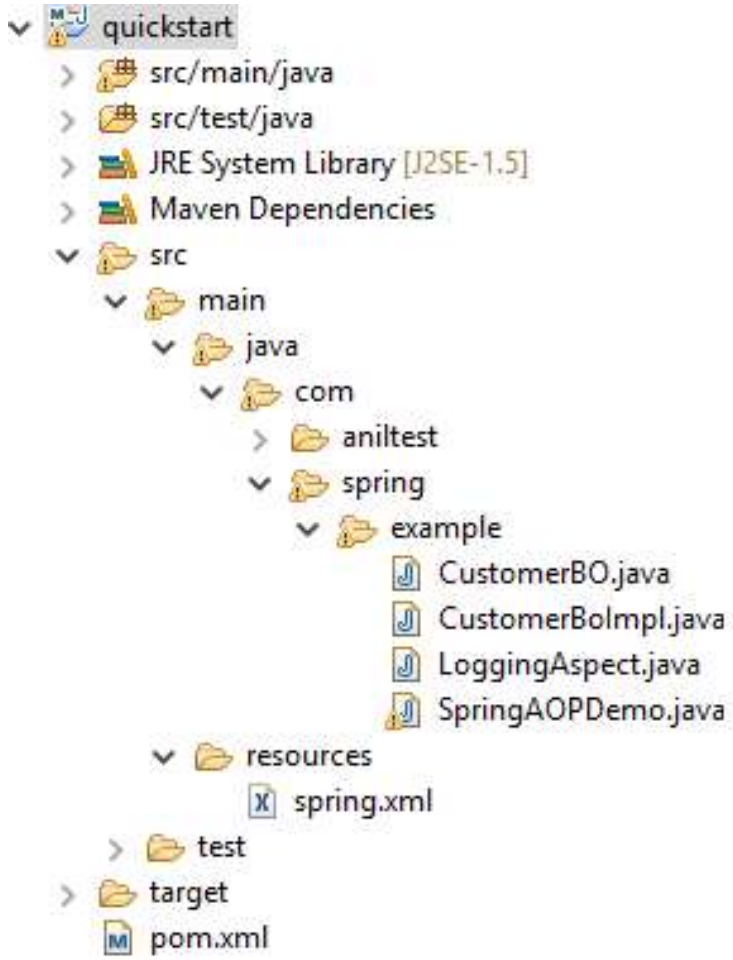
Spring AOP using Annotation

In this tutorial, we show you how to integrate AspectJ annotation with Spring AOP framework. In simple, Spring AOP + AspectJ allow you to intercept method easily.

Common AspectJ annotations:

1. **@Before** – Run before the method execution
2. **@After** – Run after the method returned a result
3. **@AfterReturning** – Run after the method returned a result, intercept the returned result as well.
4. **@AfterThrowing** – Run after the method throws an exception
5. **@Around** – Run around the method execution, combine all three advices above.

Directory Structure Maven Project



Pom.xml dependencies:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <spring.version>3.2.3.RELEASE</spring.version>
</properties>

<dependencies>
  <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-core</artifactId>
          <version>${spring.version}</version>
      </dependency>
      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-context</artifactId>
          <version>${spring.version}</version>
      </dependency>
      <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-aop</artifactId>
          <version>${spring.version}</version>
      </dependency>
      <dependency>
          <groupId>org.aspectj</groupId>
          <artifactId>aspectjrt</artifactId>
          <version>1.6.11</version>
      </dependency>
      <dependency>
          <groupId>org.aspectj</groupId>
          <artifactId>aspectjweaver</artifactId>
          <version>1.6.11</version>
      </dependency>
```

Spring Bean: CustomerBO (interface)

```
package com.spring.example;

public interface CustomerBO {

      public void addCustomer();
      public String addCustomerReturnValue();
      public void addCustomerThrowingException() throws Exception;
      public void addCustomerAround(String name,String phone_no);
}
```

Spring Bean: CustomerBoImpl (Business class or target object)

```
package com.spring.example;

public class CustomerBoImpl implements CustomerBO {

      public void addCustomer() {
            System.out.println("..Inside addCustomer target object (business
object) method...");
      }
```

```
	public String addCustomerReturnValue() {
		System.out.println("..Inside addCustomerReturnValue target object
method...");
		return "Customer Info added successfully!!";
	}

	public void addCustomerThrowingException() throws Exception {
		System.out.println("..Inside addCustomerThrowingException target
object method...");
		throw new Exception("Ops!! Something is wrong!");

	}

	public void addCustomerAround(String name,String phone_no) {
		System.out.println("..Inside addCustomerAround target object
method...");
		System.out.println("Name is:"+name);
		System.out.println("Mobile No is:"+phone_no);
	}
}
```

Spring Bean: LoggingAspect (Aspect class)

```
package com.spring.example;

import java.util.Arrays;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class LoggingAspect {

	//Before Advice
	@Before("execution(* com.spring.example.CustomerBO.addCustomer(..))")
//pointcut expression
	public void logBefore(JoinPoint joinpoint){

		System.out.println("Aspect logBefore is running..");
		System.out.println("Currently processing Business
method:"+joinpoint.getSignature().getName());
		System.out.println("---------------------------------------");
	}

	//After Advice
	@After("execution(* com.spring.example.CustomerBO.addCustomer(..))")
//pointcut expression
	public void logAfter(JoinPoint joinpoint){
		System.out.println("Aspect logAfter is running..");
```

```
            System.out.println("Processed Business method
was:"+joinpoint.getSignature().getName());
            System.out.println("---------------------------------------");
      }

