Spring Framework

Back-end Programming

김기정 (bangry313@gmail.com)

목차 (Table of Contents)

1. Spring Framework 소개

Rod Johnson

1.1 Spring Framework 등장 배경 (1/2)

✓ 2000년대 초반 EJB는 자바 엔터프라이즈 애플리케이션을 개발하기 위한 표준 기술로 인정되었다.

- 자바 기반 서버 기술을 총망라한 종합선물 세트로 공공기관 및 금융권 EJB 기반 프로젝트 유행
- 컨테이너 기술, 분산 기술, 설정에 의한 쉬운 트랜잭션 관리, ORM 기반 엔티티빈 지원 등
- EJB 표준을 구현한 대표적 벤더 (BEA-WebLogic, IBM-WebSphere, TMAX-JEUS 등)

✓ 그러나, EJB에서 제공하는 많은 기능에도 불구하고 EJB를 적용하는 것은 현실적으로 너무 어려웠다.

- EJB 서버가 굉장히 고가이며, 개발자들을 지옥으로 끌어들일 만큼 어려웠으며, EJB 인터페이스에 맞게 종속적으로 개발하여야 한다.

✓ EJB 지옥에서 반기를 든 2명의 구원자 등장

- POJO(Plain Old Java Object)를 외치며 EJB에 종속된 무거운 객체를 버리고, 옛날의 순수한 자바, 객체 지향으로 돌아가자.
- Rod Johnson(로드 존슨)은 자신의 저서(J2EE Design and Development)에서 EJB의 문제점을 지적하고, EJB를 사용하지 않고 훨씬 단순하면서 충분히 고품질의 확장 가능한 엔터프라이즈 애플리케이션을 개발하는 방법을 오픈 소스(3만 라인)로 공개하였다.
  - 핵심 개념(BeanFactory, ApplicationContext, POJO, IoC, DI)과 기반코드를 포함한 이것이 현재의 Spring 의 모태가 되었다.
- Gavin King(개빈 킹)은 EJB 엔티티빈을 비판하며 Hibernate Framework를 제작하여 오픈 소스로 공개하였다.
  - 자바 표준 기관에서 Gavin King 영입하여 Hibernate를 ORM 표준인 JPA 정제하여 발표

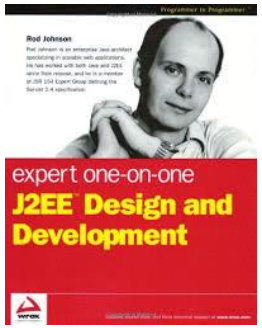
✓ 책 출간 후 Juergen Hoeller(유겐 휠러), Yann Caroff(얀 카로프)가 Rod Johnson에게 오픈소스 프로젝트 제안

- Spring 이름은 EJB 겨울을 넘어 새로운 봄의 시작이라는 의미로 명명되었다.

1.1 Spring Framework 등장 배경 (2/2)

Rod Johnson

EJB가 문제가 많아 이번에
'EJB 없이 J2EE 개발하기'
책을 냈으니 한번 보세요



오! 대박~
잘 만들었는데?
이거 오픈 소스로
함 만들어볼까?

Juergen Hoeller

Yann Caroff

ㅇㅇ~
EJB가 겨울 같았으니까
이름을
"봄" 으로 하는게 어때?

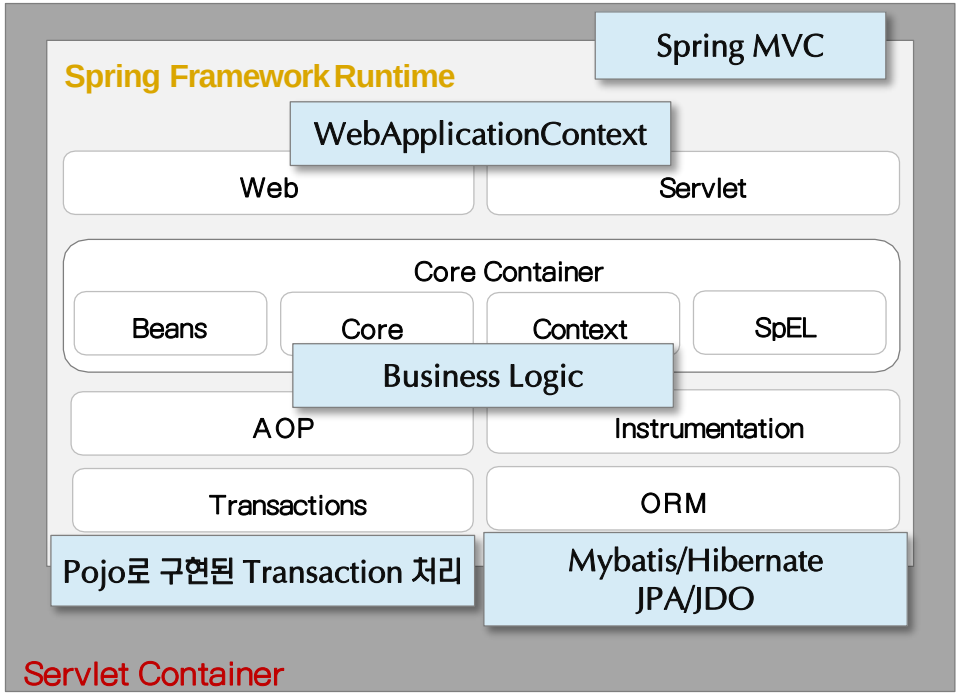
1.2 Spring Framework 개요 (1/8)

✓ **Spring은 자바 언어 기반의 엔터프라이즈 애플리케이션 개발을 위한 오픈 소스 경량 애플리케이션 프레임워크이다.**

- Spring은 Layered Architecture 기반의 애플리케이션 개발 시 모든 계층을 모두 지원한다.
- Spring은 애플리케이션 수준의 기반(설계) 구조를 제공하므로 개발자는 업무 로직 개발에만 전념할 수 있다.

✓ **Spring 핵심 컨셉은 객체지향 프로그래밍의 장점을 극대화하고, 단점을 상쇄하여 높은 품질의 애플리케이션을 개발할 수 있도록 도와주는 프레임워크이다.**

- Spring은 다형성을 극대화해서 객체 지향 프로그래밍을 잘 할 수 있도록 도와주는 도구이다
  - 하나의 기술이 아니라 객체지향 여러 기술들의 집합이다.
- Spring의 제어의 역전(IoC), 의존관계 주입(DI)은 다형성을 활용해서 역할과 구현을 편리하게 다룰 수 있도록 지원한다.

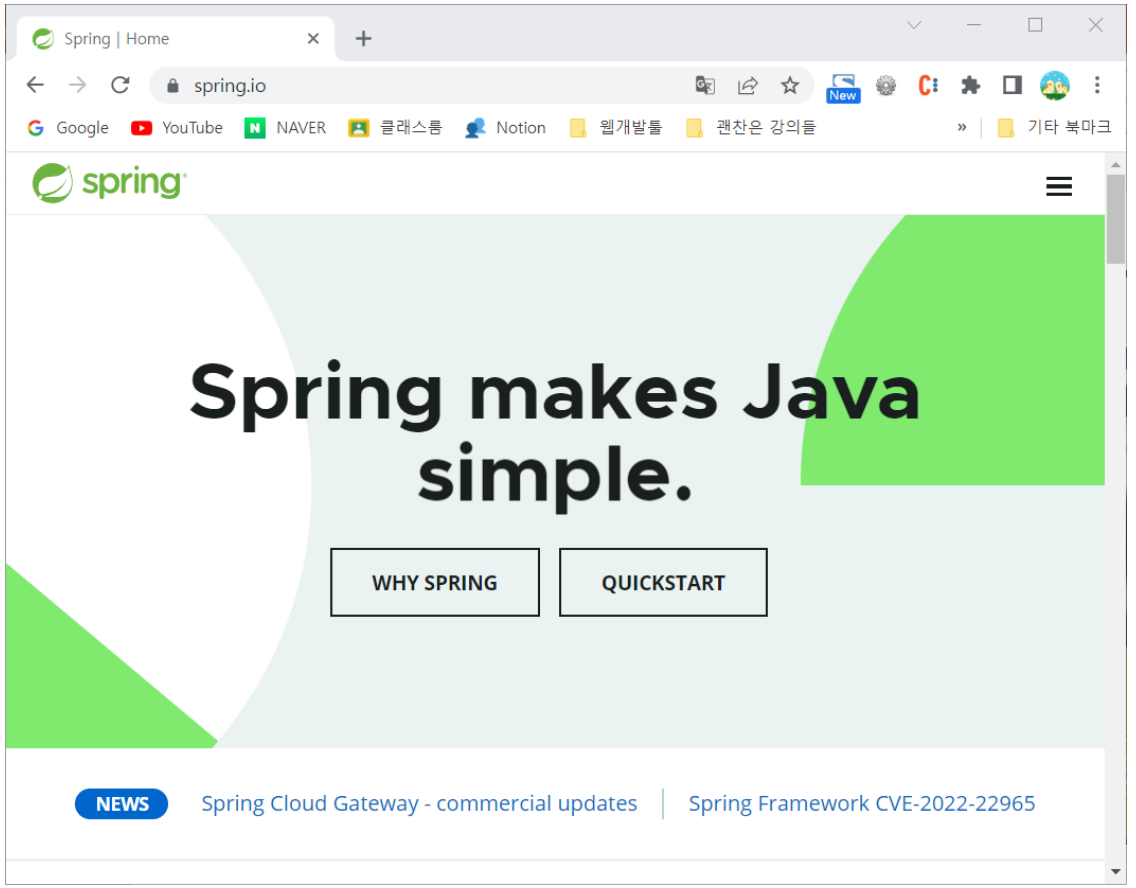


1.2 Spring Framework 개요 (2/8) – Release History

✓ 2003년 6월 아파치 2.0 라이선스로 공개

✓ Release History

- 2004년 스프링 1.0 출시 - XML 설정
- 2006년 스프링 2.0 출시 - XML 편의 기능 지원
- 2009년 스프링 3.0 출시 - 자바 코드로 설정
- 2013년 스프링 4.0 출시 - Java8 지원
- 2014년 스프링 부트 1.0 출시 - Tomcat 내장
- 2017년 스프링 5.0, 스프링 부트 2.0 출시 – Reactive Programming(비동기, 이벤트기반 프로그래밍) 지원
- 2020년 스프링 5.2.x, 스프링 부트 2.3.x 출시
- 2022년 9월 현재 스프링 5.3.23, 스프링 부트 2.7.4

