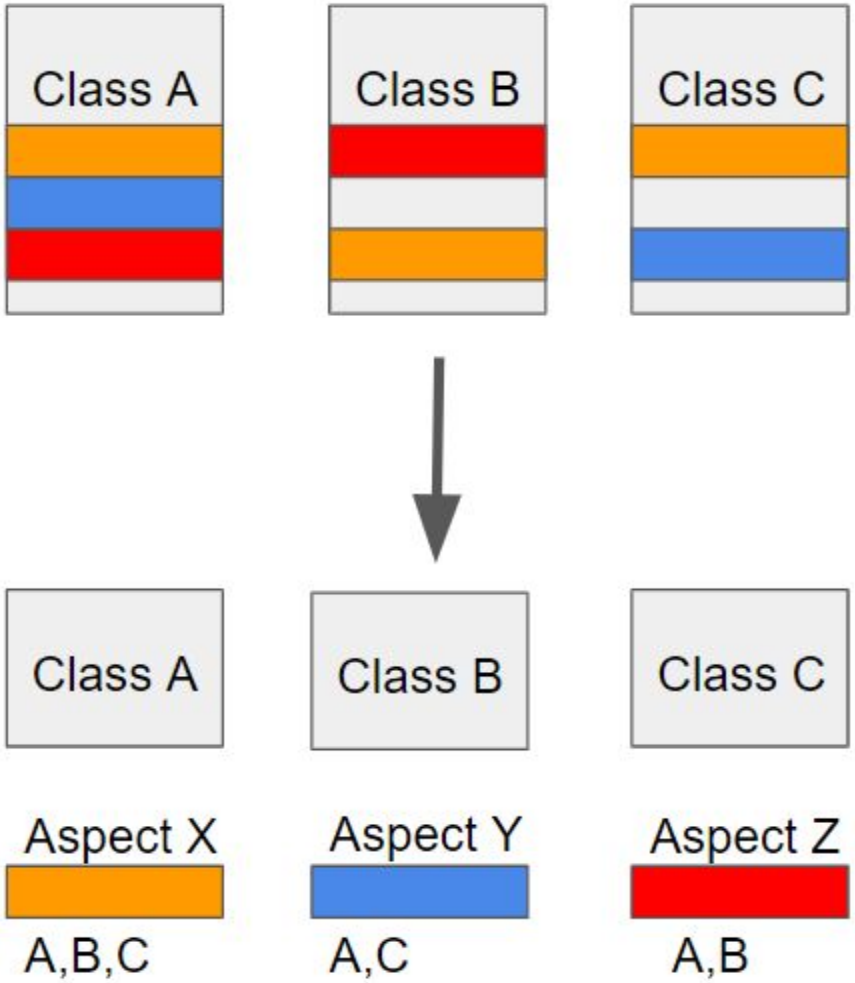
About Spring Framework

AOP IoC POJO DI 스프링삼각형



1.관점지향 프로그래밍

2.어떤 로직을 기준으로 핵심적인 관점, 부가적인 관점으로 나누어 그 관점을 기준으로 모듈화
ex)부가적:DB연결,로깅, 파일 입,출력

3. 흩어진 관심사를 Aspect로 모듈화 하고 핵심적인 비즈닉스 로직에서 분리하여 재사용 하는 목적

AOP 주요 개념

- **Aspect :** 위에서 설명한 흩어진 관심사를 모듈화 한 것. 주로 부가기능을 모듈화 함.
- **Target : Aspect**를 적용하는 곳 **(**클래스, 메서드 **..)**
- **Advice :** 실질적으로 어떤 일을 해야할 지에 대한 것, 실질적인 부가기능을 담은 구현체
- **JointPoint : Advice**가 적용될 위치, 끼어들 수 있는 지점. 메서드 진입 지점, 생성자 호출 시점, 필드에서 값을 꺼내올 때 등 다양한 시점에 적용가능
- **PointCut : JointPoint**의 상세한 스펙을 정의한 것. **'A**란 메서드의 진입 시점에 호출할 것'과 같이 더욱 구체적으로 **Advice**가 실행될 지점을 정할 수 있음

스프링 **@AOP**를 사용하기 위해서는 다음과 같은 의존성을 추가해야 한다.

