12. Spring boot AOP (Aspect Oriented Programming)

AOP

- In simple term, it helps to intercept the method invocation. And we can perform some task before and after the method.
- **AOP** allow us to focus on business logic by handling boilerplate and repetitive code like logging, transaction management etc.

>>>> What I mean to say .

If we have multiple business logic part and we want some logging for the same.

>>>> then we have to apply logging for each and every business logic

>>>>Same goes for transactional let's begin transaction and commit or rollback or end the transaction.

>>>> we have to do for multiples times for every business logic.

AOP Can handle all these part .

- So, **Aspect** is a module which handle this repetitive or boilerplate code.
- Helps in achieving reusability, maintainability of the code.

Used during:

- Logging
- Transactional Management
- Security etc..

>>If we want logging on 100 of places you can just put it into AOP module

>>it will handle it whether you want to put this check before or after the business logic.

>>changes at one place and impact will show at every places.

Dependency you need to add in pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

Simple Exa

```
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {
    @Before("execution(public String
com.springProject.SpringCourseProject.controller.EmployeeController.fetchEmployee())")
    public void beforeMethod(){
//ADVICE
        System.out.println("Inside Before Method Aspect");
    }
}
```

```
package com.springProject.SpringCourseProject.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @GetMapping(path="/fetchEmployee")
    public String fetchEmployee(){
        System.out.println("Employee Fetched.");
        return "Employee Fetched.";
    }
}
```

after hintting:http://localhost:8080/api/fetchEmployee

output:

Inside Before Method Aspect

Employee Fetched.

- A class annotated with @Aspect Spring boot get's to understand that this class has contains list of functions which could execute @before and @after as Business logic

Pointcut

-("execution(public String com.springProject.SpringCourseProject.controller.EmployeeController.fetchEmployee())")

- Its an Expression, which tells where an **ADVICE[METHOD DEFINATION]** should be applied.
- Pointcut helps to identify which business logic method or class is going to execute this advice whether @Before and @After.

1.POINTCUT EXECUTION

1. *** >>> Wildcard STAR**

("execution(public String com.springProject.SpringCourseProject.controller.EmployeeController.fetchEmployee())")

(*) wildcard : matches any single item

- Matches any return type
- **' * '** is combination of return type and access modifier's which matches any single item.
- **' * '** it will check each and every method which contains modifier's like public, protected and private and return type will be ignored.

```
 @Before("execution( *
com.springProject.SpringCourseProject.controller.EmployeeController.fetchEmployee())")
    public void beforeMethod(){
        System.out.println("Inside Before Method Aspect");
    }
```

"execution(* com.springProject.SpringCourseProject.controller.EmployeeController.*(String))"

- Matches any method with single parameter String
- Method Name will be ignored because we have put **' * '** at the place of method name.

```
 @Before("execution( *
com.springProject.SpringCourseProject.controller.EmployeeController.*(String))")
```

```
    public void beforeMethod(){
        System.out.println("Inside Before Method Aspect");
    }
```

"execution(String com.springProject.SpringCourseProject.controller.EmployeeController.fetchEmployee(*))"

- Matches fetchEmployee method that take any single parameter.

2. (..) wildcard : matches 0 or More item

"execution(String com.springProject.SpringCourseProject.controller.EmployeeController.fetchEmployee(..))")

- Matches fetchEmployee method that take any 0 or More parameters

Match this method

```
@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @GetMapping(path="/fetchEmployee")
    public String fetchEmployee(){
        System.out.println("Employee Fetched.");
        return "Employee Fetched.";
    }
}
```

"execution(String com.springProject.SpringCourseProject..fetchEmployee())")

```
@Before("execution( * com.springProject.SpringCourseProject..fetchEmployee(String))")
    public void beforeMethod(){
        System.out.println("Inside Before Method Aspect");
    }
```

- Matches fetchEmployee method in 'com.SpringCourseProject' package and subPackage classes.

"execution(String com.springProject.SpringCourseProject..*())")

- Matches any method in 'com.SpringCourseProject' package and subPackage classes.

