Spring Framework

5. IoC와 DI

Tacademy

CONTENTS

학습 목표

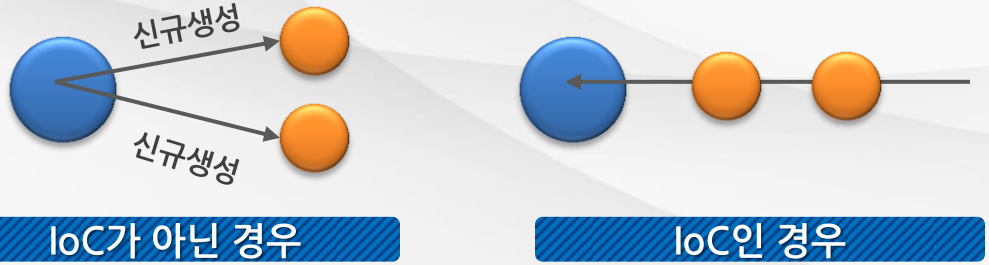
- IoC(Inversion of Control)에 대하여 이해할 수 있습니다.
- DI(Dependency Injection)에 대하여 이해할 수 있습니다.
- Spring DI 컨테이너에 대하여 이해할 수 있습니다.

1. IoC(Inversion of Control)

IoC의 개념

IoC(제어권의 역전)이란, 객체의 생성, 생명주기의 관리까지 모든 객체에 대한 제어권이 바뀌었다는 것을 의미한다.

- 컴포넌트 의존관계 결정 (component dependency resolution), 설정(configuration) 및 생명주기(lifecycle)를 해결하기 위한 디자인 패턴(Design Pattern)



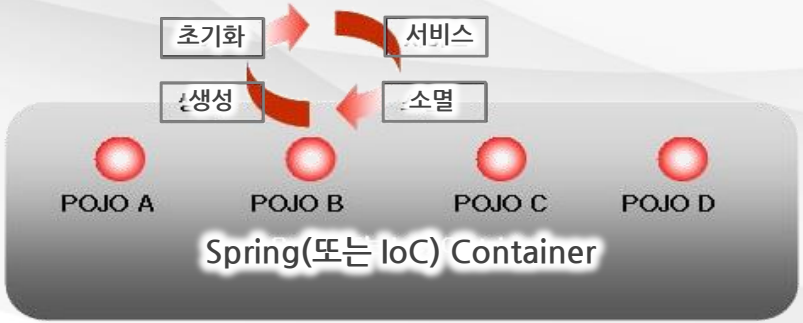
IoC 컨테이너

- 스프링 프레임워크도 **객체에 대한 생성 및 생명주기를 관리**할 수 있는 기능을 제공하고 있음. 즉, IoC 컨테이너 기능을 제공한다.

IoC 컨테이너는 객체의 생성을 책임지고, 의존성을 관리한다.

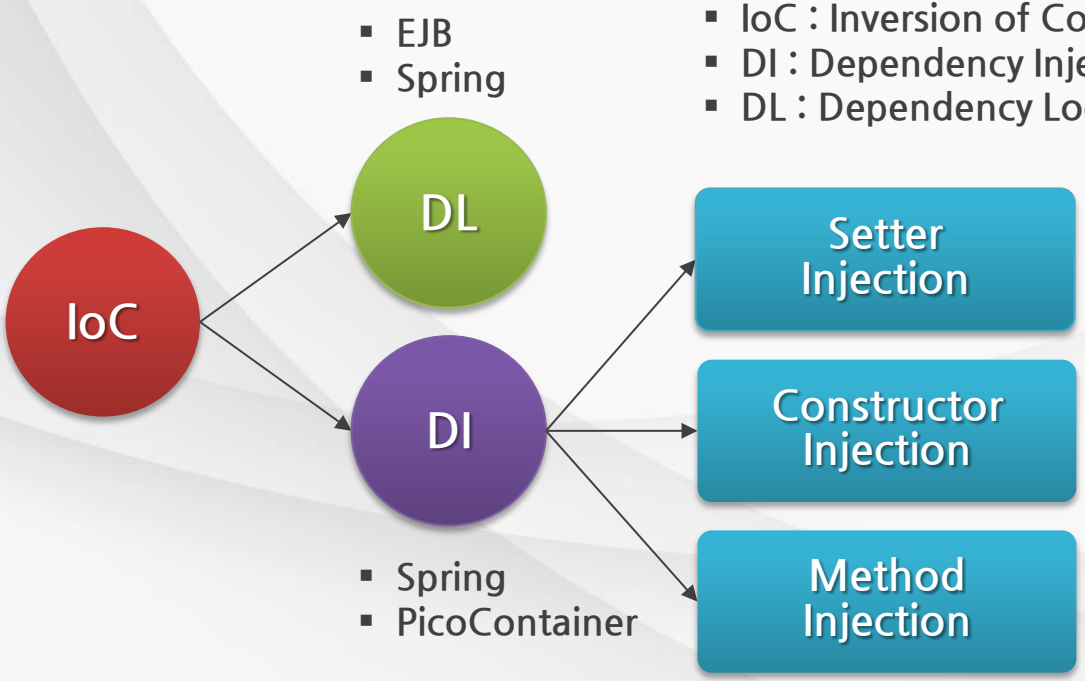
POJO의 생성, 초기화, 서비스, 소멸에 대한 권한을 가진다.

