AspectJ Load Time Weaving in Spring

Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects. (Such concerns are often termed crosscutting concerns in AOP literature.)

Lets start how to configure load time weaving in Spring framework.

applicationContext.xml Configuration

```
<bean id="logAspect" class="com.codesenior.web.template.config.OperationLogger"
     factory-method="aspectOf"/>
<context:load-time-weaver aspectj-weaving="on" />
<aop:aspectj-autoproxy proxy-target-class="true">
    <aop:include name="logAspect"/>
</aop:aspectj-autoproxy>
```

aspectOf: The aspect is a singleton object and is created outside the Spring container. A solution with XML configuration is to use Spring's factory method to retrieve the aspect. With this configuration the aspect will be treated as any other Spring bean and the autowiring will work as normal.

aop.xml configuration

```
<!DOCTYPE aspectj PUBLIC "-//AspectJ//DTD//EN" "http://www.eclipse.org/aspectj/dtd/aspectj.dtd">
<aspectj>
    <weaver options="-verbose -showWeaveInfo -Xset:weaveJavaxPackages=true -Xreweavable">
    <!-- only weave classes in our application-specific packages-->
        <include within="com.codesenior.telif.security.*" />
        <include within="com.codesenior.web.template.controller.*" />
        <!--aspects should also be defined, otherwise aspectOf not found error occurs-->
        <include within="com.codesenior.web.template.config.OperationLogger"/>
    </weaver>

    <aspects>
        <!-- weave in just this aspect -->
        <aspect name="com.codesenior.web.template.config.OperationLogger"/>
    </aspects>
</aspectj>
```

Which packages are weaved in load time weaving, we configure by using <include element as above. The meaning of the dot and star (.*), all classes of the security package and controller package. If controller package has another package then we should use double dot and star character: `com.codesenior.web.template.controller..*`

OperationLogger File

```
package com.codesenior.web.template.config;

import com.codesenior.telif.generic.dao.service.model.Logon;
import com.codesenior.telif.generic.dao.service.model.UserOperation;
import com.codesenior.telif.generic.dao.service.service.LogonService;
```

```java
import com.codesenior.telif.generic.dao.service.service.OperationTypeService;
import com.codesenior.telif.generic.dao.service.service.UserOperationService;
import com.codesenior.telif.generic.dao.service.service.UserService;
import com.codesenior.telif.util.DomainUtil;
import com.codesenior.web.template.exception.GlobalExceptionHandler;
import org.apache.logging.log4j.LogManager;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;
import javax.servlet.http.HttpServletRequest;
import java.util.Date;
import static com.codesenior.telif.security.util.LoggedUserUtil.loggedUsername;


@Aspect
public class OperationLogger {
     private static org.apache.logging.log4j.Logger logger =
LogManager.getLogger(OperationLogger.class);
     @Autowired
     private LogonService logonService;
     @Autowired
     private OperationTypeService operationTypeService;
     @Autowired
     private UserService userService;
     @Autowired
     private UserOperationService userOperationService;

     @After("execution(* com.codesenior.web.template.controller..*(..)) " +
             "&& @annotation(org.springframework.web.bind.annotation.RequestMapping)")
     public void logBefore(JoinPoint joinPoint) throws Throwable {
         if (loggedUsername() != null)
             logger.info("Kullanici " + loggedUsername() + " " +
joinPoint.getSignature().getName() + " metodu çağırdı");
         UserOperation userOperation = new UserOperation();

         userOperation.setExplanation(joinPoint.getSignature().getName() + " metodu "
                 + getMethodArgsValues(joinPoint)+" parametre değerleriyle çağrıldı");
         userOperation.setOperationType(operationTypeService.get("name", "METHOD"));
         userOperation.setUser(userService.getUser(loggedUsername()));
         userOperation.setStartDate(new Date());
         userOperation.setEndDate(new Date());
         userOperationService.save(userOperation);
     }

     private String getMethodArgsValues(JoinPoint joinPoint) {
         String result = "";
         Object[] signatureArgs = joinPoint.getArgs();
         for (Object signatureArg : signatureArgs) {
             if(signatureArg instanceof String)
             result += signatureArg+", ";
         }
         return result.replaceAll("(,\\s)$","");//son kismi sildirmek icin
     }

     @AfterReturning(pointcut =
             "execution(* com.codesenior.telif.security." +
                     "CustomUsernamePasswordAuthenticationFilter.attemptAuthentication(..))",
returning = "result")
     public void after(JoinPoint joinPoint, Object result) {
         logonService.save(new Logon(((Authentication) result).getName(), new Date(),
                 getClientIpAddress(joinPoint)));
     }

