XML schema-based AOP support

Summary

If you are unable to use Java 5, or simply prefer an XML-based format, then Spring 2.0 also offers support for defining aspects using the new "aop" namespace tags. The exact same pointcut expressions and advice kinds are supported as when using the @AspectJ style

Description

Declaring an aspect

Using the schema support, an aspect is simply a regular Java object defined as a bean in your Spring application context. An aspect is declared using the <aop:aspect> element, and the backing bean is referenced using the ref attribute.

```
<bean id="adviceUsingXML" class="egovframework.rte.fdl.aop.sample.AdviceUsingXML" />
<aop:config>
        <aop:aspect ref="adviceUsingXML">
       ...
        </aop:aspect>
</aop:config>
```

The bean backing the aspect ("aBean" in this case) can of course be configured and dependency injected just like any other Spring bean.

Declaring a pointcut

The pointcut is used to designate the join points to decide when the advice is implemented. Spring AOP only supports the method execution join point for method execution. The pointcut in Spring designates the method execution point of the bean.

Next example is declares the pointcut name 'targetMethod' which corresponds to any method execution of the class that ends with the sample name of sub-packages of egovframework.rte.fdl.aop.sample. The pointcut declares within the <aop:config> element. The expression of pointcut can be used the same as the AspectJ pointcut expression language.

```
<aop:config>
        <aop:pointcut id="targetMethod"
                expression="execution(* egovframework.rte.fdl.aop.sample.*Sample.*(..))" />
</aop:config>
```

Declaring advice

An advice is the actual implementation of aspects that is injected in the join point to work. The same five advice kinds are supported as for the @AspectJ style, and they have exactly the same semantics.

Before advice

Before advice is declared inside an <aop:aspect> using the <aop:before> element.

Next is the XML examples that declares before advice. Before advice, beforeTargetMethod() before is executed before the pointcut declared as targetMethod().

```
<aop:aspect ref="adviceUsingXML">
        <aop:before pointcut-ref="targetMethod" method="beforeTargetMethod" />
</aop:aspect>
```

Next is the class that implements before advice. The beforeTargetXML() method that executes the before advice prints the class name and method name with the pointcut.

```
public class AdviceUsingXML {
    ...
    public void beforeTargetMethod(JoinPoint thisJoinPoint) {
        System.out.println("AdviceUsingXML.beforeTargetMethod executed.");

        Class clazz = thisJoinPoint.getTarget().getClass();
        String className = thisJoinPoint.getTarget().getClass().getSimpleName();
        String methodName = thisJoinPoint.getSignature().getName();

        System.out.println(className + "." + methodName + " executed.");
    }
    ...
}
```

After returning advice

After returning advice runs when a matched method execution completes normally. It is declared inside an <aop:aspect> using <aop:after-returning> element.

Next is an example that uses the After returning advice. The afterReturningTargetMethod() advice is executed after the pointcut defined as targetMethod(). The result of executing the targetMethod() pointcut is stored at retVal variable and passed.

```
<aop:aspect ref="adviceUsingXML">
    <aop:after-returning pointcut-ref="targetMethod"
                         method="afterReturningTargetMethod" returning="retVal" />
</aop:aspect>
```

Next is the class that implements after returning advice. The afterReturningTargetMethod() that executes after returning advice prints the returning value of each pointcut.

```
public class AdviceUsingXML {
    ...
    public void afterReturningTargetMethod(JoinPoint thisJoinPoint,
            Object retVal) {
        System.out.println("AdviceUsingXML.afterReturningTargetMethod executed." +
             return value is [" + retVal + "]");
    }
    ...
}
```

After throwing advice

After throwing advice executes when a matched method execution exits by throwing an exception. It is declared inside an <aop:aspect> using the <aop:after-returning> element

Next is an example that uses the After throwing advice. The afterThrowingTargetMethod() advice is executed after the pointcut defined as targetMethod(). The exception occured in the targetMethod() pointcut is stored at retVal variable and passed.

```
<aop:aspect ref="adviceUsingXML">
        <aop:after-throwing pointcut-ref="targetMethod"
                method="afterThrowingTargetMethod" throwing="exception" />
</aop:aspect>
```

Next is the class that implements After throwing advice. The afterReturningTargetMethod() that executes after throwing advice binds the exception once more for the users to distinguish easily.

```
public class AdviceUsingXML {
    ...
    public void afterThrowingTargetMethod(JoinPoint thisJoinPoint,
            Exception exception) throws Exception {
        System.out.println("AdviceUsingXML.afterThrowingTargetMethod executed.");
        System.err.println("Error", exception);
        throw new BizException("Error", exception);
    }
    ...
}
```

After (finally) advice

After (finally) advice runs however a matched method execution exits. It is declared inside the <aop:aspect> element using the <aop:after> element. After advice is