개발자들이 직접 POJO를 생성할 수 있지만 컨테이너에게 맡긴다.



IoC의 분류

- IoC : Inversion of Control
- DI : Dependency Injection
- DL : Dependency Lookup



DL(Dependency Lookup)과 DI(Dependency Injection)

DL (Dependency Lookup) 의존성 검색	저장소에 저장되어 있는 Bean에 접근하기 위해 컨테이너가 제공하는 API를 이용하여 Bean을 Lookup 하는 것
DI (Dependency Injection) 의존성 주입	각 클래스간의 의존관계를 빈 설정(Bean Definition) 정보를 바탕으로 컨테이너가 자동으로 연결해주는 것

- DL 사용시 컨테이너 종속성이 증가하여, 주로 DI를 사용함

Setter Injection | Constructor Injection | Method Injection

2. DI(Dependency Injection)

DI의 개념

각 클래스간의 의존관계를 빈 설정 (Bean Definition)
정보를 바탕으로 컨테이너가 자동으로 연결해주는 것을 말함

- 개발자들은 단지 빈 설정파일에서 의존관계가 필요하다는 정보를 추가하면 된다.
- 객체 레퍼런스를 컨테이너로부터 주입 받아서, 실행 시에 동적으로 의존관계가 생성된다.
- 컨테이너가 흐름의 주체가 되어 애플리케이션 코드에 의존관계를 주입해 주는 것이다.

DI (Dependency Injection) 장점	▪ 코드가 단순해진다. ▪ 컴포넌트 간의 결합도가 제거된다.

DI의 유형

Setter Injection — Setter 메서드를 이용한 의존성 삽입

- 의존성을 입력 받는 setter 메서드를 만들고 이를 통해 의존성을 주입한다.

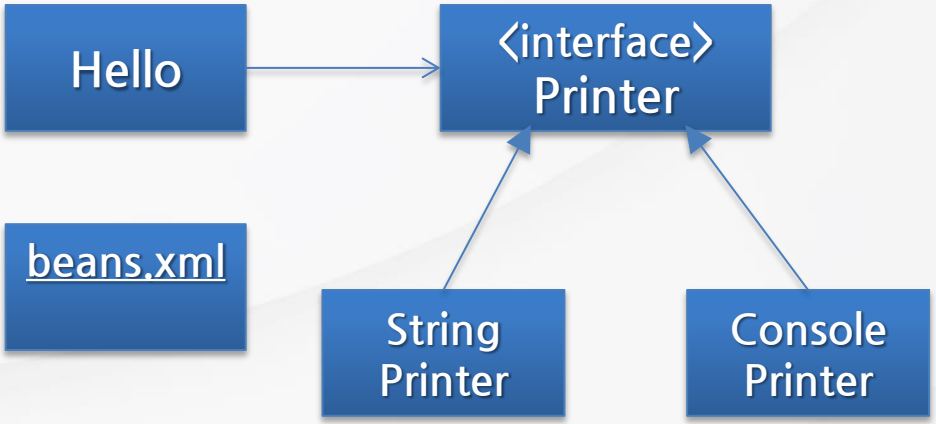
Constructor Injection — 생성자를 이용한 의존성 삽입

- 필요한 의존성을 포함하는 클래스의 생성자를 만들고 이를 통해 의존성을 주입한다.

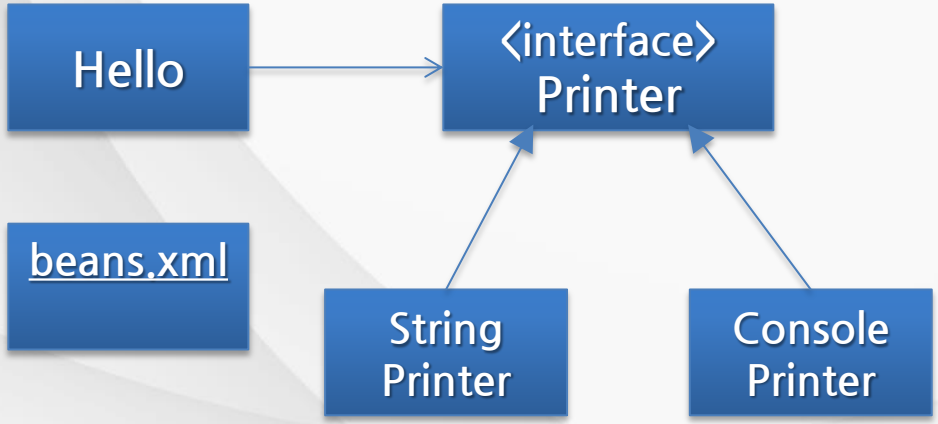
Method Injection — 일반 메서드를 이용한 의존성 삽입

- 의존성을 입력 받는 일반 메서드를 만들고 이를 통해 의존성을 주입한다.

