dynamic-proxy-to-access-mbean

Example dynamic-proxy-to-access-mbean can be browsed at https://github.com/apache/tomee/tree/master/examples/dynamic-proxy-to-access-mbean

Help us document this example! Click the blue pencil icon in the upper right to edit this page.

Example

Acessing MBean is something simple through the JMX API but it is often technical and not very interesting.

This example simplify this work simply doing it generically in a proxy.

So from an user side you simple declare an interface to access your MBeans.

Note: the example implementation uses a local MBeanServer but enhancing the example API it is easy to imagine a remote connection with user/password if needed.

ObjectName API (annotation)

Simply an annotation to get the object

```
package org.superbiz.dynamic.mbean;

import java.lang.annotation.Retention;
import java.lang.annotation.Target;

import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;

@Target({TYPE, METHOD})
@Retention(RUNTIME)
public @interface ObjectName {
    String value();

    // for remote usage only
    String url() default "";
    String user() default "";
    String password() default "";
}
```

DynamicMBeanHandler (thr proxy implementation)

```
package org.superbiz.dynamic.mbean;

import javax.annotation.PreDestroy;
import javax.management.Attribute;
import javax.management.MBeanAttributeInfo;
import javax.management.MBeanInfo;
import javax.management.MBeanServer;
import javax.management.MBeanServerConnection;
import javax.management.ObjectName;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;
import java.io.IOException;
import java.lang.management.ManagementFactory;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

/**
 * Need a @PreDestroy method to disconnect the remote host when used in remote mode.
 */
public class DynamicMBeanHandler implements InvocationHandler {
    private final Map<Method, ConnectionInfo> infos = new ConcurrentHashMap<Method,
ConnectionInfo>();

    @Override public Object invoke(Object proxy, Method method, Object[] args) throws
Throwable {
        final String methodName = method.getName();
        if (method.getDeclaringClass().equals(Object.class) && "toString".equals
(methodName)) {
            return getClass().getSimpleName() + " Proxy";
        }
        if (method.getAnnotation(PreDestroy.class) != null) {
            return destroy();
        }

        final ConnectionInfo info = getConnectionInfo(method);
        final MBeanInfo infos = info.getMBeanInfo();
        if (methodName.startsWith("set") && methodName.length() > 3 && args != null &&
args.length == 1
            && (Void.TYPE.equals(method.getReturnType()) || Void.class.equals(method
.getReturnType()))) {
            final String attributeName = attributeName(infos, methodName, method
```

```
.getParameterTypes()[0]);
            info.setAttribute(new Attribute(attributeName, args[0]));
            return null;
        } else if (methodName.startsWith("get") && (args == null || args.length == 0)
&& methodName.length() > 3) {
            final String attributeName = attributeName(infos, methodName, method
.getReturnType());
            return info.getAttribute(attributeName);
        }
        // operation
        return info.invoke(methodName, args, getSignature(method));
    }

    public Object destroy() {
        for (ConnectionInfo info : infos.values()) {
            info.clean();
        }
        infos.clear();
        return null;
    }

    private String[] getSignature(Method method) {
        String[] args = new String[method.getParameterTypes().length];
        for (int i = 0; i < method.getParameterTypes().length; i++) {
            args[i] = method.getParameterTypes()[i].getName();
        }
        return args; // note: null should often work...
    }

    private String attributeName(MBeanInfo infos, String methodName, Class<?> type) {
        String found = null;
        String foundBackUp = null; // without checking the type
        final String attributeName = methodName.substring(3, methodName.length());
        final String lowerName = Character.toLowerCase(methodName.charAt(3)) +
methodName.substring(4, methodName.length());

        for (MBeanAttributeInfo attribute : infos.getAttributes()) {
            final String name = attribute.getName();
            if (attributeName.equals(name)) {
                foundBackUp = attributeName;
                if (attribute.getType().equals(type.getName())) {
                    found = name;
                }
            } else if (found == null && ((lowerName.equals(name) && !attributeName
.equals(name))
                || lowerName.equalsIgnoreCase(name))) {
                foundBackUp = name;
                if (attribute.getType().equals(type.getName())) {
                    found = name;
                }
            }
```

```
        }

        if (found == null && foundBackUp == null) {
            throw new UnsupportedOperationException("cannot find attribute " +
attributeName);
        }

        if (found != null) {
            return found;
        }
        return foundBackUp;
    }

    private synchronized ConnectionInfo getConnectionInfo(Method method) throws
Exception {
        if (!infos.containsKey(method)) {
            synchronized (infos) {
                if (!infos.containsKey(method)) { // double check for synchro
                    org.superbiz.dynamic.mbean.ObjectName on = method.getAnnotation
