  boolean reset = this.elapsedTimeStats
                           .add(elapsedTime);
  if(reset) {
   // Long.MAX_VALUE overflow
   errors.set(0);
  } else if(error){
   if(errors.incrementAndGet() < 0) {
    // Long.MAX_VALUE overflow
    this.elapsedTimeStats.reset();
   }
  }
 }
...
```

Figure 6: Using NumericalCounterStatisticsAggregator and NumericCounterStatistics

Updating the Configuration of Cúram JMX

The next step is to add the new MBean to the list of MBeans to be instantiated by the JMX infrastructure.

Depending on where the MBean is located (Web or Enterprise Java Beans (EJB) container) modify the corresponding application property:

- `curam.jmx.configured_mbeans_ejb` – for MBeans residing in the EJB container
- `curam.jmx.configured_mbeans_web` – for MBeans residing in the Web container

See Cúram JMX Configuration Guide for more details.

Instrumenting Application Code

The application code needs to be instrumented to push data to the custom MBean. In order to minimize overhead check that JMX monitoring is turned on before pushing statistics to the MBean.

```
public void instrumentedMethod() {
 long startTime = System.currentTimeMillis();
 ...
 // do processing
 ...
 // check that JMX monitoring is enabled before
 // updating the MBean
 if(CuramJMXUtil.isJmxMonitoringEnabled()) {
  MyStats.updateStats("item",
       System.currentTimeMillis() - startTime);
 }
}
```

Figure 7: Pushing elapsed time statistics to the custom MBean

Another possible instrumentation is to add execution statistics to the existing JMX services such as transaction tracing and in-flight transaction data.

```
public Result instrumentedMethod(String param) {
 try {
  return CuramJMXUtil.runAndRecord(new Callable<Result>(){
 public Result call() throws Exception {
  return myMethod(param);
 }}, "myMethod",
 TransactionInfo.getProgramUser());
 } catch (CuramJMXUtil.CallableException e) {
  throw new AppRuntimeException(e.getCause());
 }
}
```

Figure 8: Pushing execution statistics to existing JMX services

Notices

Privacy policy

Trademarks