| **Spring AOP** | **AspectJ** |
| --- | --- |
| Implemented in pure Java | Implemented using extensions of Java programming language |
| No need for separate compilation process | Needs AspectJ compiler (ajc) unless LTW is set up |
| Only runtime weaving is available | Runtime weaving is not available. Supports compile-time, post-compile, and load-time Weaving |
| Less Powerful – only supports method level weaving | More Powerful – can weave fields, methods, constructors, static initializers, final class/methods, etc… |
| Can only be implemented on beans managed by Spring container | Can be implemented on all domain objects |
| Supports only method execution pointcuts | Support all pointcuts |
| Proxies are created of targeted objects, and aspects are applied on these proxies | Aspects are weaved directly into code before application is executed (before runtime) |
| Much slower than AspectJ | Better Performance |
| Easy to learn and apply | Comparatively more complicated than Spring AOP |

**AOP Concepts/Terms**

**Cross Cutting Concerns**

The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic.

**Aspect** –

A standard code/feature that is scattered across multiple places in the application and is typically different than the actual Business Logic (EX: Logging, Transaction management, Caching, Security).

Each aspect focuses on a specific cross-cutting functionality

**Joinpoint** –

it’s a particular point during execution of programs like method execution, constructor call, or field assignment

**Advice** –

The action taken by the aspect in a specific joinpoint

**Point Cut** -

This is a set of one or more join points where an advice should be executed.

**Weaving** –

The process of linking aspects with targeted objects to create an advised object

**Target**

This is the Actual Business Logic Object being advised by one or more Aspects.