

Spring DI

다들 내용

- ✓ Spring IoC ?
 - ✓ Container
 - ✓ Dependency 관리
 - ✓ Annotation 활용
-

IoC

✓ IoC (Inversion of Control) : 제어 역행

- 기존 : 인스턴스의 생성부분을 개발자가 소스상에서 직접 처리
-

```
public class Main {  
    public static void main(String[] args) {  
        HamSandwiches sand = new HamSandwiches();  
        sand.info();  
    }  
}
```

IoC

- **IoC** : 인스턴스의 생명주기 관리를 개발자가 아닌 **컨테이너가** 처리함
-

xml 파일

```
<beans>
```

```
    <bean id="ham" class="test.HamSandwiches" />
```

```
</beans>
```

java 파일

```
ISandwiches sand = (HamSandwiches)context.getBean("ham");  
sand.info();
```

IoC

✓ **DI (Dependency Inject) : 의존 주입**

- **Spring**에서 **IoC**를 제공하는 형태 중 하나(**DL, DI**)

- 종류 :

 - : **Setter Injection**

 - : **Contructor Injection**

* 의존 : 객체간의 의존관계를 의미

기존 코드 작성 방식

- ✓ 필요한 위치에서 객체 생성
 - ✓ 인터페이스를 활용한 객체 생성
 - ✓ 별도의 조립기 클래스를 활용하여 객체 생성
-

- ✓ 개선방법
 - **Spring** 에서 제공하는 **DI** 활용
 - 개발자가 코드에서 직접 객체 생성하지 않는 방식
 - **XML** 환경 설정파일 또는 어노테이션을 이용하여 객체를 주입
-

Container

✓ Spring Container

- Spring 프레임워크에서 **Container** 기능을 제공해 주는 **클래스**를 의미
 - **Container** : **Bean** 클래스를 관리(생성 , 삭제등) 하는 주체
 - **Bean** : **Spring** 에서 관리되는 클래스 객체를 나타냄
 - **Container** 초기화 방법 : 설정 정보 **Xml** 파일을 읽고 **Container** 에 로딩
 - **ApplicationContext**
 - **ClassPathXmlApplicationContext**
 - **FileSystemXmlApplicationContext**
 - **XmlWebApplicationContext**
-

Container

클래스명	설명
ClassPathXmlApplicationContext	클래스패스 설정 경로로부터 Xml 설정파일 로딩
FileSystemXmlApplicationContext	지정된 파일 시스템으로부터 Xml 설정파일 로딩
XmlWebApplicationContext	웹어플리케이션 컨텍스트로부터 Xml 설정파일 로딩

pom.xml 작성

- ✓ **dependency** 추가
 - **spring-context** 추가
-

`<dependency>`

`<groupId>org.springframework</groupId>`

`<artifactId>spring-context</artifactId>`

`<version>X.x.x.RELEASE</version>`

`</dependency>`

Container - XML

✓ Spring XML 파일

- Spring 은 XML 설정정보를 참조하여 여러가지 **Container Service**를 제공
 - 유지보수 작업 시 **XML** 파일을 조정
-

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans ... >
```

```
    <bean id="ham" class="di.container.HamSandwiches">
```

```
        <property name="title" value="햄" />
```

```
    </bean>
```

```
</beans>
```

Container - Bean

✓ Bean 클래스

- Spring 프레임웍에 의해 **LifeCycle**이 관리되는 클래스
- 일반 **POJO** 기반의 클래스
- XML에 **<bean />** 태그를 이용하여 등록
- **<bean>** 태그의 속성
 1. **id** : 여러 개의 **Bean** 클래스를 식별하기 위한 이름 설정
 2. **name** : **id** 속성과 동일한 의미, 별칭 설정이 가능
 3. **class** : 사용하려는 **Bean** 클래스의 패키지명을 포함한 클래스명

Container - Bean

❖ id 와 name 속성의 차이

name 속성은 쉼표, 공백, 세미콜론 등으로 별칭 설정 가능

```
<bean id="ham" name="h,a,m" class="test.Ham" />
```

```
<bean id="ham" name="h;a;m" class="test.Ham" />
```

```
<bean id="ham" name="h a m" class="test.Ham" />
```

```
<bean id="ham" name="h,a;m" class="test.Ham" />
```

Container – Bean 적용 코드

java 파일

```
public class HamSandwiches implements ISandwiches {  
    public HamSandwiches( ) {  
        System.out.println("HamSandwiches 생성자 호출");  
    }  
}
```

xml 파일

