Aspect Oriented Programming: Spring AOP and AspectJ

Nischitha M J[1], Dr. Ramakanth Kumar P[2]

[1]M. Tech CSE, RV College of Engineering, Bengaluru
[2]Prof. and Head of Dept, CSE, RV College of Engineering, Bengaluru

---***---

Abstract - *Aspect-oriented programming (AOP) is the programming method which complements the Object-Oriented Programming to specify non-functional requirements like security, logging, transaction management. These requirements are known as crosscutting concerns which are critical issues that cannot be easily modularized by traditional programming methods leading to complex code. This paper discusses AOP concept, different frameworks of AOP: Spring AOP and AspectJ, the weaving processes and comparison of frameworks.*

Keywords **-** Aspect Oriented Programming; Spring AOP; AspectJ; Weaving.

1. INTRODUCTION

A traditional code for the software is comprised of multiple different components. All of these components is responsible for implementing the system's fundamental purpose. Most issues, however, are critical for the whole system, such as error management, security, logging, transaction management, synchronization, and hence they cross-cut multiple components. Trying to implement such cross-cutting issues is a difficult problem that traditional methods to programming, like Procedural-Oriented Programming (POP) and Object Oriented Programming (OOP), will not quite easily modularise.

Inadequacy of modularity of code generally leads to a complex code and tangled. As a consequence, recently emerged Aspect-Oriented Programming (AOP) as a potential novel strategy to deal with this problem. Gregor Kiczales coined the word in1997[1] as a compliment to the OOP, instead of a substitution for it[2]. A general understanding of the technical importance would derive from the descriptive sense of the word "aspect." AOP is a programming methodology aimed at addressing cross-cutting problems during effective application modularisation. By properly addressing crosscutting concerns[3] it improves device characteristics such as usability, readability and modularity[5].
Figure 1 shows the Connection between AOP,OOP,POP.

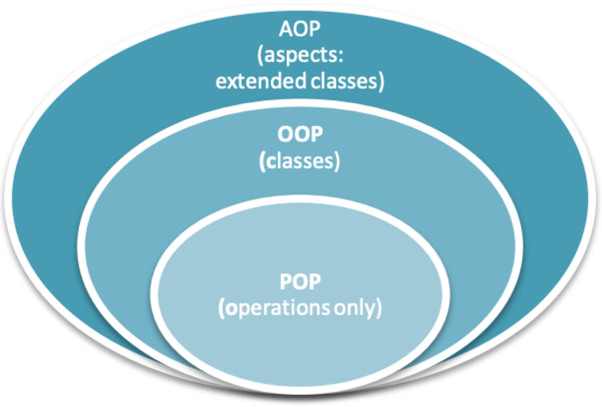

Figure 1: Connection between AOP,OOP,POP

AOP differentiates clearly between two forms of concerns:
• Primary concern: describes elements or artifacts in the physical world. In OOP, every of those components is identified by a class.

• Crosscutting concern: corresponds to the system design element provided by different components of the device. Hence its implementation among them is repeated, which affects the modularity of code[4].

Security is One of the example of a cross-cutting : while a Forum package's primary function is to view and maintain forum messages, some type of protection must be enforced to ensure that only admins can accept or remove posts. AOP allows you to move the security (or any other) aspect into your own package and leave clear responsibilities on the other objects, probably not implementing any security itself.

The most common frameworks of AOP are Spring AOP and AspectJ. The Spring AOP uses XML-based specification to implement AOP, as well as use of annotations that are viewed by using a library provided for parsing and matching by AspectJ. AspectJ is a Java programming extension created at the PARC research centre. AspectJ It uses Java as syntax and incorporates IDE functionalities to display cross-cutting configuration. It has its own compiler and weaver, allowing complete AspectJ language to be used on it.

2. AOP TERMINOLOGIES

AOP provides complete support for modularization of program ; Instead of spreading the code relevant to a non-functional condition or problem across the entire program[7], developers should position it within a separate section [8]. The common terminologies used in AOP are as follows:

• **Aspect:** An aspect is a class that implements java enterprise application concerns which cuts through multiple classes.
• **Join point**: It is a point in a program such as exception handling, Method execution etc.
• **Advice**: Advices are the actual actions taken for a particular join point. Advices are the methods that gets executed when certain a join point meets a matching pointcut in application.

The types of Advices are as follows:

- **After:** Runs after the advised method completes regardless of the outcome, whether successful or not.
- **Before:** Runs before the advised method is invoked.
- **Around:** This is the strongest advice among all the advice since it wraps around and runs before and after the advised method. This type of advice is used where we need frequent access to a method or database like- caching.
- **AfterReturning:** Runs after the advised method successfully completes i.e. without any runtime exceptions.
- **AfterThrowing:** Runs after the advised method throws a Runtime Exception.

