SpEL – Spring Expression Language & AOP

Presented by

Sagar
Java Consultant

spring

Spring SpEL

•**Definition:**

Spring Expression Language (SpEL) allows to query and manipulate objects in the Spring IoC container.

•**Purpose:**

- •Conditional logic in annotations, property resolution, and dynamic expressions in configuration.

- ✓ The Spring EL evaluated or executed during the bean creation time.
- ✓ In addition, all Spring expressions are available via XML or annotation.

SpEL Features

Dynamic Querying:

Querying and filtering data using SpEL expressions.
For instance, use SpEL to dynamically evaluate and retrieve properties of beans.

```
<bean id="emp" class="com.mycom.spring.spel.properties.Employee">
    <property name="id" value="1020"></property>
    <property name="name" value="Nesha"></property>
    <property name="address" value="#{address}"></property>
    <property name="state" value="#{address.state}"></property>

</bean>
```

Arithmetic and Logical Operations:

SpEL supports arithmetic operations (+, -, *, /) and logical operations (&&, ||, !), allowing to perform computations and logical checks within expressions.

```
// evaluates to 6
result = parser.parseExpression("3 * 2").getValue(Integer.class);
System.out.println("Product : " +result);
```

Spring SpEL

String Manipulation:

SpEL provides methods for string operations such as substring(), length(), toUpperCase(), etc.
Ex:

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("'Hello World'.concat('!')");
String message = (String) exp.getValue();
```

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("new String('hello world').toUpperCase()");
String message = exp.getValue(String.class);
```

Object Navigation:

You can traverse through object properties and collections using SpEL. For example, you can access nested properties and iterate over collections.

**Demo

Aspect Oriented Programming - Concerns

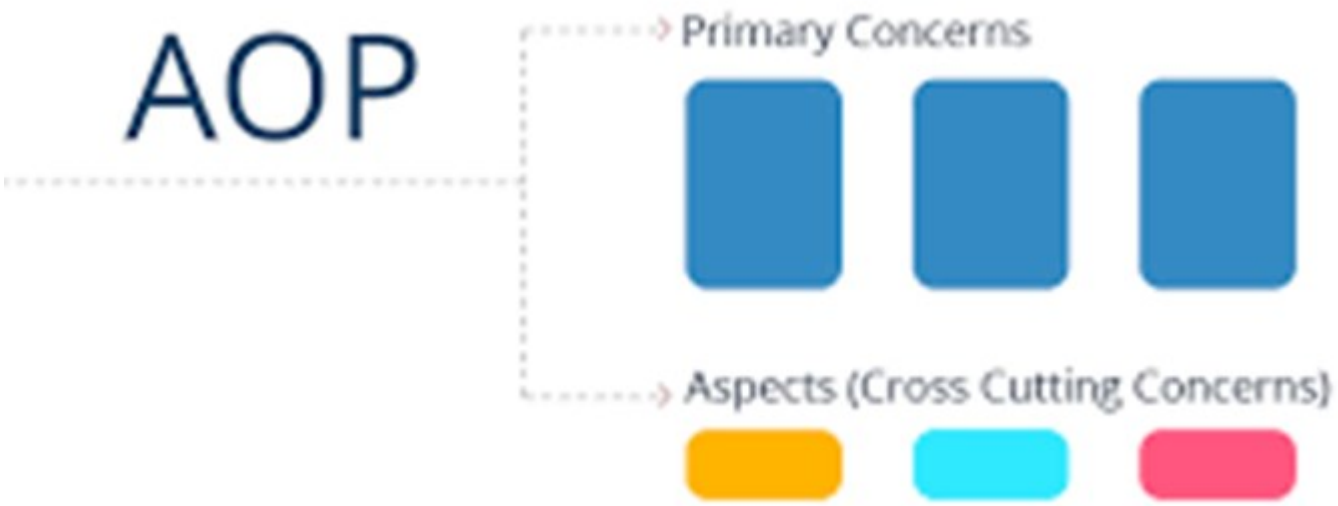
Concern : Background functionality of an application apart from business flow.

Ex : Logging messages – Web Server, Hibernate, Spring – apart from app result

These concerns are spread across the application modules.