```
<bean id="ham" class="di.container.HamSandwiches" />
```

Container – ApplicationContext

Java 코드

```
String contextPath = "di/container/container.xml";
```

```
ApplicationContext context =
```

```
    new ClassPathXmlApplicationContext(contextPath);
```

예제 작성

di.container

DI

✓ Dependency Injection

- 각 **Bean** 간의 의존관계 설정을 **xml** 파일에 등록
 - 프로그램 코드에서는 직접 빈을 획득하기 위한 코드를 사용할 필요가 없음
 - **Container** 가 자체적으로 필요한 객체를 넘겨줘서 사용하는 방식
 - 사용방식
 1. **Constructor Injection**
 2. **Setter Injection**
-

Constructor Injection

✓ Constructor Injection : 생성자를 활용한 값 설정

1. 문자열을 매개변수로 받는 생성자

```
public SetMenu(String description) {  
    this.description = description;  
}
```

```
<bean id="menu" class="di.constructInjection.SetMenu" >  
    <constructor-arg>  
        <value>샌드위치 : 햄, 음료 : 콜라</value>  
    </constructor-arg>  
</bean>
```

Constructor Injection

2. 객체를 매개변수로 받는 생성자

```
public SetMenu(ISandwiches sandwiches) {  
    코드 생략....  
}
```

```
<bean id="ham" class="di.constructInjection.HamSandwiches" />  
<bean id="menu" class="di.constructInjection.SetMenu" >  
    <constructor-arg>  
        <ref bean="ham" />  
    </constructor-arg>  
</bean>
```

Constructor Injection

3. 여러개의 매개변수를 받는 생성자

```
public SetMenu(String description, int price) {  
    코드 생략....  
}
```

```
<bean id="menu" class="di.constructInjection.SetMenu" >  
    <constructor-arg>  
        <value>샌드위치 : 햄, 음료 : 콜라</value>  
    </constructor-arg>  
    <constructor-arg><value>100</value></constructor-arg>  
</bean>
```

Constructor Injection

4. 매개변수 타입 지정 생성자

```
public SetMenu(String description, int price) {  
    코드 생략....  
}
```

```
<bean id="menu" class="di.constructInjection.SetMenu" >  
    <constructor-arg>  
        <value type="java.lang.String">샌드위치 : 햄, 음료 : 콜라</value>  
    </constructor-arg>  
    <constructor-arg type="int" value="100" />  
</bean>
```

예제 작성

di.constructInjection

Setter Injection

✓ **Setter Injection** : **set** 메소드를 활용한 값 설정

1. 문자열을 매개변수로 받는 **set** 메서드

```
public void setDescription(String description) {  
    this.description = description;  
}
```

```
<bean id="menu" class="di.setterInjection.SetMenu" >  
    <property name="description">  
        <value>샌드위치 : 햄, 음료 : 콜라</value>  
    </property >  
</bean>
```

Setter Injection

2. 객체를 매개변수로 받는 **set** 메서드

```
public void setSandwiches(ISandwiches sandwiches) {  
    코드 생략....  
}
```

```
<bean id="ham" class="di.setterInjection.HamSandwiches" />  
<bean id="menu" class="di.setterInjection.SetMenu">  
    <property name="sandwiches "><ref bean="ham"/></property>  
</bean>
```

예제 작성

`di.setterInjection`

Dependency 응용

✓ 집합객체 설정

XML	타입
<list>	java.util.List, 배열
<set>	java.util.Set
<map>	java.util.Map
<props>	Java.util.Properties

Dependency 응용

- list

```
public void setSandwichesList(List<ISandwiches> sandwichesList) {  
    코드생략...  
}
```

```
<property name="sandwichesList">  
    <list>  
        <ref bean="hamSandwiches"/>  
        <ref bean="cheeseSandwiches"/>  
    </list>  
</property>
```

Dependency 응용

- set

```
public void setSandwichesSet(Set<ISandwiches> sandwichesSet) {  
    코드생략...  
}
```

```
<property name="sandwichesSet">  
    <set>  
        <ref bean="hamSandwiches" />  
        <ref bean="cheeseSandwiches" />  
    </set>  
</property>
```

Dependency 응용

- map

```
public void setSandwichesMap(Map<String, ISandwiches> sandMap) {  
    코드생략...  
}
```

```
<property name="sandwichesMap">  
    <map>  
        <entry key="ham" value-ref="hamSandwiches" />  
        <entry key="cheese" value-ref="cheeseSandwiches" />  
    </map>  
</property>
```

Dependency 응용

- prop

