Spring AOP

ASPECT ORIENTED PROGRAMMING

Spring AOP - Introduction

- Aspect – Unit of Modularity
- Entails breaking down program logic into distinct parts called so-called concerns. The functions that span multiple points of an application are called cross-cutting concerns
- Cross-cutting concerns are conceptually separate from the application's business logic.

AOP in Business

- Logging
- Transaction Management – Declarative
- Caching
- Security

AOP Terminologies

- Aspect
- Advice
- Join Point
- Pointcut
- Introduction
- Target Object
- Weaving

AOP Advice

- *This is the actual action to be taken either before or after the method execution.*
- *This is actual piece of code that is invoked during program execution by Spring AOP framework.*

Types of Advices

- Before - Run advice before the a method execution.
- After - Run advice after the a method execution regardless of its outcome.
- After Returning - Run advice after the a method execution only if method completes successfully.
- After throwing - Run advice after the a method execution only if method exits by throwing an exception.
- Around- Run advice before and after the advised method is invoked.

Advices - Examples

- @Before("execution(* *.*(..))")
- @After("execution(* *.*(..))")
- @AfterReturning("execution(* *.*(..))")
- @AfterThrowing("execution(* *.*(..))")
- @Around("execution(* *.*(..))")

Aspects Implementation

- XML Schema Based
- Annotaion Based

Join Point

- An Advice is applied to different program execution points, which are called join points.
- An Advice to take the correct action, it often requires detailed information about join points.

Accessing Join Point Information

- Kind
- Method signature
- Argument values
- Target Object
- Proxy Object

AspectJ Precedence

- More than one Aspect classes – need Precedence
- How to implement?
  - Ordered Interface
  - Order Annotation

Point Cut Expressions

- It is a powerful expression language that can match various kinds of join points.
- Indicate which method should be intercept, by method name or regular expression pattern.
- expression(<method scope> <return type> <fully qualified class name>.*(parametes))

Continue..,

- *method scope: Advice will be applied to all the methods having this scope. For e.g., public, private, etc. Please note that Spring AOP only supports advising public methods.*
- *return type: Advice will be applied to all the methods having this return type.*

Continue..,

- *fully qualified class name: Advice will be applied to all the methods of this type. If the class and advice are in the same package then package name is not required*
- *parameters: You can also filter the method names based on the types. Two dots(..) means any number and type of parameters.*

Pointcut Examples

- *execution(* com.aspects.pointcut.DemoClass.*(..)) : This advice will be applied to all the methods of DemoClass.*
- *execution(* DemoClass.*(..)): You can omit the package if the DemoClass and the advice is in the same package.*
- *execution(public * DemoClass.*(..)): This advice will be applied to the public methods of DemoClass.*

Pointcut Examples

- *execution(public int DemoClass.*(..)): This advice will be applied to the public methods of DemoClass and returning an int.*
- *execution(public int DemoClass.*(int, ..)): This advice will be applied to the public methods of DemoClass and returning an int and having first parameter as int.*
- *execution(public int DemoClass.*(int, int)): This advice will be applied to the public methods of DemoClass and returning an int and having both parameters as int.*

Reusing Pointcut Expression

```
@Pointcut("execution(* *.*(..))")
private void loggingOperation()
{}

@Before("loggingOperation()")
public void logBefore(JoinPoint joinPoint)
{}
```

PointCut Class Aspect

```
@Aspect
Class PointCut
{
@Pointcut("execution(* *.*(..))")
private void loggingOperation()
{}
}
```

```
@AfterReturning(
pointcut = "PointCut.loggingOperation()",
returning = "result")
public void logAfterReturning(JoinPoint joinPoint, Object result) {
}
```

Type Signature Pattern

- Within - pointcut expressions matches all join points within certain types.
- pointcut matches all the method execution join points within the com.msoftgp.spring package

within(com.msoftgp.spring.*)

- To match the join points within a package and its subpackage, you have to add one more dot before the wildcard.

within(com.msoftgp.spring..*)

- *The pointcut expression matches the method execution join points within a particular class:*

within(com.msoftgp.spring.ArithmeticCalculatorImpl)

- *Again, if the target class is located in the same package as this aspect, the package name can be omitted.*

within(ArithmeticCalculatorImpl)

- *You can match the method execution join points within all classes that implement the ArithmeticCalculator interface by adding a plus symbol.*

within(ArithmeticCalculator+)

Combining Pointcuts

```
@Aspect
public class CalculatorPointcuts {
@Pointcut("within(ArithmeticCalculator+)")
public void arithmeticOperation() {}
@Pointcut("within(UnitCalculator+)")
public void unitOperation() {}
@Pointcut("arithmeticOperation() || unitOperation()")
public void loggingOperation() {}
}
```

Pointcut Parameters

- *execution(* *.*(..)) && target(target) && args(a,b)*
- *Public void logBefore(Object target,int a,int b)*

AOP Introduction

- Special type of Advice
- Same effect as multiple inheritance
- Mechanism used – Dynamic Proxy
- @DeclareParents

Load Time Weaving AspectJ

- Purpose – Additional Pointcuts ,apply aspects

to objects created outside the Spring IoC container

- Weaving is the process of applying aspects to your target objects.
- Weaving happens at runtime through dynamic proxies.
- Supports compile time and load time weaving.

Types of AspectJ Weaving

- Complie time weaving
- Load time weaving
  - By AspectJ Weaver
  - Spring Load time Weaver
- Application Used : Caching

Configuring AspectJ Aspects in Spring

- *Factory-method is aspectof()*

Injecting Spring Beans into Domain Beans

- Wiring Spring objects and Domain Objects
- Injection of Spring beans is a cross cutting concern
- Annotate the Domain class with @Configurable

THANK YOU