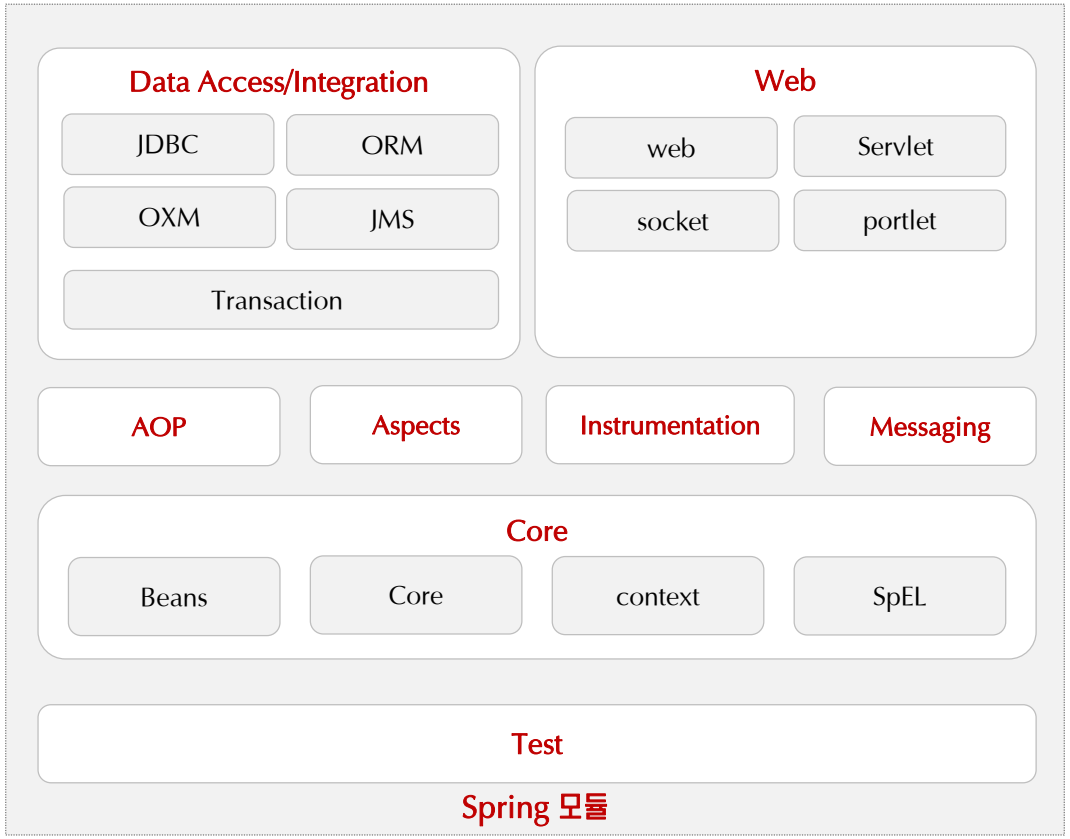


1.2 Spring Framework 개요 (3/8) – 구성 요소

✓ Spring은 EJB가 제공하는 다양한 서비스를 지원하는 경량 프레임워크이다.

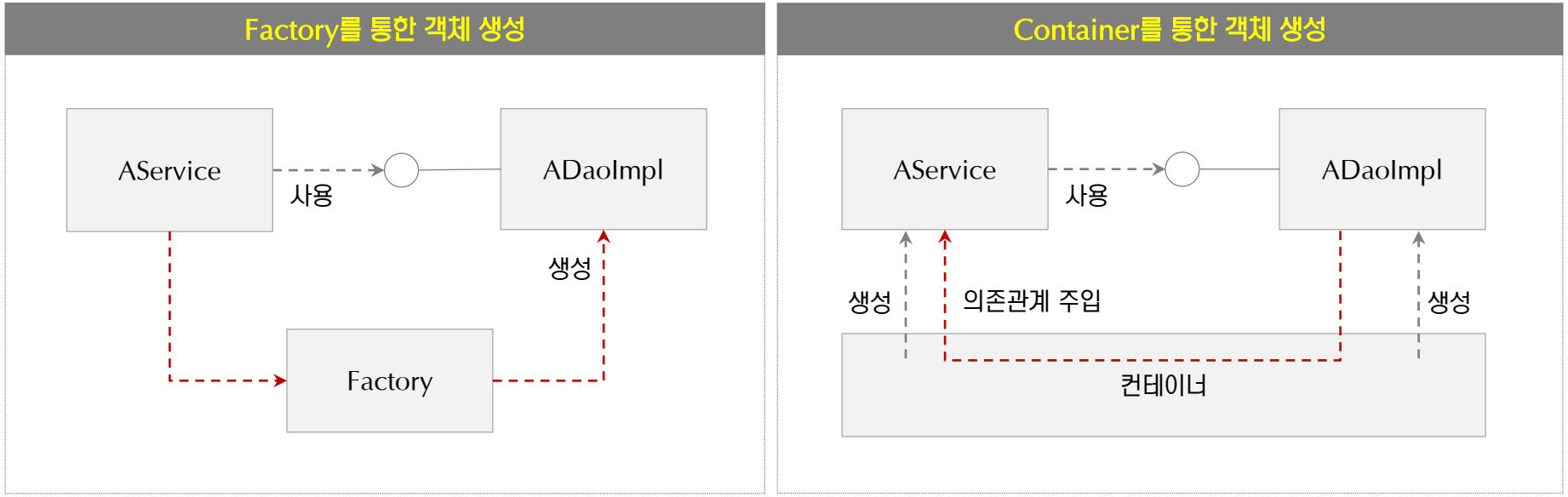
- EJB와 달리 기술과 환경에 의존적인 부분을 제거하고, 복잡성을 해결하기 위해 나온 경량 프레임워크이다.
- POJO 클래스와 인터페이스를 이용하는 심플한 구조이기 때문에 진입장벽이 높지 않고, 빠르게 서버 애플리케이션을 개발할 수 있다.

✓ Spring은 수십만 라인의 복잡한 코드로 구성되어 있으나, 20여개의 모듈로 구성되어 있어 작게 시작하여 필요한 만큼 모듈을 사용할 수 있다.



1.2 Spring Framework 개요 (4/8) – 경량 컨테이너 (Lightweight Container)

- ✓ Spring은 애플리케이션 객체의 생명주기(Lifecycle)와 설정(Configuration)을 관리하는 경량 컨테이너이다.
- ✓ Spring은 객체를 생성하고, 의존관계 설정(Dependency Injection)을 통해 객체 간의 관계를 설정한다.
- ✓ 애플리케이션 모든 계층의 객체가 관리 대상이며, 각 계층이나 서비스들 간의 의존관계를 관리한다.
- ✓ 의존관계 설정(Dependency Injection)을 통해 인터페이스 기반의 업무 로직을 컴포넌트로 모듈화 한다.

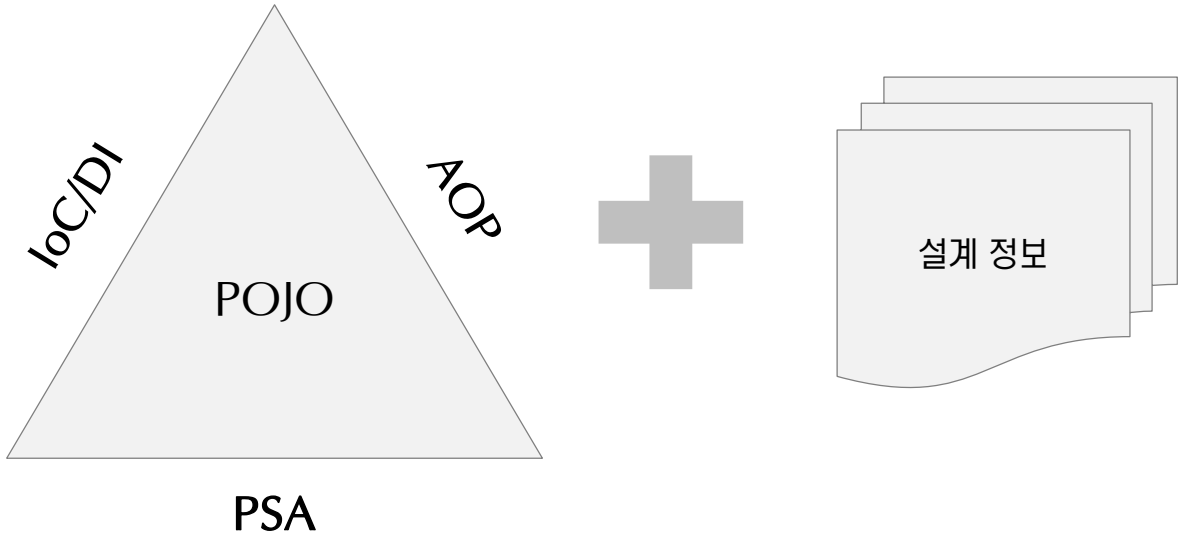


1.2 Spring Framework 개요 (5/8) – POJO 프레임워크 (1/2)

- ✓ Spring은 업무 로직과 엔터프라이즈 기술의 복잡함을 분리하여 처리 할 수 있도록 지원한다.
- ✓ 다시 말해, Spring은 특정 프레임워크나 기술에 얽매이지 않는 POJO로 업무 로직을 구성할 수 있도록 해준다.
- ✓ POJO(Plain Old Java Object)란 특별한 기술에 종속되지 않은 순수한 자바 객체를 말한다.
- ✓ POJO로 개발하면 테스트가 쉽고, 객체지향 설계 적용이 용이하여 개발 생산성 및 이식성이 향상된다.

1.2 Spring Framework 개요 (6/8) – POJO 프레임워크 (2/2)

- ✓ Spring은 업무 로직을 POJO만으로 개발할 수 있는 POJO 프레임워크이다.
- ✓ Spring은 POJO로 개발할 수 있도록 IoC/DI , AOP, PSA 기술을 지원한다.
- ✓ Spring 애플리케이션은 POJO를 이용해서 만든 업무 코드와 설계정보로 구성된다.
- ✓ 설계정보는 POJO 사이의 관계 및 동작방법을 설정한다.



1.2 Spring Framework 개요 (7/8) – Spring 프로젝트들

Spring Boot

Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.

Spring Framework

Provides core support for dependency injection, transaction management, web apps, data access, messaging, and more.

Spring for GraphQL

Spring for GraphQL provides support for Spring applications built on GraphQL Java.

Spring Session

Provides an API and implementations for managing a user's session information.

Spring Data

Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.

Spring Cloud

Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.

Spring Integration

Supports the well-known Enterprise Integration Patterns through lightweight messaging and declarative adapters.

Spring HATEOAS

Simplifies creating REST representations that follow the HATEOAS principle.

Spring Cloud Data Flow

Provides an orchestration service for composable data microservice applications on modern runtimes.