```
public void setSandwichesProp(Properties sandwichesProp) {  
    코드생략...  
}
```

```
<property name="sandwichesProp">  
    <props>  
        <prop key="ham">햄 샌드위치</prop>  
        <prop key="cheese">치즈 샌드위치</prop>  
    </props>  
</property>
```

예제 작성

di.collection

다중 파일 설정

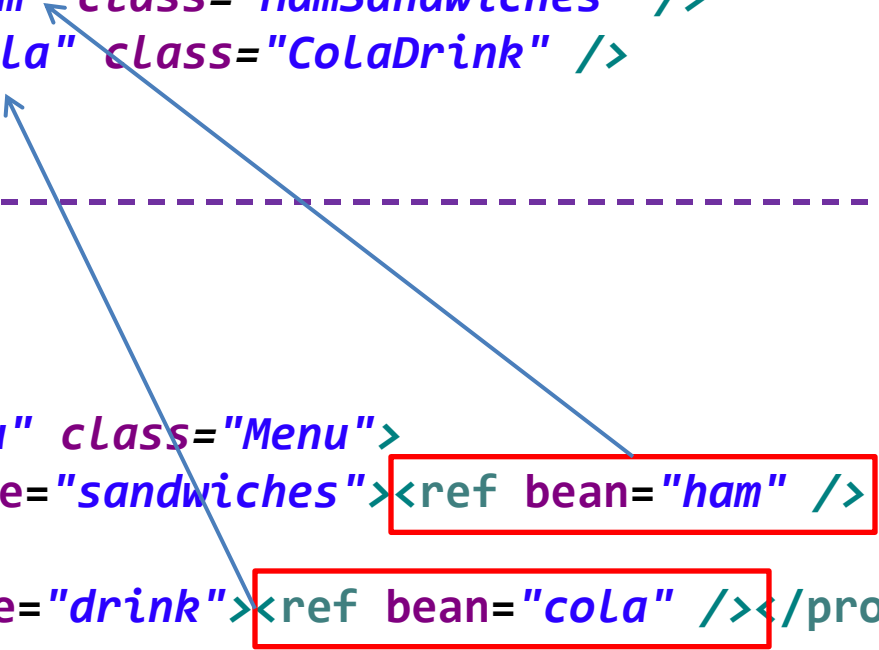
- ✓ Xml에 설정되는 내용이 많은 경우 여러 개의 설정 파일로 관리

- config1.xml

```
<beans ..">  
    <bean id="ham" class="HamSandwiches" />  
    <bean id="cola" class="ColaDrink" />  
</beans>
```

- config2.xml

```
<beans ..">  
    <bean id="menu" class="Menu">  
        <property name="sandwiches"><ref bean="ham" />  
        </property>  
        <property name="drink"><ref bean="cola" /></property>  
    </bean>  
</beans>
```



사용하기

- ✓ 자바에서 호출 시 처리

```
-----  
ApplicationContext context =  
    new ClassPathXmlApplicationContext(  
        new String[] {"di/configDivide/config1.xml",  
                      "di/configDivide/config2.xml"});  
-----
```

- ✓ Xml 파일에서 처리

```
<import resource="classpath:config1.xml" />  
<beans ..">  
    <bean id="menu" class="Menu">  
        <property name="sandwiches"><ref bean="ham" />  
        </property>  
        <property name="drink"><ref bean="cola" /></property>  
    </bean>  
</beans>  
-----
```


예제 작성

di.configDivide

Annotation

DI 자동 주입

- ✓ **XML** 파일이 너무 커지는 것을 방지
 - ✓ 자동 주입 기능 사용시 스프링이 알아서 의존 객체를 찾아서 주입
 - ✓ 자동 주입 기능 사용 방법
 - **XML**파일에 **<context:annotation-config />** 설정을 추가
 - **Java**파일에 의존주입대상에 **@Autowired** 또는 **@Resource** 설정
-

XML 설정

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config />

</beans>
```

@Autowired

✓ Java 설정

- 변수 설정

```
@Autowired  
private IDrink drink;
```

- 생성자 설정

```
@Autowired  
public Menu(ISandwiches sandwiches) {  
    this.sandwiches = sandwiches;  
}
```

- set 메서드 설정

```
@Autowired  
public setSandwiches(ISandwiches sandwiches) {  
    this.sandwiches = sandwiches;  
}
```