Next is an example of using after(finally) advice. The afterTargetMethod() advice is executed to any case of normal completion and exception of the pointcut declared as targetMethod(). Generally, it executes the same task as relieving the resources.

```
<aop:aspect ref="adviceUsingXML">
        <aop:after pointcut-ref="targetMethod" method="afterTargetMethod" />
</aop:aspect>
```

Next is the class that implements After (finally) advice. The afterTargetMethod() that executes After (finally) advice prints the message that indicates the execution of after advice.

```
public class AdviceUsingXML {
    ...
    public void afterTargetMethod(JoinPoint thisJoinPoint) {
        System.out.println("AdviceUsingXML.afterTargetMethod executed.");
    }
    ...
}
```

Around advice

Around advice runs "around" a matched method execution. It is declared inside the <aop:aspect> element using the <aop:around> element. The around advice is used in case that processes normal completion and exceptions. The tasks such as resource release are included.

```
<aop:aspect ref="adviceUsingXML">
        <aop:around pointcut-ref="targetMethod" method="aroundTargetMethod" />
</aop:aspect>
```

Next is the class that implements the around advice. The aroundTargetMethod() advice is the parameter that passes ProceedingJoinPoint. It executes the pointcut through calling proceed() method. It shows that the result of executing the pointcut, retVal can be returned by converting within the around advice.

```
public class AdviceUsingXML {
    ...
    public Object aroundTargetMethod(ProceedingJoinPoint thisJoinPoint)
            throws Throwable {
        System.out.println("AdviceUsingXML.aroundTargetMethod start.");
        long time1 = System.currentTimeMillis();
        Object retVal = thisJoinPoint.proceed();

        System.out.println("ProceedingJoinPoint executed. return value is ["
            + retVal + "]");
```

```
        retVal = retVal + "(modified)";
        System.out.println("return value modified to [" + retVal + "]");

        long time2 = System.currentTimeMillis();
        System.out.println("AdviceUsingXML.aroundTargetMethod end. Time("
            + (time2 - time1) + ")");
        return retVal;
    }
    ...
}
```

Implementing aspects

Use the test code to confirm normal activities of defined aspects. AdviceTest class test is done by classifying the case of normal execution and exception.

Normal execution

The testAdvice() function shows the example that the method is executed normally. The before, after returning, after finally, around advice is applied to the someMethod() of the AdviceSample class within the egovframework.rte.fdl.aop.sample package.

```
public class AdviceTest{
    @Resource(name = "adviceSample")
    AdviceSample adviceSample;

    @Test
    public void testAdvice() throws Exception {
        SampleVO vo = new SampleVO();
        ..
        String resultStr = adviceSample.someMethod(vo);

        assertEquals("someMethod executed.(modified)", resultStr);
    }
}
```

The result log of executing the test code is as following.

```
AdviceUsingXML.beforeTargetMethod executed.
AdviceSample.someMethod executed.
AdviceUsingXML.aroundTargetMethod start.
AdviceUsingXML.afterReturningTargetMethod executed. return value is [someMethod executed.]
AdviceUsingXML.afterTargetMethod executed.
ProceedingJoinPoint executed. return value is [someMethod executed.]
return value modified to [someMethod executed.(modified)]
AdviceUsingXML.aroundTargetMethod end. Time(12)
```

The order where the advice is applied is as following.

- @Before

- @Around (before executing the method)
- Method

- @AfterReturning
- @After(finally)
- @Around (after executing the method)

Note that @Around advice can change the method return value but the after returning advice can only refer to and cannot change the return value.

Other cases

testAnnotationAspectWithException() function shows examples of method errors. The someMethod() method of AnnotationAdviceSample class within the egovframework.rte.fdl.aop.sample applies before, after throwing, after finally, around advice.

```
public class AdviceTest{
    @Resource(name = "adviceSample")
    AdviceSample adviceSample;

    @Test
    public void testAdviceWithException() throws Exception {
        SampleVO vo = new SampleVO();
        // set the flag to generate exceptions
         vo.setForceException(true);
        ..
        try {
            // when the forceException flag of vo is true, it forces - / by zero situation
             resultStr = adviceSample.someMethod(vo);

            fail("this line cannot be executed because the exception is forced.");
        } catch (Exception e) {
            ..
        }
    }
}
```

The result log from executing the test code is as following:

```
AdviceUsingXML.beforeTargetMethod executed.
AdviceSample.someMethod executed.
AdviceUsingXML.aroundTargetMethod start.
AdviceUsingXML.afterThrowingTargetMethod executed.
Error.
java.lang.ArithmeticException: / by zero
...
```

The order of applying the advice in the console log output is as following:

- @Before
- @Around (before executing the method)
- Method (ArithmeticException is generated)
- @AfterThrowing
- @After(finally)

The advice declared as after is executed even if the exception occurs. The After Throwing advice can reset the error message, generate new exceptions to pass.

Reference

- Spring 2.5 Reference Documentation