



제2장 스프링 프레임워크 기초 개념

스프링(Spring) 역사



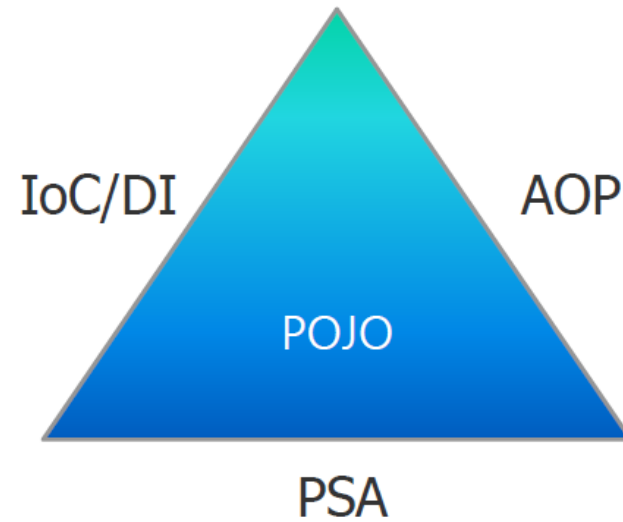
- ▶ 2002년 로드 존슨(Rod Johnson)이 출판한 “Expert One-on-One J2EE Design and Development”에 선보인 코드를 기반으로 시작
- ▶ 2003년 6월 아파치 2.0 라이선스로 공개됨
- ▶ 버전 약력
 - ▶ 1.0: 2004년 3월
 - ▶ 2.0: 2006년 10월
 - ▶ 2.5: 2007년 11월
 - ▶ 3.0: 2009년 12월
 - ▶ 3.1: 2011년 12월
 - ▶ 4.0: 2013년 12월
 - ▶ 5.0: 2017~

스프링(Spring) 이란? (1/4)

- ▶ 오픈 소스 프레임워크
 - ▶ 엔터프라이즈 애플리케이션 개발의 복잡성을 줄여주기 위한 목적
 - ▶ EJB 사용으로 수행되었던 모든 기능을 일반 POJO(Plain Old Java Object) 를 사용해서 가능하게 함
 - ▶ www.springframework.org or <https://spring.io>
- ▶ 간략 정의
 - ▶ 자바 엔터프라이즈 개발을 위한 오픈소스 경량 애플리케이션 프레임워크
 - ▶ Spring은 가벼운 의존 역행(**lightweight dependency injection**)과 관점 지향 컨테이너(**aspect-oriented container**) 및 프레임워크

스프링(Spring) 이란? (2/4)

▶ Spring 의 삼각형



스프링(Spring) 이란? (3/4)

▶ 가벼움 (Lightweight)

- ▶ 20 여개의 모듈로 세분화
- ▶ 불필요하게 무겁지 않음, 불필요한 복잡함의 반대되는 개념
- ▶ Spring이 필요로 하는 처리 능력(성능)은 무시할 정도

▶ 의존성 주입 (Dependency Injection)

- ▶ 의존성 주입 기술을 통해서 느슨한 결합(*loose coupling*) 유지
- ▶ 객체 스스로가 의존 객체를 생성하고 검색하는 대신에 수동적으로 의존관계가 주어짐
- ▶ JNDI (Java Naming & Directory Interface)의 반대 개념
 - ▶ 컨테이너로부터 객체가 의존관계를 검색하는 대신에 컨테이너가 바로 초기화 시에 객체에게 의존 관계를 주입

스프링(Spring) 이란? (4/4)

▶ 관점 지향 (Aspect-oriented)

- ▶ 애플리케이션 비즈니스 로직과 시스템 서비스(감사 및 트랜잭션 관리 등)를 분리하는 개발 기법
- ▶ 애플리케이션 객체는 수행하고자 하는 비즈니스 로직만을 구현하며, 로깅이나 트랜잭션 관리와 같은 시스템 관심사는 전혀 책임이 없음

▶ 컨테이너 (Container)

- ▶ 애플리케이션 객체의 생명주기(lifecycle)와 설정을 담고 관리한다는 측면
- ▶ 애플리케이션 객체가 생성되는 방법, 설정되는 방법, 연관을 맺는 방법을 지정

▶ 프레임워크

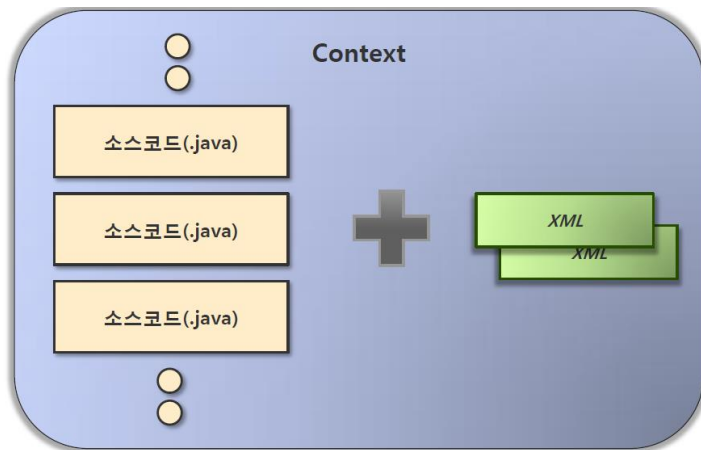
- ▶ 단순한 컴포넌트로부터 복잡한 애플리케이션을 설정하고 구성하는 것이 가능
- ▶ 애플리케이션 객체는 XML 파일 형태로 선언적으로(declaratively) 구성
- ▶ 인프라 기능 (트랜잭션 관리, 저장 프레임워크 통합 등)의 많은 부분을 제공

