고용노동부 한국산업인력공단 HUMAN RESOURCES DEVELOPMENT SERVICE OF KOREA

Spring Framework의 이해

- 강사 김현오 -

한국소프트웨어기술진흥협회 KOSTA Korea Software Technology Association


1. 스프링 프레임워크 개요

1.1 스프링 프레임워크 개요

1.2 스프링 시작하기

- 스프링의 등장 배경
- POJO 프레임워크
- 스프링 프레임워크

스프링의 등장 배경 (1/2)

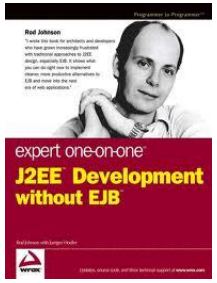
✓ 자바 엔터프라이즈 애플리케이션 개발 표준인 EJB

> *EJB* 를 사용하면 애플리케이션 작성을 쉽게 할 수 있다. 저수준의 트랜잭션이나 상태관리, 멀티 쓰레딩, 리소스 풀링과 같은 복잡한 저수준의 *API* 따위를 이해하지 못하더라도 아무 문제 없이 애플리케이션을 개발할 수 있다.
>
> *- Enterprise JavaBean 1.0 Specification, Chapter 2 Goals*

✓ 그러나, 현실은 너무 어려웠다.

✓ EJB의 대안, 스프링 프레임워크

- Rod Johnson은 자신의 저서에서 EJB를 사용하지 않고 엔터프라이즈 애플리케이션을 개발하는 방법을 소개하였고, 이것이 스프링 프레임워크의 모태가 되었다.



스프링의 등장 배경 (2/2)