Spring Security

Protects your application with comprehensive and extensible authentication and authorization support.

Spring REST Docs

Lets you document RESTful services by combining hand-written documentation with auto-generated snippets produced with Spring MVC Test or REST Assured.

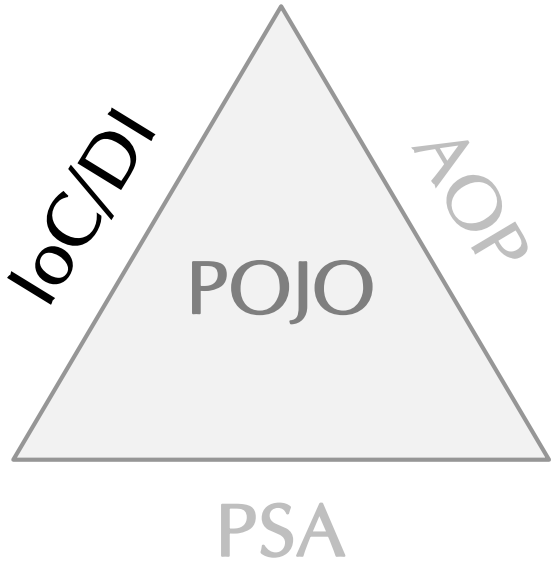
Spring Batch

Simplifies and optimizes the work of processing high-volume batch operations.

1.2 Spring Framework 개요 (8/8) – 다양한 오픈소스 프레임워크

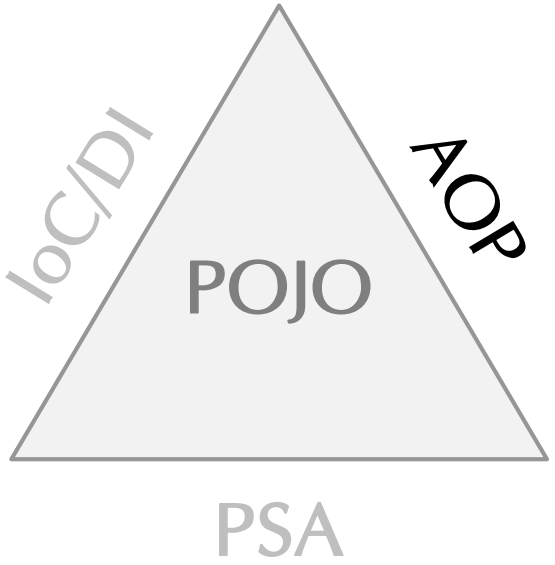
1.3 Spring Framework 핵심 기술 (1/6) – IoC/DI

- ✓ Inversion Of Control(제어의 역전)를 줄여서 IoC라고 한다.
  - 객체의 생성 및 의존관계 설정을 자바 코드에서 하는 것이 아니라 외부 컨테이너를 이용하여 설정
- ✓ IoC는 객체지향 언어에서 객체의 생성 및 객체 간의 연결 관계를 런타임 시에 결정하게 하는 방법이다.
- ✓ IoC는 객체 간의 관계를 느슨하게 연결하도록 한다 (loose coupling)
- ✓ DI는 Dependency Injection의 약자로 IoC의 구현 방법 중 하나이다.



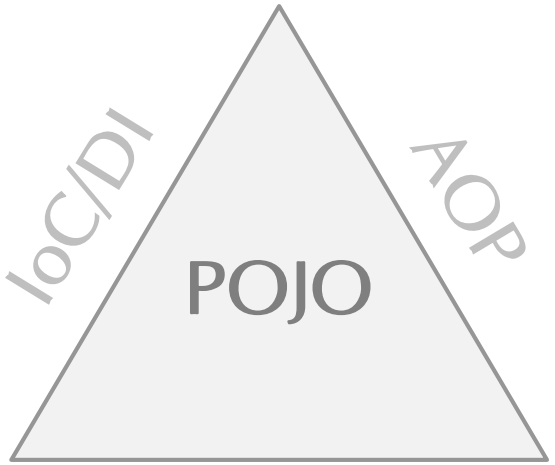
1.3 Spring Framework 핵심 기술 (2/6) - AOP

✓ Aspect Oriented Programming를 줄여서 AOP 라고 한다.
✓ AOP는 관심사의 분리를 통해서 소프트웨어 모듈성을 향상시키고자 하는 프로그래밍 패러다임이다.
✓ 소스코드 레벨에서 관심사의 모듈화를 지향하는 프로그래밍 방법이나 도구를 포함한다.



1.3 Spring Framework 핵심 기술 (3/6) – PSA

- ✓ Portable Service Abstraction을 줄여서 PSA라고 한다.
- ✓ 환경과 세부 기술의 변화에 관계없이 어떤 기술을 내부에 숨기고, 일관된 방식으로 기술에 접근할 수 있게 해주는 설계 원칙으로 개발자에게 개발 편의성을 제공해주는 서비스이다
- ✓ 데이터베이스에 관계 없이 동일하게 적용할 수 있는 트랜잭션 처리 방식을 예로 들 수 있다.



PSA

1.3 Spring Framework 핵심 기술 (4/6) - 컨셉

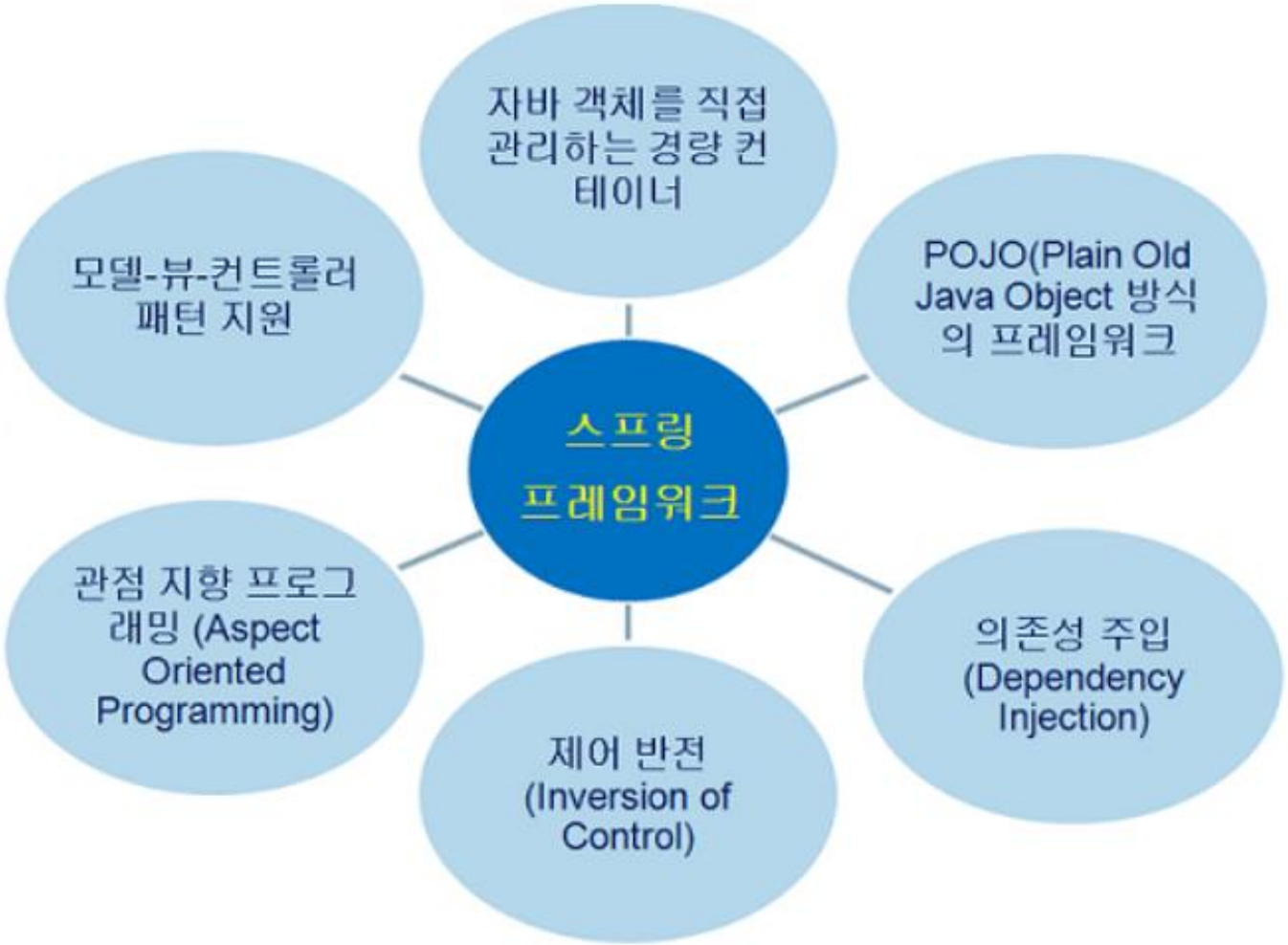
✓ 순수한 자바 스타일, 순수한 객체 지향 프로그래밍 스타일로 돌아가자!

- POJO 용어 등장함.
- Spring은 객체 지향 언어가 가지는 강력한 특징 (추상화, 캡슐화, 상속, 다형성)을 잘 살려내는 프레임워크이다.
  - 다형성 : 역할(인터페이스)과 구현(클래스)을 분리하여 개발하면 애플리케이션이 단순해지고, 유연하며 변경이 용이해 진다.

