2024/10/01 2일차

제어역전 (IoC : Inversion of Control)

- 기존에는 자신이 의존하는 객체를 생성/획들할 책임을 자기가 가지고 있었다.
- 제어역전은 자신이 의존하는 객체를 획득하는 방법이 역전되어서(자기가생성/획득할 책임이 없다.) 외부존재스프링의경우 Spring Container)로부터 제공받는다.

의존성 주입(DI: Dependency Injection)

- 제어역전의 방식중 하나다.
- 자신이 의존하는 객체를 외부존재로부터 Setter 메소드나 생성자 메소드를 통해서 전달받는다.(의존성 주입)

제어역전 흐름

기존

OrderService객체가 UserDao,OrderDao를 의존한다.

OrderService 객체 → 의존 → UserDao,OrderDao객체

의존하는 객체를 직접 생성하기

OrderService 객체 → 객체생성→ UserDao,OrderDao객체

제어역전

의존성 주입으로 의존하는 객체 획득하기

OrderService 객체 ←의존성주입 ← UserDao,OrderDao객체

스프링 컨테이너

프로그램에서 컨테이너는 무엇무엇을 담고 있는것

톰켓 자바 서블릿 컨테이너 ex: index.jsp → 컴파일→ 객체생성

Context란 말이 들어가면 객체를 담고있는 곳

- 스프링의 가장 핵심적인 객체다.
- `설정정보`를 읽어서 `객체를 생성`하고, 객체간의 의존관계를 분석해서 `의존성주입으로 객체를 조립` 한다.

  `* 스프링 컨테이너는 객체를 생성하고 객체를 조립하는 곳이다. + 관리`

스프링 컨테이너 종류

스프링 컨테이너에 있는 인터페이스

- **BeanFactory**
  - spring-bean.jar 모듈에서 제공하는 스프링 컨테이너다.
  - 스프링 컨테이너의 가장 기본적인 기능(객체 생성, 의존성 주입, 생명주기 관리)만 제공한다.
  - 구현 클래스 (이런게 있더라~ 암기 필요 X)
    - XmlBeanFactory
- **ApplicationContext**
  - spring-context.jar 모듈에서 제공하는 스프링 컨테이너다.
  - BeanFactory가 제공하는 기능에 덧붙여서 AOP,메세지 리소스처리(국제화처리),이벤트 처리, EJB 연동 등 더 다양한 기능을 제공한다.
  - 구현클래스 (이런게 있더라~ 암기 필요 X)
    - ClassPathXmlApplicationContext, FileSystemXmlApplicationContext, GenericXmlApplicationContext
      AnnotationConfigApplicationContext
- **WebApplicationContext (실무에서 사용)**
  - spring-web,jar 모듈에서 제공하는 스프링 컨테이너다.
  - ApplicationContext가 제공하는 모든 기능을 제공한다.
  - 웹 애플리케이션에 최적화된 스프링 컨테이너다.
  - 구현 클래스 (이런게 있더라~ 암기 필요 X)
    - XmlWebApplicationContext, AnnotationConfigWebApplicationContext (요즘은 Annotation 기반으로 추가됨)

```
스프링 설정파일
            + 스프링 컨테이너가 생성할 객체를 지정한다.
                <bean id="식별자명" class="패키지명을 포함한 클래스이름"></bean
>

            + 스프링 컨테이너가 조립에 필요한 정보를 제공한다.
                <property name="멤버변수이름" ref="bean태그의 식별자명"></pro
perty>
                <property name="멤버변수이름" value="기본자료형 값"></propert
y>

                <constructor-arg name="생성자의 매개변수이름" ref="bean태그의
식별자명"/>
                <constructor-arg name="생성자의 매개변수이름" value="기본 자
```

```
료형 값"/>

                class UserService {
                    // 의존하는 객체가 조립되는 멤버변수(사실은 세터메소드 이름)
                    private UserDao userDao;
                    // 의존성 주입에 사용되는 Setter 메소드
                    public void SetUserDao(UserDao userDao){
                        this.userDao = userDao;
                    }
                }

사용법
bean은 생성
property 조립

<bean id="식별자명" class="x.y.z클래스이름">
    <property name="멤버변수이름" ref="bean태그의 식별자명"></property>
    <property name="멤버변수이름" value="기본자료형 값"></property>
</bean>

<property name="messageSender" ref="sms"/> ref : 뭐 조립할꺼니  property가
있어야지 주입이 실행됨
```

