SolrSnmp

SNMP Monitoring of Solr Statistics (through JBoss)

There are likely several ways of doing this, including finding some libraries (open source or otherwise) that expose MBean attributes through SNMP. However, one of the easiest methods is to use the snmp-adapter.sar that comes with JBoss 4.0.5.GA and above. This page shows how to add an SNMP-adapter that exposes one attribute, the "standard" QueryHandler's request count.

Setup

The snmp-adapter.sar can be found in the server/all/deploy directory. You'll want to copy it into your server instance's deploy directory.

Private Enterprise Number (OID)

In order to use SNMP monitoring, you will need to have your own Private Enterprise Number (OID). You can apply for one at IANA's website, and if it's approved it will show up in the IANA's master list.

Configuring the exposed attributes

Using your OID (let's say it's 12345678 for XYZ Corp.), you'll want to edit the attributes.xml included in the snmp-adapter.sar to add something like the following:

```
<mbean name="com.xyz:name=SolrStats" oid-prefix=".1.3.6.1.4.1.12345678.1">
  <attribute name="QueryHandler_Standard_Requests" oid=".1"/>
</mbean>
```

Creating the MBean

You'll want to create an MBean interface that looks something like this:

```
public interface SolrStatsMBean {
  public long getQueryHandler_Standard_Requests();
}
```

And for the implementation, I have chosen to make most of the accessor methods static because in my implementation other pieces of code access the stats as well. The implementation looks like this:

```
public class SolrStats implements SolrStatsMBean {
  protected static CoreContainer coreContainer;
  protected static final String DEFAULT_CORE_NAME = "";

  public static void initialize(CoreContainer container) {
    coreContainer = container;
  }

  public long getQueryHandler_Standard_Requests() {
    return SolrStats.getSolrInfoMBeanValue(SolrInfoMBean.Category.QUERYHANDLER, "standard", "requests");
  }

  private static String getSolrInfoMBeanValue(SolrInfoMBean.Category category, String entryName, String
statName) {
    Map<String, SolrInfoMBean> registry = coreContainer.getCore(DEFAULT_CORE_NAME).getInfoRegistry();
    for (Map.Entry<String, SolrInfoMBean> entry : registry.entrySet()) {
      String key = entry.getKey();
      SolrInfoMBean solrInfoMBean = entry.getValue();
      if ((solrInfoMBean.getCategory() != category) ||
          (!entryName.equals(key.trim()))) {
        continue;
      }
      NamedList<?> nl = solrInfoMBean.getStatistics();
      if ((nl != null) && (nl.size() > 0)) {
        for (int i = 0; i < nl.size(); i++) {
          if (nl.getName(i).equals(statName)) {
            return nl.getVal(i).toString();
          }
        }
      }
    }
    return null;
  }

  public static String getSolrInfoMBeanValueString(SolrInfoMBean.Category category, String entryName, String
statName) {
    String s = getSolrInfoMBeanValue(category, entryName, statName);
    return (s == null) ? "" : s;
  }

  public static long getSolrInfoMBeanValueLong(SolrInfoMBean.Category category, String entryName, String
statName) {
    String num = getSolrInfoMBeanValue(category, entryName, statName);
    try {
      return (num != null) ? Long.parseLong(num) : -1L;
    }
    catch (NumberFormatException e) {
      return -2L;
    }
  }
}
```

Note that I've stripped out the imports and a bunch of exception-handling code in order to keep this code section as brief as possible, and names have been changed to protect the innocent. You'll want to handle potential null returns, etc.

Initializing and Registering the MBean

Solr uses a filter to direct its traffic (SolrDispatchFilter), and inside that filter is a member variable that contains the Solr cores (of type org.apache.solr.core.CoreContainer). Because that member variable is not exposed in any way, you'll have to extend the filter in order to get to the CoreContainer. You should end up with code that looks like this:

```
public class SolrWithSNMPDispatchFilter extends SolrDispatchFilter {
  private List<ObjectName> mbeanNames = new ArrayList<ObjectName>();

  @Override
  public void init(FilterConfig config) throws ServletException {
    super.init(config);
    SolrStats.initialize(super.cores);
    registerMBean("com.xyz:name=SolrStats", new SolrStats());
  }

  @Override
  public void destroy() {
    deregisterMBeans();
    super.destroy();
  }

  private void registerMBean(String objectName, Object bean) throws ServletException {
    MBeanServer mbs = MBeanServerLocator.locateJBoss();
    try {
      ObjectName on = new ObjectName(objectName);
      mbs.registerMBean(bean, on);
      mbeanNames.add(on);
    }
    catch (NotCompliantMBeanException e) { throw new ServletException(e); }
    catch (MBeanRegistrationException e) { throw new ServletException(e); }
    catch (InstanceAlreadyExistsException e) { throw new ServletException(e); }
    catch (MalformedObjectNameException e) { throw new ServletException(e); }
  }

  private void deregisterMBeans() {
    try {
      MBeanServer mbs = MBeanServerLocator.locateJBoss();
      for (ObjectName on : mbeanNames) {
        mbs.unregisterMBean(on);
      }
    }
    catch (MBeanRegistrationException e) { e.printStackTrace(); }
    catch (InstanceNotFoundException e) { e.printStackTrace(); }
  }
}
```

Again, you may not want to do your error-handling that way, but you get the picture. You'll note where we use the JNDI name the same way we specified it in attributes.xml.

In order to make sure the filter is used, you must deploy your classes (MBean classes and filter) along with Solr. For now we'll assume you've done that by placing them inside the solr.war.

You must also edit the web.xml in the solr.war file to change the SolrRequestFilter to use your extended filter:

```
<filter>
  <filter-name>SolrRequestFilter</filter-name>
  <filter-class>com.xyz.web.filter.SolrWithSNMPDispatchFilter</filter-class>
</filter>
```

Testing the exposed SNMP attributes

Once that's done, simply deploy your new war file into JBoss and start it up. A good way to test SNMP locally is by using NET-SNMP. With that installed, you can run commands like this:

```
./snmpwalk.exe -v 1 -c public 127.0.0.1:1161 .1.3.6.1.4.1.12345678.1
```

Note the use of the Private Enterprise Number in the OID! A command like that (or a specific value using "snmpget") should yield results like this if everything's working:

```
SNMPv2-SMI::enterprises.12345678.1.1 = Gauge32: 4
```

Troubleshooting

One problem I've noticed with the SNMP adapter SAR is that although it seems to be able to figure out how to expose "String" and "long" types properly, it does not expose "double" types properly (and it gives no errors about them either, which is troublesome). That means that attributes such as the standard query-handler's avgTimePerRequest statistic should probably be exposed as Strings. Of course, if you can get the double values working, please feel free to edit this wiki and enlighten us all.

Another thing I've noticed is that when an SNMP attribute cannot be exposed, the NET-SNMP snmpwalk will quit walking upon the first variable that gives it cannot retrieve, however the following attributes may still be exposed properly.