Behind the application logic / flow / module to know the status of the application

Spring AOP



Spring AOP - Terminology

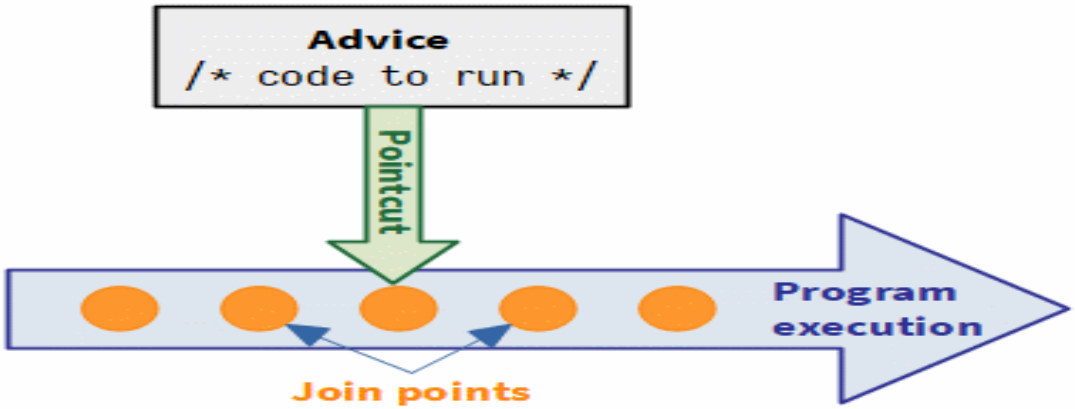
Aspect: A modularization of a cross-cutting concern. Implemented in Spring as Advisors or interceptors

Joinpoint: Point during the execution of execution.

Advice: Action taken at a particular joinpoint.

Pointcut: A set of joinpoints specifying where advice should be applied.

Introduction: Adding methods or fields to an advised class.



spring

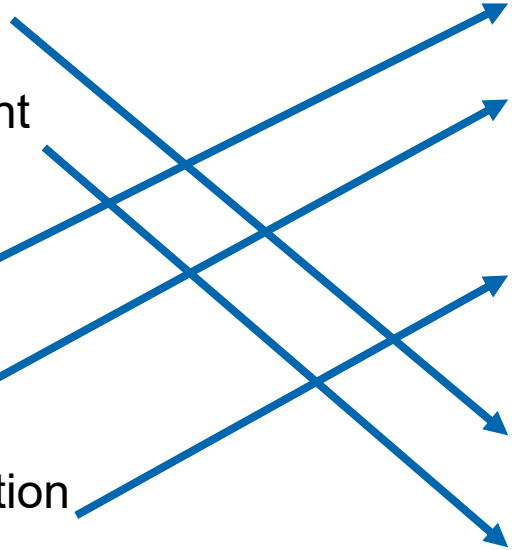
Spring AOP - Terminology

Terms	Explanations
1. Pointcut	A. Well defined point of execution in a program - Action to be taken at Join Points
2. Join Point	B. Well-modularized cross-cutting concern
3. Advice	C. Adding methods or fields to an advised class
4. Aspect	D. Set of Join Points
5. Introduction	E. Well-defined point of execution in a program

Spring AOP - Terminology

Terms

1. Pointcut
2. Join Point
3. Advice
4. Aspect
5. Introduction



Explanations

A. Well defined point of execution in a program - Action to be taken at Join Points

B. Well-modularized cross-cutting concern

C. Adding methods or fields to an advised class

D. Set of Join Points

E. Well-defined point of execution in a program

Spring AOP - Example

Student.java

```
public void setAge(Integer age) {
   this.age = age;
}
public Integer getAge() {
   System.out.println("Age : " + age );
   return age;
}
public void setName(String name) {
   this.name = name;
}
public String getName() {
   System.out.println("Name : " + name );
   return name;
}
```

MainApp.java

```
Student student = (Student) context.getBean("student");
student.getName();
student.getAge();
```

Spring AOP - Example

Bean Configurations in applicationContext.xml

```
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

<aop:aspectj-autoproxy/>

<!-- Definition for student bean -->
<bean id = "student" class = "com.mycom.springaop.bean.Student">
    <property name = "name"  value = "Vinoothna" />
    <property name = "age"  value = "21"/>
</bean>

<!-- Definition for logging aspect -->
<bean id = "logging" class = "com.mycom.springaop.aspect.StudentLoggerAspect"/>
```

Output on Console

```
INFO: Loading XML bean definitions from class path resource [applicationContext.xml]
Going to setup student profile.
Name : Vinoothna
Going to setup student profile.
Age : 21
```

Spring AOP - Example

StudentLoggerAspect.java

```
@Aspect
public class StudentLoggerAspect {
    /*Following is the definition for a Pointcut to select
     *  all the methods available. So advice will be called
     *  for all the methods.
    */
    @Pointcut("execution(* com.mycom.springaop.bean.*.*(..))")
    private void selectAll(){}

    /*
     * This is the method that executes
     * before a selected method execution.
    */
    @Before("selectAll()")
    public void beforeAdvice(){
        System.out.println("Going to setup student profile.");
    }
}
```

Any Class

Any Method

Thank You