Chapter 1. JOnAS and JMX, registering and manipulating MBeans

Table of Contents

By Jonny Way.

1.1. Introduction

JMX (Java Management eXtensions) is an API for managing, among other things, J2EE applications. JOnAS (version 4 and above) integrates the MX4J open-source JMX server and registers a number of MBeans. The web-based JonasAdmin application acts as a JMX client, enabling viewing and manipulation of these MBeans.

It maybe desirable for an application to expose its own MBeans via the JMX server to allow application management (using, for example, MC4J). JOnAS currently does not provide a prebuilt method for registering MBeans with its JMX server. The intent of this document is to illustrate one method of registering application-specific MBeans with the JOnAS JMX server based on the m-let service.

1.2. ServletContextListener

The basic task of registering an MBean with the JOnAS JMX server is accomplished by the following implementation of the ServletContextListener [http://java.sun.com/j2ee/1.4/docs/api/javax/servlet/ServletContextListener.html] interface. This implementation reads a number of MLet files, which specify the MBeans to be registered, and attempts to register those beans during the web application context initialization.

```
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Iterator;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.StringTokenizer;

import javax.management.InstanceAlreadyExistsException;
import javax.management.InstanceNotFoundException;
import javax.management.ReflectionException;
import javax.management.MBeanServer;
import javax.management.MBeanException;
import javax.management.MBeanRegistrationException;
import javax.management.MBeanServerFactory;
import javax.management.MalformedObjectNameException;
import javax.management.NotCompliantMBeanException;
import javax.management.ObjectInstance;
import javax.management.ObjectName;
import javax.management.loading.MLet;
import javax.management.loading.MLetMBean;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContext;
import javax.servlet.ServletContextListener;

import org.apache.log4j.Logger;
```

```
/**
 * ServletContextListener designed to register JMX MBeans into
 * an existing JMX server when the application starts up. The
 * MBeans to be loaded are specified in the MLet files whose
 * names are given in the servlet context parameter with the name mletFiles,
 * in a semi-colon delminated list (although this is not really
 * needed as multiple mlets can be specifyed in one file, it might
 * be useful). Note that the filename are relative to the WEB-INF
 * directory of the servlet context being listened to.
 *
 * Note, that this listener should precede (in web.xml) any other that depend
 * on the MBeans being registered.
 *
 * Note that the MBean registration is sensitive to classloader issues. For
 * example, when registering the MBeans in the JMX server provided by
 * the Jonas application server any libraries required by the MBeans need
 * to be in the central lib directory (lib/ext).
 *
 *
 * @author Jonny Wray
 *
 */
public class MBeanRegistrationListener implements ServletContextListener {

        private static final String MLET_DOMAIN = "MBeanRegistrationListener";
        private static final String MLET_BEAN_NAME = MLET_DOMAIN+":Name=MLet";
        private static final String MLETFILE_INITPARAM_NAME = "mletFiles";
        private static final Logger log = Logger.getLogger(MBeanRegistrationListener.class);


        private MBeanServer lookForExistingServer(){
                List mbeanServers = MBeanServerFactory.findMBeanServer(null);
                if(mbeanServers != null && mbeanServers.size() > 0){
                        return (MBeanServer)mbeanServers.get(0);
                }
                return null;
        }

        private MBeanServer getMBeanServer(){
                MBeanServer server = lookForExistingServer();
                if(server == null){
                        server = MBeanServerFactory.createMBeanServer(MLET_DOMAIN);
                }
                return server;
        }


        public void contextDestroyed(ServletContextEvent arg0) {
                log.info("Destroy event");
                // Anything that needs to be done here on deregistering of the
                // web application?
        }


        private List getMLetURLs(String filenames){
                List urls = new ArrayList();
                StringTokenizer tok =
                        new StringTokenizer(filenames, ";");
                while(tok.hasMoreTokens()){
                        String filename = tok.nextToken();
                        URL configURL =
 Thread.currentThread().getContextClassLoader().getResource(filename);
                        if(configURL == null){
                                log.error("Cound not load MLet file resource from "+filename
+" using current thread context classloader");
                        }
                        else{
                                urls.add(configURL);
                        }
                }
                return urls;
        }

        private List getMLetURLs(ServletContext context, String filenames){
                List urls = new ArrayList();
                StringTokenizer tok =
                        new StringTokenizer(filenames, ";");
                while(tok.hasMoreTokens()){
                        String filename = "/WEB-INF/"+tok.nextToken();
                        URL configURL = null;
                        try {
                                configURL = context.getResource(filename);
                        }
```

```
                        catch (MalformedURLException e) {
                                log.error("URL for "+filename+" is malformed", e);
                        }
                        if(configURL == null){
                                log.error("Cound not find MLet file resource from "+filename
+" in servlet context");
                        }
                        else{
                                urls.add(configURL);
                        }
                }
                return urls;
        }

        /**
         * Dynamically register the MBeans specified in the list
         * of MLet files (relative to /WEB-INF/) specified in servlet context parameter
         * mletFiles as a semi-colon deliminated list of file names.