로그레벨

- **로그(Log)란 프로그램 개발이나 운영 시 발생하는 문제점을 추적하거나 운영 상태를 모니터링하기 위한 텍스트이다.**
- **메세지 중요도**

종류

`Fatal` 가장 심각한 장애

`Error` 오류

`Warn` 경고

`Info` 정보

`Debug` 상세한 정보

`Trace` 가장 상세한 정보

로그레벨	log.fatal(메세지)	log.error(메세지)	log.warn(메세지)	log.info(메세지)	log.debug(메세지)
`Fatal`	o	x	x	x	x
`Error`	o	o	x	x	x
`Warn`	o	o	o	x	x

`Info`	o	o	o	o	x
`Debug`	o	o	o	o	o
`Trace`					

개발 환경에서는 Info 나 Debug로 설정

*** 소나큐브 : 소스코드 품질 검사**

객체지향 개발원칙

S - 단일 책임원칙

O - 개방폐쇄 원칙 : 확장에는 열려있고 변화에는 닫혀있게 설계해라

L - 리스코프 치환 원칙

I - 인터페이스 분리 원칙

D- 의존성 주입원칙

자바는 항상 느슨한 결합이 필요하다. 느슨한결합에는 인터페이스가 필요하다 . 느슨한 결합은 확장에 편리한듯 코드를 바꿨을때 여파가 적다인듯

스프링의 핵심적인 가치 : 인터페이스를 이용한 느슨한결합과 제어역전을 활용한 의존성 주입 `핵심은 제어역전 IoC`

오후 수업

```
<bean id ="service2" class="di_4.spring.NotificationService">

    </bean>
매개변수 1개인 생성자가 있어야함
<bean id ="service2" class="di_4.spring.NotificationService">
        <constructor-arg name="" ref="">
    </bean>
매개변수 2개인 생성자가 있어야함
<bean id ="service2" class="di_4.spring.NotificationService">
        <constructor-arg name="" ref="">
        <constructor-arg name="" ref="">
    </bean>

매개변수 3개인 생성자가 있어야함
<bean id ="service2" class="di_4.spring.NotificationService">
        <constructor-arg name="" ref="">
```

```
        <constructor-arg name="" ref="">
        <constructor-arg name="" ref="">
    </bean>

Setter 메소드를 이용한 의존성 주입 (매개변수 하나도없는 생성자를 이용한것)

        아래코드는 매개변수 하나가 있는 생성자 이용. 여러개도 마찬가지
        <bean id ="service1" class="di_4.spring.NotificationService">
        <constructor-arg name="x" value="10"/>   매개변수가 있을때
        <property name="messageSender" ref="sms"/>
        </bean>


    <!--
        생성자 메소드를 이용한 의존성 주입
        <bean id ="service2" class="di_4.spring.NotificationService">
            <constructor-arg name="messageSender" ref="sms">
        </bean>

        * NotificationService에는 매개변수 한 개짜리 생성자가 필요하다
        * 생성자의 매개변수 이름은 messageSender다.
        * 생성자의 매개변수 타입은 sms객체를 전달받을 수 있는 타입이어야한다.

        <bean id ="service2" class="di_4.spring.NotificationService">
            <constructor-arg name="messageSender" ref="sms">
            <constructor-arg name="customerDao" ref="cDao">
        </bean>

        * NotificationService에는 매개변수 한 개짜리 생성자가 필요하다
        * 생성자의 매개변수 이름은 messageSender와 customerDao다.
        * 생성자의 매개변수 타입은 sms객체, cDao객체를 전달받을 수 있는 타입이어야한
다.
    -->
    <bean id ="service2" class="di_4.spring.NotificationService">
        <constructor-arg name="messageSender" ref="sms">
    </bean>
```

컴포넌트 : 수정없이 사용할 수 있는 반제품상태에 객체
애플리케이션에 조립함

@Component 설정파일을 설정할 필요가 없음

스프링 컨테이너의 컴포넌트 스캔

- **스프링 컨테이너는 빈으로 등록될 준비가 끝난 클래스를 스캔하여, 빈으로 등록해 주는 것이다.**
  - **빈(Bean)**
    - **스프링에서 스프링 컨테이너가 생성한 객체를말한다**
    - **스프링 컨테이너가 라이프 사이클을 관리한다.(생성하고 유지보관한다.)**
    - **의존성 주입에 이용한다.**
- **스프링 컨테이너의 스캔 대상 클래스**
  - 아래의 어노테이션이 부착된 클래스는 스캔대상 클래스다.
    - @Component
    - @Repository
    - @Service
    - @ContorIler
    - @RestController
    - @ContrdlerAdvice
    - @RestControllerAdvice
    - @Configuration