DI를 이용한 클래스 호출방식



Setter Injection



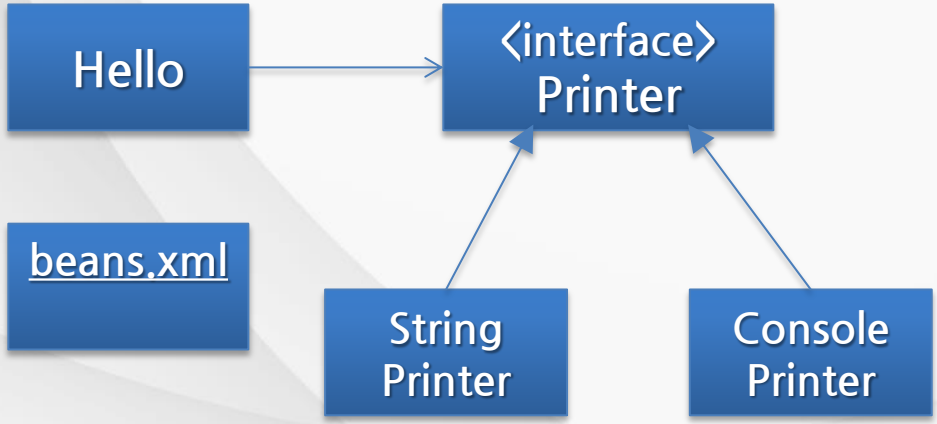
Hello.java

```
1  package bean;
2
3  import java.util.List;
4
5  public class Hello {
6      String name;
7      Printer printer;
8
9      public Hello() { }
10
11     public void setName(String name) {
12         this.name = name;
13     }
14
15     public void setPrinter(Printer printer) {
16         this.printer = printer;
17     }
```

beans.xml

```
11  <bean id="hello" class="bean.Hello">
12      <property name="name" value="Spring" />
13      <property name="printer" ref="printer" />
14  </bean>
15
16  <bean id="printer" class="bean.StringPrinter" />
17  <bean id="consolePrinter" class="bean.ConsolePrinter" />
```

Constructor Injection



Hello.java

```
1  package bean;
2
3  import java.util.List;
4
5  public class Hello {
6      String name;
7      Printer printer;
8
9      public Hello() { }
10
11     public Hello(String name, Printer printer) {
12         this.name = name;
13         this.printer = printer;
14     }
```

beans.xml

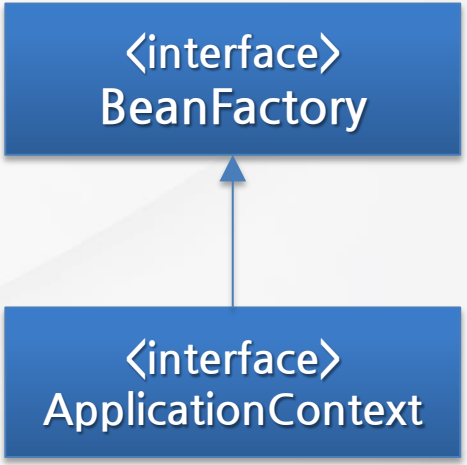
```
11
12     <bean id="hello" class="bean.Hello">
13         <constructor-arg index="0" value="Spring" />
14         <constructor-arg index="1" ref="printer" />
15     </bean>
16
17     <bean id="printer" class="bean.StringPrinter" />
18     <bean id="consolePrinter" class="bean.ConsolePrinter" />
```

3. Spring DI 컨테이너

Spring DI 컨테이너의 개념

Spring DI 컨테이너가 관리하는 객체를 빈(bean)이라고 하고,
이 빈(bean)들을 관리한다는 의미로 컨테이너를
빈 팩토리 (BeanFactory)라고 부른다.