2.POINTCUT WITHIN

Within : matches all method within any class or package.

- Here we are only giving class path and packagepath
- we are not pointing for specific method.

@Before("@within(com.springProject.SpringCourseProject.controller.EmployeeController")

- This pointcut will run for each method in the classEmployeeController.

@Before("@within(com.springProject.SpringCourseProject.controller..")

- (..) This pointcut will run for each method in this package and subpackage

```
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {
    //Any class which has this @Service annotations, All it's method match's will
happen.
    @Before("@within(org.springframework.stereotype.Service)")
    public void fetchEmployeeSalaryBeforeMethod(){
        System.out.println("Inside Before Method Aspect of
fetchEmployeeSalaryBeforeMethod");
    }

}
```

```
package com.springProject.SpringCourseProject.Services;

import org.springframework.stereotype.Service;

@Service
public class EmployeeUtil {

    public String fetchEmpSalary(){
        System.out.println("Employee Salary Fetched");
        return "Employee Salary Fetched.";
```

```
    }
}
```

```
package com.springProject.SpringCourseProject.controller;

import com.springProject.SpringCourseProject.Services.EmployeeUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    EmployeeUtil employeeUtil;

    @GetMapping(path="/fetchEmployeeSalary")
    public String fetchEmployeeSalary(){
        System.out.println("Inside Method of fetchEmployeeSalary of
EmployeeController.");
        return employeeUtil.fetchEmpSalary();
    }
}
```

Hitting API : http://localhost:8080/api/fetchEmployeeSalary

OUTPUT:

Inside Method of fetchEmployeeSalary of EmployeeController.

Inside Before Method Aspect of fetchEmployeeSalaryBeforeMethod

Employee Salary Fetched

3.POINTCUT @annotation :

- matches any that is annotated with given annotation.

```
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;

@Component
@Aspect
```

```
public class LoggingAspect {
    @Before("execution(*
com.springProject.SpringCourseProject.controller.EmployeeController.fetchEmployee())")
    public void beforeMethod(){
        System.out.println("Inside Before Method Aspect");
    }
    @Before("@annotation(org.springframework.web.bind.annotation.GetMapping)")
    public void beforeMethodForAnnotations(){
        System.out.println("Inside Before Method Aspect of beforeMethodForAnnotations");
    }

}
```

```
package com.springProject.SpringCourseProject.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {
    @GetMapping(path="/fetchEmployee")
    public String fetchEmployee(){
        System.out.println("Employee Fetched.");
        return "Employee Fetched.";
    }
}
```

Hitting API : http://localhost:8080/api/fetchEmployee

OUTPUT :

Inside Before Method Aspect

Inside Before Method Aspect of beforeMethodForAnnotations

Employee Fetched.

4.POINTCUT args :

@Before("args(String,int)")

- Args: matches any method with particular arguments (or parameters)
- If instead of primitive type, we need object, then we can give like this.

>>>>>**@Before("args(com.springProject.SpringCourseProject.DTO.Employee)")**

```java
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {
    @Before("@annotation(org.springframework.web.bind.annotation.GetMapping)")
    public void beforeMethodForAnnotations(){
        System.out.println("Inside Before Method Aspect of beforeMethodForAnnotations");
    }

    @Before("args(String, int)")
    public void beforeMethodForArgs(){
        System.out.println("Inside Before Method Aspect of beforeMethodForArgs");
    }

}
```

```java
package com.springProject.SpringCourseProject.Services;

import org.springframework.stereotype.Service;

@Service
public class EmployeeUtil {

    public String fetchEmployeeNameAndSalary(String name,int sal){
        String result = "EmployeeName: "+name+" Employee Salary: "+String.valueOf(sal);
        System.out.println(result);
        return result;
    }
}
```

```java
package com.springProject.SpringCourseProject.controller;

import com.springProject.SpringCourseProject.Services.EmployeeUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping(value = "/api/")
```

```
public class EmployeeController {    @GetMapping(path="/fetchEmployeeNameAndSalary")
    public String fetchEmployeeNameAndSalary(){
        System.out.println("Inside Method of fetchEmployeeNameAndSalary of
EmployeeController.");
        return employeeUtil.fetchEmployeeNameAndSalary("SAURAV SAXENA",1200000);
    }
}
```