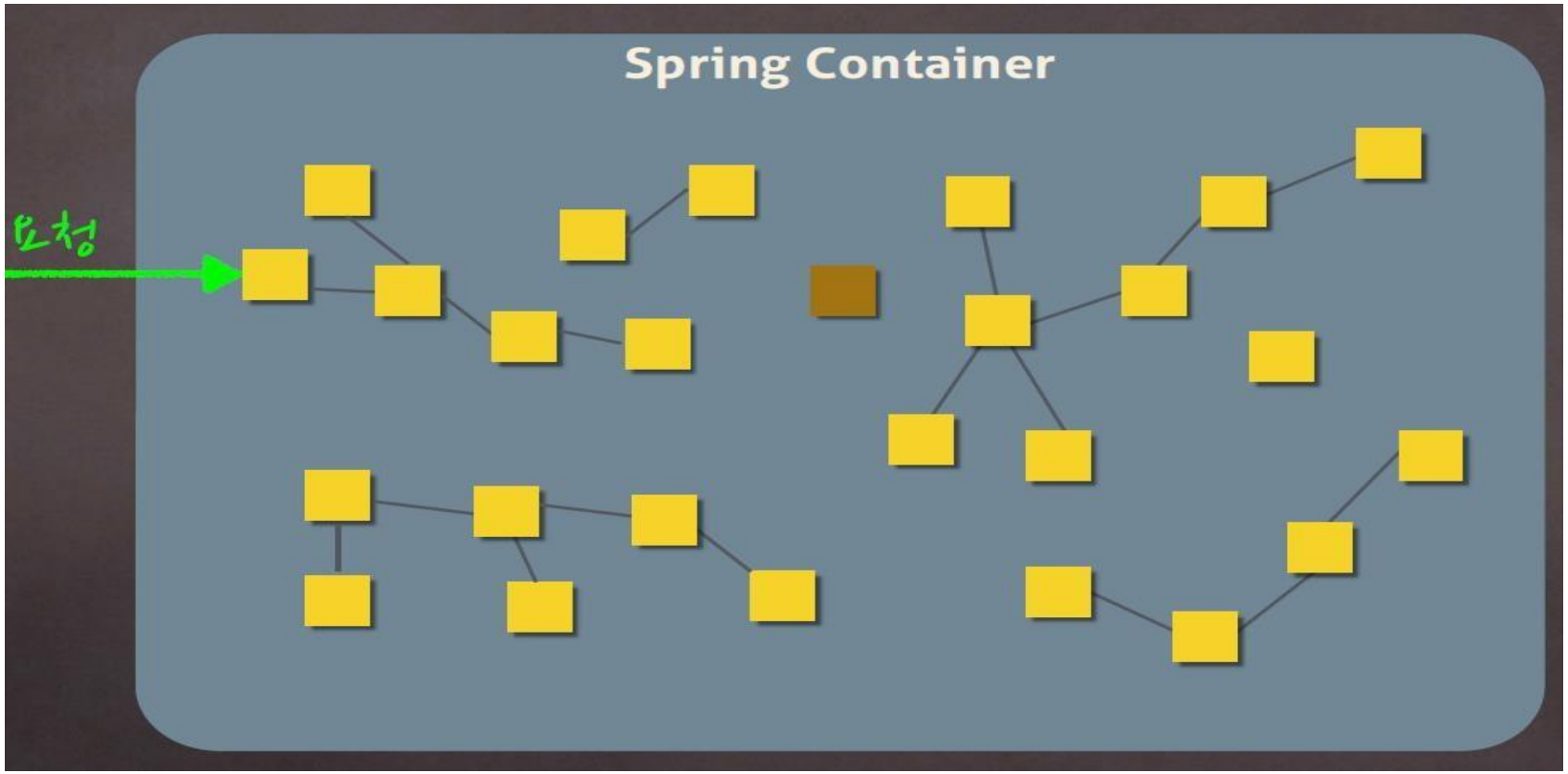
✓ Spring의 DI Container가 객체 지향 프로그래밍을 잘 할 수 있도록 도와준다.

✓ Spring은 품질 좋은 객체 지향 애플리케이션을 쉽게 개발할 수 있도록 도와주는 범용 프레임워크이다.

1.3 Spring Framework 핵심 기술 (5/6) - 요약



1.3 Spring Framework 핵심 기술 (6/6) - 요약



Java 클래스는 Spring 컨테이너 의해 생성, 관리될 수 있도록 POJO로 작성하고, Spring 프로그래밍 모델(IOC/DI, AOP, PSA)을 준수하며, 엔터프라이즈 관련 기술을 사용할 때는 Spring이 지원하는 Service와 API를 사용한다.

2. Spring Container, Spring Bean

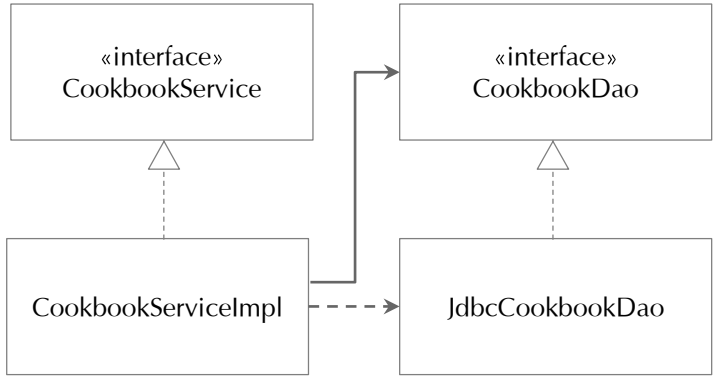
2.1 IoC (1/6)

✓ 일반적인 애플리케이션들은 필요한 객체를 자바 소스코드에서 직접 생성하여 사용한다.
✓ 자바 객체지향 프로그래밍에서는 인터페이스와 추상 클래스를 사용하여 객체 간의 의존성을 줄일 수 있다.
✓ 그러나, 소스코드에 하드 코딩 된 객체 생성 로직은 객체 간의 의존성을 강제한다.
✓ 결국, 의도하지는 않았지만 객체 간의 결합도는 높아진다.

```java
public class CookbookServiceImpl implements CookbookService {

    private CookbookDao cookbookDao;

    public CookbookServiceImpl() {
        this.cookbookDao = new JdbcCookbookDao();
     }

    @Override
    public List<Cookbook> findAllCookbooks() {
        //
        return cookbookDao.retrieveAll();
    }

    *** 중략 ***
}

```

CookbookServiceImpl.java

클래스 다이어그램

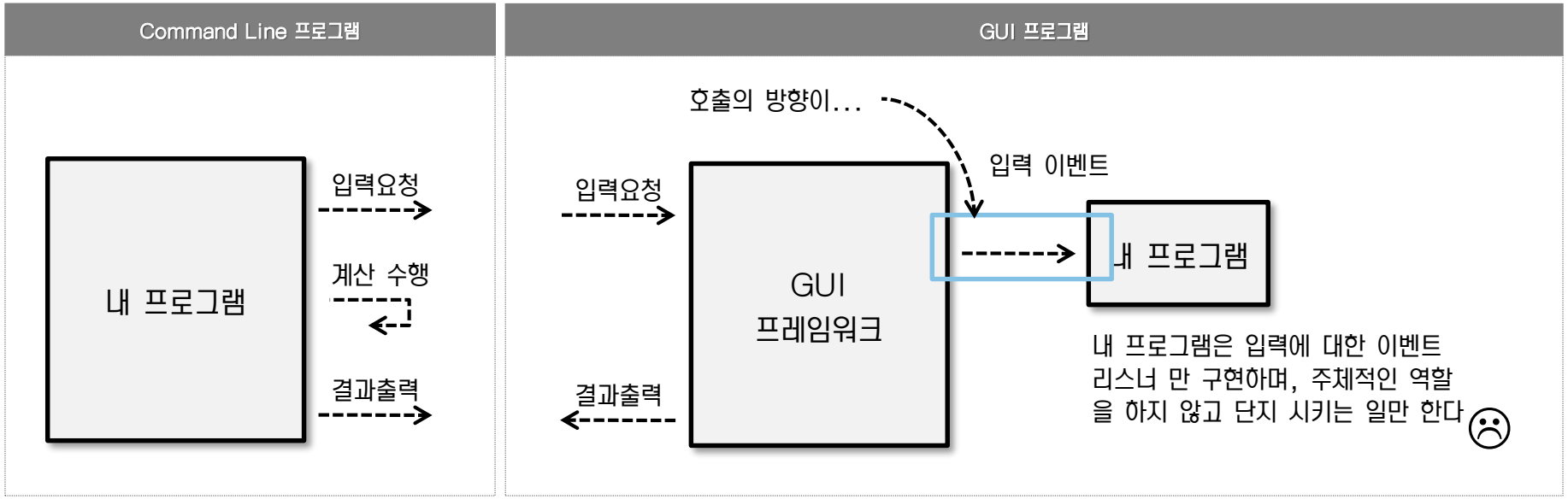


코드설명

new 생성자를 이용하여 객체를 생성한다.

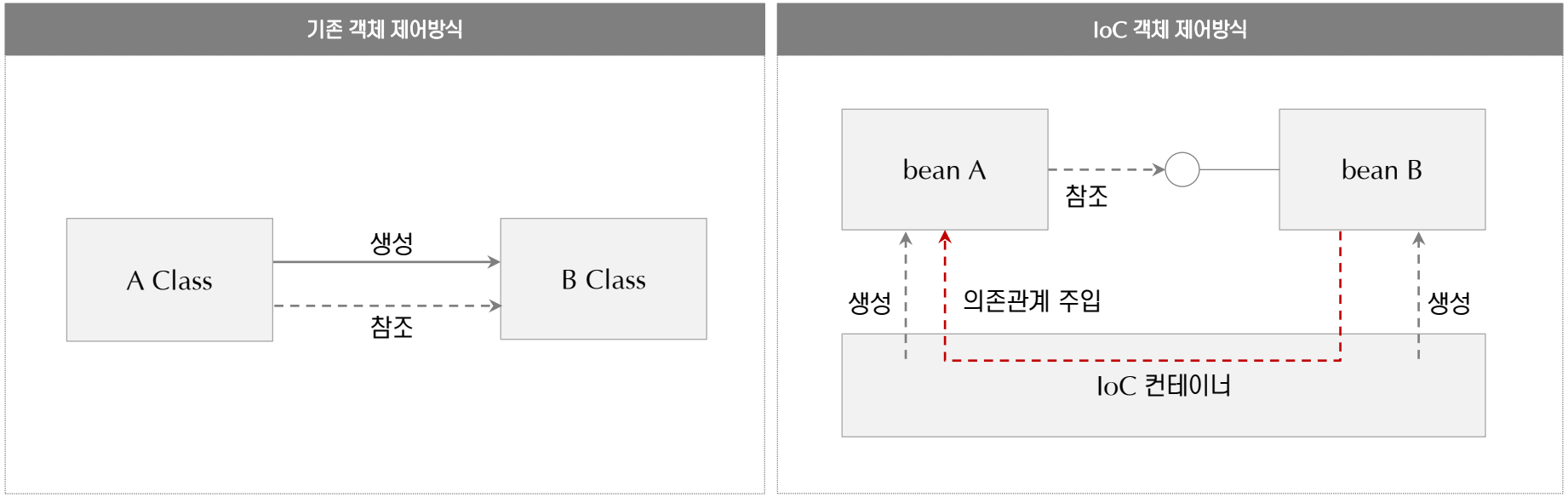
2.1 IoC (2/6)

- ✓ IoC(Inversion of Control)는 통제 방향의 변경을 의미한다.
- ✓ 이는 프레임워크의 일반적인 특징으로 어떤 일을 제어하는 주체를 변경하는 것을 말한다.
- ✓ 최초의 Command Line 프로그램은 애플리케이션 클래스가 모든 것을 제어한다.
- ✓ 그와 달리, GUI 프로그램은 프레임워크가 제어하며 내 프로그램에서는 이벤트리스너 만 구현한다.



