8주차 발표_민서연

Spring : AOP

01 Spring

엔터프라이즈용 Java 애플리케이션 개발을 편하게 할 수 있게 해주는
오픈 소스 경량급 프레임워크

경량급: 스프링을 사용했을때에 개발자가 작성해야 할 코드가 상대적으로 단순하다는 것
불필요한 복잡한 코드 제거하여 복잡성 낮춤.

01 Spring

특징!

POJO 프로그래밍 지향

01 Spring

POJO 프로그래밍 지향

POJO: Plain Old Java Object
특정 환경에 종속적이지 않으며,
순수 java 만을 통해서 생성한 객체를 의미
= java 및 java의 스펙에 정의된 기술만 사용한다!
= 외부의 라이브러리나 외부의 모듈을 가져오면 X

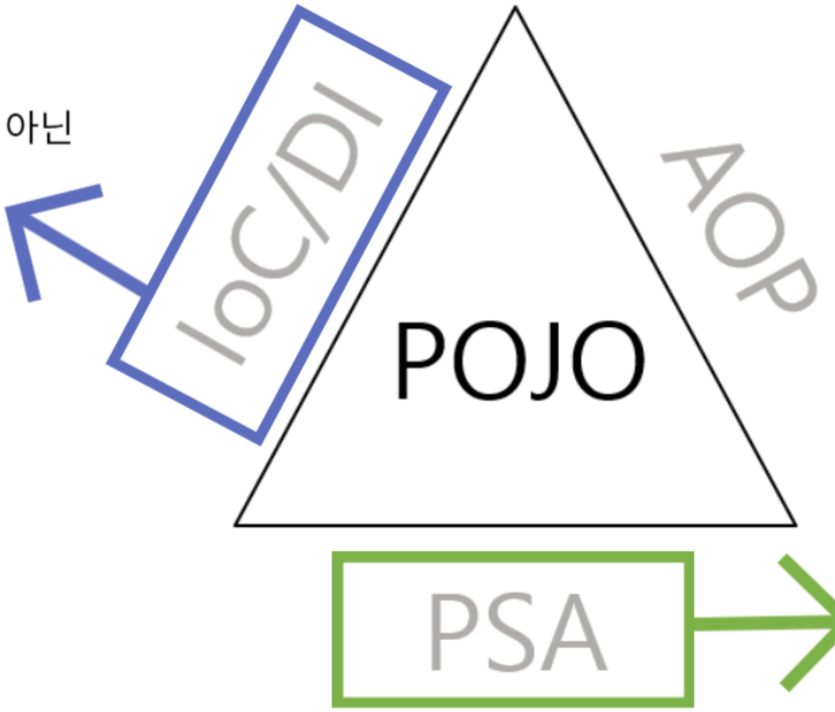
필요성

- 특정 환경이나 기술에 종속적이지 않으면 재사용이 가능하고 확장 가능한 유연한 코드 작성이 가능함
- 저수준 레벨의 기술과 환경에 종속적인 코드를 제거하여 코드가 간결해지고 디버깅하기도 쉬움
- 객체 지향적인 설계를 제한 없이 적용할 수 있음

01 Spring

POJO 프로그래밍 지향

- IoC
: Inversion of Control
제어의 역전
메소드, 객체 호출 작업을 개발자가 아닌
외부에서 결정
- DI
: Dependency Injection
객체를 직접 생성하는 것이 아니라
외부에서 생성한 후 주입