Rod Johnson

EJB 개구려서 내가 이번에
*'EJB*안쓰고 개발하기 ' 책 냈으니
흥아들 한 번만 봐주삼~

오*!* 대박~ 이생퀴
잘만들었는데*?*
이거 오픈소스로
함 만들어보까*?*

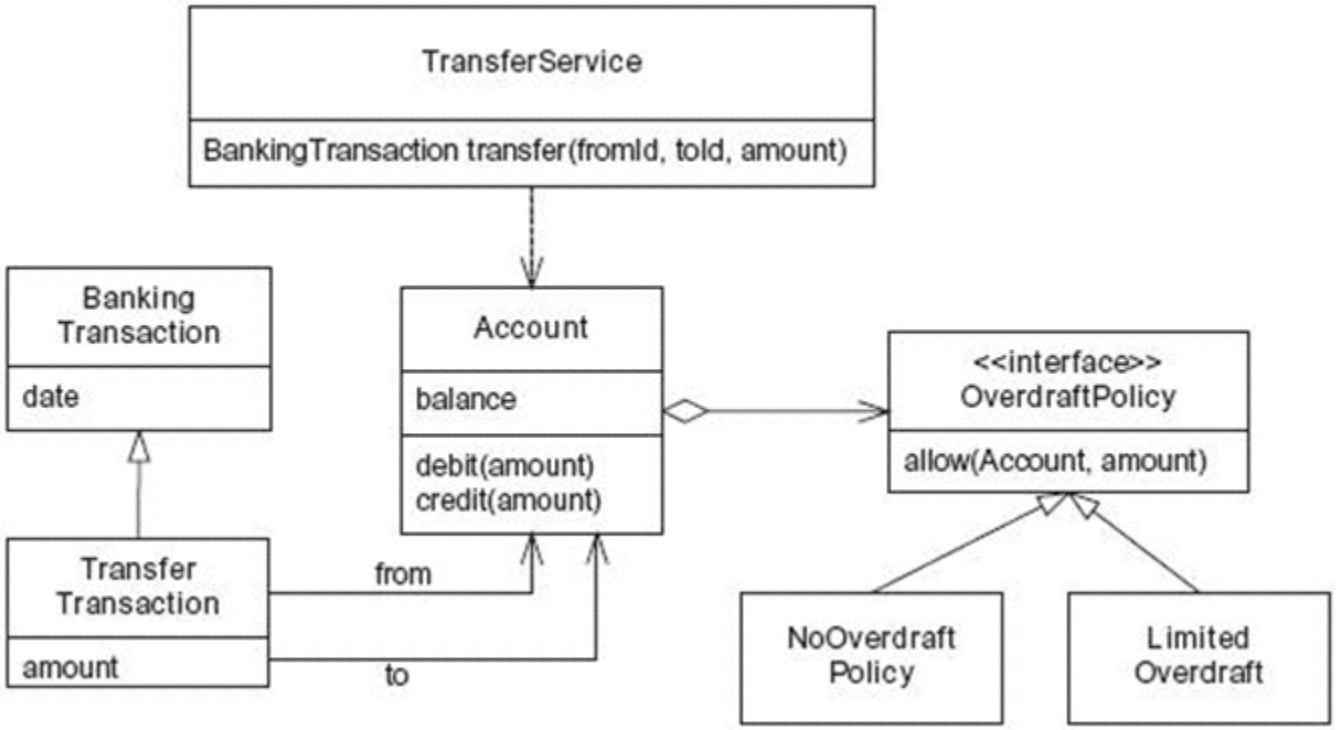
Juergen Hoeller

Yann Caroff

ㅇㅇ~
*EJB*가 개겨울 같았으니까
이름을
봄 '으로 하는게 어때*?*

POJO + Framework

- ✓ 점차 POJO + 경량 프레임워크을 사용하기 시작
- ✓ POJO
  - 특정 프레임워크나 기술에 의존적이지 않은 자바 객체
  - 특정 기술에 얽매이지 않기 때문에 생산성, 이식성 향상
- ✓ 경량 프레임워크
  - EJB가 제공하는 서비스를 지원해 줄 수 있는 프레임워크 등장
  - Hibernate, JDO, iBatis, Spring



스프링 프레임워크 (1/2)