*** 다양한 어노테이션이 존재하는 이유**

- **어노테이션 종류별로 스캔전략을 다르게 수립할 수 있다.**
- **어노테이션별로 특정한 부가적인 작업을 수행할 수 있게 한다.**

  @Repository는 데이터베이스 엑세스 작업을 담당하는 클래스에 부착하는데,
  이 어노테이션이 붙어있으면 데이터베이스 엑세스 작업중 예외가 발생했을 때
  스프링의 DateAccessException 예외를 변환해서 발생시킨다.

  즉, 데이터베이스 엑세스 예외가 딱 한 종류의 예외만 발생시킬 수 있게된다.(예외처리가 단순해진다.)

@Autowired

비었을때 오류

2개되었을때 오류뜸

스프링 컨테이너 컴포넌트 스캔

```
프로젝트 구조

kr.co.jhta <- 패키지명
---...BoardDao.Java
---BoardService.java
 --App.java
 src/main/resources
```

```
 ---contet
     --tl.xml


스프링 컨테이너 스캔/자동묶기 대상으로 설정하기

@Repository  해당 클래스를 스캔대상으로 지정한다
publc class BoardDao{
     public void insert(Board board) {...}
     public void delete(int tno){int no}
}

    @Service  해당 클래스를 스캔대상으로 지정한다
    public class BoardService{

    @Autosired 자동의존성주입이 가능하게 설정한다.(자동 묶기 불가.))
    private BoarDao boardDao;

    public void addNewBoard(Board boar) {
    boardDao.insert(board);
    }
    스프링 빈 설정파일
    <context:comppnent_scan

}
```

스프링 컨테이너 컴포넌트 스캔

스프링 컨테이너의 컴포넌트 스캔

- 스프링 컨테이너는 빈으로 등록될 준비가 끝난 클래스를 스캔하여, 빈으로 등록해 주는 것이다.

* 빈(Bean)
  + 스프링에서 스프링 컨테이너가 생성한 객체를 말한다.
  + 스프링 컨테이너가 라이프사이클을 관리한다.(생성하고 유지보관한다.)
  + 의존성 주입에 이용한다.

- 스프링 컨테이너의 스캔 대상 클래스

* 아래의 어노테이션이 부착된 클래스는 스캔대상 클래스다.

@Component
@Repository
@Service
@Controller
@RestController
@ControllerAdvice
@RestControllerAdvice
@Configuration

*** 다양한 어노테이션이 존재하는 이유**

+ 어노테이션 종류별로 스캔전략을 다르게 수립할 수 있다.

+ 어노테이션별로 특정한 부가적인 작업을 수행할 수 있게 한다.

@Repository는 데이터베이스 엑세스 작업을 담당하는 클래스에 부착하는데,
이 어노테이션이 붙어있으면 데이터베이스 엑세스 작업중 예외가 발생했을 때
스프링의 DataAccessException 예외를 변환해서 발생시킨다.
즉, 데이터베이스 엑세스 예외가 딱 한 종류의 예외만 발생시킬 수 있게된다.(예외처리가 단순해진다.)

스프링 컨테이너 컴포넌트 스캔

프로젝트 구조

```
src/main/java
--- kr.co.jhta
    --- BoardDao.java
    --- BoardService.java
    --- App.java
src/main/resources
--- context
    --- di.xml
```

스프링 컨테이너 **스캔/자동빈묶기** 대상으로 설정하기

```
@Repository
public class BoardDao {
  public void insert(Board board) { .... }
  public void delete(int no) { ... }
}
```

```
@Service
public class BoardService {

  @Autowired
  private BoardDao boardDao;

  public void addNewBoard(Board board) {
    boardDao.insert(board);
  }
}
```

해당 클래스를 스캔대상으로 지정한다.

자동 의존성주입이 가능하게 설정한다. (자동 빈 묶기)

스프링 빈 설정파일

```
<context:component-scan
    base-package="kr.co.jhta" />
```

스프링 컨테이너 컴포넌트 스캔