2.1 IoC (3/6)

✓ 객체 간 결합도가 높으면, 해당 클래스가 변경될 때 결합된 다른 클래스도 같이 수정될 가능성이 있다.
✓ IoC는 객체 생성 책임을 컨테이너에게 위임하여, 객체 간의 결합도를 낮춘다. (loose coupling)
✓ IoC를 통한 객체 제어 방식은 기존 로직에서 객체를 생성하는 로직을 제거한다.
✓ IoC는 구현하는 방법에 따라, Dependency Lookup과, Dependency Injection 방법이 있다.



2.1 IoC (4/6) – Dependency Injection (1/3)

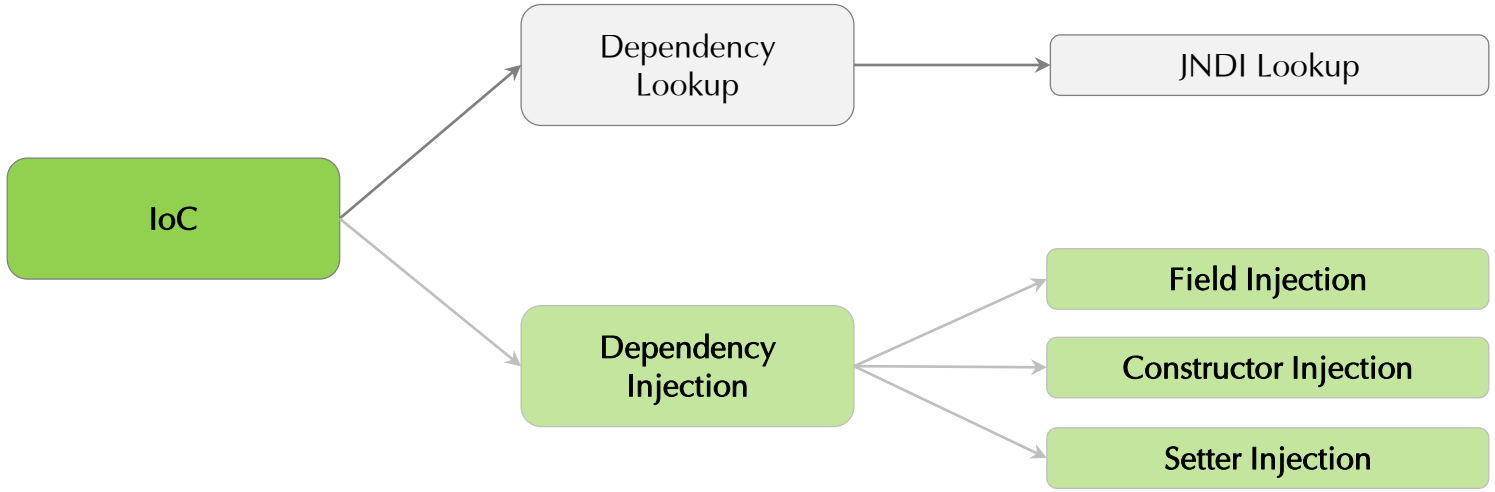
✓ Dependency Lookup 방식은 EJB나 Spring 에서 JNDI 리소스를 얻는 방식이다.

- 컨테이너가 제공하는 Lookup Context 를 통해서 필요한 자원이나 객체를 얻을 수 있다.
- 그러나, 이 방식은 결과적으로 컨테이너 API에 대한 의존성을 높이게 된다.

✓ Dependency Injection은 컨테이너가 직접 의존관계를 객체에 설정할 수 있도록 지정하는 방식이다.

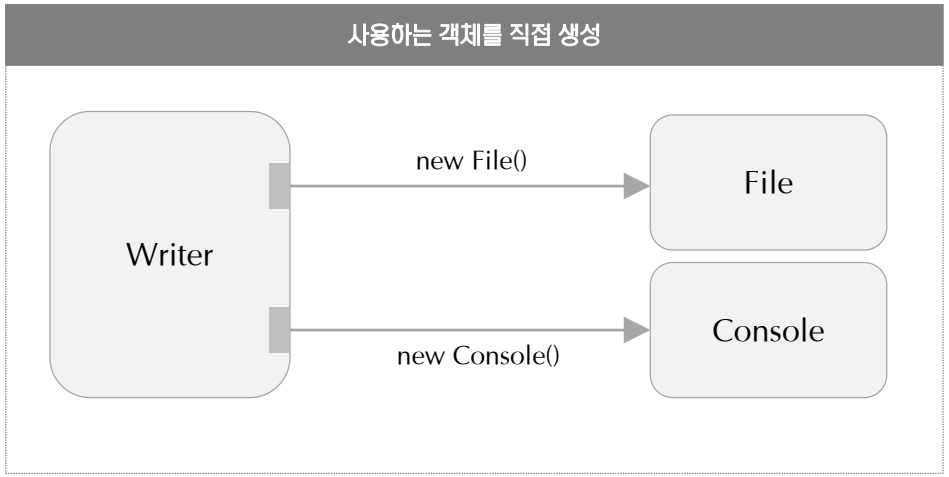
- 외부(컨테이너)에서 부품(객체)을 조립하듯이 애플리케이션 개발하는 방식이다.
- Injection 방법에는 Field Injection, Constructor Injection, Setter Injection 이 있다.

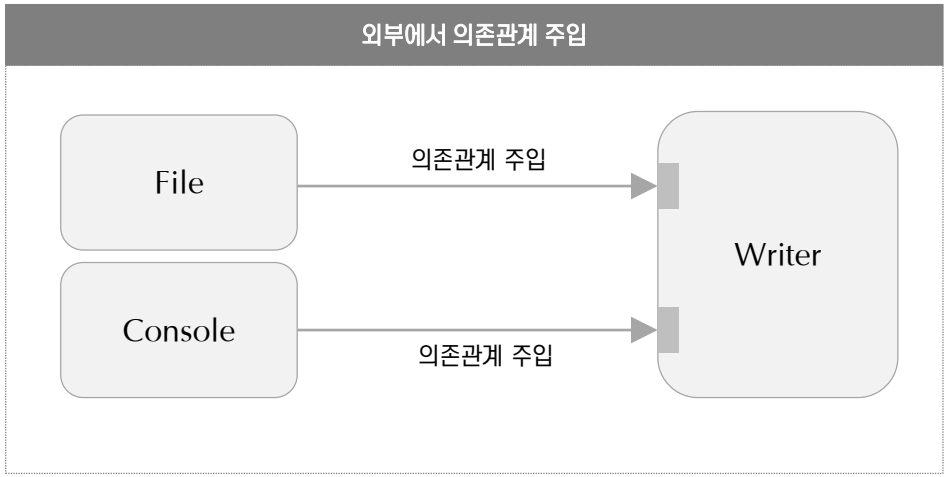
✓ Spring은 컨테이너와의 의존성을 줄이기 위해서는 DI 방식을 사용한다.



2.1 IoC (5/6) – Dependency Injection (2/3)

✓ 의존 관계 주입 Dependency Injection을 줄여서 DI라고 한다.
✓ DI를 통해, 외부 컨테이너에서 객체를 생성하는 시점에 참조하는 객체에게 의존관계를 제공한다.
✓ 객체가 인터페이스만 알고 있으므로, 느슨한 결합(loose coupling)이 가능하다.





객체는 인터페이스에 의한 의존관계만을 알고 있으며,
구현 클래스에 대한 차이를 모르므로 서로 다른 구현체로 대체가 가능하다.

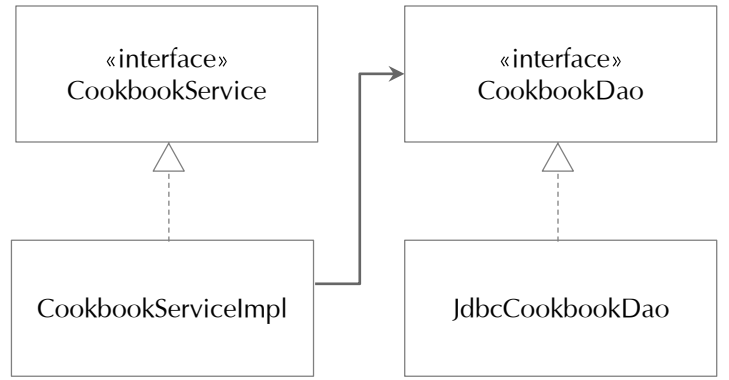
2.1 IoC (6/6) – Dependency Injection (3/3)

- ✓ 사용하는 객체 생성을 IoC 컨테이너에게 위임하여, 객체 생성 로직이 없어졌다 (관심사의 분리)
- ✓ 객체 생성에 대한 주도권을 객체를 필요로 하는 곳으로 넘겨주어, 필요할 때 필요한 곳에서 객체를 생성하는 방법이다.
- ✓ Spring은 Field Injection, Constructor Injection, Setter Injection 등 세가지 DI 패턴을 모두 지원한다.
- ✓ Spring은 의존관계를 설정하는 방법으로 XML, Annotation, 자바 소스코드를 사용하는 방법을 제공한다.

```java
public class CookbookServiceImpl implements CookbookService {

    private CookbookDao cookbookDao;

    public void setCookbookDao(CookbookDao cookbookDao) {
        this.cookbookDao = cookbookDao;
    }

    @Override
    public List<Cookbook> findAllCookbooks() {
        //
        return cookbookDao.retrieveAll();
    }

    ......
}
```

CookbookServiceImpl.java

클래스 다이어그램



코드설명

cookbookDao 객체를 new 생성 없이 IoC 컨테이너로 부터 Setter 방식으로 Injection 한다.