• **Pointcut**: Pointcut are the expressions that are matched with join point to determine that advices needs to be executed or not.
• **Weaving**: Weaving is the process of linking an object with other application types or object to create an advised object. For AspectJ it can be executed in the compiling time, post compile time or Load time and for Spring AOP it can be executed during the running of the program [6].

3. AOP FRAMEWORKS

3.1 Spring AOP

Spring AOP enables Aspect Oriented programming in Spring Applications. In AOP aspects enable modularisation of concerns like Transaction Management, Logging, security that cut across multiple types of objects. An AOP provides the way that dynamically adds the cross cutting concerns before, after or around actual logic using simple pluggable configurations. It makes easy to maintain the code in present as well as future as it remove the concerns without recompiling the complete code simply by changing the configurations files.

The Target object is an object on which advices are applied and in spring a subclass is created at runtime for the target method is overwritten and advices are included based on their configurations. A proxy which is an object that is created after applying advice that are target object.

The proxy based mechanism is used in SpringAOP.It creates a proxy object that wraps around the original object and takes advice relevant to the call of the method. Proxy objects can be created manually in the XML file via proxy factory bean or auto proxy configuration, and destroyed when the execution is complete. Proxy objects are used to enrich the real object 's Original behaviour. Spring AOP Uses java for implementation. Any type of special compilation is not needed. Currently Spring AOP supports only method executions.

3.2 AspectJ

AspectJ(tm) is a simple and practical extension to the Java(tm) programming language that adds to Java aspect-oriented programming capabilities.

AspectJ supports two kinds of crosscutting implementation.

Static Crosscutting: It is possible to define new operations on existing types. This is known as static crosscutting since it affects the static type signature of the program.

Dynamic Crosscutting: It possible to define additional implementation to run at certain well- defined points in the execution of the program.

4. WEAVING

There are a few different methods in our application to insert the AOP code, but one common denominator is that they all need some sort of extra step to add to our code. That extra step is known as weaving.

4.1 Compile Time Weaving

This is the Simplest Weaving approach. In Compile Time Weaving the aspects compilation and the compilation of the source code are done directly using an AspectJ compiler if there are both the aspect source code and the code that is using aspects inside.

4.2 Post Compile Weaving or Binary Weaving

In Post Compile Time Weaving the input will be in the form of jar files or class files where each of which may contain classes and aspects. The compiler produces woven byte code (as class files or as a jar depending on the compiler options). Then deploying the resulting byte code in any standard compliant VM.

Figure 2 shows the Build-time binary weaving. To produce the woven system the input to aspectJ compiler(ajc) are the Java files and the Aspect byte-code files.

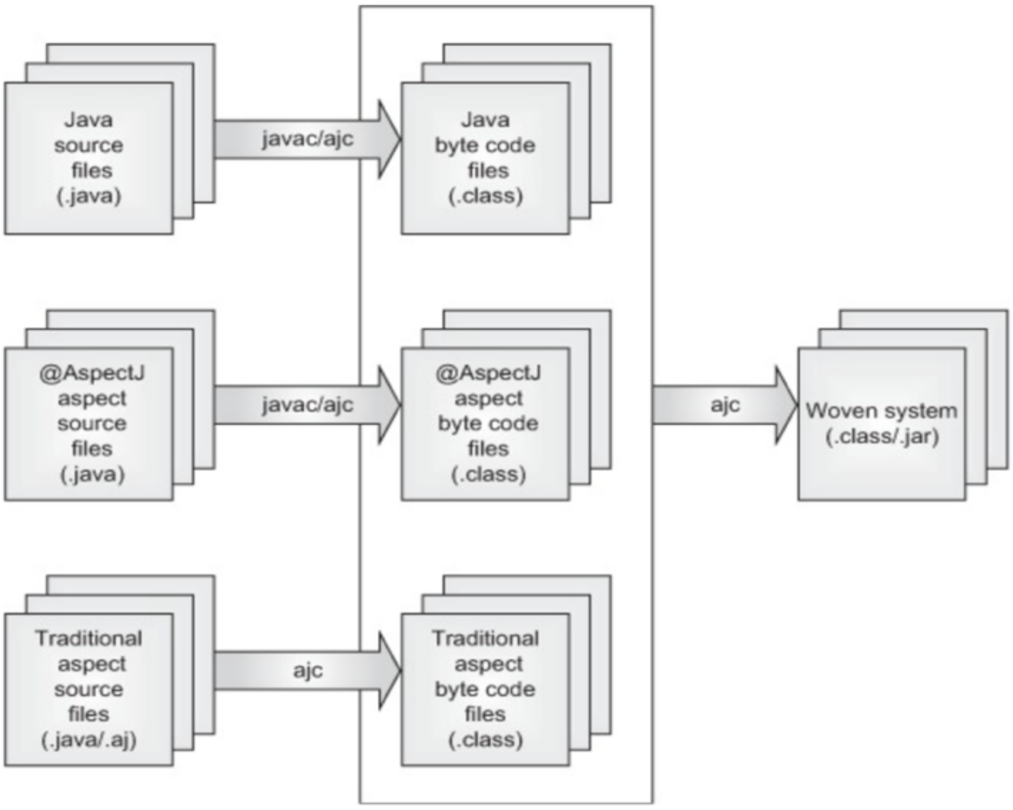
Java source files (.java)
javac/ajc
Java byte code files (.class)
@AspectJ aspect source files (.java)
javac/ajc
@AspectJ aspect byte code files (.class)
ajc
Woven system (.class/.jar)
Traditional aspect source files (.java/.aj)
ajc
Traditional aspect byte code files (.class)

