[1주차] 서버 기초

- **Spring Framework의 주요 특징**
  - **IOC(Inversion of Control) 컨테이너 소개**
    1. **IOC 란?**

       : 제어의 역전이라는 뜻으로, 개발자가 제어해야 할 요소들을 Spring Framework에서 대신 제어해준다는 것을 의미한다.

    2. **ApplicationContext**

       : IOC를 지원하기 위해 Spring에서 'ApplicationContext' 라는 컨테이너를 제공한다.

       ApplicationContext는 애플리케이션의 컴포넌트를 생성하고 조립하며, 객체의 라이프 사이클을 관리한다.

    3. **ApplicationContext의 과정**

       1) 객체의 생성 및 관리

          : ApplicationContext를 사용하여 빈(Bean)을 생성하고 관리한다. Bean은 Spring이 제어하며 개발자는 객체 생성과 관리를 직접 처리하지 않는다.

          ex) 개발자가 *@Service*, *@Repository*, *@Controller* 와 같은 어노테이션을 사용해 간단하게 객체를 정의하면 Spring이 알아서 이 객체들을 생성한다.

          또, Bean의 생성되고 초기화되며 애플리케이션 종료 시 소멸되는 모든 과정이 Spring에 의해 자동으로 처리된다.

       2) 의존성 관리

          : 객체 간의 의존성을 Spring이 주입(DI)한다. 객체가 필요로 하는 다른 객체를 직접 생성하거나 찾는 대신 Spring 컨테이너가 의존성을 주입해준다.

       3) 제어 흐름의 역전

          : 개발자가 코드의 제어 흐름을 결정하지 않고, 프레임워크가 객체의 라이프 사이클 및 실행 흐름을 관리한다.

- **DI(Dependency Injection)의 개념과 장점**

  1. **DI란?**

     : 객체 간의 의존성을 프레임워크가 주입하는 개념이다. 즉, 객체를 직접 생성하는 것이 아니라 외부에서 생성 후 주입시켜 주는 방식이라고 할 수 있다.

  2. 의존성 주입 방법

     ⇒ ***@Autowired라는 어노테이션을 통해 의존성 주입을 명시할 수 있다.***

     1) 생성자를 통한 의존성 주입 (Construct Injection)

     : 클래스의 생성자를 정의하고, 의존하는 객체를 매개변수로 받아 필드에 할당한다. 오직 생성자 주입만이 final 키워드를 사용할 수 있다. final 키워드를 사용해서 값이 한 번 할당되면 변경할 수 없기에 객체의 불변성이 보장된다.

```
@RestController
public class DIController {

    MyService myService;

    @Autowired
    public DIController(MyService myService) {
        this.myService = myService;
    }
}
```

     → 롬복 라이브러리 사용 시 @RequiredArgsConstructor 어노테이션을 사용해 간단하게 코드를 작성할 수 있다. (final 키워드가 붙은 주입에만 생성자를 만들어준다.)

```
@RequiredArgsConstructor
@RestController
public class DIController {

    private final MyService myService;
}
```

2) 필드 객체 선언을 통한 의존성 주입 (Field Injection)

: 주로 '@Autowired' 어노테이션을 사용하여 필드에 의존성을 주입한다.

```
@RestController
public class DIController {

    @Autowired
    private MyService myService;
}
```

간편하게 의존 관계 주입이 가능하지만 참조 관계를 눈으로 확인하기 어렵고, 순환 참조를 막을 수 없습니다.

ex) A가 B를 가지고 있고, B가 A를 가지고 있어 실행 전까지 에러를 잡을 수 없다.

또, 생성자 주입을 뺀 나머지(필드 주입, setter 주입)은 모두 생성자 이후에 호출되므로, 필드에 final 키워드를 사용할 수 없다.

3) setter를 통한 의존성 주입 (Setter Injection)

: Setter 메서드를 정의하고 해당 메서드를 통해 의존하는 객체를 주입한다.

```
@RestController
public class DIController {

    MyService myService;

    @Autowired
    public void setMyService(MyService myService) {
        this.myService = myService;
    }
}
```

현재, 위의 3가지 방법 중 가장 권장되는 주입 방식은 1번 생성자 주입 방법을 권장하고 있다고 한다. 프레임워크에 의존하지 않고 순수 JAVA 언어의 특징을 잘 살리는 방법이라고 한다.

- **AOP(Aspect-Oriented Programming) 기초**

  : AOP는 관점 지향 프로그래밍이다.

  프로그래밍에 대한 관점을 핵심 관점, 부가 관점으로 구분해 각각 하나의 관점으로 보는 것을 의미한다.

  ex) 고객으로부터 상품 정보 등록 요청을 받아 데이터베이스에 저장하고, 그 상품을 조회하는 로직을 구현할 때

  <핵심 기능>

  1. 상품 정보를 데이터베이스에 저장
  2. 저장된 상품 정보 데이터를 보여주는 코드

  <부가 기능>

  1. 로깅 : 메서드의 호출과 반환 값을 로깅하는 작업은 애플리케이션에서 중요하다. AOP를 사용하면 모든 메서드 호출에 대한 로그를 중앙에서 관리할 수 있다.
  2. 트랜잭션 관리 : 데이터베이스 트랜잭션을 관리할 때 AOP를 사용하여 트랜잭션의 시작과 종료, 롤백 등을 처리할 수 있다.
  3. 보안 : 인증과 권한 부여와 같은 보안 관련 작업을 AOP를 이용해 모듈화하고 중앙에서 관리할 수 있습니다.
  4. 예외처리 : 특정 로그를 남기거나 특정한 방식으로 처리하는 작업을 AOP로 모듈화할 수 있습니다.

참고 블로그) https://velog.io/@dkwktm45/Spring-AOP를-알고-사용-방법을-알자

https://velog.io/@jinyeong-afk/기술-면접-Spring-IoC-Inversion-of-Control와-DI-Dependency-Injection에-대하여