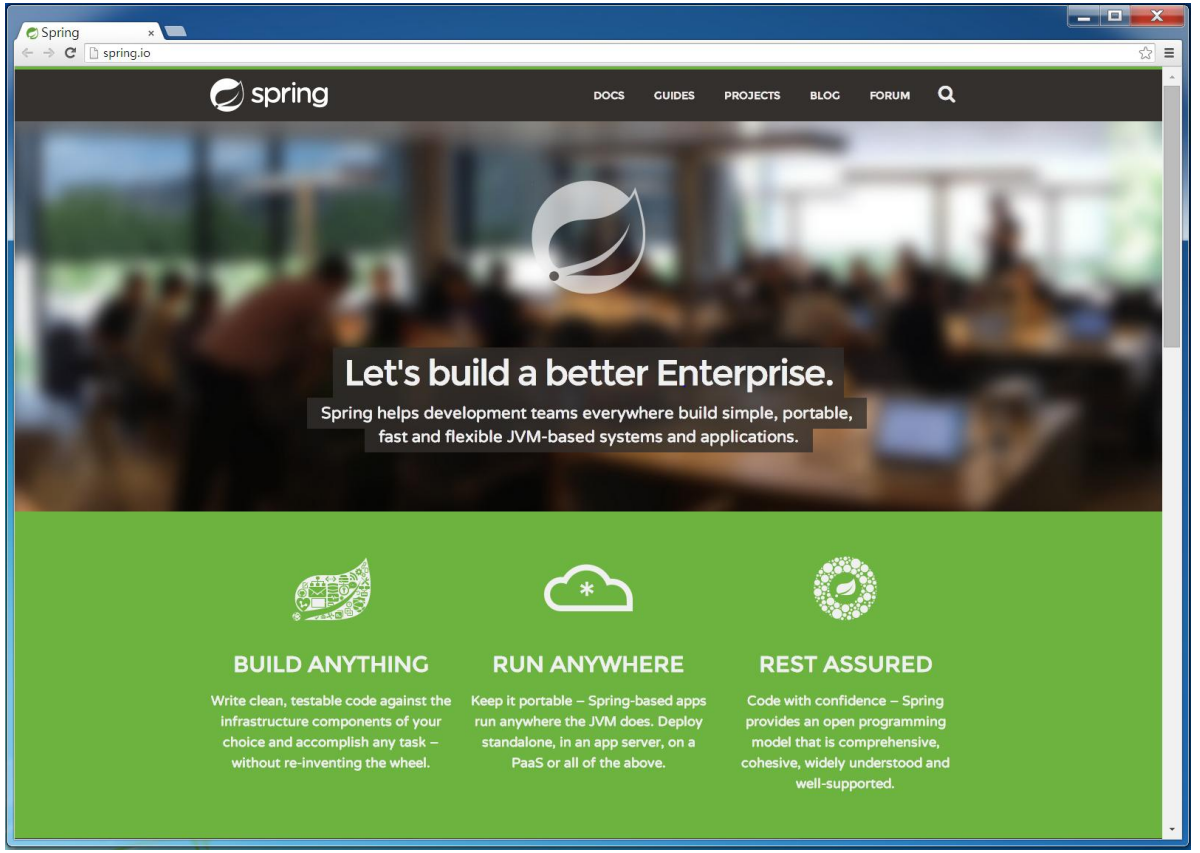
✓ **스프링 프레임워크란?**

- 자바로 엔터프라이즈 애플리케이션을 만들 때 포괄적으로 사용할 수 있는 Programming 및 Configuration 모델을 제공해주는 프레임워크
- 그러니, 주로 하는 일은 애플리케이션 수준에 인프라 스트럭처를 제공하는 일이고…
- 쉽게 말하면, 개발자가 귀찮은 일에 신경 쓰지 않고 비즈니스 로직 개발에만 전념할 수 있게 해준다.

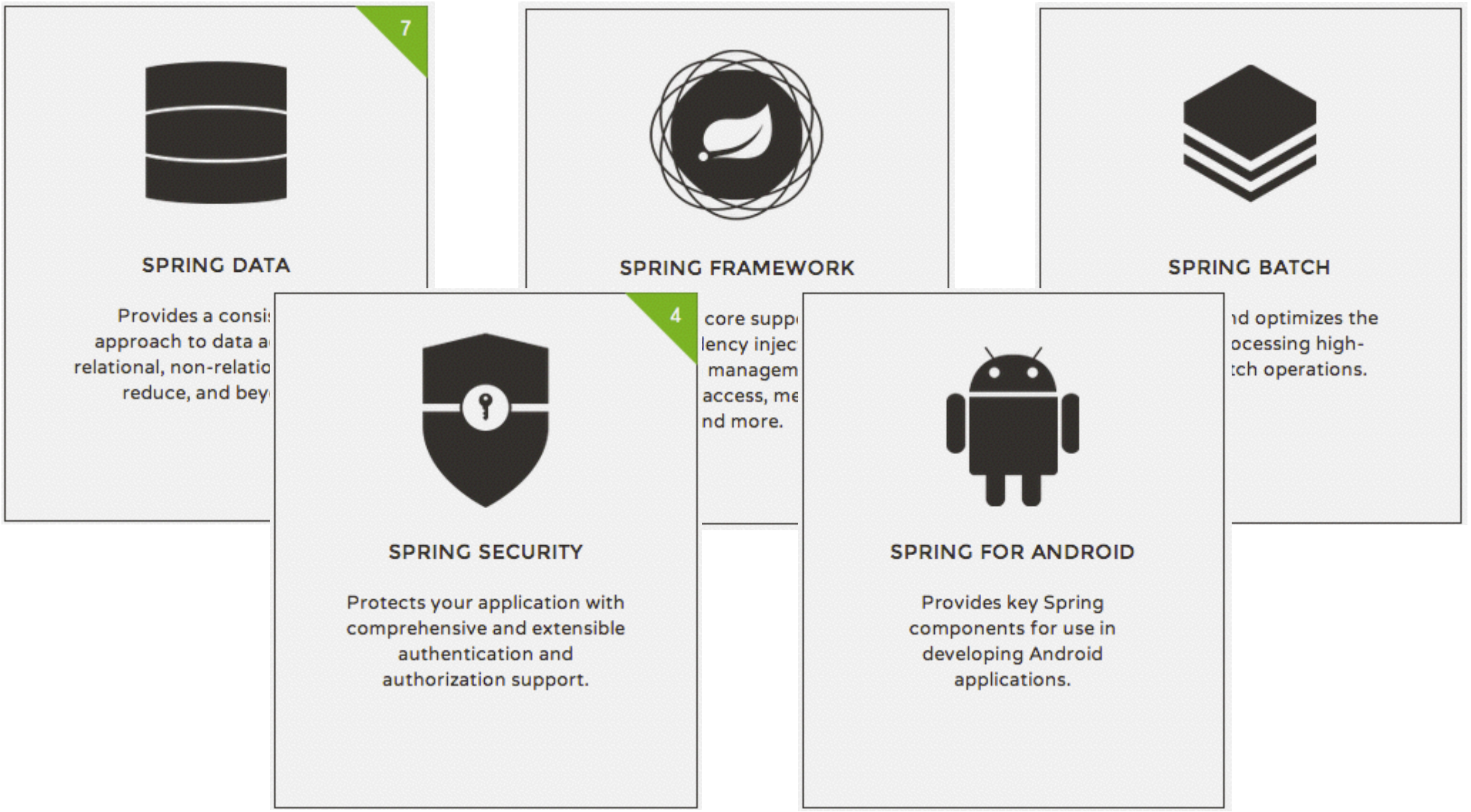
*엔터프라이즈 시스템이란 서버에서 동작하며 기업과 조직의 업무를 처리해주는 시스템을 말한다.