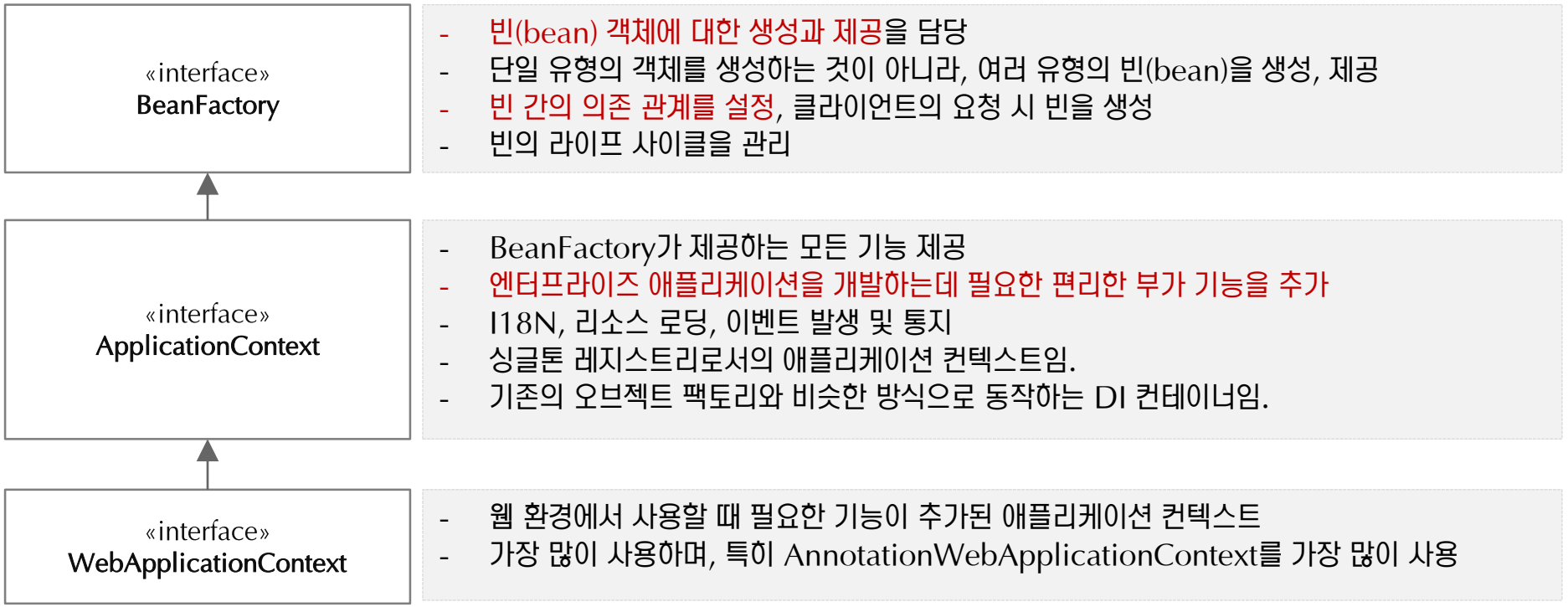
2.2 Spring IoC (1/3) – Spring IoC 용어

✓ 빈(Bean)이란, Spring이 IoC 방식으로 관리하는 자바 객체를 의미한다.
✓ IoC 컨테이너는 빈의 생성과 제어의 관점에서 빈 팩토리(Bean Factory)라고도 한다.
✓ 애플리케이션 컨텍스트(Application Context)는 Spring이 제공하는 애플리케이션 지원 기능을 모두 포함하는 의미이다.
✓ Spring 프레임워크는 IoC 컨테이너와 애플리케이션 컨텍스트를 모두를 포함한다.

IoC 용어	설명
빈 (bean)	Spring이 IoC 방식으로 관리하는 오브젝트 관리 되는 오브젝트라고도 함. Spring이 직접 그 생성과 제어를 담당하는 오브젝트만을 빈(bean)이라고 함.
빈 팩토리 (bean factory)	Spring이 IoC를 담당하는 핵심 컨테이너로 빈을 등록, 생성하고, 조회하고 돌려주고, 그 외에 부가적으로 빈을 관리하는 기능을 담당함. 보통은 이 빈 팩토리를 바로 사용하지 않고 이를 확장한 애플리케이션 컨텍스트를 이용합니다.
애플리케이션 컨텍스트 (application context)	빈 팩토리를 확장한 IoC 컨테이너로 빈을 등록하고 관리하는 기본적인 기능은 빈 팩토리와 동일함. Spring이 제공하는 각종 부가 서비스를 추가로 제공함. Spring DI Container.
설정정보/설정 메타정보 (configuration metadata)	애플리케이션 컨텍스트 또는 빈 팩토리가 IoC를 적용하기 위해 사용하는 설정정보
Spring 프레임워크	IoC 컨테이너, 애플리케이션 컨텍스트를 포함해서 Spring이 제공하는 모든 기능 칭함.

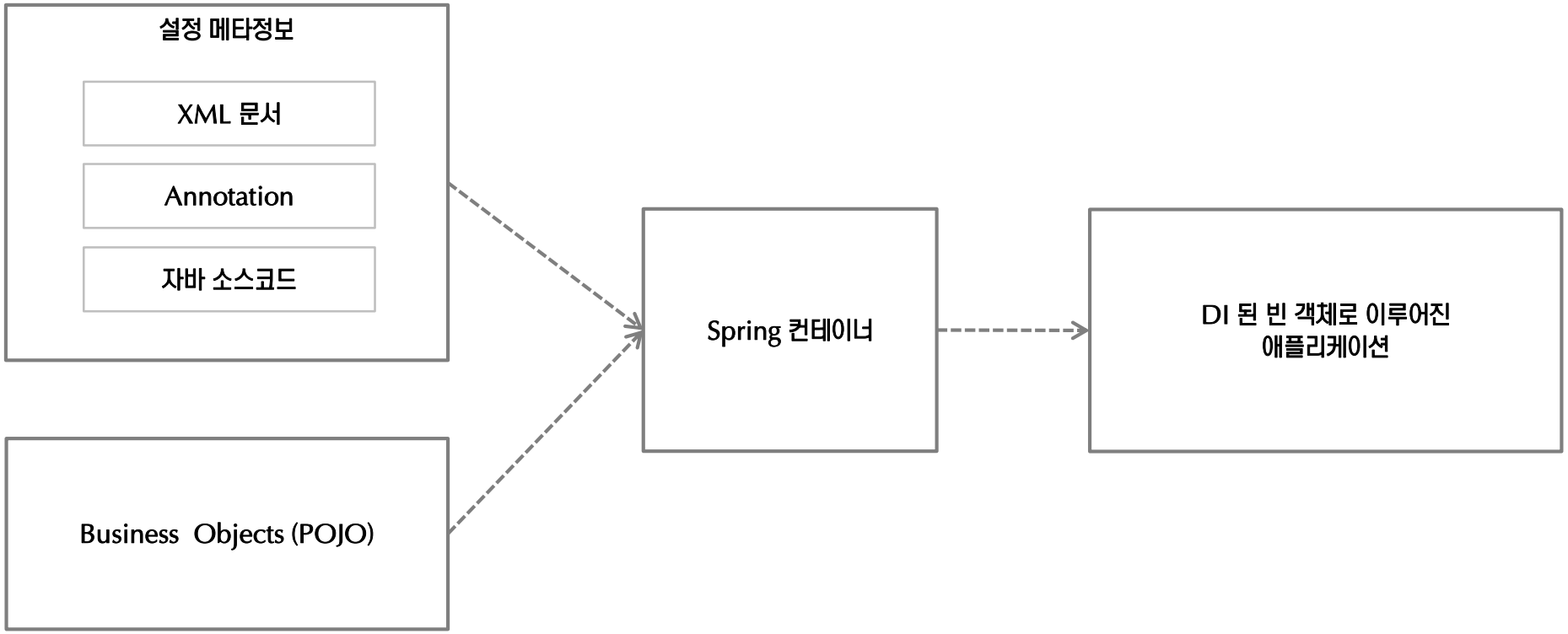
2.2 Spring IoC (2/3) – Spring IoC 컨테이너

- ✓ AppConfig 처럼 객체를 생성하고, 관리하면서 의존관계를 설정해 주는 것을 IoC 컨테이너 또는 DI 컨테이너라 한다.
- ✓ 자바 코드 대신 컨테이너가 객체에 대한 제어권을 갖고 있어, Spring 컨테이너를 IoC 컨테이너 또는 DI 컨테이너라고 한다.
- ✓ Spring에서 IoC를 담당하는 컨테이너는 BeanFactory, ApplicationContext가 있다.
- ✓ Spring은 기본적으로 따로 설정하지 않으면 내부에서 생성하는 빈 객체를 모두 싱글톤으로 관리한다.



2.2 Spring IoC (3/3) – 설정 메타정보

✓ 하나의 애플리케이션은 Spring 컨테이너에 의해 POJO 클래스와 설정 메타정보가 결합되어 만들어진다.
✓ 설정 메타정보는 애플리케이션을 구성하는 객체와 객체사이의 상호 의존성을 포함한다.
✓ Spring은 XML설정 파일과 Annotation 설정, 자바 소스코드로 메타정보의 설정이 가능하다.



2.3 Spring XML 빈(bean) 관리 (1/3) – 빈 선언

- ✓ XML 방식으로 Spring 빈을 설정할 수 있다.
- ✓ 이 방법은 단순하여 사용하기 쉽다. 초창기에 많이 사용한 방식으로 빈의 설정 메타 정보를 XML문서 형태로 기술한다.
- ✓ XML 설정파일에 <bean> 태그를 작성하여 빈을 선언한다.
- ✓ Spring은 빈을 선언함과 동시에 <property> 태그의 ref 요소를 이용하여 의존관계를 설정한다.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
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-4.0.xsd">

<bean id="cookbook" class="com.namoo.yorizori.dao.jdbc.JdbcCookbookDao"/>
<bean id="cookbookService" class="com.namoo.yorizori.service.impl.CookbookServiceImpl">
    <property name="cookbookDao" ref="cookbook" />
</bean>
</beans>
```

applicationContext.xml

코드설명

[Line11] <bean> 의 id 속성을 이용하여 빈을 선언한다.
[Line12] <property> 의 ref 속성 값과 동일한 id 또는 name을 갖고 있는 빈을 찾아서 cookbookService빈의 cookbookDao프로퍼티에 주입하도록 설정한다.

