

Understanding Aspect Oriented programming (AOP)

By [Abhimanyu](#) February 2, 2013

[Featured, JEE/J2EE](#) [2 Comments](#)

Spring MVC based login application using JDBC template

Spring MVC application flow and integration with Thymeleaf

Understanding Dependency Injection (DI)

Create your first application in Thymeleaf with Spring MVC

Integration of Hibernate with Spring MVC and Maven

Introduction to Spring Framework

91
SHARES



Share with friends



Twitter



Subscribe Now

Aspect Oriented Programming (AOP) allows modularization of crosscutting concerns such as transaction management, Security, Logging etc. In other words, AOP aims at increasing modularity of the application, by separating the crosscutting concerns from the actual business logic implementations.

Crosscutting concerns are nothing but the functionalities that are not actually a part of your problem domain, such as logging.

Why do we need AOP when OOP is already there?

Suppose, there are three classes, A, B and C. Each class has its own business methods and I want my logging

Receive our updates to your inbox Email

[Subscribe Now](#)

Understanding
Spring MVC
based
application
flow

Logger object, though it does not add any business value when called from inside any of the business methods of any of the classes A, B and C.

Spring MVC
BeanNameView
example

Configure
multiple
View
Resolvers in
Spring

Any change in the method signature (**Method signature** is part of the method declaration. It is the combination of the method name and the parameter list), might require a corresponding change in all the classes A, B and C, wherever the Logger method call has been done.

Read
property
files with
PropertyPlace

These unnecessary dependencies inside the classes finally gave way to Aspect Oriented Programming. AOP not only removes these unnecessary dependencies, but it also increases application modularity and provides us with easily configurable handle inside spring context XML.

Internationali
with
ReloadableRes
MessageSource

Understanding
Transactional
annotation
in Spring

How does AOP work?

Content
Negotiation
in Spring
Framework

Let us take the same scenario as I have explained above. Suppose, there are three classes, A, B and C. Each class has it's own business methods and I want my logging method to be called at the start and

Easy
integration
of Spring
MVC with
Angular JS

successful end of these methods. Now, with AOP concept into picture, I create a new class

Read
property
files with
PropertiesFac

LoggingAspect instead of the conventional Logger class. For now, assume *LoggingAspect* is a class with some extra privileges (We will be going with the actual implementation and examples later). With *LoggingAspect*, I do not need to instantiate

Access

Receive our updates to your inbox Email

Subscribe Now

Framework**LDAP
Authentication
with Core
Spring LDAP**

configuration, and executes the configured aspects on the concerned methods/objects. Spring makes sure the configured aspects are called before/after the methods that have been configured.

**Performance
tuning of
Spring based
application**

An **XML based aspect configuration** looks something like the following:

```

<!-- Aspect -->
<bean id="logAspect" class="com.jcombat.aspect.LoggingAspect" />
<aop:config>
  <aop:aspect id="aspectLogging" ref="logAspect" >
    <!-- @Before --> any return type
    <aop:pointcut id="pointCutBefore"
      expression="execution(* com.jcombat.customer.bq.addCustomer(..))" />
    <aop:before method="logBefore" pointcut-ref="pointCutBefore" />
  </aop:aspect>
</aop:config>

```

The business method before which the logBefore method of LoggingAspect class will get called

Advice

Each of the comments in the snapshot have their explanation as you go ahead.

Adding the same above snapshot below with some additional comments:

```

<!-- Aspect -->
<bean id="logAspect" class="com.jcombat.aspect.LoggingAspect" />
<aop:config>
  <aop:aspect id="aspectLogging" ref="logAspect" >
    <!-- @Before --> Aspect class method or the Advice method
    <aop:pointcut id="pointCutBefore"
      expression="execution(* com.jcombat.customer.bq.addCustomer(..))" />
    <aop:before method="logBefore" pointcut-ref="pointCutBefore" />
  </aop:aspect>
</aop:config>

```

Any method parameter

Pointcut expression

The same AOP configuration can be very well achieved with **AspectJ Annotations**. Equivalent **AspectJ implementation with annotation** will be like:

```

1 package com.jcombat.aspect;
2
3 import org.aspectj.lang.JoinPoint;
4 import org.aspectj.lang.annotation.Aspect;
5 import org.aspectj.lang.annotation.Before;
6
7 @Aspect

```

Receive our updates to your inbox

Email

Subscribe Now

There are a few **AOP terminologies** that needs to be kept in mind.