스프링(Spring) 맛보기 (1/6)

- ▶ 1단계 : 환경 구성
 - ▶ JDK 8 설치
 - ▶ <http://java.sun.com/javase/downloads/index.jsp>
 - ▶ Eclipse 설치
 - ▶ <http://www.eclipse.org/downloads/>
- ▶ 2단계 : 코딩
 - ▶ Eclipse 실행
 - ▶ Java 프로젝트 생성

스프링(Spring) 맛보기 (2/6)

▶ Spring 구성



스프링(Spring) 맛보기 (3/6)

▶ 소스코드 : 인터페이스와 구현 클래스

```
public interface HelloWorldService {  
    void sayHello();  
}
```

```
public class HelloWorldServiceImpl implements HelloWorldService {  
    private String hello;    // sayHello 메소드에서 사용하는 속성  
  
    public HelloWorldServiceImpl() {} // 빈 생성자  
  
    //hello 속성을 파라미터로 가지는 생성자  
    public HelloWorldServiceImpl(String hello) {  
        this.hello = hello;  
    }  
    @Override  
    public void sayHello() { // hello 속성을 출력하는 메소드  
        System.out.println(hello);  
    }  
    public void setHello(String hello) { //hello 속성의 setter  
        this.hello = hello;  
    }  
}
```

스프링(Spring) 맛보기 (4/6)

▶ 소스코드 : Spring 설정 XML파일 네임스페이스

```
<? xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       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-3.0.xsd">
  <bean id="helloService" class="ejkim.spring.HelloWorldServiceImpl">
    <property name="hello" value="안녕하세요." />
  </bean>
</beans>
```

- **beans** : Spring 설정 파일의 루트 요소
- **bean** : Spring 컨테이너에게 어떠한 클래스를 설정해야 하는지를 표시
- **property** : 속성에 값을 세팅할 때 사용(**setter** 호출), **setHello** 오퍼레이션 호출

// 아래의 코드와 동일한 기능

```
HelloWorldService helloWorldService = new HelloWorldServiceImpl();
helloWorldService.setHello("안녕하세요.");
```

스프링(Spring) 맛보기 (5/6)

▶ 소스코드

- ▶ 생성자
- ▶ HelloWorldServiceImpl 클래스의 기본 생성자를 없애는 경우
- ▶ 파라미터가 있는 생성자에 속성을 세팅하는 경우

```
<bean id="helloWorldService" class="ejkim.spring.HelloWorldServiceImpl">  
    <constructor-arg value="안녕하세요." />  
</bean>
```

```
// 아래의 코드와 동일한 기능  
HelloWorldServiceImpl helloWorldService = new HelloWorldServiceImpl("안녕하세요.");
```

스프링(Spring) 맛보기 (6/6)