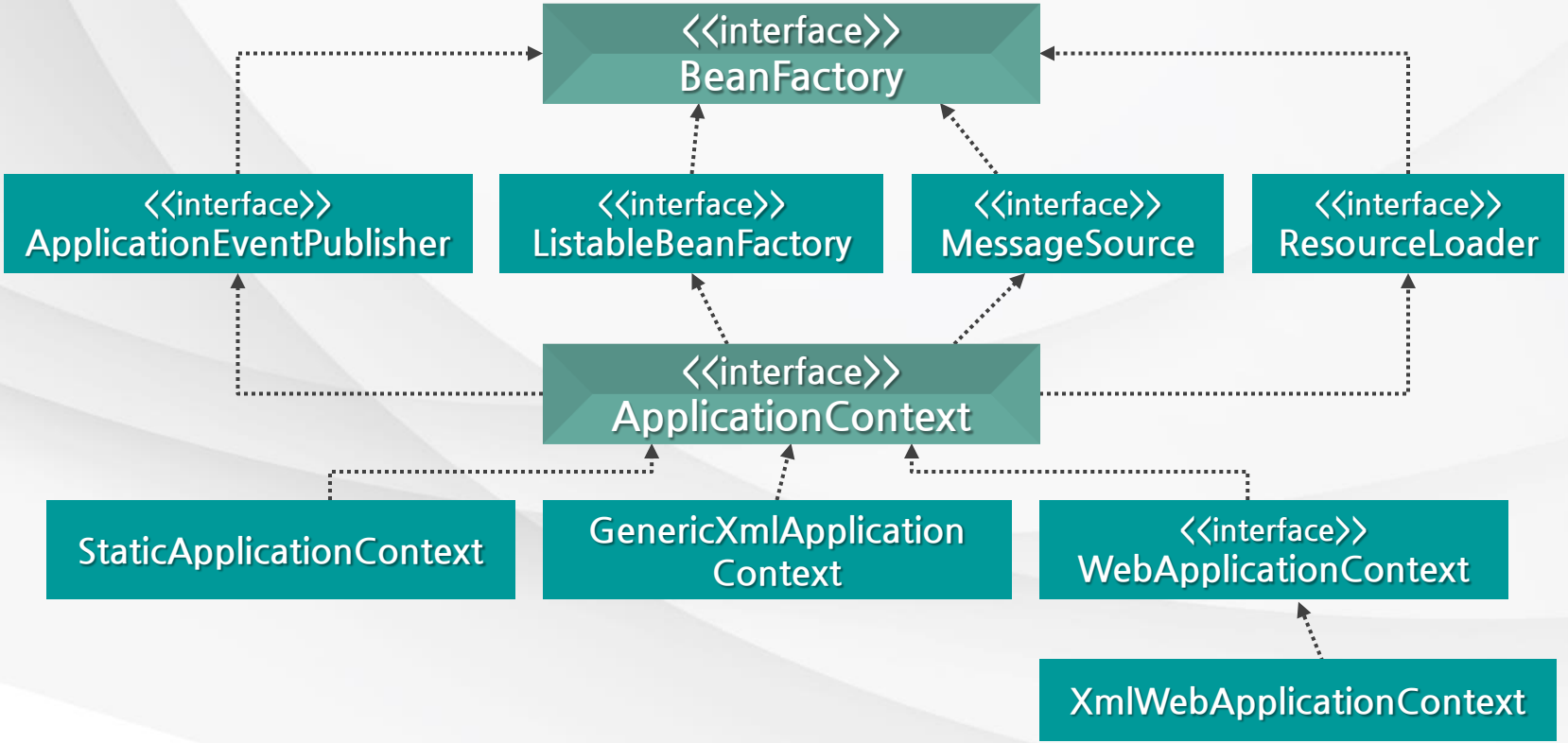
- 객체의 생성과 객체 사이의 런타임(run-time) 관계를 DI 관점에서 볼 때는 컨테이너를 BeanFactory라고 한다.
- Bean Factory에 여러 가지 컨테이너 기능을 추가하여 애플리케이션 컨텍스(ApplicationContext)라고 부름



BeanFactory와 ApplicationContext

BeanFactory	▪ Bean을 등록,생성, 조회, 반환 관리함 ▪ 보통은 BeanFactory를 바로 사용하지 않고, 이를 확장한 ApplicationContext를 사용함 ▪ getBean() 메서드가 정의되어 있음
Application Context	▪ Bean을 등록,생성, 조회, 반환 관리하는 기능은 BeanFactory와 같음 ▪ Spring의 각종 부가 서비스를 추가로 제공함 ▪ Spring이 제공하는 ApplicationContext 구현 클래스가 여러 가지 종류가 있음

BeanFactory와 ApplicationContext



학습정리

지금까지 [IoC와 DI]에 대해서 살펴보았습니다.

IoC(Inversion of Control)

제어의 역전, IoC 컨테이너, DL, DI

DI(Dependency Injection)

- 클래스 간의 의존관계를 컨테이너가 주입
- Setter Injection, Constructor Injection

Spring DI 컨테이너

BeanFactory, ApplicationContext