Aspect: Aspect is the modularization of a concern that cuts across multiple objects. Logging and Transaction management are the best examples of cross-cutting concern in Java. In Spring AOP, aspects are implemented using regular classes (the schema-based approach, with *XML based configuration*) or regular classes annotated with the *@Aspect* annotation (*@AspectJ* style), for which AspectJ needs to be integrated with Spring.

Join Point: In Spring AOP, a Join Point always represents a method execution. This represents a point in your application where you can plug-in AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework.

Advice: Action taken by an Aspect at a particular join point. Different types of advice include “around”, “before”, “after”, “after returning” and “after throwing” advice. **Around advice** is the most important and powerful of all, where the advice code gets executed before and after the execution of the JoinPoint methods. It is the responsibility of around advice to invoke the join point methods, and finally return the respective values if the JoinPoint method returns any.

Pointcut: Advice is associated with a pointcut expression and runs at any join point matched by the

Receive our updates to your inbox Email

Subscribe Now

that, *set of Join Points is called a PointCut.*

Target object: The object that is being advised by one or more aspects. Also referred to as the *advised object*.

Weaving: Weaving is the process of linking aspects with other objects to create an advised object. This can be done at compile time, load time, or at runtime. Spring AOP performs weaving at the runtime.

There is something great with [AOP proxies](#) getting created in the process. For any of your queries, we would be as glad to help you.

**We frequently publish on such topics.
Subscribe us now and stay updated. Also get
the FREE copy of masterful Ebook on RESTful
& SOAP Web Services and avail the chance to
win attractive geeky T-Shirts.**

Subscribe

[aspect](#) [basic concepts](#) [dependency injection](#)
[Inversion of Control](#) [IoC](#) [j2ee](#) [jee](#)
[logging](#)

Related Posts

No Preview

No Preview


No Preview

Receive our updates to your inbox Email

Subscribe Now

Written by **Abhimanyu**

Owner/Administrator at jCombat, a passionate tech blogger and a senior programmer with an extensive end-to-end development experience with wide range of technologies.

**Comments****Community** **Login** ▾ **Recommend** **Share****Sort by Best** ▾

LOG IN WITH

 jCombatOR SIGN UP WITH DISQUS 

By signing up, you agree to the Disqus [Basic Rules](#), [Terms of Service](#), and [Privacy Policy](#).

**DHIRENDRA GOHIL** • 5 months ago

Hi Abhimanyu, its great to learning the simplest, easy to understand & relate explanation of any

Receive our updates to your inbox Email**Subscribe Now**



Abhimanyu MOD  **DHIRENDRA GORIL**

• 5 months ago

Oh sure, Dhirendra! However, you might have to wait a while. But it's really great to hear that this article helped. Thanks a lot!

^ | v • Reply • Share ›

ALSO ON JCOMBAT

Spring Security LDAP Authentication

2 comments • 2 years ago •

Avatar **Abhimanyu** — Hi

ApplicationContext vs WebApplicationContext

5 comments • 2 years ago •

Avatar **Abhimanyu** — Yes and

GET MORE
STUFF LIKE
THIS

IN YOUR
INBOX

Receive our updates to your inbox [Email](#)

[Subscribe Now](#)

we respect your privacy
and take protecting it
seriously

**Whats new on
Spring!**

**Understanding
OAuth2 token
authentication**

**First RESTful
Service with Spring
Boot**

**How to load
properties using
Spring Boot
@ConfigurationProp**

jCombat is also on



**Whats new on Core
Java!**

**Introduction to
Executor
Framework**

**Understanding
Connection Pooling**

**Understanding
Generics in Java**

Receive our updates to your inbox Email

Subscribe Now

**Remove elements
from a JSON object
in DOJO**

**Introduction to
jsRender**

**Whats new on
Hibernate!**

**Constructor based
HQL to improve
performance**

**Enabling Entity and
Query cache in
Hibernate**

**Criteria Query with
an object property
as restriction**

**Whats new on
AngularJS!**

**RequireJS with
AngularJS
application**

**Custom Directives in
AngularJS**

**AngularJS digest
cycle and watch
function**

Receive our updates to your inbox Email

Subscribe Now

notification whenever we publish something new.

Tips and Tricks

- **6 best tools for quick and efficient technical blogging**
- **Boost your speed with these Notepad++ shortcuts**
- **Increase the maximum file upload size in WordPress**

Our Partners



Useful Links

- [About Me](#)
- [Contact Us](#)
- [Learn with us](#)
- [Write a testimonial](#)

Happy to Connect!

Testimonials We Respect

I really like it. Keep going author is doing good job posting this kind of stuff examples and easy explanations

Avinash
NTT data

jCombat Copyright © 2017.