```java
public class CookbookServiceImpl implements CookbookService {

    private CookbookDao cookbookDao;

    public void setCookbookDao(CookbookDao cookbookDao) {
        this.cookbookDao = cookbookDao;
    }
    ......
}
```

CookbookServiceImpl .java

코드설명

[Line5] XML 에서 프로퍼티로 의존관계를 설정하였으므로, setter가 필요하다.

2.3 Spring XML 빈(bean) 관리 (2/3) – Autowiring (1/2)

✓ XML 방식으로 빈을 선언하면, 자동 엮음(Autowiring)을 이용하여 Spring 빈 사이의 의존관계를 설정할 수 있다.
✓ Autowiring은 명시적으로 프로퍼티나 생성자 파라미터 지정 없이 컨테이너가 DI 설정을 추가한다.
✓ 대표적인 Autowiring 방식으로 byName과 byType이 있다.

Autowiring 유형	설명
byName	빈(bean)의 이름(ID)이 엮여지는 속성의 이름과 같은 것을 컨테이너에서 찾는다. 만일 일치하는 빈(bean)이 없으면 속성은 엮여지지 않은 상태로 남게 된다.
byType	빈(bean)의 타입이 엮여지는 속성의 타입과 같은 것을 컨테이너에서 단일한 빈(bean)을 찾는다. 만일 일치하는 빈(bean)이 없으면 속성은 엮여지지 않은 상태로 남게 된다. 일치하는 빈(bean)이 두 개 이상 이면 org.springframework.beans.factory.UnsatisfiedDependencyException 예외가 발생한다.
constructor	엮여질 빈(bean) 생성자들 중에 하나의 파라미터들로 컨테이너의 하나 이상의 빈(bean)들을 일치시킨다. 모호한 빈(bean)이나 모호한 생성자의 경우, org.springframework.beans.fact-tory.UnsatisfiedDependcyException 예외가 발생한다.
autodetect	처음에 constructor 를 사용해서 자동 엮음(autowiring)을 시도하고 그 다음에 byType으로 자동 엮음(autowiring)을 시도한다. 모호한 부분은 constructor와 byType 과 같이 동일한 방식으로 처리한다.

2.3 Spring XML 빈(bean) 관리 (3/3) – Autowiring (2/2)

- ✓ 자동 엮음(Autowiring)을 이용할 경우, Autowiring 방식에 대한 선언이 필요하다.
- ✓ 각 빈마다 Autowiring 방식을 지정할 수 있다.
- ✓ default-autowire를 사용하면 선언된 모든 빈에 적용할 수 있다.

Autowiring 를 적용하지 않은 빈 선언

```
<beans xmlns=http://www.springframework.org/schema/beans *** >

<bean id="cookbook"
class="com.namoo.yorizori.dao.jdbc.JdbcCookbookDao"/>
<bean id="cookbookService“
class="com.namoo.yorizori.service.impl.CookbookServiceImpl">
    <property name="cookbookDao" ref="cookbook" />
</bean>
</beans>
```

applicationContext.xml

DI 가 필요한 클래스

```
public class CookbookServiceImpl implements CookbookService {

    private CookbookDao cookbookDao;

    public void setCookbookDao(CookbookDao cookbookDao) {
        this.cookbookDao = cookbookDao;
    }
    *** 중략 ***
}
```

CookbookServiceImpl .java

autowire를 이용하여 빈의 Autowiring 방식 설정

```
<beans xmlns=http://www.springframework.org/schema/beans*** >

<bean id="cookbook"
class="com.namoo.yorizori.dao.jdbc.JdbcCookbookDao" />
<bean id="cookbookService"
class="com.namoo.yorizori.service.impl.CookbookServiceImpl"
autowire="byType"/>
</beans>
```

applicationContext.xml

default-autowire 을 이용한 모든 빈의 Autowiring 방식 설정

```
<beans xmlns=http://www.springframework.org/schema/beans***
default-autowire="byType">

<bean id="cookbook"
class="com.namoo.yorizori.dao.jdbc.JdbcCookbookDao" />
<bean id="cookbookService"
class="com.namoo.yorizori.service.impl.CookbookServiceImpl" />
</beans>
```

applicationContext.xml

autowire 속성을 byName 으로 설정한다면, CookbookServiceImpl의 setCookbookDao를 setCookbook으로 변경해야 합니다. 그 이유는, JdbcCookbookDao 빈을 cookbook으로 선언하였기 때문에 Spring이 Setter Injection 방식으로 CookbookServiceImpl의 cookbookDao 에 주입하기 위해서는 해당 이름의 setter가 필요하기 때문이다.

2.4 Spring Annotation 빈(bean)관리 (1/8) – 빈 선언 (1/2)

- ✓ 클래스에 Annotation을 추가하는 방식으로 Spring 빈을 설정할 수 있다.
- ✓ 빈으로 사용할 클래스에 특별한 Annotation을 부여해 주면 자동으로 빈으로 등록한다.
- ✓ "객체 빈 스캐너"로 빈 스캐닝을 통해 자동으로 빈으로 등록해 준다.
- ✓ 빈 스캐너는 기본적으로 빈의 아이디로 클래스 이름(클래스 이름의 첫 글자만 소문자로 바꾼 것)을 사용한다.

```java
@Component
public class CookbookServiceImpl implements CookbookService {

    @Autowired
    private CookbookDao cookbookDao;

    @Autowired
    private RecipeDao recipeDao;

    @Override
    public List<Cookbook> findAllCookbooks() {
        //
        return cookbookDao.retrieveAll();
    }

    ......
}
```

CookbookServiceImpl .java

코드설명

[Line1] 자동 빈 등록이 가능한 스테레오타입 어노테이션을 클래스에 추가한다.

2.4 Spring Annotation 빈(bean)관리 (2/8) – 빈 선언 (2/2)

- ✓ 어노테이션으로 빈 선언 시 <context:component-scan> 설정이 필요하다.
- ✓ Context 네임스페이스의 태그를 처리할 수 있는 핸들러가 컨테이너 인프라 빈을 등록한다.
- ✓ 컨테이너 인프라 빈의 빈 스캐너 기능을 통해 @Component와 같은 스테레오타입의 클래스를 빈으로 등록한다.
- ✓ Component-scan은 빈 스캐너 기능과 <context:annotation-config>의 기능을 포함한다.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
xsi:schemaLocation="http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc-4.0.xsd
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-4.0.xsd">


<!-- Service Component Scan -->
<context:component-scan base-package="com.namoo.yorizori.service" />

<!-- DAO Component Scan -->
<context:component-scan base-package="com.namoo.yorizori.dao" />

</beans>
```

applicationContext.xml

2.4 Spring Annotation 빈(bean)관리 (3/8) – Stereotype Annotation

✓ Spring은 ApplicationContext의 BeanDefinition를 자동적으로 등록하여,
✓ @Repository, @Service, @Controller, @Component 스테레오타입 클래스들을 자동으로 인식한다.
✓ Spring이 관리하는 모든 컴포넌트를 계층 별로 빈의 특성이나 종류에 따라 스테레오타입 어노테이션을 사용한다.
✓ 이를 통해, AOP Pointcut 표현식을 사용하여 특정 어노테이션을 선언한 클래스만을 선정할 수 있다.

특정 계층에서 사용하는 스테레오타입 어노테이션

스테레오타입 어노테이션	설명
@Repository	데이터 액세스 계층의 DAO 또는 Repository 클래스에 사용 DataAccessException 자동변환과 같은 AOP의 적용 대상을 선정하기 위해 사용하기도 함
@Service	서비스 계층의 클래스에 사용
@Controller	프리젠테이션 계층의 MVC 컨트롤러에 사용. Spring 웹 서블릿에 의해 웹 요청을 처리하는 컨트롤러 빈으로 선정
@Component	위의 계층 구분을 적용하기 어려운 일반적인 경우에 사용

2.4 Spring Annotation 빈(bean)관리 (4/8) – Injection Annotation

✓ 어노테이션을 이용하여 Spring 빈 사이의 의존관계를 설정할 수 있다.
✓ 지원하는 표준 스펙에 따라 @Resource, @Autowired, @Inject 등 3가지로 분류한다.
✓ Spring은 기본적으로 byType 방식으로 객체 타입에 맞는 Bean을 검색하여 의존성을 주입한다.

Injection 어노테이션	설명
@Resource	JSR-250 표준 Annotation, Spring 2.5 부터 지원 JSR-250은 JNDI를 이용한 datasource 등의 인젝션을 위한 사양임 필드, Setter에 사용 가능
@Autowired	Spring 2.5부터 지원 Spring에서만 사용가능, required 속성을 통해 DI 여부 조정 필드, Setter, Constructor에 사용 가능
@Inject	JSR-330 표준, Spring 3.0부터 사용 가능 프레임워크에 종속적이지 않음, javax.inject-x.x.x.jar 필요 필드, Setter, Constructor에 사용 가능

2.4 Spring Annotation 빈(bean)관리 (5/8) – DI (1/2)

- ✓ 클래스의 참조할 객체에 @Autowired와 같은 특정 어노테이션을 추가한다.
- ✓ 어노테이션으로 객체 사이의 의존관계를 설정하기 위해선 <context:annotation-config> 설정이 필요하다.
- ✓ Annotation-config는 관계설정, 후처리 등의 기능이 있는 컨테이너 인프라 빈을 등록한다.
- ✓ Annotation-config는 빈을 등록하는 기능이 없으므로, 빈 선언은 추가한다.

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
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-4.0.xsd">
    <context:annotation-config />
    <bean id="cookbook" class="com.namoo.yorizori.dao.jdbc.JdbcCookbookDao" />
    <bean id="cookbookService" class="com.namoo.yorizori.service.impl.CookbookServiceImpl" />
</beans>
```

applicationContext.xml

코드설명

[Line8] @Autowired 를 처리하기 위한 빈을 등록한다.
[Line9-10] xml 방식으로 빈을 선언한다.

```java
@Component
public class CookbookServiceImpl implements CookbookService {

    @Autowired
    private CookbookDao cookbookDao;

    @Override
    public List<Cookbook> findAllCookbooks() {
        //
        return cookbookDao.retrieveAll();
    }
```

CookbookServiceImpl .java

코드설명

[Line1] applicationContext.xml 파일에 stereotype annotation을 처리 할 수 있는 빈을 등록하지 않았으므로 무시된다.
[Line4] 프로퍼티에 의존성 주입을 위해 Injection Annotation을 선언한다.

2.4 Spring Annotation 빈(bean)관리 (6/8) – DI (2/2)

✓ <context:component-scan> 은 <context:annotation-config>의 기능을 포함한다.

✓ 따라서, component-scan 선언 시 annotation-config 설정은 생략 가능하다.

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
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-4.0.xsd">
     <!-- Service Component Scan -->
     <context:component-scan base-package="com.namoo.yorizori.service" />
     <!-- DAO Component Scan -->
     <context:component-scan base-package="com.namoo.yorizori.dao" />
</beans>
```

applicationContext.xml

코드설명

[Line9, 11] @Component 와 @Autowired 를 처리하기 위한 빈을 등록한다.
base-package로 빈 스캔 및 DI 하려고 하는 빈들을 선택한다.

```java
@Component
public class CookbookServiceImpl implements CookbookService {

    @Autowired
    private CookbookDao cookbookDao;

    @Override
    public List<Cookbook> findAllCookbooks() {
        //
        return cookbookDao.retrieveAll();
    }
```

CookbookServiceImpl .java

코드설명

[Line1] applicationContext.xml 파일에 stereotype annotation을 처리 할 수 있는 빈을 등록하였으므로, 빈으로 등록된다.
[Line4] 프로퍼티에 의존성 주입을 위해 Injection Annotation을 선언한다.