동일한 객체가 두개 이상일 경우

```
<bean id="cheese" class="CheeseSandwiches">
    <property name="title" value="햄" />
</bean>
<bean id="ham" class="HamSandwiches">
    <property name="title" value="햄" />
</bean>
```

```
@Autowired
public void setSandwiches(ISandwiches sandwiches) {
    this.sandwiches = sandwiches;
}
```

예외발생

expected single matching bean but found 2: cheese,ham

@Qualifier 활용

```
<bean id="cheese" class="CheeseSandwiches">
    <qualifier value="cheese" />
    <property name="title" value=" 치즈 " />
</bean>

<bean id="ham" class="HamSandwiches">
    <property name="title" value=" 햄 " />
</bean>
```

@Autowired

@Qualifier("cheese")

```
public void setSandwiches(ISandwiches sandwiches) {
    this.sandwiches = sandwiches;
}
```

이름이 같은 빈 찾기

```
<bean id="cheese" class="CheeseSandwiches">
    <property name="title" value=" 치즈" />
</bean>

<bean id="ham" class="HamSandwiches">
    <property name="title" value=" 햄" />
</bean>
```

@Autowired

```
public void setSandwiches(ISandwiches cheese) {
    this.sandwiches = cheese;
}
```

객체 찾는 순서

1. 타입이 같은 빈을 검색하여 한 개면 그 빈 객체를 사용
 2. 두개이상이면 **@Qualifier**가 명시되어 되어있는 빈객체를 찾는다. 명시된 값과 같은 빈 객체를 사용
 3. 두개이상이고 **@Qualifier**가 없을 경우 이름이 같은 빈객체를 찾아서 사용
 4. 위의 경우에 해당하는 객체가 없을 경우 예외가 발생함
-

@Resource

✓ Java 설정

- 변수 설정

```
@Resource(name="cola")  
private IDrink drink;
```

- 생성자 설정 : 제공되지 않음

- set 메서드 설정

```
@Resource(name="ham")  
public Menu(ISandwiches sandwiches) {  
    this.sandwiches = sandwiches;  
}
```

객체 설정

```
<bean id="ham" class="HamSandwiches">  
    <property name="title" value="햄" />  
</bean>
```

```
@Resource(name="ham")
```

```
public void setSandwiches(ISandwiches sandwiches) {  
    this.sandwiches = sandwiches;  
}
```

Name 속성이 없는 경우

이름이 없을 경우 동일한 타입 객체를 선택

```
<bean id="ham" class="HamSandwiches">  
    <property name="title" value="햄" />  
</bean>
```

@Resource

```
public void setSandwiches(ISandwiches sandwiches) {  
    this.sandwiches = sandwiches;  
}
```

동일 타입이 여러 개인 경우

변수명과 동일한 객체를 선택

```
<bean id="cheese" class="CheeseSandwiches">
    <property name="title" value="치즈" />
</bean>
<bean id="ham" class="HamSandwiches">
    <property name="title" value="햄" />
</bean>
```

@Resource

```
public void setSandwiches(ISandwiches cheese) {
    this.sandwiches = sandwiches;
}
```

@Qualifier 활용

```
<bean id="cheese" class="CheeseSandwiches">
    <qualifier value="ch" />
    <property name="title" value=" 치즈 " />
</bean>

<bean id="ham" class="HamSandwiches">
    <property name="title" value=" 햄 " />
</bean>
```

@Resource

@Qualifier("ch")

```
public void setSandwiches(ISandwiches sandwiches) {
    this.sandwiches = sandwiches;
}
```

component-scan

- ✓ xml 파일 설정을 통해서 자동으로 빈으로 사용 될 객체를 등록한다.
 - `<context:component-scan base-package="kr.co.mlec" />`
 - 지정된 패키지 하위의 모든 패키지를 스캔하여 빈으로 등록
 - 빈으로 등록되려면 자바 클래스에서 어노테이션을 사용해야 한다.
 - **@Component, @Controller, @Service, @Repository**
-

코드작성

Xml 파일

```
<context:component-scan base-package="_11_di" />
```

설정값이 없는 경우 클래스이름의 첫자를 소문자로 적용한 빈으로 등록

```
@Component
```

```
public class ColaDrink implements IDrink
```

```
@Component("ham")
```

```
public class HamSandwiches implements ISandwiches
```

```
@Resource(name="colaDrink")
```

```
private IDrink drink;
```

```
@Resource(name="ham")
```

```
public void setSandwiches(ISandwiches sandwiches)
```