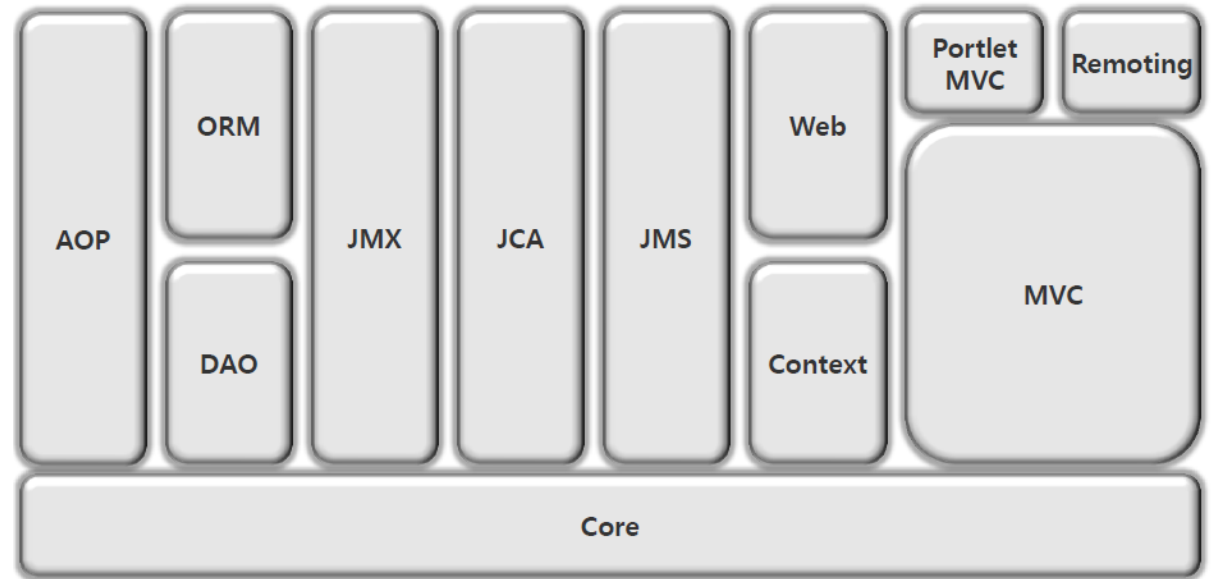
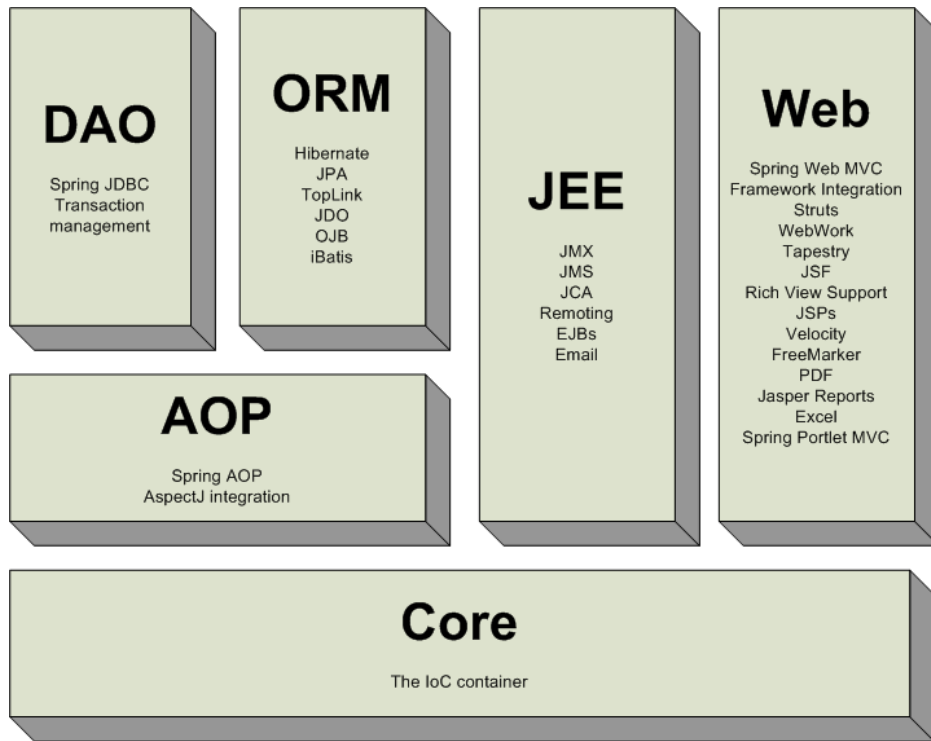
▶ 소스코드 : 클라이언트 코드

