DI
(Dependency Injection)

강 사 : 임 정 훈

목차

1 DI(Dependency Injection) 개념

2 IoC/DI 특 . 장점

3 ApplicationContext

4 Autowiring

DI(Dependency Injection)

DI(Dependency Injection)

1. Object 참조를 외부로부터 주입 (XML 파일) 받고 이를 통해서 다른 Object 와 역동적으로 의존관계가 형성 되게 하는 기법
2. 객체를 생성 , 취득하는 코드를 직접 만들지 않음
3. spring 에서 의존관계 주입 방법
   1) 생성자 (Constructor) 를 이용한 의존성 주입
   2) 설정 메소드 (Setter) 를 이용한 의존성 주입

DI(Dependency Injection)

1) 생성자 (Constructor) 를 이용한 의존성 주입

```
public class ServiceImpl {

private DaoImpl daoImpl; // 멤버변수

        // 생성자 의존관계 주입 : 외부 클래스의 객체 주입
        public ServiceImpl (DaoImpl daoImpl){
              this.daoImpl = daoImpl;
        }

        public void write(DbTable article){  System.out.println("Se
              rviceImpl.write() 메서드 실행 "); daoImpl.insert(article
              ); // dao 메서드 호출
        }
}
```

DI(Dependency Injection)

2) 설정메서드 (Setter) 를 이용한 의존성 주입

```
public class ServiceImpl {

        private DaoImpl daoImpl; // 멤버 변수

        // 의존 관계 설정 방식 : Setter()
        public void setDaoImpl (DaoImpl daoImpl) {
                this.daoImpl = daoImpl;
        }

        public void write(DbTable table){  System.out.println("ServiceI
                mpl.write() 메서드 실행 "); daoImpl.insert(table);
        }
}
```

DI(Dependency Injection)

스프링 설정파일

```
<! – 생성자를 이용한 의존성 주입 -->
<bean id="serviceImpl" class="spring.di.ch03.ServiceImpl">
    <constructor-arg   ref="daoImpl"/>
 </bean>
 <!-- ref 속성 : 다른 bean 의 id 를 참조하는 속성 -->
 <bean id="daoImpl" class="spring.di.ch03.DaoImpl"/>

<! – 설정메서드를 이용한 의존성 주입 -->
 <bean id =“ serviceImpl" class="spring.di.ch03.ServiceImpl">
    <property name =“ daoImpl “ ref=“ daoImpl"/>
 </bean>
 <bean id =“ daoImpl" class="spring.di.ch03.DaoImpl"/>
```

IoC/DI 특 . 장점

IoC/DI 특 . 장점

1. Object 의존관계에 대한 객체지향 프로그래밍 모델
2. 유연성과 확장성이 뛰어난 코드 작성을 위한 객체지향 설계원칙과 디자인 패턴의 핵심
3. Spring 프레임워크에서 동작하는 모듞 코드는 IoC/DI 방식을 따른다 .
4. Spring 을 이해하고 효율적으로 사용하는데 기본이 되는 가장 중요한 메커니즘이다 .

스프링 컨테이너

스프링 컨테이너

- 스프링 설정파일로부터 스프링 객체 (빈) 를 생성하는 역할
- 스프링 컨테이너 구현체

1) BeanFactory 인터페이스
2) ApplicationContext 인터페이스

BeanFactory 인터페이스 보다 많은 기능 제공

xml 파일을 읽어서 스프링 컨테이너 생성

```
AbstractApplicationContext    context = new
   ClassPathXmlApplicationContext("applicationContext.xml");
```

ApplicationContext

ApplicationContext 구현 클래스

1. ClassPathXmlApplicationContext
   - 클래스패스에 위치한 xml 파일로부터 설정 정보 로딩
2. FileSystemXmlApplicationContext
   - 파일 시스템에 위치한 xml 파일로부터 설정 정보 로딩
3. XmlWebApplicationContext
   - 웹 애플리케이션에 대한 xml 파일 설정 정보 로딩
   - Spring MVC 에서 적용

ApplicationContext

ClassPathXmlApplicationContext 이용 컨테이너 생성

```
// 1. 스프링 설정파일로부터 스프링 컨테이너 생성
    AbstractApplicationContext context =
                new ClassPathXmlApplicationContext("applicationContext.xml");
// 2. JVM 종료될 때 ApplicationContext 를 종료하는 작업
    context .registerShutdownHook();
// 3. 스프링 컨테이너에서 빈 ( 객체 ) 를 가져온다 .
    MessageBean messageBean = (MessageBean) context .getBean("messageBean");
// 4. 스프링 빈으로 메서드 호출  mes
    sageBean .sayHello(" 홍길동 ");
// 5. 어플리케이션 종료시 컨테이너에 존재하는 모듞 빈 ( 객체 ) 닫기
    context .close();
```