(org.superbiz.dynamic.mbean.ObjectName.class);
                    if (on == null) {
                        Class<?> current = method.getDeclaringClass();
                        do {
                            on = method.getDeclaringClass().getAnnotation(org.
superbiz.dynamic.mbean.ObjectName.class);
                            current = current.getSuperclass();
                        } while (on == null && current != null);
                        if (on == null) {
                            throw new UnsupportedOperationException("class or method
should define the objectName to use for invocation: " + method.toGenericString());
                        }
                    }
                    final ConnectionInfo info;
                    if (on.url().isEmpty()) {
                        info = new LocalConnectionInfo();
                        ((LocalConnectionInfo) info).server = ManagementFactory
.getPlatformMBeanServer(); // could use an id...
                    } else {
                        info = new RemoteConnectionInfo();
                        final Map<String, String[]> environment = new HashMap<String,
String[]>();
                        if (!on.user().isEmpty()) {
                            environment.put(JMXConnector.CREDENTIALS, new String[]{
on.user(), on.password() });
                        }
                        // ((RemoteConnectionInfo) info).connector =
JMXConnectorFactory.newJMXConnector(new JMXServiceURL(on.url()), environment);
                        ((RemoteConnectionInfo) info).connector = JMXConnectorFactory
.connect(new JMXServiceURL(on.url()), environment);

                    }
```

```
                    info.objectName = new ObjectName(on.value());

                    infos.put(method, info);
                }
            }
        }
        return infos.get(method);
    }

    private abstract static class ConnectionInfo {
        protected ObjectName objectName;

        public abstract void setAttribute(Attribute attribute) throws Exception;
        public abstract Object getAttribute(String attribute) throws Exception;
        public abstract Object invoke(String operationName, Object params[], String
signature[]) throws Exception;
        public abstract MBeanInfo getMBeanInfo() throws Exception;
        public abstract void clean();
    }

    private static class LocalConnectionInfo extends ConnectionInfo {
        private MBeanServer server;

        @Override public void setAttribute(Attribute attribute) throws Exception {
            server.setAttribute(objectName, attribute);
        }

        @Override public Object getAttribute(String attribute) throws Exception {
            return server.getAttribute(objectName, attribute);
        }

        @Override
        public Object invoke(String operationName, Object[] params, String[]
signature) throws Exception {
            return server.invoke(objectName, operationName, params, signature);
        }

        @Override public MBeanInfo getMBeanInfo() throws Exception {
            return server.getMBeanInfo(objectName);
        }

        @Override public void clean() {
            // no-op
        }
    }

    private static class RemoteConnectionInfo extends ConnectionInfo {
        private JMXConnector connector;
        private MBeanServerConnection connection;

        private void before() throws IOException {
```

```
            connection = connector.getMBeanServerConnection();
        }

        private void after() throws IOException {
            // no-op
        }

        @Override public void setAttribute(Attribute attribute) throws Exception {
            before();
            connection.setAttribute(objectName, attribute);
            after();
        }

        @Override public Object getAttribute(String attribute) throws Exception {
            before();
            try {
                return connection.getAttribute(objectName, attribute);
            } finally {
                after();
            }
        }

        @Override
        public Object invoke(String operationName, Object[] params, String[]
signature) throws Exception {
            before();
            try {
                return connection.invoke(objectName, operationName, params, signature
);
            } finally {
                after();
            }
        }

        @Override public MBeanInfo getMBeanInfo() throws Exception {
            before();
            try {
                return connection.getMBeanInfo(objectName);
            } finally {
                after();
            }
        }

        @Override public void clean() {
            try {
                connector.close();
            } catch (IOException e) {
                // no-op
            }
        }
    }
```

```
}
```

Dynamic Proxies

DynamicMBeanClient (the dynamic JMX client)

```
package org.superbiz.dynamic.mbean;
```

```
import org.apache.openejb.api.Proxy;
import org.superbiz.dynamic.mbean.DynamicMBeanHandler;
import org.superbiz.dynamic.mbean.ObjectName;
```

```
import javax.ejb.Singleton;
```

```
/**
 * @author rmannibucau
 */
@Singleton
@Proxy(DynamicMBeanHandler.class)
@ObjectName(DynamicMBeanClient.OBJECT_NAME)
public interface DynamicMBeanClient {
    static final String OBJECT_NAME = "test:group=DynamicMBeanClientTest";
```

```
    int getCounter();
    void setCounter(int i);
    int length(String aString);
}
```