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class HelloApp {
    public static void main(String[] args) throws Exception {
        BeanFactory factory = new XmlBeanFactory(
            new FileSystemResource("hello.xml"));
        HelloWorldService helloWorldService = (HelloWorldService) factory.getBean("helloWorldService");
        helloWorldService.sayHello();
    }
}
```

- **BeanFactory** : Spring 컨테이너, **hello.xml** 파일을 컨테이너로 로딩
- **BeanFactory .getBean** : "helloWorldService"에 해당하는 참조를 조회

Spring 모듈(1/6)



* Core 컨테이너 위에 모든 모듈들이 구성
- 컨테이너는 빈(Beans)이 어떻게 생성되고, 설정되고, 관리되는지를 정의

Spring 모듈(2/6)

▶ Core 컨테이너

- ▶ Spring 프레임워크의 기본적인 기능을 제공
- ▶ 기본적인 Spring 컨테이너이며, DI의 기본이 되는 BeanFactory를 포함

▶ Application context 모듈

- ▶ Core 모듈의 BeanFactory는 Spring을 컨테이너로 만들지만, context 모듈은 Spring을 프레임워크로 만듦
- ▶ BeanFactory 개념을 상속, 메시지 지원, 애플리케이션 생명주기 이벤트 추가
- ▶ Email, JNDI 접근, EJB 통합, 리모팅, 스케줄링과 같은 엔터프라이즈 서비스를 제공
- ▶ Velocity와 FreeMarker와 같은 템플릿 기반 프레임워크와 통합을 지원

Spring 모듈(3/6)

▶ AOP 모듈

- ▶ 관점 지향 프로그래밍 (aspect-oriented programming) 을 강력하게 지원
- ▶ Spring 을 사용한 애플리케이션에 대한 관점(aspect)을 개발하는데 기초를 제공
 - ▶ AOP는 애플리케이션 객체에 대한 느슨한 결합을 지원
- ▶ AOP Alliance 인터페이스 (aopalliance.sf.net)를 기반으로 한 aspect 구축과 AspectJ에 대한 지원을 포함

▶ JDBC(Java Database Connectivity) 추상화와 **DAO** (Data Access Object) 모듈

- ▶ DB 접근과 관련된 코드들을 없애고, 단순하고 깔끔하게 코딩할 수 있는 추상화를 제공
 - ▶ DB 자원을 닫는데 실패로 인한 문제점들을 방지
- ▶ 여러 DB 서버가 제공하는 에러 메시지 위에 의미 있는 예외 계층을 마련해 둠
 - ▶ 암호화되고 내부적인 SQL 에러 메시지를 해석하지 않아도 됨
- ▶ Spring 애플리케이션에 있는 객체에 대해서 트랜잭션 관리 서비스를 제공하기 위해 Spring AOP 모듈을 사용

Spring 모듈(4/6)

▶ 객체 관계 매핑 통합(ORM Integration) 모듈

- ▶ 몇 가지 ORM(Object-Relational Mapping) 도구에 대한 DAO(Data Access Object) 구축의 편의성을 제공하여 DAO 구축을 지원
