

Spring IoC, DI





IoC와 DI



Spring

IoC

- Inversion of Control의 약자로 제어의 역행이라는 의미
- 프로그램을 구동하는데 필요한 객체 생성에 대한 생성, 변경 등의 관리를 개발자가 아닌 프로그램을 구동하는 컨테이너에서 직접 관리하는 것
- 스프링은 IoC 구조를 통해 구동 시 필요한 객체의 생성부터 생명주기 까지 해당 객체에 대한 관리를 직접 수행

IoC 컨테이너의 역할

객체의 생명주기와 의존성 관리

- 생명주기 : 생성 → 초기화 → 사용 → 소멸
- 의존성 : 개발자가 직접 객체를 생성할 수 있지만 해당 권한을 컨테이너에 맡김으로써 소스코드 구현의 시간을 단축



Spring

IoC 컨테이너와 Bean 객체

컨테이너	설명
Bean	<ul style="list-style-type: none">- 스프링이 IoC 방식으로 관리하는 객체- 스프링이 직접 생성과 제어를 담당하는 객체
BeanFactory	<ul style="list-style-type: none">- 자바 빈 객체의 등록 및 관리- <code>getBean()</code> 메소드를 통해 객체를 가져옴
ApplicationContext	<ul style="list-style-type: none">- BeanFactory의 확장 개념- Spring의 각종 부가 서비스를 제공- 일반적인 IoC 컨테이너를 의미
GenericXml ApplicationContext	<ul style="list-style-type: none">- ApplicationContext를 구현한 클래스- 일반적인 XML 형태의 문서를 읽어 컨테이너 역할을 수행



Spring

Spring DL

의존성 검색이란 뜻으로 컨테이너가 제공하는 API 함수로 필요한 Bean을 검색해서 사용하는 방식

Spring DI

의존성 주입이란 뜻으로 IoC 구현의 핵심 기술로, 사용하는 객체를 직접 생성하는 것이 아니라 컨테이너가 빈의 설정정보를 읽어와 자동으로 해당 객체를 연결하는 것
장점

- 개발자가 작성해야 할 코드가 단순해짐
- 각 객체 간의 종속관계(결합도)를 해소 할 수 있음



Spring

결합도(객체 간의 종속 관계)

한 클래스에서 필드 객체를 생성 할 때 발생하는 두 객체 간의 관계를 말하며, 각 객체 간의 내용이 수정될 경우 영향을 미치는 정도

A Class와 B Class가 있다.

A Class 작성시 B Class 객체의 메소드를 이용하여 기능을 작성하는 경우, B Class의 생성자나, 메소드가 변경되는 경우 A Class의 일부 정보도 반드시 수정해야 하는 상황이 발생하는데, 이를 '두 객체간 결합도가 강하다'라고 표현한다.



Spring

Spring DI



Spring

Spring DI



Spring

Spring DI



Spring

Spring DI



Spring

Spring DI 종류

1. **Setter 메소드를 통한 의존성 주입**
 - 의존성을 주입 받는 setter 메소드를 만들고, 이를 통해 의존성을 주입
2. **생성자를 통한 의존성 주입**
 - 필요한 의존성을 포함하는 클래스에 생성자를 만들고, 이를 통해 의존성을 주입
3. **메소드를 통한 의존성 주입**
 - 의존성을 입력 받는 일반 메소드를 만들고 이를 통해 의존성을 주입



Spring

Spring DI 종류 – setter메소드를 통한 의존성 주입

setter 메소드를 통한 의존관계가 있는 bean을 주입하려면 <property> 태그 사용

```
<bean id="객체이름" class="클래스 풀네임">  
    <property name="name" value="OOO"/>  
    <property name="name" ref="OOO"/>  
</bean>
```

- id 속성은 객체의 이름을 의미(자바코드 : Student student)
- class 속성은 해당 클래스의 패키지명을 포함한 풀네임
- name 속성은 클래스에 선언한 변수의 이름(String name, School school)
- value 속성은 해당 변수에 대입할 값(객체타입이 아닌 일반 값)
- ref 속성은 객체타입의 변수에 bean을 주입



Spring

Spring DI 종류 – setter메소드를 통한 의존성 주입 예

```
<bean id="school1" class="student.model.vo.School" />  
<bean id="student" class="student.model.vo.Student">  
    <property name="name" value="홍길동"/>  
    <property name="school" ref="school1"/>  
</bean>
```

자바코드

```
School school1 = new School();  
Student student = new Student();  
student.setName("홍길동");  
student.setSchool(school1);
```