Figure 2 : Build Time Weaving

4.3 Load Time Weaving

When the behaviour of the application is to be monitored to check whether there are any performance bottlenecks, in this situation once the requirements have been completed it not required to keep the aspects in system. In these type of cases the compile time weaving or binary weaving are not to be used. The Load Time Weaving(LTW) allows the build system to be left intact and serves the purpose by weaving the aspects needed. The LTW acts in similar way as binary weaving. The LTW also takes input in the form of byte code. The LTW does not need any extra step of weaving rather to perform weaving the as the VM loads the classes the VM is setup, so it is known as LTW.

The load-time weaver requires extra information to determine that what aspects to weave. Even though weaver can analyse the classpath to identify aspects present, it will be the expensive method because every class on classpath have to be checked if it was an aspect. It may not be desirable to weave all aspects found on classpath. The weaver therefore defines each element directly on the classpath in the aop.xml file. Although several classpath entries will contain an aop.xml file, they are all used by the weaver by merging the details inside them.

Figure 3 shows the Load-time weaving arrangement. The classes participating and the aspects information is given in aop.xml for load time weaving. Then the interception of weaver for loading class to weaving is done using aspects

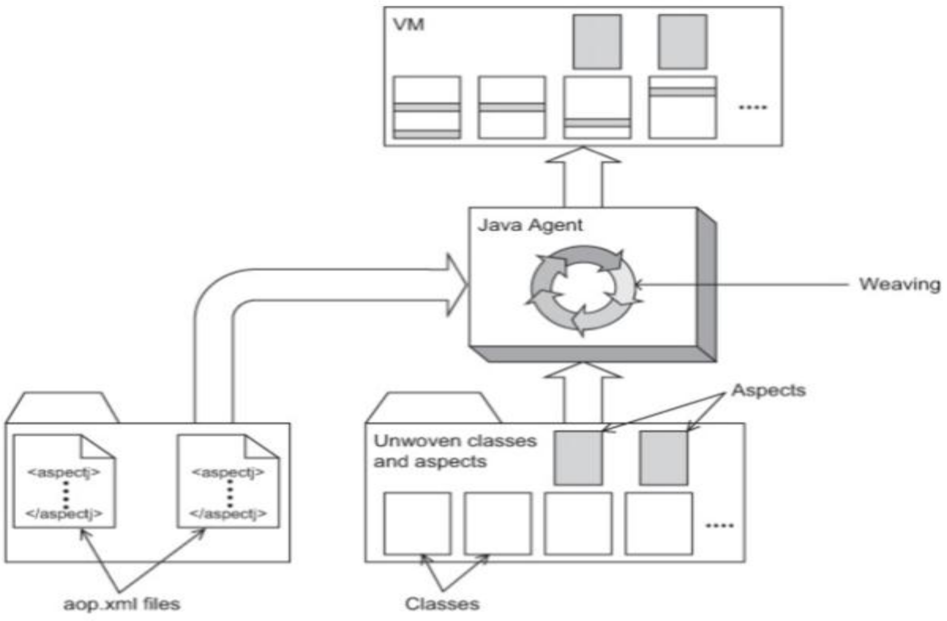

Figure 3: Load Time Weaving Arrangement

4.4 Runtime Weaving

Runtime weaving is used by Spring AOP. Here during the application execution the aspects are woven using proxies of the targeted object. This is done using either JDK proxy or CGLIB proxy as shown in Figure 4 .

1. JDK dynamic proxy – Spring AOP 's best way is JDK dynamic proxy. When the targeted object has a single interface, JDK dynamic proxy is used.
2. CGLIB proxy – CGLIB is used if the target object does not enforce an interface.