- ▶ ORM 도구를 직접적으로 구현하지는 않았지만, Hibernate, JPA (Java Persistence API), JDO (Java Data Objects), iBATIS SQL Map 등을 포함하는 몇 가지 유명한 ORM 프레임워크로의 훅(hook)을 제공
 - ▶ Spring의 트랜잭션 관리는 JDBC 뿐만 아니라 이들 각각의 ORM 프레임워크에 대해서도 지원
- ▶ Spring의 템플릿 기반 JDBC 추상화의 지원은 ORM과 persistence 프레임워크에 대해서 유사한 추상화를 제공

▶ Java Management Extensions (JMX)

- ▶ 관리를 위한 자바 애플리케이션 내부 동작에 대한 모니터링은 애플리케이션을 제품화하는 중요한 부분
- ▶ 애플리케이션의 빈(bean)을 JMX MBeans으로 쉽게 변환시켜 줌
 - ▶ 실행 애플리케이션에 대한 모니터링과 재설정이 가능

Spring 모듈(5/6)

- ▶ Java EE Connector API (JCA)
 - ▶ JCA는 메임프레임이나 DB를 포함하여 다양한 기업형 정보 시스템과 자바 애플리케이션과의 통합에 대한 표준적인 방법을 제시
 - ▶ JDBC가 DB 접근에 초점이 두는 것을 제외하고는 대부분이 JDBC와 유사
 - ▶ JCA는 레거시 시스템에 대한 연결의 일반적인 API
 - ▶ Spring은 JDBC와 유사하게 세부사항을 추상화 시켜 코드를 템플릿화 함
- ▶ Spring MVC 프레임워크
 - ▶ Spring의 느슨한 결합 기능을 애플리케이션 웹 레이어에 적용
- ▶ Spring Portlet MVC
 - ▶ 포틀릿(Portlet) 기반 애플리케이션은 단일 웹 페이지에 몇 가지 기능의 조각들을 통합함
 - ▶ 즉각적으로 여러 애플리케이션에 하나의 뷰를 제공
 - ▶ 자바의 Portal API를 지원하는 컨트롤러들을 제공하여 Spring MVC 기반에 구축

Spring 모듈(6/6)

▶ 웹 모듈

- ▶ Spring MVC와 Portlet MVC는 Spring 애플리케이션 컨텍스트 로딩시 고려 대상
 - ▶ Spring의 웹 모듈은 Spring MVC와 Spring Portlet MVC에 대한 특별한 지원 클래스들을 제공
- ▶ 그 외에 multipart 파일 업로드와 요청 파라미터와 비즈니스 객체의 바인딩과 같은 웹에 종속된 기능을 지원
- ▶ Apache Struts와 JSF(Java Server Faces)와의 통합 지원

▶ Remoting

- ▶ 자바 객체를 원격 객체화 시키는 기능을 지원
- ▶ 원격으로 객체를 접근 시 로컬에 있는 POJO 처럼 애플리케이션에서 원격 객체로의 접근(wiring)작업을 단순화시킴

▶ Java Message Service (JMS)

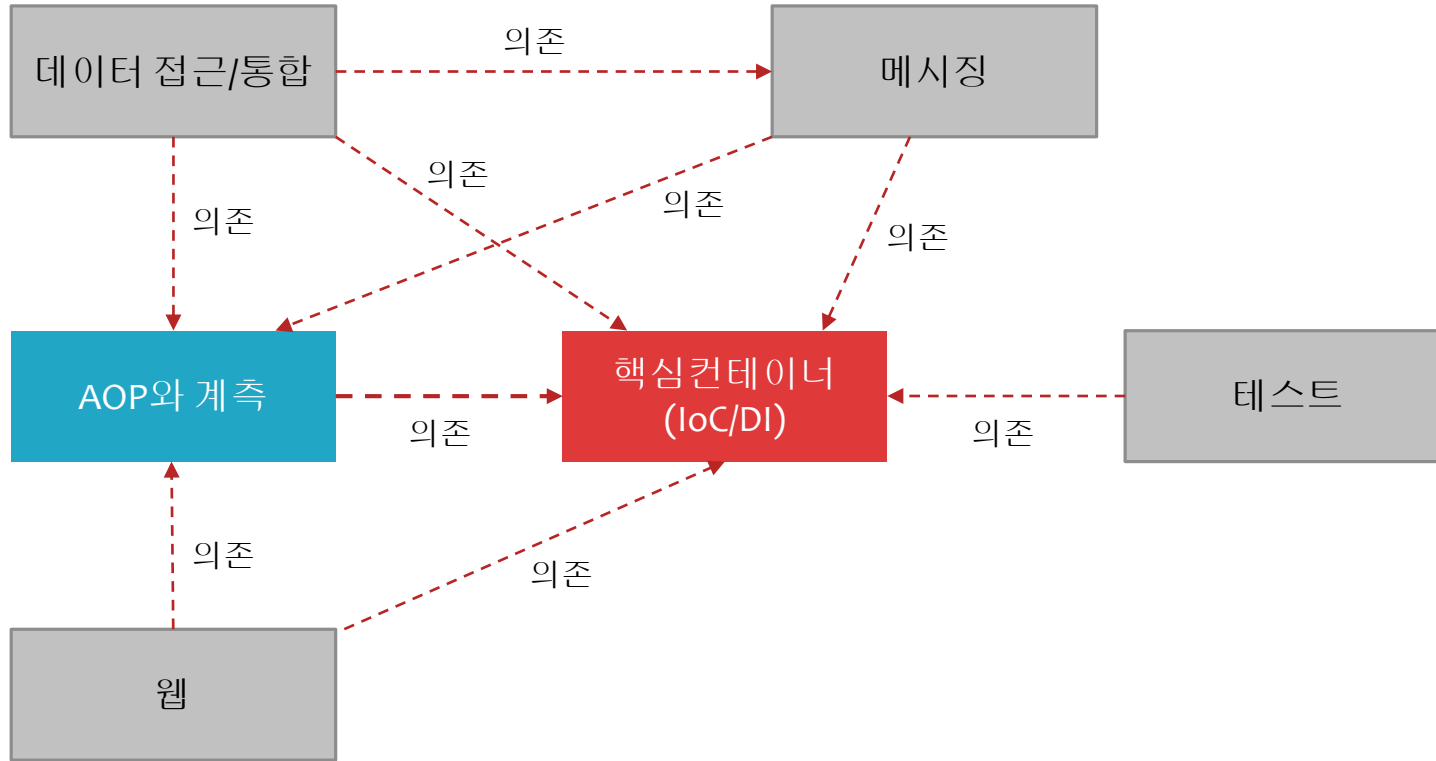
- ▶ JMS 메시지 큐와 토픽에 메시지 전송을 지원
 - ▶ 비동기 메시지를 소비하는(consume) 기능을 가진 메시지 드리븐(Driven) POJO를 생성하는데 지원

스프링 모듈 그룹

- ▶ 스프링이 엔터프라이즈 애플리케이션 개발의 모든 측면을 다룸
- ▶ 스프링 모듈 그룹

모듈 그룹	설명