         *
         * The algorithm looks for already running JMX servers and uses
         * the first it comes across. If no servers are running, then
         * it creates one.
         *
         * Note, the interface does not define any exceptions to be
         * thrown. Currently, any exceptions thrown during registration
         * are logged at error level and then ignored. This seems
         * reasonable, as these may or may not be a fatal event. In
         * this way the registration process reports its failure and
         * the application context initialization continues.
         */
        public void contextInitialized(ServletContextEvent arg0) {
                log.info("Initializing event");
                String filenames =
 arg0.getServletContext().getInitParameter(MLETFILE_INITPARAM_NAME);
                if(filenames != null && filenames.length() > 0){
                        MBeanServer server = getMBeanServer();
                        if(server != null){
                                try{
                                        ObjectName name = new ObjectName(MLET_BEAN_NAME);
                                        if(!server.isRegistered(name)){
                                                log.info("Creating new MLetMBean for dynamic
 registration");
                                                MLetMBean mletService = new MLet();
                                                server.registerMBean(mletService, name);
                                        }
                                        List urls = getMLetURLs(arg0.getServletContext(),
 filenames);
                                        for(int i=0;i < urls.size();i++){
                                                URL url = (URL)urls.get(i);
                                                try {
                                                        log.info("Registering MBeans from
 MLet file "+url);
                                                        Set loadedMBeans =
 (Set)server.invoke(name, "getMBeansFromURL",
                                                                        new Object[]{url},
 new String[]{URL.class.getName()});


 processRegisteredMBeans(loadedMBeans);
                                                }
                                                catch (InstanceNotFoundException e) {
                                                        log.error("Unable to register MBeans
 from MLet file "+url, e);
                                                }
                                                catch (MBeanException e) {
                                                        log.error("Unable to register MBeans
 from MLet file "+url, e);
                                                }
                                                catch (ReflectionException e) {
                                                        log.error("Unable to register MBeans
 from MLet file "+url, e);
                                                }
                                        }
                                }
                                catch(MalformedObjectNameException e){
                                        log.error("Unable to register the MLetMBean", e);
                                }
                                catch(NotCompliantMBeanException e){
                                        log.error("Unable to register the MLetMBean", e);
                                }
                                catch(MBeanRegistrationException e){
                                        log.error("Unable to register the MLetMBean", e);
                                }
```

```
                                catch(InstanceAlreadyExistsException e){
                                        log.error("Unable to register the MLetMBean", e);
                                }
                        }
                        else{
                                log.error("MBeanServer not found and could not be created.
 Not registering MBeans.");
                        }
                }
                else{
                        log.error("No mletFiles servlet context parameter found.");
                }
        }

        private void processRegisteredMBeans(Set loadedMBeans) {
                log.debug("Loaded beans: "+loadedMBeans.size());
                Iterator it = loadedMBeans.iterator();
                while(it.hasNext()){
                        Object o = it.next();
                        if(o instanceof ObjectInstance){
                                ObjectInstance inst = (ObjectInstance)o;
                                log.info("Registered: "+inst.getObjectName());
                        }
                        else if(o instanceof Throwable){
                                Throwable err = (Throwable)o;
                                log.error("Error registering MBeans", err);
                        }
                }
        }

}
```

1.3. Configuration

In order to use the above ServletContextListener, it must be configured in the web.xml of the web application that wants to register the MBeans. For example, the following lines added to the web.xml will result in the registeration of the MBeans specified in the application.mlet file under the WEB-INF directory. Multiple MLet files can be specified in a comma-separated list.

```
        <context-param>
                <param-name>mletFiles</param-name>
                <param-value>application.mlet</param-value>
        </context-param>

        <listener>
                <listener-class>net.fiveprime.jmx.MBeanRegistrationListener</listener-class>
        </listener>
```

An example MLet file to load an extension (detailed below) of the HibernateService MBean is:

```
<mlet code="ConfigurableHibernateService"
                name="HibernateService:Name=HibernateService"
                archive="mbeans.jar">
        <arg type="java.lang.String" value="hibernate.cfg.xml">
        <arg type="java.lang.String" value="hibernate/HibernateSessionFactory">
        <arg type="java.lang.String" value="DefaultDS">
</mlet>
```

1.4. Library Dependences

Registration of MBeans results in their construction by the JMX server. As such, any classes the MBean is dependent on must be available to the JMX server, in lib/ext.