2.4 Spring Annotation 빈(bean)관리 (7/8) – @Resource

- ✓ @Resource 를 이용하여 필드에 주입하거나, setter 에 주입한다.
- ✓ @Autowired 와 동일한 기능을 처리하지만, JNDI를 이용한 datasource 등의 Injection을 위한 JSR-250 표준이다.
- ✓ 동일한 타입의 빈이 여러 개일 경우 name을 통해서 빈을 구분한다.
- ✓ 이는 byName 방식으로 선언된 Bean의 이름(ID)를 검색하여 의존성을 주입한다.

필드 주입

```java
@Repository
public class JdbcCookbookDao implements CookbookDao {
    @Resource(name="dataSource2")
    private DataSource dataSource;

    @Override
    public List<Cookbook> retrieveAll() {
    ......
}
```

JdbcCookbookDao.java

Setter 주입

```java
@Repository
public class JdbcCookbookDao implements CookbookDao {
    private DataSource dataSource;

    @Resource(name = "dataSource2")
    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }
        ......
}
```

JdbcCookbookDao.java

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
 ..... >

<!-- DAO Component Scan -->
<context:component-scan base-package="com.namoo.yorizori.dao" />

<!-- Embedded Database -->
<jdbc:embedded-database id="dataSource1" type="H2">
    <jdbc:script location="classpath:schema.ddl" />
    <jdbc:script location="classpath:init-data.sql" />
</jdbc:embedded-database>

<!-- Embedded Database -->
<jdbc:embedded-database id="dataSource2" type="HSQL">
    <jdbc:script location="classpath:schema.ddl" />
    <jdbc:script location="classpath:init-data.sql" />
</jdbc:embedded-database>

</beans>
```

applicationContext.xml

DataSource 형의 빈이 dataSource1과 dataSource2 두 개 선언된 경우, 명시적으로 주입할 빈의 이름을 지정해야 한다. 만약, DataSource 의 빈이 dataSource1인 빈 하나만 존재하는 경우, Spring은 이름이 다르더라도 동일한 형을 찾아 dataSource에 의존성을 주입한다.

2.4 Spring Annotation 빈(bean)관리 (8/8) – @Autowired

✓ @Autowired 를 이용하여 생성자 또는 필드, 일반 메소드에 의존성을 주입할 수 있다.
✓ 동일한 타입의 bean이 여러 개일 경우에는 @Qualifer("이름")으로 빈을 식별한다.
✓ 생성자가 여러 개일 경우에는 하나의 생성자에만 적용 가능하다.

생성자 주입

```java
@Component
public class CookbookServiceImpl implements CookbookService {

    private CookbookDao cookbookDao;

    @Autowired
    public CookbookServiceImpl(
              @Qualifier("cookbook") CookbookDao cookbookDao,
                                     RecipeDao recipeDao) {
        this.cookbookDao = cookbookDao;
        this.recipeDao = recipeDao;
    }
    ......
```

CookbookServiceImpl.java

필드 주입

```java
@Component
public class CookbookServiceImpl implements CookbookService {

    @Autowired
    @Qualifier("cookbook")
    private CookbookDao cookbookDao;

    @Override
    public List<Cookbook> findAllCookbooks() {
    ......
```

CookbookServiceImpl.java

2.5 Spring 빈 Scope (1/4)

- ✓ Spring 빈의 생성범위에는 singleton, prototype, request, session 이 있다.
- ✓ Spring 빈은 기본적으로 모든 빈을 Singleton Bean 으로 생성한다 (싱글톤 컨테이너).
- ✓ 따라서, Spring 컨테이너가 제공하는 빈의 인스턴스는 항상 동일하다.
- ✓ Spring 컨테이너가 항상 새로운 인스턴스를 반환하게 만들고 싶다면 scope을 prototype 으로 설정해야 한다.

Scope	설명
singleton	Spring 컨테이너 당 하나의 인스턴스 만 생성 (디폴트)
prototype	컨테이너에 빈을 요청할 때 마다 새로운 인스턴스 생성
request	HTTP 요청 별로 새로운 인스턴스를 생성
session	HTTP 세션 별로 새로운 인스턴스를 생성

XML 방식으로 선언한 빈의 Scope 설정

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
 ......>

    <bean id="recipeDao"
          class="com.namoo.yorizori.dao.jdbc.JdbcRecipeDao"
          scope="prototype" />
```

JdbcRecipeDao.xml

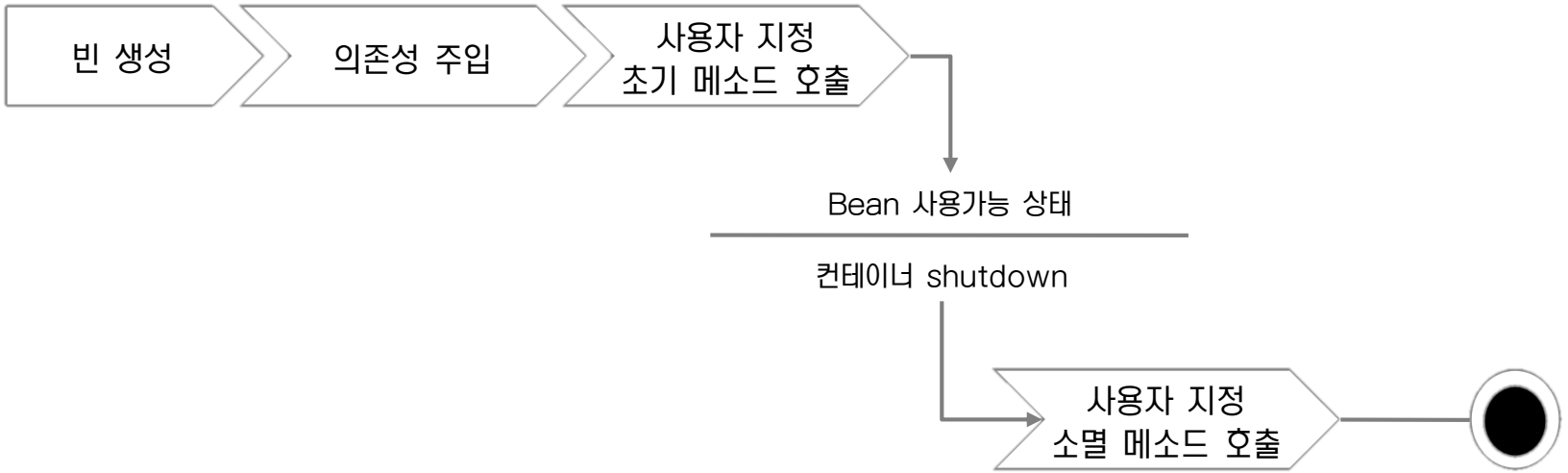
Annotation 방식으로 선언한 빈의 Scope 설정

```java
@Repository
@Scope("prototype")
public class JdbcRecipeDao implements RecipeDao {

    @Resource
    private DataSource dataSource;
     //...
}
```

JdbcRecipeDao.java

2.5 Spring 빈 Scope (2/4) – Spring 빈 생명주기

✓ Spring 컨테이너는 인스턴스의 생성과 삭제 시점에 호출되는 메소드의 설정이 가능하다.
✓ 빈의 초기화 메소드는 DI를 통해 모든 프로퍼티가 세팅 된 후에만 가능한 초기화를 지원해 준다.
✓ 빈 소멸 메소드는 컨테이너가 종료될 때 호출되어 종료 전에 처리해야 하는 작업을 수행한다.



2.5 Spring 빈 Scope (3/4) – 빈의 초기화메소드

✓ XML 방식으로 빈을 설정했다면, 빈의 init-method 속성에 초기화 메소드를 선언한다.

✓ 어노테이션 방식으로 선언한 경우, 빈 클래스의 초기화 메소드에 @PostConstruct를 추가한다.

✓ @Bean(init-method)와 Spring의 초기화 메소드 콜백을 이용할 수 있다.

@PostConstruct로 사용자 초기화 메소드 설정

```java
@Component
public class CookbookServiceImpl implements CookbookService {

    @Autowired
    private CookbookDao cookbookDao;

    @PostConstruct
    public void initCookbookService() {
        //
        System.out.println("@PostConstruct 빈 초기화시  메소드  - Start -");
        List<Cookbook> books = cookbookDao.retrieveAll();
        for (Cookbook cookbook : books) {
            System.out.println("@PostConstruct 요리책 목록 ::" + cookbook.getName());
        }
        System.out.println("@PostConstruct 빈 초기화시  메소드 - End -");
    }
    ......
```

CookbookServiceImpl .java

init-method 로 사용자 초기화 메소드 설정

```xml
<beans xmlns="http://www.springframework.org/schema/beans" .... >

    <bean id="cookbookService"
    class="com.namoo.yorizori.service.impl.CookbookServiceImpl"
         init-method="initCookbookService" />
......
```

CookbookServiceImpl .java

초기화 메소드 테스트 케이스

```java
public class CookbookService_DI_Test {
    @Test
    public void testInitCookbookServicee() {
        ApplicationContext ac = new
          ClassPathXmlApplicationContext(
        "applicationContext.xml");
    }
}
```

수행결과

ApplicationContext가 Bean생성 및 DI 설정 후 initCookbookService 메소드를 호출하여, 요리책목록을 console 창에 출력한다.

```
@PostConstruct 빈 초기화시  메소드  - Start -
@PostConstruct 요리책 목록 ::요리조리북
@PostConstruct 빈 초기화시  메소드 - End -
```

2.5 Spring 빈 Scope (4/4) – 빈의 제거 메소드

- ✓ XML 방식으로 빈을 설정하였다면 빈에 destroy-method 속성에 제거 메소드를 선언한다.
- ✓ 어노테이션방식으로 선언한 경우, 빈 클래스의 초기화 메소드에 @PreDestroy 를 추가한다.
- ✓ @Bean(destroy-method)와 Spring의 제거 메소드 콜백을 이용할 수 있다.

@preDestroy로 사용자 제거 메소드 설정

```java
@Component
public class CookbookServiceImpl implements CookbookService {

    @Autowired
    private CookbookDao cookbookDao;

    @PreDestroy
    public void destroyCookbookService() {
        //
        System.out.println("@PreDestroy 메소드  - Start -");
        List<Cookbook> books = cookbookDao.retrieveAll();
        for (Cookbook cookbook : books) {
            System.out.println("@PreDestroy 요리책 목록 ::" + cookbook.getName());
        }
        System.out.println("@PreDestroy 메소드 - End -");
    }.....
```

CookbookServiceImpl .java

destroy-method 로 사용자 제거 메소드 설정

```xml
<beans xmlns="http://www.springframework.org/schema/beans" .... >

    <bean id="cookbookService"
    class="com.namoo.yorizori.service.impl.CookbookServiceImpl“
         destroy-method="destroyCookbookService"/>
......
```

CookbookServiceImpl .xml

End of Document

✓ Q&A

감사합니다...