핵심컨테이너	스프링의 기반을 이루는 모듈 포함. 이 그룹에 spring-core 와 spring-beans 모듈은 스프링 DI기능과 IoC 컨테이너 구현을 제공한다. spring-expression 모듈은 스프링 애플리케이션에서 애플리케이션 객체 설정에 사용하는 SpEL(스프링표현언어)지원을 제공한다.
AOP와 계층	AOP(관점지향프로그래밍)과 클래스 계층을 지원하는 모듈을 포함. spring-aop 모듈은 AOP기능을, spring-instrument 모듈은 클래스 계층 지원을 제공한다.
메시징	메시지 기반 애플리케이션을 쉽게 개발하도록 도와주는 spring-messaging 모듈을 포함
데이터 접근/통합	데이터베이스나 메시징 공급자와의 상호 작용을 쉽게 해주는 모듈을 포함. spring-jdbc 모듈은 JDBC를 사용한 데이터베이스 사용을 단순화 해주고, spring-orm 모듈은 하이버네이트나 JPA와 같은 ORM(객체-관계 매핑)프레임워크 통합을 제공한다. Spring-jms 모듈은 JMS공급자와의 상호작용을 쉽게 만들어준다. spring-tx 모듈은 프로그램을 통해 트랜잭션 관리를 선언적으로 할 수 있다.
웹	spring-web , spring-webmvc , spring-webflux , spring-websocket 모듈이 있다. spring-webmvc 모듈은 서블릿 기반의 웹애플리케이션과 RESTful 웹서비스 개발을 쉽게 해주며, spring-webflux 모듈은 반응형 웹 애플리케이션과 RESTful웹서비스 개발을 쉽게 해준다. spring-websocket 모듈은 웹소켓 프로토콜을 사용하는 웹 애플리케이션 개발을 지원하며 spring-web 모듈은 모든 웹 모듈이 공통으로 사용하는 클래스와 인터페이스를 정의한다.
테스트	단위테스트와 통합 테스트를 도와주는 spring-test 모듈이 있다.

○○○ 스프링 모듈 간의 상호 의존관계 ○○○



스프링 프레임워크 (Spring Framework)

▶ 스프링 프레임워크(Spring Framework)

- ▶ 자바 플랫폼을 위한 오픈소스 애플리케이션 프레임워크
 - ▶ 엔터프라이즈 개발 용이
 - ▶ 복잡하고 실수하기 쉬운 Low Level에 대한 관심을 줄이고, 비즈니스 로직 개발에 전념할 수 있도록 지원
 - ▶ 오픈소스
 - ▶ 오픈소스로 기능을 추가 하거나 수정이 가능(국내 전자정부 프레임워크가 스프링을 기반으로 제작됨)
- ▶ 애플리케이션 프레임워크
 - ▶ 애플리케이션 전 영역을 포괄하는 범용적인 프레임워크를 뜻함
 - ▶ 빠른 개발 과정을 목표
- ▶ 경량급 프레임워크 *경량프레임워크의 종류 : Hibernate, JDO, iBatis, **Spring**
 - ▶ 단순한 웹컨테이너에서도 엔터프라이즈 개발의 고급기술 사용가능
 - ▶ J2EE(Java 2 Enterprise Edition) 엔터프라이즈 환경에서 EJB(Enterprise Java Beans)를 대체할 수 있도록 만들어진 경량급(light-wegit) 프레임워크

*EJB : 분산 비즈니스 애플리케이션을 위한 자바 표준 아키텍처

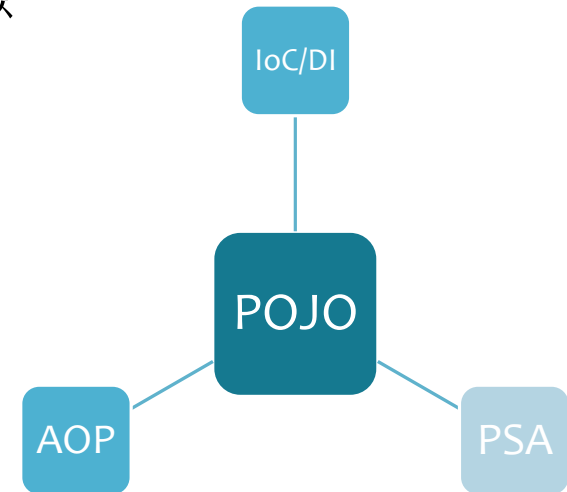
스프링 프레임워크 (Spring Framework)-계속

▶ 주요 특징

- ▶ 가벼움(lightweight)
 - 제공되는 기능은 EJB와 비슷한 수준 내지 이상을 제공하지만 가볍다(코어 파일 : 2.6M)
- ▶ 의존성 주입(Dependency Injection)
 - 의존성 주입 기술을 통해 느슨한 결합(loose coupling)유지