```

```
    private String getClientIpAddress(JoinPoint joinPoint) {
        return DomainUtil.getClientIpAddr(getHttpServletRequest(joinPoint));
    }

    private HttpServletRequest getHttpServletRequest(JoinPoint joinPoint) {
        Object[] args = joinPoint.getArgs();
        HttpServletRequest request = null;
        for (Object arg : args)
            if (arg instanceof HttpServletRequest) return (HttpServletRequest) arg;

        return request;
    }
}
```

Maven Dependencies

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>4.2.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.8.9</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-agent</artifactId>
    <version>2.5.6</version>
</dependency>

<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.9</version>
</dependency>

<!--It is used for -javaagent jvm argument-->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument</artifactId>
    <version>3.2.4.RELEASE</version>
</dependency>

<!--If you are using Tomcat 6.X or Tomcat 7.X you should use this
dependency for TomcatInstrumentableClassLoader. -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-instrument-tomcat</artifactId>
    <version>3.0.4.RELEASE</version>
</dependency>
```

Using -javaagent argument in JUnit test:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.19.1</version>
    <configuration>
        <forkMode>once</forkMode>
        <skipTests>true</skipTests><!--you can set this value as false-->
        <!--necessary for test -->
        <argLine>
```

```
            -javaagent:${user.home}/.m2/repository/org/springframework/spring-
agent/2.5.6/spring-agent-2.5.6.jar
            -
javaagent:${user.home}/.m2/repository/org/aspectj/aspectjweaver/1.8.9/aspectjweaver-1.8.9.jar
        </argLine>
    </configuration>
</plugin>
```

Tomcat Configuration

Historically, Apache Tomcat's default class loader did not support class transformation which is why Spring provides an enhanced implementation that addresses this need. Named **TomcatInstrumentableClassLoader**, the loader works on Tomcat 6.0 and above.

Note: Do not define **TomcatInstrumentableClassLoader** anymore on Tomcat 8.0 and higher. Instead, let Spring automatically use Tomcat's new native InstrumentableClassLoader facility through the TomcatLoadTimeWeaver strategy.

If you still need to use **TomcatInstrumentableClassLoader**, it can be registered individually for each web application as follows:

Copy org.springframework.instrument.tomcat.jar into `$CATALINA_HOME/lib`, where `$CATALINA_HOME` represents the root of the Tomcat installation)

Instruct Tomcat to use the custom class loader (instead of the default) by editing the web application context file:

```
<Context path="/myWebApp" docBase="/my/webApp/location">
    <Loader
       loaderClass="org.springframework.instrument.classloading.tomcat.TomcatInstrumentableClas
sLoader"/>
</Context>
```

Apache Tomcat (6.0+) supports several context locations:

```
server configuration file - $CATALINA_HOME/conf/server.xml
default context configuration - $CATALINA_HOME/conf/context.xml - that affects all deployed web
applications
per-web application configuration which can be deployed either on the server-side
at $CATALINA_HOME/conf/[enginename]/[hostname]/[webapp]-context.xml or embedded inside
the web-app archive at META-INF/context.xml
```

For efficiency, the embedded per-web-app configuration style is recommended because it will impact only applications that use the custom class loader and does not require any changes to the server configuration. See the Tomcat 6.0.x documentation for more details about available context locations.

Alternatively, consider the use of the Spring-provided generic VM agent, to be specified in Tomcat's launch script (see above). This will make instrumentation available to all deployed web applications, no matter what ClassLoader they happen to run on.

For Maven project, the location of context xml file per web application is src/main/resources/META-INF/context.xml

If Tomcat doesn't find this location, you can use maven war plugin as follows:

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>2.6</version>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
                <classpathPrefix>lib/</classpathPrefix>
            </manifest>
        </archive>
        <webResources>
            <resource>
                <directory>${project.basedir}/src/main/resources</directory>
            </resource>
        </webResources>
        <warName>mywar</warName>
    </configuration>
</plugin>
```