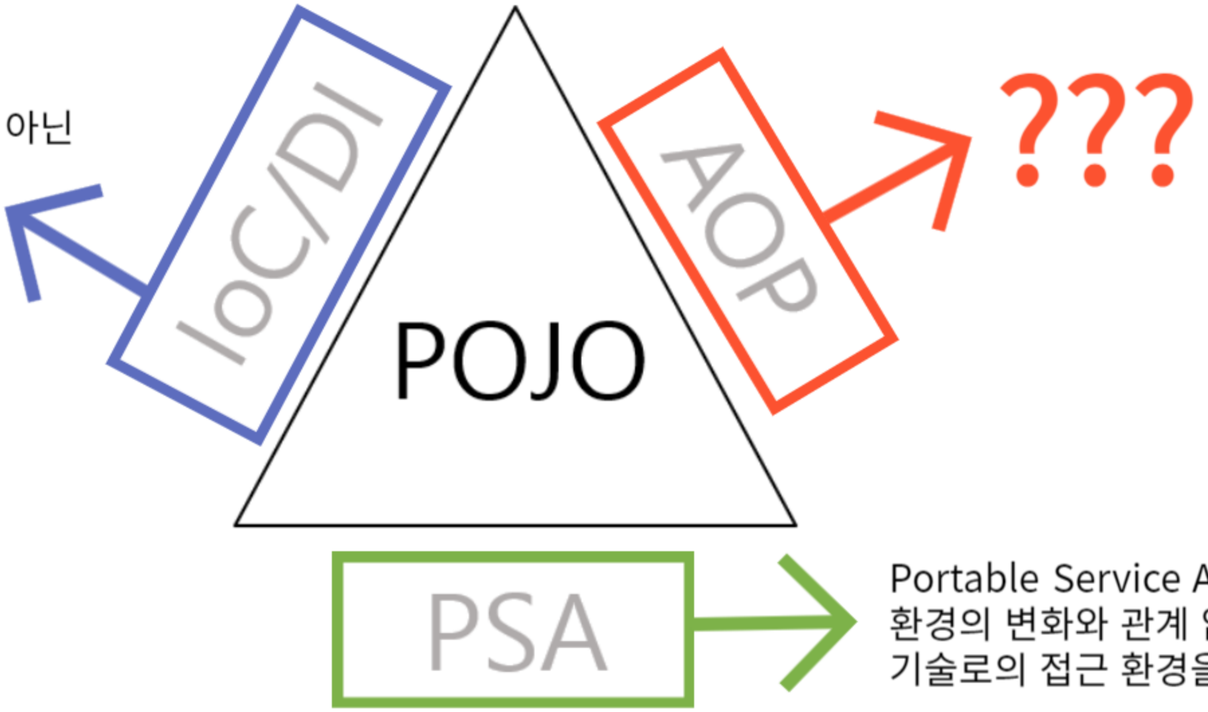


Portable Service Abstraction
환경의 변화와 관계 없이 일관된 방식의
기술로의 접근 환경을 제공하는 추상화 구조

01 Spring

POJO 프로그래밍 지향

- IoC
: Inversion of Control
제어의 역전
메소드, 객체 호출 작업을 개발자가 아닌
외부에서 결정
- DI
: Dependency Injection
객체를 직접 생성하는 것이 아니라
외부에서 생성한 후 주입



Portable Service Abstraction
환경의 변화와 관계 없이 일관된 방식의
기술로의 접근 환경을 제공하는 추상화 구조

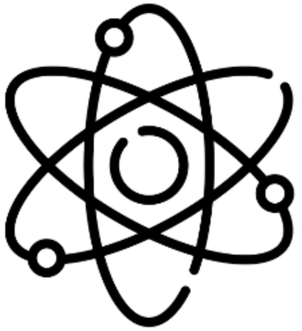
02 AOP

Aspect Oriented Programing

어떤 로직을 기준으로 핵심적인 관점, 부과적인 관점으로 나누어서 보고
그 관점을 기준으로 각각 모듈화 하겠다는 것
= 관점 지향 프로그래밍

**모듈화
어떤 공통된 로직이나 기능을 하나의 단위로 묶는 것

02 AOP



핵심 비지니스 로직

적용하고자 하는 핵심적인 관점

부가적인 로직

핵심 로직을 실행하기 위한 것들
(DB 연결, 로깅, 파일 입출력 등)

02 AOP

AOP

인프라 혹은 부가기능의 모듈화
ex) 모니터링 및 로깅, 동기화, 오류 검사 및 처리
각각의 모듈들의 주 목적 외에 필요한 부가적인 기능들

OOP

비즈니스 로직의 모듈화
모듈화의 핵심 단위는 비즈니스 로직

공통된 기능을 재사용하는 기법

이때 OOP는 재사용 하는 방법으로 상속이나 위임을 사용하는데,
전체적 애플리케이션에서 여기저기 사용되는 부가기능을 상속이나 위임으로 처리하기엔 깔끔한 모듈화가 어려움!

그래서~ AOP가 나옴!

02 AOP

주요 개념

Aspect
: 흩어진 관심사를 모듈화 한 것. 주로 부가 기능을 모듈화 함

Target
: Aspect를 적용하는 곳 (클래스, 메서드 등등)

Advice
: 실질적으로 어떤 일을 해야할 지에 대한 것, 실질적인 부가기능을 담은 구현체

Join Point
: Advice가 적용될 위치, 끼어 들 수 있는 지점.
  메서드 진입 지점, 생성자 호출 지점 등등

PointCut
: Join Point의 상세한 스펙을 정의한 것

02 AOP

특징

- 프록시 패턴 기반의 AOP 구현체
  -> 프록시 객체를 쓰는 이유: 접근 제어 및 부가기능을 추가하기 위해

- 스프링 빈에만 AOP 적용 가능

- 모든 AOP 기능을 제공하는 것이 아닌 스프링 IoC와 연동하여
  엔터프라이즈 애플리케이션에서 가장 흔한 문제에 대한 해결책 지원

02 AOP

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

AOP 어노테이션을 사용하기 위해서는
위와 같은 의존성을 추가해야함!

02 AOP

주요 어노테이션

@Aspect
: 해당 클래스를 Aspect로 사용하겠다는 것을 명시

@Before
: 대상 메서드가 실행되기 전에 Advice를 실행

@AfterReturning
: 대상 메서드가 정상적으로 실행되고 반환된 후에 Advice를 실행

@AfterReturning
: 대상 메서드가 정상적으로 실행되고 반환된 후에 Advice를 실행

@After
대상 메서드가 실행된 후에 Advice 실행