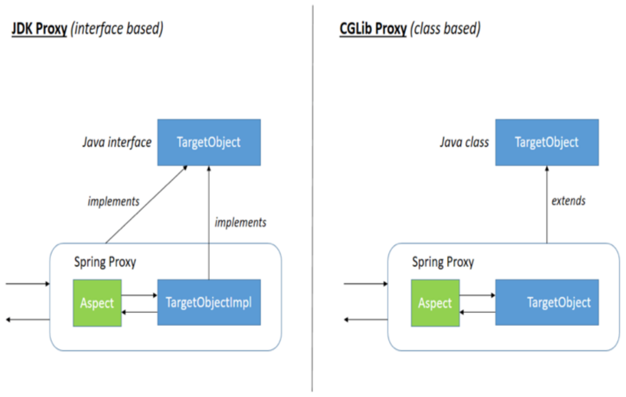

Figure 4: Spring AOP Process

5. COMPARISION OF SPRING AOP AND ASPECTJ

Spring AOP intends to have a basic Spring IoC-wide AOP implementation to address the problems that programmers experience. It is applied to beans which a Spring container manages and not aimed to give a complete AOP solution. The Spring AOP uses Runtime Weaving. Cross-cutting concerns cannot be applied across classes that are "final" because they cannot be overridden and would result in an exception in run time. Spring AOP can be only applied to method execution join point and it does not support method call, constructor call, Constructor Execution, Object initialisation, Field reference, static method, final methods

AspectJ is the main AOP technique that intends to provide complete AOP solution. This is more stable but also much more complex than Spring AOP. Also notable is that AspectJ can be extended to all domain objects. AspectJ uses three types of weaving: Load time weaving, Post Compile weaving or compile time weaving. Unlike Spring AOP, the target object does not need to be subclassified, and thus also supports many other join points. AspectJ directly weaves the cross-cutting concerns into the actual code before run time.

Performance wise Spring AOP is must slower than AspectJ but Spring AOP is easy to learn and apply whereas AspectJ is little complicated than Spring AOP.

6. CONCLUSIONS

For an Enterprise application, which is typically divided into layers: Data Access Layer, Business Logic Layer, UI Layer. These Layers will typically have some non-functional requirements like security, logging, Transaction management. Logging to write the error or information to log files, profiling to see how the application is performing. These are called cross cutting concerns as these are required across the all the layer as well as across the applications in enterprise. So, this is where the AOP is used. Using Cross cutting concerns the AOP address all the non-functional requirements which are common across the enterprises allowing to reuse. The Focus on Business logic without worrying about the non-functional requirements is one of the main advantages of AOP.

Both of the frameworks of AOP: Spring AOP and AspectJ are fully compatible with each other. Even though Spring AOP does not provide Complete AOP solution, but when the application uses spring container it is better to use Spring AOP as it easy to learn and apply. When it is required to exploit the AOP to maximum then AspectJ is the best choice as it allows to use wide variety of join points. AspectJ is considered to be 8-35 times faster than AOP in Spring. When there are thousands of aspects the runtime weaving is not feasible, and AspectJ would be better to implement AOP.

REFERENCES

[1] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.Videira Lopes, J.- M. Loingtier, J. Irwin, “Aspect Oriented Programming”, In Proc. Europ. Conf. on Object-Oriented Prog.(ECOOP), Finnland, Springer Verlag LNCS 1241, June 1997.
[2] I. Boticki, M. Katic, S. Martin, “Exploring the Educational Benefits of Introducing Aspect-Oriented Programming Into a Programming Course,” , IEEE Transactions on Education, vol.56, no.2, pp.217-226, May 2013.
[3] T. Zukai, P. Zhiyong, “Survey of Aspect-Oriented Programming Language , Journal of Frontiers of Computer Science and Technology, 2010, vol.4, no.1, pp 1-19.
[4] L. Madeyski, L. Szala, “Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study,” IET Software, 2007 , vol. 1, no.5, pp. 180-187.
[5] J. Viega, J. Vuas, “Can aspect-oriented programming lead to more reliable software?,” IEEE Software, 2000, vol.17, no.6, pp. 19-21.
[6] D. Zhengyan, “Aspect Oriented Programming Technology and the Strategy of Its Implementation,” In Proceedings of International Conference on Intelligence Science and Information Engineering (ISIE), 2011, pp.457,460, 20-21.
[7] S.R. Chidamber, C.F. Kemerer, “A metrics suite for object oriented design,” IEEE Transaction on Software Engineering, 1994, vol. 20, no. 6, pp. 476–493.
[8] S.L. Tsang, S. Clarke, E.L.A. Baniassad, “An evaluation of aspect oriented programming for Java-based real-time systems development,” In Proceedings of the International Symposium of Object-Oriented Real-Time Distributed Computing ISORC, 2004, Vienna, Austria, pp. 291–300.