DynamicMBeanClient (the dynamic JMX client)

```
package org.superbiz.dynamic.mbean;

import org.apache.openejb.api.Proxy;

import javax.annotation.PreDestroy;
import javax.ejb.Singleton;


@Singleton
@Proxy(DynamicMBeanHandler.class)
@ObjectName(value = DynamicRemoteMBeanClient.OBJECT_NAME, url =
"service:jmx:rmi:///jndi/rmi://localhost:8243/jmxrmi")
public interface DynamicRemoteMBeanClient {
    static final String OBJECT_NAME = "test:group=DynamicMBeanClientTest";

    int getCounter();
    void setCounter(int i);
    int length(String aString);

    @PreDestroy void clean();
}
```

The MBean used for the test

SimpleMBean

```
package org.superbiz.dynamic.mbean.simple;
```

```
public interface SimpleMBean {
    int length(String s);
```

```
    int getCounter();
    void setCounter(int c);
}
```

Simple

```
package org.superbiz.dynamic.mbean.simple;
```

```
public class Simple implements SimpleMBean {
    private int counter = 0;
```

```
@Override public int length(String s) {
    if (s == null) {
        return 0;
    }
    return s.length();
}
```

```
@Override public int getCounter() {
    return counter;
}
```

```
    @Override public void setCounter(int c) {
        counter = c;
    }
}
```

DynamicMBeanClientTest (The test)

```
package org.superbiz.dynamic.mbean;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import org.superbiz.dynamic.mbean.simple.Simple;

import javax.ejb.EJB;
import javax.ejb.embeddable.EJBContainer;
import javax.management.Attribute;
import javax.management.ObjectName;
import java.lang.management.ManagementFactory;

import static junit.framework.Assert.assertEquals;

public class DynamicMBeanClientTest {
    private static ObjectName objectName;
    private static EJBContainer container;

    @EJB private DynamicMBeanClient localClient;
    @EJB private DynamicRemoteMBeanClient remoteClient;
```

```
    @BeforeClass public static void start() {
        container = EJBContainer.createEJBContainer();
    }

    @Before public void injectAndRegisterMBean() throws Exception {
        container.getContext().bind("inject", this);
        objectName = new ObjectName(DynamicMBeanClient.OBJECT_NAME);
        ManagementFactory.getPlatformMBeanServer().registerMBean(new Simple(),
objectName);
    }

    @After public void unregisterMBean() throws Exception {
        if (objectName != null) {
            ManagementFactory.getPlatformMBeanServer().unregisterMBean(objectName);
        }
    }

    @Test public void localGet() throws Exception {
        assertEquals(0, localClient.getCounter());
        ManagementFactory.getPlatformMBeanServer().setAttribute(objectName, new
Attribute("Counter", 5));
        assertEquals(5, localClient.getCounter());
    }

    @Test public void localSet() throws Exception {
        assertEquals(0, ((Integer) ManagementFactory.getPlatformMBeanServer()
.getAttribute(objectName, "Counter")).intValue());
        localClient.setCounter(8);
        assertEquals(8, ((Integer) ManagementFactory.getPlatformMBeanServer()
.getAttribute(objectName, "Counter")).intValue());
    }

    @Test public void localOperation() {
        assertEquals(7, localClient.length("openejb"));
    }

    @Test public void remoteGet() throws Exception {
        assertEquals(0, remoteClient.getCounter());
        ManagementFactory.getPlatformMBeanServer().setAttribute(objectName, new
Attribute("Counter", 5));
        assertEquals(5, remoteClient.getCounter());
    }

    @Test public void remoteSet() throws Exception {
        assertEquals(0, ((Integer) ManagementFactory.getPlatformMBeanServer()
.getAttribute(objectName, "Counter")).intValue());
        remoteClient.setCounter(8);
        assertEquals(8, ((Integer) ManagementFactory.getPlatformMBeanServer()
.getAttribute(objectName, "Counter")).intValue());
    }
```

```
    @Test public void remoteOperation() {
        assertEquals(7, remoteClient.length("openejb"));
    }

    @AfterClass public static void close() {
        if (container != null) {
            container.close();
        }
    }
}
```