HITTING API : http://localhost:8080/api/fetchEmployeeNameAndSalary

OUTPUT:

Inside Before Method Aspect of fetchEmployeeSalaryBeforeMethod

EmployeeName: SAURAV SAXENA Employee Salary: 1200000

5. @args: @Before("@args(org.springframework.stereotype.Service)")

matches any method with particular parameters and that parmeter class is annotated with particular annotation In this case @Service.

```
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {
    @Before("@args(org.springframework.stereotype.Service)")
    public void beforeMethodForA(){
        System.out.println("Inside Before Method Aspect of
beforeMethodForArgsForAnnotationsServices");
    }

}
```

```
package com.springProject.SpringCourseProject.DTO;


import org.springframework.stereotype.Service;

@Service
//@Data
public class EmployeeDTO {
    String name;
```

```
    int sal;
    int EmpId;

    public void setName(String name) {
        this.name = name;
    }

    public void setSal(int sal) {
        this.sal = sal;
    }

    public void setEmpId(int empId) {
        EmpId = empId;
    }

    @Override
    public String toString() {
        return "EmployeeDTO{" +
                "name='" + name + '\'' +
                ", sal=" + sal +
                ", EmpId=" + EmpId +
                '}';
    }
}
```

not working for me

6. Target: Matches any method on particular instances of a class.

```
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {

    //Any class which has this @Service annotations, All it's method match's will
happen.
    @Before("@within(org.springframework.stereotype.Service)")
    public void fetchEmployeeSalaryBeforeMethod(){
        System.out.println("Inside Before Method Aspect of
fetchEmployeeSalaryBeforeMethod");
    }
   @Before("target(com.springProject.SpringCourseProject.Services.EmployeeUtil)")
    public void beforeMethodForTargetPointCut(){
```

```
        System.out.println("Inside Before Method Aspect of
beforeMethodForTargetPointCut");
    }

}
```

```
package com.springProject.SpringCourseProject.DTO;


import org.springframework.stereotype.Service;

@Service
//@Data
public class EmployeeDTO {
    String name;
    int sal;
    int EmpId;

    public void setName(String name) {
        this.name = name;
    }

    public void setSal(int sal) {
        this.sal = sal;
    }

    public void setEmpId(int empId) {
        EmpId = empId;
    }

    @Override
    public String toString() {
        return "EmployeeDTO{" +
                "name='" + name + '\'' +
                ", sal=" + sal +
                ", EmpId=" + EmpId +
                '}';
    }
}
```

```
package com.springProject.SpringCourseProject.Services;

import com.springProject.SpringCourseProject.DTO.EmployeeDTO;
import org.springframework.stereotype.Service;

@Service
public class EmployeeUtil {
```

```
    public String fetchEmployeeNameAndSalary(String name,int sal){
        String result = "EmployeeName: "+name+" Employee Salary: "+String.valueOf(sal);
        System.out.println(result);
        return result;
    }

    public String fetchEmployeeDTO(EmployeeDTO employeeDTO){
        System.out.println(employeeDTO.toString());
        return employeeDTO.toString();
    }
}
```

```
package com.springProject.SpringCourseProject.controller;

import com.springProject.SpringCourseProject.DTO.EmployeeDTO;
import com.springProject.SpringCourseProject.Services.EmployeeUtil;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {

    @Autowired
    EmployeeUtil employeeUtil;

    @PostMapping(path = "fetchEmployeeDTO")
    public String fetchEmployeeDTO(@RequestBody EmployeeDTO employeeDTO){
        System.out.println("Inside fetchEmployeeDTO Method Of Employee Controller");
        return employeeUtil.fetchEmployeeDTO(employeeDTO);
    }
}
```

OUTPUT:

Hitting API : http://localhost:8080/api/fetchEmployeeDTO

POST

PAYLOAD:{

"name":"SAURAV SAXENA",

"sal":120000,

"empId":1

}

Inside fetchEmployeeDTO Method Of Employee Controller

Inside Before Method Aspect of beforeMethodForTargetPointCut

Inside Before Method Aspect of fetchEmployeeSalaryBeforeMethod

Inside EmployeeUtil fetchEmployeeDTO details fetched: EmployeeDTO{name='SAURAV SAXENA', sal=120000, EmpId=1}

7. In Target we can also apply interface

@Before("target(com.springProject.SpringCourseProject.interfaces.IEmployee)")

```
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {
@Before("target(com.springProject.SpringCourseProject.interfaces.IEmployee)")
    public void beforeMethodForTargetPointCutAppliedInterface(){
        System.out.println("Inside Before Method Aspect of
beforeMethodForTargetPointCutAppliedInterface");
    }

}
```

```
package com.springProject.SpringCourseProject.controller;


import com.springProject.SpringCourseProject.interfaces.IEmployee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {
    @Autowired
    @Qualifier("TempEmployee")
    IEmployee iEmployeeObj;

    @GetMapping(path="/fetchEmployee")
    public String fetchEmployee(){
        iEmployeeObj.fetchEmployeeMethod();
        return "Employee Fetched.";
    }
```

```
}
```

```
package com.springProject.SpringCourseProject.interfaces;

public interface IEmployee {
    public void fetchEmployeeMethod();
}
```

```
package com.springProject.SpringCourseProject.interfaceImple;

import com.springProject.SpringCourseProject.interfaces.IEmployee;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
@Qualifier("PermanentEmployee")
public class PermanentEmployee implements IEmployee {
    @Override
    public void fetchEmployeeMethod() {
        System.out.println("Inside fetchEmployeeMethod Of permanentEmployee");
    }
}
```

```
package com.springProject.SpringCourseProject.interfaceImple;

import com.springProject.SpringCourseProject.interfaces.IEmployee;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
@Qualifier("TempEmployee")
public class TempEmployee implements IEmployee {
    @Override
    public void fetchEmployeeMethod() {
        System.out.println("Inside fetchEmployeeMethod Of TempEmployee");
    }
}
```

OUTPUT:

HITING API: http://localhost:8080/api/fetchEmployee

Inside Before Method Aspect

Inside Before Method Aspect of beforeMethodForAnnotations

Inside Before Method Aspect of beforeMethodForTargetPointCutAppliedInterface
Inside fetchEmployeeMethod Of TempEmployee

8. Combining two pointcuts using: [&& boolean AND] and [|| boolean OR]

```
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {
    @Before("execution(*
com.springProject.SpringCourseProject.controller.EmployeeController.*())"
            +
            "&& @within(org.springframework.web.bind.annotation.RestController)"
     )
    public void beforeAndMethod(){
        System.out.println("Inside beforeAndMethod Method Aspect");
    }

    @After("execution(*
com.springProject.SpringCourseProject.controller.EmployeeController.*())"
            +
            "|| @within(org.springframework.stereotype.Component)"
    )
    public void afterORMethod(){
        System.out.println("Inside afterORMethod Method Aspect");
    }

}
```

```
package com.springProject.SpringCourseProject.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {
    @GetMapping(path="/fetchEmployee")
    public String fetchEmployee(){
```

```
        System.out.println("Employee Fetched.");
        return "Employee Fetched.";
    }
}
```

hitting API: http://localhost:8080/api/fetchEmployee

OUTPUT

Inside Before Method Aspect of beforeMethodForAnnotations

Employee Fetched.