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

```java
@Component
@Aspect
public class PerfAspect {

    @Around("execution(* com.saelobi..*.EventService.*(..))")
    public Object logPerf(ProceedingJoinPoint pjp) throws Throwable{
        long begin = System.currentTimeMillis();
        Object retVal = pjp.proceed(); // 메서드 호출 자체를 감쌈
        System.out.println(System.currentTimeMillis() - begin);
        return retVal;
    }
}
```

Around 어노테이션은 타겟 메서드를 감싸서 특정 Advice를 실행한다는 의미.
실행된 시간을 측정하기 위한 로직 구현
com.saelobi..*.EventService.*(..)) -> 패키지 경로 , EventService 객체의 모든 메소드에 이 Aspect를 적용한다는 의미

```java
public interface EventService {

    void createEvent();

    void publishEvent();

    void deleteEvent();
}
```

```java
@Component
public class SimpleEventService implements EventService {

    @Override
    public void createEvent() {
        try {
            Thread.sleep(1000);
        } catch(InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Created an event");
    }

    @Override
    public void publishEvent() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e){
            e.printStackTrace();;
        }
        System.out.println("Published an event");
    }

    public void deleteEvent() {
        System.out.println("Delete an event");
    }
}
```

```java
@Service
public class AppRunner implements ApplicationRunner {

    @Autowired
    EventService eventService;

    @Override
    public void run(ApplicationArguments args) throws Exception {
        eventService.createEvent();
        eventService.publishEvent();
        eventService.deleteEvent();
    }
}
```

```
Created an event
1003
Published an event
1000
Delete an event
0
```

의존성 주입(Dependency Injection)

- 어떤 객체에 Spring Container가 또 다른 객체와 의존성을 맺어주는 행위

의존성을 주입하지 않은 경우
클래스 내부에서 Gun객체를 생성했기 때문에
autowired

<Gun.java>

```
public class Gun {
    ...
    ...
}
```

<Soldier.java>

```
public class Soldier {
    private Gun gun;

    public Soldier() {
        gun = new Gun();
    }
}
```

의존성을 주입 한 경우
컨테이너에 Bean으로 등록하고

<Gun.java>

```
@Component // 스프링 컨테이너에 Bean으로 등록
public class Gun {
    ...
    ...
}
```

<Soldier.java>

```
public class Soldier {
    @Autowired // 스프링 컨테이너에 있는 Gun 타입의 Bean을 주입
    private Gun gun;
}
```

제어의 역전

- 의존성 주입의 상위개념
- 객체의 생성부터 소멸까지 객체의 모든 생명주기를 개발자가 아닌 컨테이너가 담당
- 스프링 컨테이너가 필요에 따라 개발자 대신 Bean들을 관리해주는 행위
- 일반적인 상황에서는 개발자가 직접 객체를 제어한다.
  ex) new 연산자를 통해 객체를 생성
- 스프링에서는 xml 혹은 어노테이션 방식으로 스프링 컨테이너에 Bean객체를 등록하기만 하면, 스프링 컨테이너에서 Bean의 생명주기(생성 -> 의존성 설정 -> 초기화 -> 소멸)를 전부 관리 해준다.
- 객체애 대한 제어권이 컨테이너로 역전되기 때문에 제어의 역전이라고 한다.

장점

- 개발자는 객체 관리에 덜 신경쓸 수 있게 되어 다른 부분에 더 집중 할 수 있다.
- 약한 결합을 이용하여 객체 간 의존 관계를 쉽게 변경 할 수 있다.
- 즉 코드의 재사용성과 유지보수성을 높인다

클라이언트가 서버에 웹 페이지를 요청하면 서블릿 컨테이너에서 사용자 요청 별 스레드를 생성하고, URL에 매핑된 서블릿 객체를 만들어서 파라미터 값을 전달해주며 실행 시킨다.
(서블릿 컨테이너가 서블릿 안의 로직이 전개될 수 있기까지의 과정을 대신 해주기 때문이다)

스프링도 이와 같이 컨테이너가 존재하며 싱글톤 패턴의 객체 생성을 지향한다.

즉 빈 객체는 하나만 생성되어 계속 재사용된다.

의존도를 높이고 결합도를 낮춘다.

POJO(Plain Old Java Object)

- @Configuration, @Bean을 붙여 자바 구성 클래스를 만들거나
- @Component, @Service, @Controller를 붙여 자바 컴포넌트를 구성
- 복잡하고 객체지향의 개념을 흐리게 만드는 EJB 대신 객체지향적인 원리로 만들어진 java 언어의 개념에 충실하게 여러 논리를 구현할 수 있는 방법이 POJO이다.
- jquery가 많이 무거워져 순수 자바스크립트인 바닐라js로 돌아가는것과 비슷한 양상
- impl, extends를 최소화(의존도를 낮추기위해)
- 특정 규약에 종속되지 않는다.
- 환경이나 기술에 종속되지 않는다.
- 원리를 철저히 준수한다.
- 대표적인 예시가 getter/setter로 이루어진 java bean이다.

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import com.example.demo.SequenceGenerator;

@Configuration
public class SequenceGeneratorConfiguration {

    @Bean
    public SequenceGenerator sequenceGenerator() {

        SequenceGenerator seqgen = new SequenceGenerator();
        seqgen.setPrefix("30");
        seqgen.setSuffix("A");
        seqgen.setInitial(100000);
        return seqgen;
    }
}
```