ResponseTimeAspect

Purpose

The purpose of this aspect is to report arbitrary method execution response time to a JMX server.
It is easily configurable to gather the metrics on a per class, package, application, etc basis and thus report averages metrics as well.

This aspect has been demonstrated during JavaOne 2004 in the "AOP in J2EE environments with AspectWerkz" session.
Download the JavaOne 2004 slides.

Usage

In order to be able to use this aspect, you need to declare it in your **META-INF/aop.xml** file and bind the advice named **"monitor"** of type **"around"** to any method execution pointcuts (actually it can work for any pointcut type if you want to).

One JMX mbean will be created per target method/constructor/field signature by default. See the next section for more configuration about the aspect parameter **"granularity"**.

The mbeans will be registered *lazyly* in the default (local) JMX server. For use with BEA WebLogic Server 6, 7 and 8 and for other schemes, see the next section about the aspect parameter **"mbeanServer"**.

Here is a sample *META-INF/aop.xml* file:

```
<!DOCTYPE aspectwerkz PUBLIC
    "-//AspectWerkz//DTD//EN"

"http://aspectwerkz.codehaus.org/dtd/aspectw
erkz.dtd">
<aspectwerkz>
    <system id="aware-jmx">
        <!-- make use of defaults
granularity and server -->
        <aspect
class="org.codehaus.aware.jmx.ResponseTimeAs
pect">
            <advice name="monitor"
type="around"

bind-to="execution(!static *
test.jmx.ResponseTimeObject.*(..)) AND
!execution(* *..*.freeze(..))"/>
        </aspect>
  </system>
</aspectwerkz>
```

Reporting response time averages

To change the granularity of the reported metrics, which defaults to *one mbean per target method/constructor/field signature* you need to use the *aspect parameter* **"granularity"**.

This parameter will specify a class that implements the *org.codehaus.aware.jmx.ResponseTimeAspect.IGranularity* i nterface below:

```
public class ResponseTimeAspect {
    /**
     * Granularity interface to determine
which mbean to update given a specific
joinpoint signature.
     *
     */
    public static interface IGranularity {
        /**
         * Compute a suitable key given a
signature. The mbeans will be register on a
per key basis, and it is thus
         * advised to have suitable
hashcode(), equals() and toString()
implementation for it.
         *
         * @param signature
         * @return
         */
        public Object getKeyFor(Signature
signature);
    }
}
```

AWare provides an implementation which is able to compute a global average. Here is how it has been written. You can refer to AspectWerkz documentation about the *org.codehaus.aspectwerkz.joinpoint.Signature* interface to write your own:

```
public class AverageGranularity implements
IGranularity {
    public Object getKeyFor(Signature
signature) {
        // to compute a global average, just
always return the same value
        return "Average";
    }
}
```

This class is provided in AWare as org.codehaus.aware.jmx.ResponseTimeAspect.AverageGranularity. Since it is a static inner class, we will have to use it under the name "org.codehaus.aware.jmx.ResponseTimeAspect$AverageGranularity" (with a **$** sign) :

To use your custom granularity the *META-INF/aop.xml* file will look like the following. Note that we are using the ResponseTimeAspect twice, once with default granularity and once with average granularity, and thus we specify an aspect **"name"**.

```
<!DOCTYPE aspectwerkz PUBLIC
    "-//AspectWerkz//DTD//EN"

"http://aspectwerkz.codehaus.org/dtd/aspectw
erkz.dtd">
<aspectwerkz>
    <system id="aware-jmx">
        <!-- make use of defaults
granularity and server -->
        <aspect
class="org.codehaus.aware.jmx.ResponseTimeAs
pect">
            <advice name="monitor"
type="around"
```

```
bind-to="execution(!static *
test.jmx.ResponseTimeObject.*(..)) AND
!execution(* *..*.freeze(..))"/>
        </aspect>
        <!-- make use of custom granularity
-->
        <aspect
class="org.codehaus.aware.jmx.ResponseTimeAs
pect" name="MyAverageResponseTimeAspect">
            <param name="granularity"
value="org.codehaus.aware.jmx.ResponseTimeAs
pect$AverageGranularity"/>
            <advice name="monitor"
type="around"

bind-to="execution(!static *
test.jmx.ResponseTimeObject.*(..)) AND
!execution(* *..*.freeze(..))"/>
```

```
        </aspect>
  </system>
</aspectwerkz>
```

Using a specific JMX mbean server

The default AWare implementation is using the default JMX API to retrieve the local JMX server and register the mbeans:

```
public class JMXHelper {
    /**
     * Default driver interface using
javax.management.MBeanServerFactory.
     * See MX4JAspect if using it with MX4J
jmx server.
     *
     */
    public static class
StandardMBeanServerDriver implements
IMBeanServerDriver {
        public MBeanServer findMBeanServer()
throws Throwable {
            return
javax.management.MBeanServerFactory.createMB
eanServer();
        }
    }
}
```

This might not be convenient in your environment. For example in BEA WebLogic, you have several JMX server available when you are running in a cluster or in a managed server environment.

AWare allows you to use the mbean server you like.

For such use the aspect parameter **"mbeanServer"** and specify a class that implements the *org.codehaus.aware.jmx.JMXHelper.IMBeanServerDriver* interface:

```
public interface IMBeanServerDriver {
        public MBeanServer findMBeanServer()
throws Throwable;
    }
```

AWare provides a BEA Weblogic implementation of it thru **org.codehaus.aware.jmx.JMXHelper.WebLogicMBeanServerDriver**.

To use it, the *META-INF/aop.xml* will looks like:

```
<aspectwerkz>
    <system id="aware-jmx">
        <!-- make use of defaults
granularity and use WebLogic mbean server
custom lookup -->
        <aspect
class="org.codehaus.aware.jmx.ResponseTimeAs
pect">
            <param name="mbeanServer"
value="org.codehaus.aware.jmx.JMXHelper$WebL
ogicMBeanServerDriver"/>
            <advice name="monitor"
type="around"

bind-to="execution(!static *
test.jmx.ResponseTimeObject.*(..)) AND
!execution(* *..*.freeze(..))"/>
        </aspect>
  </system>
</aspectwerkz>
```