      //AfterReturning Advice
      @AfterReturning(
                  pointcut="execution(*
com.spring.example.CustomerBO.addCustomerReturnValue(..))",
                  returning="result") //pointcut expression with returning
value
      public void logAfterReturning(JoinPoint joinpoint,Object result){

            System.out.println("Aspect logAfterReturning is running..");
            System.out.println("Processed Business Method
was:"+joinpoint.getSignature().getName());
            System.out.println("Returned value by Processed Business Method
is:"+result);
            System.out.println("---------------------------------------");

      }

      //AfterThrowing Advice
      @AfterThrowing(
                  pointcut="execution(*
com.spring.example.CustomerBO.addCustomerThrowingException(..))",
                  throwing="error") //pointcut expression with returning
exception
      public void logAfterThrowing(JoinPoint joinpoint,Throwable error){
            System.out.println("Aspect logAfterThrowing is running..");
            System.out.println("Processed Business Method
was:"+joinpoint.getSignature().getName());
            System.out.println("Exception Thrown by Processed Business Method
is:"+error);
            System.out.println("---------------------------------------");
      }

      //Around Advice
      @Around("execution(*
com.spring.example.CustomerBO.addCustomerAround(..))") //pointcut expression
      public void logAround(ProceedingJoinPoint prdJointPoint) throws
Throwable{
            System.out.println("Aspect logAround is running..");
            System.out.println("Business
Method:"+prdJointPoint.getSignature().getName());
            System.out.println("Business Method arguments
value:"+Arrays.toString(prdJointPoint.getArgs()));

            System.out.println("Before Business Method Processing..");
            prdJointPoint.proceed();
            System.out.println("After Business Method Processed");
      }
}
```

Spring Bean Configuration File: spring.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">

    <aop:aspectj-autoproxy />

    <bean id="customerBo" class="com.spring.example.CustomerBoImpl" />

    <!-- Aspect -->
    <bean id="logAspect" class="com.spring.example.LoggingAspect" />

</beans>
```

Spring Demo Class: SpringAOPDemo (main method)

```
package com.spring.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringAOPDemo {


      public static void main(String[] args) {

      ApplicationContext appcontext=new
      ClassPathXmlApplicationContext("spring.xml");

            //Here only business method call from main method.
            CustomerBO customer=(CustomerBO)appcontext.getBean("customerBo");

            customer.addCustomer();

            customer.addCustomerReturnValue();

            try {
                  customer.addCustomerThrowingException();
            } catch (Exception e) {
                  e.printStackTrace();
            }

            customer.addCustomerAround("Anil Kumar","+918860543130");
      }

}
```

Run: Before Advice

```
ApplicationContext appcontext=new ClassPathXmlApplicationContext("spring.xml");

CustomerBO customer=(CustomerBO)appcontext.getBean("customerBo");

customer.addCustomer();
```

OUTPUT:

```
Aspect logBefore is running..
Currently processing Business method:addCustomer
---------------------------------------
..Inside addCustomer target object (business object) method...
```

Run: AfterReturn Advice

```
ApplicationContext appcontext=new ClassPathXmlApplicationContext("spring.xml");

CustomerBO customer=(CustomerBO)appcontext.getBean("customerBo");

customer.addCustomerReturnValue();
```

OUTPUT:

```
..Inside addCustomerReturnValue target object method...
Aspect logAfterReturning is running..
Processed Business Method was:addCustomerReturnValue
Returned value by Processed Business Method is:Customer Info added
successfully!!
---------------------------------------
```

Run: AfterThrowing Advice

```
ApplicationContext appcontext=new ClassPathXmlApplicationContext("spring.xml");

CustomerBO customer=(CustomerBO)appcontext.getBean("customerBo");

try {
    customer.addCustomerThrowingException();
} catch (Exception e) {
    e.printStackTrace();
}
```

OUTPUT:

```
..Inside addCustomerThrowingException target object method...
Aspect logAfterThrowing is running..
Processed Business Method was:addCustomerThrowingException
Exception Thrown by Processed Business Method is:java.lang.Exception: Ops!!
Something is wrong!
---------------------------------------
java.lang.Exception: Ops!! Something is wrong!
```

Run: Around Advice

```
ApplicationContext appcontext=new ClassPathXmlApplicationContext("spring.xml");

CustomerBO customer=(CustomerBO)appcontext.getBean("customerBo");

customer.addCustomerAround("Anil Kumar","+918860543130");
```

OUTPUT:

```
Aspect logAround is running..
Business Method:addCustomerAround
Business Method arguments value:[Anil Kumar, +918860543130]
Before Business Method Processing..
..Inside addCustomerAround target object method...
Name is:Anil Kumar
Mobile No is:+918860543130
After Business Method Processed
```

Pointcut Regular expression method matching sample:

Matching all public methods in EmployeeManager
execution(public * EmployeeManager.*(..))

Matching all public methods in EmployeeManager with return type EmployeeDTO
execution(public EmployeeDTO EmployeeManager.*(..))

Matching all public methods in EmployeeManager with return type EmployeeDTO and first parameter as EmployeeDTO
execution(public EmployeeDTO EmployeeManager.*(EmployeeDTO, ..))

Matching all public methods in EmployeeManager with return type EmployeeDTO and definite parameters
execution(public EmployeeDTO EmployeeManager.*(EmployeeDTO, Integer))

Matching all methods defined in classes inside package com.spring.example
within(com.spring.example.*)

Match all methods with a class in same package
within(EmployeeManagerImpl)

Match method with any return type and having any arguments
execution(* com.spring.example.EmployeeManager.addEmployee(..))

By: Anil Kumar