

Bloomington JUG AOP Presentation

by Dan Countryman

The majority of the information was covered off of the following sites and documents:
dynaop web site: <https://dynaop.dev.java.net/nonav/release/1.0-beta/manual/index.html>
aspectwerkz web site: <http://aspectwerkz.codehaus.org/>
Aspectwerkz Dynaic AOP for Java by David Rabinowitz:
<http://www.cs.huji.ac.il/course/2003/ood/docs/tir09.ppt>
aspectj web site: <http://eclipse.org/aspectj/>

AOP Definition

An expanding of object oriented programming to include the concept of applying (weaving) expanded behavior to objects. The idea is to find modularized or crosscutting concerns, repeated code scattered throughout an application and isolate them into classes (aspects) that can then be weaved into all of the places that need that behavior .

Terminology

Aspect

An *aspect* is the AOP equivalent of a class. Just as a class contains methods and fields, an aspect contains advice and rules for applying advice.

Advice

Advice is the code portion of an aspect, more specifically the logic that replaces your crosscutting concern.

Interceptor

An *interceptor* is the first of the two types of advice we'll talk about. An interceptor simply intercepts events. Most commonly, an interceptor intercepts method invocations. In our [logging example](#), we log messages at the beginning and end of method invocations. We can use an interceptor to execute logic such as this before and after a method invocation.

Introduction (Mixin)

Introduction is the second type of advice and involves adding functionality to a class. In our logging example, we had a `Log` instance for each class type. In some AOP frameworks, we might introduce this `Log` field to our class.

The term *mixin* is commonly used in reference to introducing a mixin class. The mixin methods are added to the target class and their implementation delegates to a mixin instance.

Pointcut

Pointcut is a term used in many AOP frameworks to refer to the aspect configuration mechanism. A pointcut is a set of points in your application where advice should be applied. For example, a pointcut may pick all setter methods in a given set of classes.

1. Dynaop

Example of a crosscutting concern from dynaop doc

Logging is a crosscutting concern.

```
class Foo {  
  
    Log log = Log.getLog(Foo.class);  
  
    void foo(String s) {  
        log.log("*** invoked foo(" + s + ")");  
        // business logic...  
        log.log("*** returned: void");  
    }  
  
    void foo(int a, int b) {  
        log.log("*** invoked foo(" + a + ", " + b + ")");  
        // business logic...  
        log.log("*** returned: void");  
    }  
}  
  
class Bar {  
  
    Log log = Log.getLog(Bar.class);  
  
    String bar(String s) {  
        log.log("*** invoked bar(" + s + ")");  
        // business logic...  
        String result = ...;  
        log.log("*** returned: " + result);  
        return result;  
    }  
}
```

Types of Implementations

1. Proxy Frameworks

Examples:

- nanning <http://nanning.codehaus.org>
- dynaop
- Spring AOP <http://www.springframework.org/docs/reference/aop.html>

Advantages:

- not intrusive
- doesn't require a different runtime
- doesn't require precompilation
- easy to apply in a small scale to an existing application or application infrastructure

2. Bytecode Manipulators

Examples:

- Aspectwerkz
- JbossAOP
- AspectJ

• Benefits:

- usually faster
- can apply to class loader of existing applications

Uses:

- transparent persistence
- transactions
- security
- performance optimization
- error handling
- logging, debugging, metrics
- extending applications

Security Example

This example will apply from outside the application security rules.

We have an example dao type interface.

```
public interface WikiInterface {

    public String viewPage(String pageName) throws Exception;
    public void savePage(String pageName, String pageString)
throws Exception;
    public void appendToPage(String pageName, String
appendString) throws Exception;
}
```

This application already exists and we now find out we want to apply security rules for the different types of actions.

Create your mixin / aspect that will handle the concern (validating security).

```
public class SecurityMixin implements SecurityInterface, ProxyAware
{

    Proxy proxy = null;
    /* (non-Javadoc)
    * @see dynaop.ProxyAware#setProxy(dynaop.Proxy)
    */
    public void setProxy(Proxy proxy) {
        this.proxy = proxy;
    }

    /* (non-Javadoc)
    * @see
    com.objectcountry.aoppres.security.SecurityInterface#validateSecurity(java.lang.String)
    */
    public void validateSecurity(String method) throws Exception {

        User u = User.getCurrentUser();
        if(u == null){
            throw(new Exception("security violation: "
+ method + " no user defined." ));
        }
        if( !Role.ADMIN_USER.equals(u.getRole())){
            throw(new Exception("security violation: " + method + " by: "
+ u.getName() ));
        }
    }
}
```

Next you have to write your interceptor. This is the code that handles the logic for overriding the behavior of the base classes behavior.

```
public class SecurityMixinInterceptor implements Interceptor {

    public Object intercept(Invocation invocation) throws Throwable
    {
        // invoke our SubjectMixin...

        SecurityInterface si = (SecurityInterface)
invocation.getProxy();
        si.validateSecurity(invocation.getMethod().
getName());

        // proceed to the target method implementation.
Object result = invocation.proceed();

        return result;
    }
}
```

Next we code the introduction (tying it all together)

```
// make wikis colorable
mixin(
    com.objectcountry.aoppres.SimpleWiki.class,
    com.objectcountry.aoppres.security.SecurityMixin.class);

interfaces( com.objectcountry.aoppres.SimpleWiki.class,
            new Class[]
{ com.objectcountry.aoppres.security.SecurityInterface.class});

interceptor(
    SimpleWiki.class,
    signature("save"),
    new SecurityMixinInterceptor()
);
```

Lets take a look at some other tools ...
Aspectwerkz

interceptor / aspect:

```
public class MyAspect {

    public void beforeaction(JoinPoint joinPoint) {
        System.out.println("before greeting...");
    }

    public void afterGreeting(JoinPoint joinPoint) {
        System.out.println("after greeting...");
    }
}
```

introduction:

```
<aspectwerkz>
  <system id="AspectWerkzExample">
    <package name="testAOP">
      <aspect class="MyAspect">
        <pointcut name="greetMethod" expression="execution(*
testAOP.HelloWorld.greet(..))"/>
        <advice name="beforeGreeting" type="before" bind-
to="greetMethod"/>
        <advice name="afterGreeting" type="after" bind-
to="greetMethod"/>
      </aspect>
    </package>
  </system>
</aspectwerkz>
```

running example:

```
java -cp $ASPECTWERKZ_HOME/lib/aspectwerkz-2.0.RC1.jar:target
-Daspectwerkz.definition.file=aop.xml testAOP.HelloWorld
```

Bibliography

1. dynaop web site: <https://dynaop.dev.java.net/nonav/release/1.0-beta/manual/index.html>
2. aspectwerkz web site: <http://aspectwerkz.codehaus.org/>
3. Aspectwerkz Dynaic AOP for Java by David Rabinowitz:
<http://www.cs.huji.ac.il/course/2003/ood/docs/tir09.ppt>
4. aspectj web site: <http://eclipse.org/aspectj/>