- ▶ 관점 지향(Asspect-Oriented Programming)
 - 애플리케이션의 비즈니스 로직과 시스템 로직(로깅, 트랜잭션)을 완전하게 분리
- ▶ 통합 프레임워크
 - 광범위한 기능의 통합 솔루션을 제공
- ▶ POJO(Plain Old Java)
 - 별도로 상속을 받거나 인터페이스를 구현할 필요없이 코드 작성이 가능하다.

Spring Framework 핵심 요소

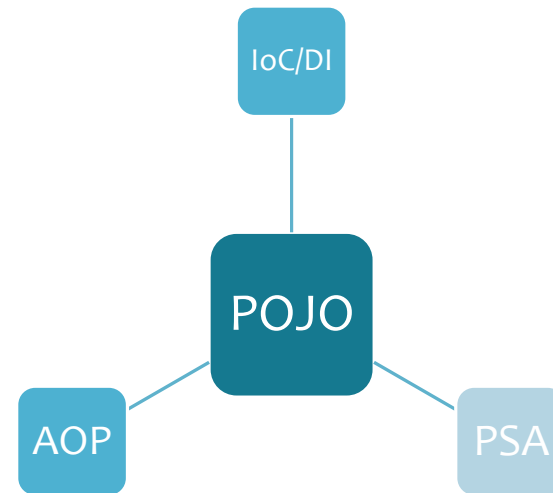
- ▶ AOP – 관점 지향 프로그래밍
- ▶ IoC / DI – IoC: 제어의 역전 , DI: 의존관계주입
- ▶ PSA – Portable Service Abstractions – 쉬운 서비스 추상화
- ▶ POJO(포조) – Plain Old Java Object로 자바 객체를 사용하겠다는 뜻. 특정 규약 및 환경에 종속되면 안됨.



Spring Framework 핵심 요소

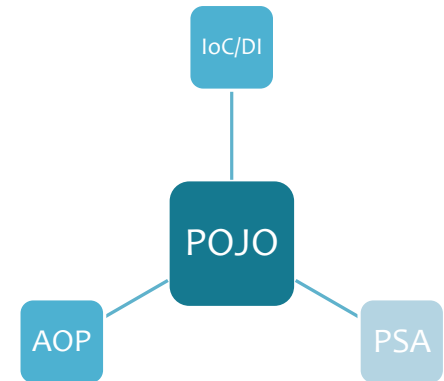
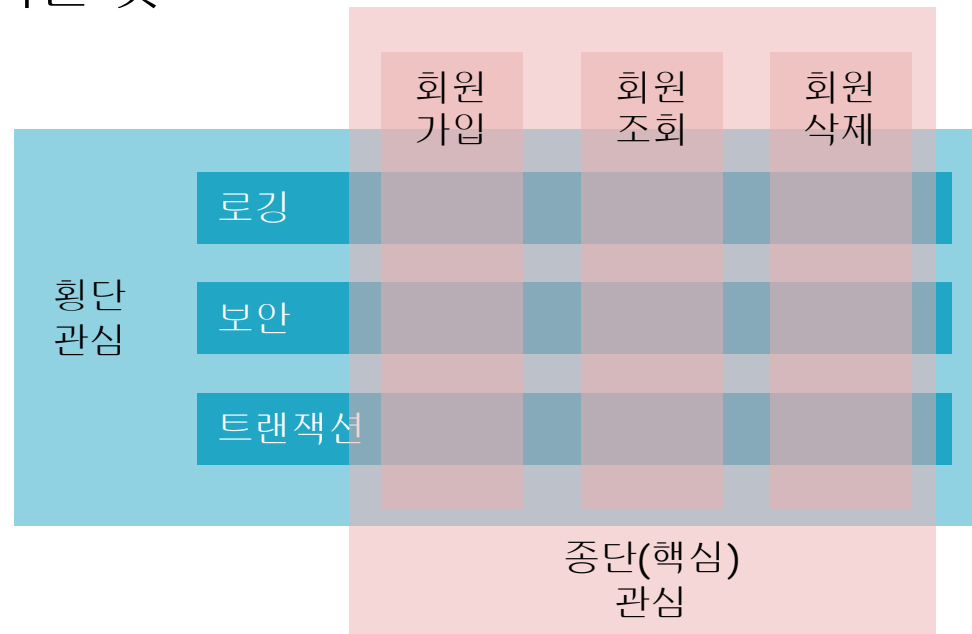
▶ POJO의 장점

- ▶ 특정 프레임워크나 기술에 의존적이지 않은 자바 객체 : 객체지향 설계를 자유롭게 적용
- ▶ EJB 프레임워크에서 정의한 특별한 인터페이스를 포함하지 않음
- ▶ EJB 컨테이너나 데이터베이스 없이 개발 가능 - 개발이 쉽고 빨라짐
- ▶ 특정 구현 기술이 얹매이지 않기 때문에 이식성 향상
- ▶ POJO로 개발된 코드는 자동화된 테스트에 유리



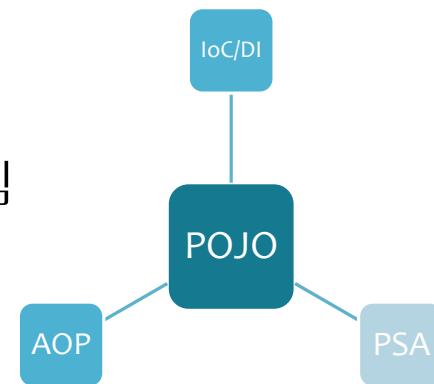
Spring Framework 핵심 요소

- ▶ AOP(Aspect-Oriented Programming)
 - ▶ 기능의 관심사를 종단 관심사와 횡단 관심사(Crosscutting Concerns)로 구분하여 핵심 관심(Core Concerns)에 집중하고, 횡단 관심사는 시스템(=Spring Framework회원)에서 담당하게 하는 것



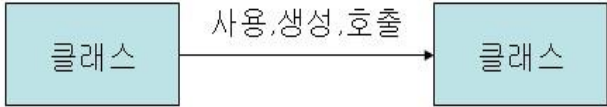
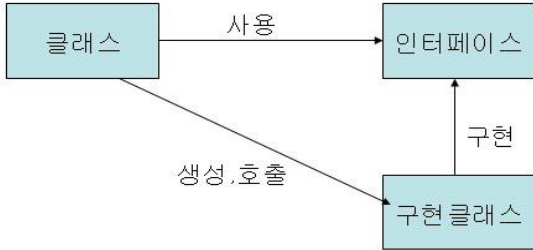
Spring Framework 핵심 요소

- ▶ IoC(Inversion of Control): 제어의 역전
 - ▶ 프로그램의 흐름을 개발자가 아닌 프레임워크가 주도한다는 의미
 - ▶ 스프링 컨테이너가 흐름의 제어권을 가짐으로서 DI, AOP 등이 가능해짐
- ▶ DI(Dependency Injection): 의존성 주입
 - ▶ 객체간의 의존성을 자신의 내부 코드가 아닌 외부의 주입으로 가능하게 하는 것.
외부의 설정파일 등을 통해 정의함
 - ▶ 의존 관계 설정이 컴파일시가 아닌 실행시에 이루어져 **모듈간의 결합도를 낮춤**
 - ▶ **코드 재사용**을 높여서 작성된 모듈을 여러 곳에서 소스코드 수정없이 사용이 가능함



Spring Framework 핵심 요소- IoC

- IoC - 객체 간의 의존관계를 객체 내부에 선언 또는 정의하지 않고, 외부의 조립기를 이용하여 의존 관계를 설명한다는 것.

호출패턴	개념도	설명
클래스 호출방식	 <pre>graph LR; A[클래스] -- "사용, 생성, 호출" --> B[클래스]</pre>	클래스 내에 선언과 구현이 한 몸이기때문에 다양한 형태로 변화가 불가능.