Spring

Spring DI 종류 – 생성자를 통한 의존성 주입

setter 메소드를 통한 의존관계가 있는 bean을 주입하려면 <constructor-arg> 태그 사용

```
<bean id="객체이름" class="클래스 풀네임">  
    <constructor-arg index="0" value="OOO"/>  
    <constructor-arg name="name" ref="OOO"/>  
</bean>
```

- name 속성은 생성자에 전달하는 매개변수의 변수명으로 설정
- 생성자 매개변수 순서에 따라 index 속성을 통해서도 접근 가능



Spring

Spring DI 종류 – setter메소드를 통한 의존성 주입 예

```
<bean id="school1" class="student.model.vo.School" />
```

- name을 이용한 방식

```
<bean id="student" class="student.model.vo.Student">  
    <constructor-arg name="name" value="홍길동"/>  
    <constructor-arg name="school" ref="school1"/>  
</bean>
```

</bean>

- index를 이용한 방식

```
<bean id="student" class="student.model.vo.Student">  
    <constructor-arg index="0" value="홍길동"/>  
    <constructor-arg index="1" ref="school1"/>  
</bean>
```

</bean>

자바코드

```
School school1 = new School();  
Student student = new Student("홍길동", school1);
```





DI Annotation



Spring

Annotation

- 대부분의 프레임워크가 그렇듯 Spring Framework 역시 XML 파일 설정이 매우 중요
- XML파일의 과도한 설정을 하게 되면 부담스러워 짐
- XML 방식 이외에도 Annotation 방식의 설정을 지원



Spring

DI Annotation

bean으로 사용될 클래스에 특별한 Annotation을 부여하고 Spring 컨테이너가 이 Annotation을 통해 자동으로 bean을 등록하는 방식으로, 빈 스캐닝(bean scanning)을 통한 자동인 bean 등록기능이라고 함

DI Annotation 방식의 장/단점

장점	단점
<ul style="list-style-type: none">- XML 문서 생성과 관리에 따른 수고를 덜어 주고 개발 속도를 향상 시킴- 개발자 간 XML 설정 파일의 충돌을 최소화	<ul style="list-style-type: none">- 어플리케이션에 등록될 bean이 어떤 것들이 있고, bean들 간의 의존관계가 어떻게 되는지 한눈에 파악 할 수 없음



Spring

bean 등록 Annotation

Annotation	설명
@Component	- 객체를 나타내는 일반적인 타입으로 <bean>태그 역할
@Controller	- Presentation Layer Annotation으로 view에서 전달된 웹 요청과 응답을 처리하는 클래스에 사용
@Service	- Service Layer Annotation으로 비즈니스 로직을 가진 클래스에 사용
@Repository	- Persistence Layer Annotation으로 영속성(파일, DB)을 가진 클래스에 사용

※ @Controller, @Service, @Repository는 특정한 객체의 역할에 대한 @Component의 구체화된 형태



Spring

bean 의존관계 주입 Annotation

Annotation	설명
@Autowired	<ul style="list-style-type: none">- 주로 변수 위에 설정하여 해당 타입 객체를 컨테이너에서 찾아서 자동으로 주입
@Inject	<ul style="list-style-type: none">- @Autowired와 동일한 기능 지원- @Autowired는 Spring 전용, @Inject는 Java 전용
@Qualifier	<ul style="list-style-type: none">- @Autowired와 같이 사용되며, 특정 객체의 이름을 이용하여 의존성을 주입할 때 사용- @Autowired를 하려는 상황에 해당 타입의 객체가 컨테이너에 2개 이상 존재하는 경우 어떤 객체를 주입할 지 결정할 수 없어 에러가 발생하는데 이를 해결함
@Resource	<ul style="list-style-type: none">- @Autowired와 Qualifier의 기능을 결합한 Annotation



Spring

<context:component-scan> 태그

- Spring 설정 파일에 애플리케이션에서 사용할 <bean>을 등록하지 않고 Annotation을 통해 자동으로 생성하기 위해 사용하는 태그
- 특정 패키지 내부의 클래스들 중 @Component Annotation이 설정된 클래스들을 자동으로 객체 생성

<context:component-scan> 태그 사용 예

<context:component-scan base-package="kr.or.iei.member" />

- kr.or.iei.member.controller → 스캔대상
- kr.or.iei.member.model.vo → 스캔대상
- kr.or.iei.board.controller → 스캔대상 아님
- kr.or.iei.board.model.service → 스캔대상 아님

