AOP

#01. 프로젝트 구성

1. 패키지 정보

`kr.hossam.aop`

항목	설정 내용
GroupId	`kr.hossam`
ArtifactId	`aop`

2. 의존성 설정

프로젝트 생성 과정에서 `dependencies` 항목에 대해 아래의 세 항목을 선택

항목 이름	구분
Spring Boot DevTools	기존 사용
Spring Boot Acturator Ops	기존 사용
Spring Web	기존 사용
Thymeleaf Template Engines	기존 사용
Lombok	신규 추가

3. 추가 설정

1) logback 설정파일 추가

`/src/main/resources/logback-spring.xml` 파일을 추가

(이전 수업에서 생성한 파일을 재사용)

2) UserAgent 라이브러리 설정

`build.gradle`에 의존성 설정 추가

```
implementation 'com.github.ua-parser:uap-java:1.6.1'
```

3) 기본 패키지 추가

패키지	기능
`kr.hossam.aop.controllers`	컨트롤러 클래스

패키지	기능
kr.hossam.aop.helpers	헬퍼 클래스
kr.hossam.aop.interceptors	인터셉터 클래스
kr.hossam.aop.services	비즈니스 로직 인터페이스
kr.hossam.aop.services.impl	비즈니스 로직 구현체

4. 파일 구성

```
├─java
│  └─kr
│      └─hossam
│          └─aop
│              │  AopApplication.java                (기본파일, 수정없음)
│              │  MyWebConfig.java                   (재사용, 수정있음)
│              │
│              ├─controllers
│              │      HomeController.java            (신규)
│              │
│              ├─helpers
│              │      UtilHelper.java                (재사용, 수정있음)
│              │
│              ├─interceptors
│              │      MyInterceptor.java             (재사용, 수정있음)
│              │
│              └─services
│                  │  MyCalcService.java             (신규)
│                  │
│                  └─impl
│                         MyCalcServiceImpl.java    (신규)
│
└─resources
    │  application.properties   (기본파일, 수정없음)
    │  logback-spring.xml       (재사용)
    │
    ├─static
    │  │  favicon.ico           (재사용)
    │  └─ robots.txt            (재사용)
    │
    └─templates
           index.html           (신규)
```

#02. Spring의 객체 의존성 주입

Interface로 비즈니스로직을 나열하고 이에 대한 구현체 클래스(~~~Impl)를 정의해 두면 스프링이 Interface에 대한 선언문을 통해 구현체의 객체 생성 및 할당을 자동으로 처리함

1) 인터페이스 정의

MyCalcService.java (실습)

```
package kr.hossam.aop.services;

public interface MyCalcService {
    public int plus(int x, int y);
    public int minus(int x, int y);
}
```

2) 인터페이스 구현체 정의

모든 구현체 클래스 정의문 위에는 @Service가 반드시 명시되어야 한다.

MyCalcServiceImpl.java (실습)

```
package kr.hossam.aop.services.impl;

import org.springframework.stereotype.Service;

import lombok.extern.slf4j.Slf4j;
import kr.hossam.aop.services.MyCalcService;

@Slf4j
@Service    // <-- 비즈니스 로직을 구현하는 모든 구현체 클래스에 명시
public class MyCalcServiceImpl implements MyCalcService {
    // 이 객체가 생성되었음을 확인하기 위해 생성자 정의함
    // 보통의 Service 구현체는 생성자를 정의하지 않음
    public MyCalcServiceImpl() {
        log.debug("MyCalcServiceImpl() 생성자 호출됨!");
    }

    @Override
    public int plus(int x, int y) {
        return x + y;
    }

    @Override
    public int minus(int x, int y) {
        return x - y;
    }
}
```

3) 컴포넌트 스캔

원래는 환경설정 클래스에 아래와 같이 지정

```
@ComponentScan({"패키지1", "패키지2", ... "패키지n"})
```

최신 Spring의 메인 클래스에 적용되어 있는 `@SpringBootApplication`이 `@ComponentScan`을 상속받고 있기 때문에 메인 클래스 하위의 모든 패키지가 자동으로 스캔된다.

> 결국은 VSCode 환경으로 SpringBoot를 사용할 경우 추가적으로 해야 하는 작업이 없다는 의미

4) 의존성 주입 받기

객체 주입

컨트롤러 클래스에서 멤버변수로 인터페이스의 객체를 선언하고 `@Autowired`를 명시한다.

```
@Autowired
private MyCalcService myCalcService;
```

생성자를 통한 의존성 주입

컨트롤러 클래스에서 `@Autowired`없이 선언한 인터페이스의 객체를 파라미터로 전달받는 생성자를 정의

생성자 상단에 `@Autowired`를 명시하여 Spring으로부터 객체를 주입받음

```
public class FooController {
    private MyCalcService myCalcService;

    @Autowired
    public FooController(MyCalcService myCalcService) {
        this.myCalcService = myCalcService;
    }
}
```

Lombok을 통한 자동 의존성 주입

Lomok이 멤버변수를 파라미터로 갖는 생성자를 자동으로 만들어주는 기능을 활용한다.

단, 객체를 자동으로 주입받을 경우 상수 형태로 선언해야 한다.

> 결국은 이 작업만 하면 된다는 뜻

```
@Controller
@RequiredArgsConstructor
public class FooController {
    private final MyCalcService myCalcService;
}
```

AopController.java (실습)

```java
package kr.hossam.aop.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import lombok.RequiredArgsConstructor;
import kr.hossam.aop.services.MyCalcService;

@Controller
@RequiredArgsConstructor
public class AopController {
    private final MyCalcService myCalcService;

    // '/home' 요청이 들어오면 실행되는 메서드
    @GetMapping("/home")
    public String home(Model model) {

        int value1 = myCalcService.plus(100, 200);
        int value2 = myCalcService.minus(200, 300);

        model.addAttribute("value1", value1);
        model.addAttribute("value2", value2);

        return "home";
    }
}
```

home.jsp (실습)

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page trimDirectiveWhitespaces="true"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>home</title>
</head>
<body>
    <h1>Hello, home</h1>
    <p>value1: ${value1}</p>
    <p>value2: ${value2}</p>
</body>
</html>
```