1.5. HibernateService Extension

The Hibernate distribution provides an implementation of HibernateServiceMBean in the class HibernateService. In the MLet file above, an extension of this class is specified that allows the HibernateService to be configured from an external file, such as the standard hibernate.cfg.xml file. There are a number of situations where it is desirable to use the Hibernate mapped classes outside of Jonas running a JMX server. This allows the Hibernate mapping files and properties to be specified in one place and used in multiple situations. If this is not needed, then the HibernateService class can be used directly.

```
import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import net.sf.hibernate.HibernateException;
import net.sf.hibernate.jmx.HibernateService;

import org.apache.commons.digester.Digester;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.xml.sax.SAXException;
/**
 * Extension of the HibernateService class to add configuration
 * ability from a Hibernate XML configuration file.
 *
 * @author Jonny Wray
 *
 */
public class ConfigurableHibernateService extends HibernateService {

        private static Log log = LogFactory.getLog(ConfigurableHibernateService.class);

        /**
         * Configure this HibernateService from an XML file
         *
         * @param filename The Hibernate XML configuration file, for example
 hibernate.cfg.xml
         * @param jndiName The JNDI name that the session factory will be registered under
         * @param datasourceName The name of the datasource used by the session factory
         * @throws HibernateException If there's a problem reading the configuration file
         */
        public ConfigurableHibernateService(String filename, String jndiName, String
 datasourceName)
                throws HibernateException{

                init(filename, jndiName, datasourceName);
                start();
        }

        private void init(String filename, String jndiName, String datasourceName) throws
 HibernateException {
                if(log.isDebugEnabled()){
                        log.debug("Configuring Hibernate JMX MBean with filename "+filename+
                                ", JNDI name "+jndiName+" and datasource "+datasourceName);
                }
                try{
                        URL url = this.getClass().getClassLoader().getResource(filename);
                        Digester mappingDigester = configureMappingDigester();
                        List results = (List)mappingDigester.parse(url.openStream());
                        Iterator it = results.iterator();
                        while(it.hasNext()){
                                StringBuffer buffer = (StringBuffer)it.next();
                                addMapResource(buffer.toString());
                                log.debug("Adding mapping resource "+buffer.toString());
                        }

                        Digester propertyDigester = configurePropertyDigester();
                        Map resultMap = (Map)propertyDigester.parse(url.openStream());
                        it = resultMap.keySet().iterator();
                        while(it.hasNext()){
```

```
                                String key = (String)it.next();
                                String value = (String)resultMap.get(key);
                                setProperty("hibernate."+key, value);
                                log.debug("Adding property ("+key+","+value+")");
                        }
                        setJndiName(jndiName);
                        setDatasource(datasourceName);
                }
                catch(IOException e){
                        throw new HibernateException(e);
                }
                catch(SAXException e){
                        throw new HibernateException(e);
                }
        }

        private Digester configureMappingDigester(){
                Digester digester = new Digester();
                digester.setClassLoader(this.getClass().getClassLoader());
                digester.setValidating(false);
                digester.addObjectCreate("hibernate-configuration/session-factory",
 ArrayList.class);

                digester.addObjectCreate("hibernate-configuration/session-factory/mapping",
 StringBuffer.class);
                digester.addCallMethod("hibernate-configuration/session-factory/mapping",
 "append", 1);
                digester.addCallParam("hibernate-configuration/session-factory/mapping", 0,
 "resource");
                digester.addSetNext("hibernate-configuration/session-factory/mapping",
 "add");

                return digester;
        }

        private Digester configurePropertyDigester(){
                Digester digester = new Digester();
                digester.setClassLoader(this.getClass().getClassLoader());
                digester.setValidating(false);
                digester.addObjectCreate("hibernate-configuration/session-factory",
 HashMap.class);

                digester.addCallMethod("hibernate-configuration/session-factory/property",
 "put", 2);
                digester.addCallParam("hibernate-configuration/session-factory/property", 0,
 "name");
                digester.addCallParam("hibernate-configuration/session-factory/property",
 1);

                return digester;
        }
}
```