인터페이스 호출방식	 <pre>graph LR; A[클래스] -- "사용" --> B[인터페이스]; C[구현 클래스] -- "구현" --> B; A -- "생성, 호출" --> C</pre>	클래스를 인터페이스와 구현클래스로 모양상의 분리. 구현클래스 교체가 용이하여 다양한 형태로 변화가능. 하지만 구현 클래스 교체 시 호출클래스의 소스에 수정이 가해짐.

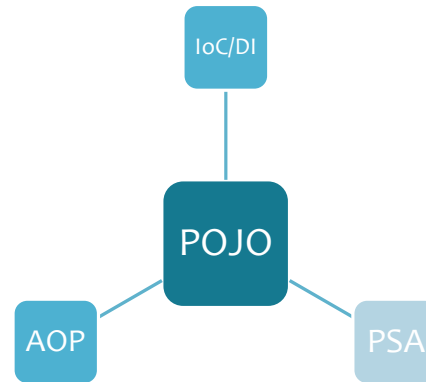
Spring Framework 핵심 요소-IoC

- IoC - 객체 간의 의존관계를 객체 내부에 선언 또는 정의하지 않고, 외부의 조립기를 이용하여 의존 관계를 설명한다는 것.

호출패턴	개념도	설명
팩토리 호출방식	<pre>graph TD; C[클래스] -- 사용 --> I[인터페이스]; C -- 호출 --> F[팩토리]; F -- 생성 --> IC[구현 클래스]; IC -- 구현 --> I;</pre>	팩토리 방식은 팩토리가 구현클래스를 생성하므로 클래스는 팩토리를 호출하는 코드로 충분. 구현클래스 변경시 팩토리만 수정하면 됨. 호출클래스에는 영향을 미치지 않음. 하지만 클래스에 팩토리를 호출하는 소스가 들어가야 함. 그것 자체가 팩토리에 의존함을 의미한다.
IoC	<pre>graph TD; C[클래스] -- 사용 --> I[인터페이스]; C -- 의존성삽입 --> F[조립기]; F -- 생성 --> IC[구현 클래스]; IC -- 구현 --> I;</pre>	팩토리 패턴의 장점을 더하여 어떠한 것에도 의존하지 않는 형태가 됨. 실행시점에 클래스간의 관계가 형성이 된다. 즉 의존성이 삽입된다는 의미로 IoC를 DI(Dependency Injection) 라는 표현으로 사용한다

Spring Framework 핵심 요소

- ▶ PSA(Portable Service Abstraction): (쉬운)서비스 추상화
 - ▶ 환경의 변화에 상관없이 일관된 방식의 기술로 접근 환경을 제공하려는 추상화 구조
 - ▶ 추상화란 하위 시스템의 공통점을 뽑아내서 분리시키는 것
 - ▶ 추상화를 하면 하위 시스템이 어떤 것인지 알지 못하거나 하위 시스템이 변해도 일관된 방법으로 접근이 가능함



주요 용어

주요 용어	설명
Spring-<모듈이름>-<스프링버전>.jar	스프링 배포판 명명규약 예) spring-aop-5.0.1.RELEASE.jar, spring-beans-5.0.1.RELEASE.jar 등
스프링 IoC 컨테이너 or 스프링컨테이너 (IoC : Inversion of Control)	- 스프링 애플리케이션에서 애플리케이션에 존재하는 객체(빈, bean)를 생성하고 의존 관계를 주입하는 일을 담당 - 애플리케이션의 설정 메타 데이터를 읽어서 애플리케이션 객체들과 그들의 의존관계를 인스턴스화 함 - 스프링컨테이너에 설정 메타데이터를 제공하는 방법 : XML, 자바 애너테이션, 자바 코드사용
객체의 의존관계(dependency)	객체가 다른 객체와 상호 작용하는 경우
DI (=IoC) (DI:Dependency Injection)	객체간의 관계를 생성자 인수나 세터 메소드 인수로 명시, 객체를 생성할 때 생성자나 세터를 통해 의존 관계를 주입하는 디자인 패턴 -Setter Injection, Constructor Injection, Method Injection