Inside afterORMethod Method Aspect

9. Named Pointcuts

```
package com.springProject.SpringCourseProject.AOP;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class LoggingAspect {
    @Pointcut("execution(*
com.springProject.SpringCourseProject.controller.EmployeeController.*())")
    public void customPointCutName(){
        //always stays empty
    }

    @Before("customPointCutName()")
    public void customPointcutBeforeMethod(){
        System.out.println("inside before customPointcutBeforeMethod");
    }

}
```

```
package com.springProject.SpringCourseProject.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.web.bind.annotation.*;

@RestController
```

```
@RequestMapping(value = "/api/")
public class EmployeeController {
    @GetMapping(path="/fetchEmployee")
    public String fetchEmployee(){

        System.out.println("Employee Fetched.");
        return "Employee Fetched.";
    }
}
```

hitting API: http://localhost:8080/api/fetchEmployee

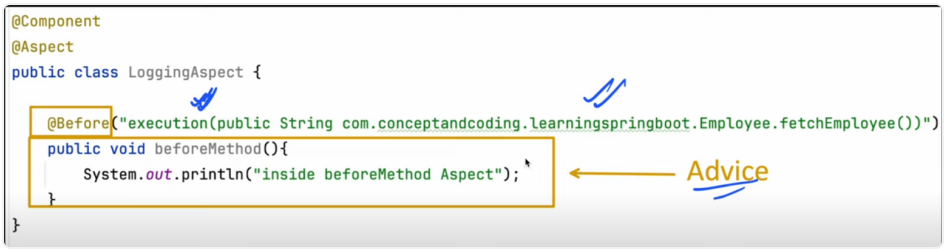
OUTPUT

inside before customPointcutBeforeMethod

Employee Fetched.

ADVICE

- It's action, which is taken @Before or @After or @Around the execution.



- @Around As the name says, it surrounds the method execution with before and after both
- Incase of @Before internally spring boot call the aspect method before calling the actual business logic
- Incase @After internally call after the execution of Business logic.
- But @Around we need to call business logic by applying **joinPoint.proceed();.**
- **joinPoint** Its generally considered a point, where actual method invocation happens.

```
@Around("execution(*
com.springProject.SpringCourseProject.controller.EmployeeController.*())")
public void AroundMethod(ProceedingJoinPoint joinPoint) throws Throwable {
    System.out.println("Inside Before Method Aspect of AroundMethod");
    joinPoint.proceed();
    System.out.println("Inside After Method Aspect of AroundMethod");
}
```

```
package com.springProject.SpringCourseProject.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping(value = "/api/")
public class EmployeeController {
    @GetMapping(path="/fetchEmployee")
    public String fetchEmployee(){

        System.out.println("Employee Fetched.");
        return "Employee Fetched.";
    }
}
```

hitting API: http://localhost:8080/api/fetchEmployee

OUTPUT

Inside Before Method Aspect of AroundMethod

Employee Fetched.

Inside After Method Aspect of AroundMethod

By now, few questions we all should have.

1. How this interception works?.
2. What if we have 1000s of pointcut, so whenever I Invoke method does matching happens with 100s of pointcut.

>>Matching with 1000s of pointcut . will cost application latency.

Let's understand the AOP flow, to get an answer of above doubts.

1. When Application startup happens

- Look for @Aspest annotation classes
- Parse the pointcut Expression

  >>>Done by PointcutParser.java class
- Stored in efficient data structure or cache after parsing.

  >>So it be easy for matching.
- Look for @Component, @Service @Conroller etc.

- For each class, it check if it's eligible for interceptions based on Pointcut expression
  >>Done By AbstractAutoProxyCreator.java class
- If yes, it creates a Proxy using **JDK Dynamic proxy** or **CGLIB[Code generation Library PROXY]** proxy this proxy class, has code, which execute advice before the method, then method execution happens and after than advice if any.

When to use JDK Dynamic proxy or CGLIB?

- When a class is implemented with some interface, then it's already a child class it's uses **JDK Dynamic proxies**
- If class is not child of interface CGLIB has capability to create sub class at this it's uses **CGLIB**.

- Note When you hitting your actual business logic Like In our case it's EmployeeUtil class
- spring boot checked if there is a proxy created for this fetchEmployeeNameAndSalary() method of EmployeeUtil class.
- Internally might Proxy class has invoked which is created at the time of application startup.
- EmployeeUtil class is not child of any class so it's uses CGLIB For creating proxy of EmployeeUtil

Detailed Expanation https://notebook.zohopublic.in/public/notes/dcr5z12bd30d9fbc2454e9881904115682227 for above AOP Internal working.