스프링 프레임워크 (2/2)

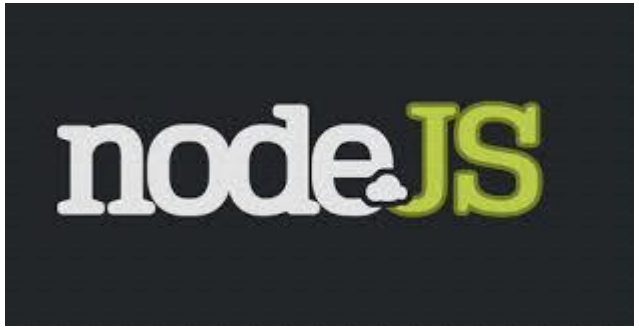
- ✓ 2003년 6월 아파치 2.0 라이선스로 공개
- ✓ Release History
  - 1.0 : 2004년 4월
  - 2.0 : 2006년 10월
  - 2.5 : 2007년 11월
  - 3.0 : 2009년 12월
  - 3.1 : 2011년 12월
  - 3.1.4 : 2013년 1월
  - 4.0.1 : 2014년 1월



스프링 프로젝트


7
SPRING DATA
Provides a consi
approach to data a
relational, non-relatio
reduce, and bey
SPRING FRAMEWORK
core supp
lency injec
managem
access, me
nd more.
SPRING BATCH
ıd optimizes the
ocessing high-
tch operations.
4
SPRING SECURITY
Protects your application with comprehensive and extensible authentication and authorization support.
SPRING FOR ANDROID
Provides key Spring components for use in developing Android applications.

오픈소스 프레임워크


node.JS


MyBatis


iBATIS


ANGULARJS
by Google


spring


avajé


GRAILS


play


LOGBack
The Generic, Reliable
Fast & Flexible
Logging Framework


HIBERNATE


1. 스프링 프레임워크 개요

1.1 스프링 프레임워크 개요

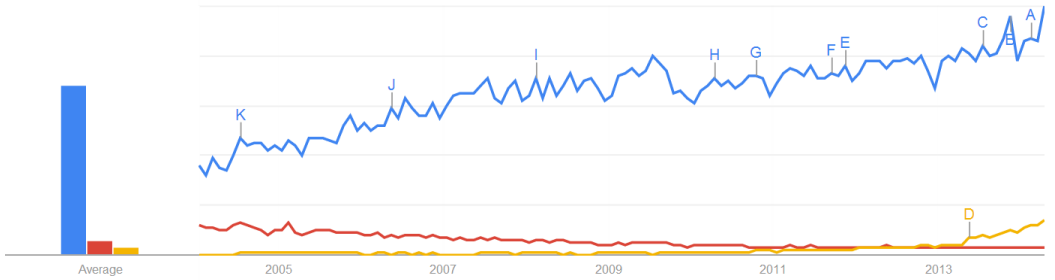
1.2 스프링 시작하기

- 실습환경 준비 (Maven, STS)
- Hello MVC

Maven

- ✓ Maven
  - 소프트웨어 프로젝트를 포괄적으로 관리할 수 있게 해주는 도구
  - Project Object Model 개념에 기반
  - 프로젝트의 빌드, 의존성 관리, 리포팅, 문서화 등의 기능제공
  - 현재 가장 많이 사용하는 소프트웨어 프로젝트 관리 도구

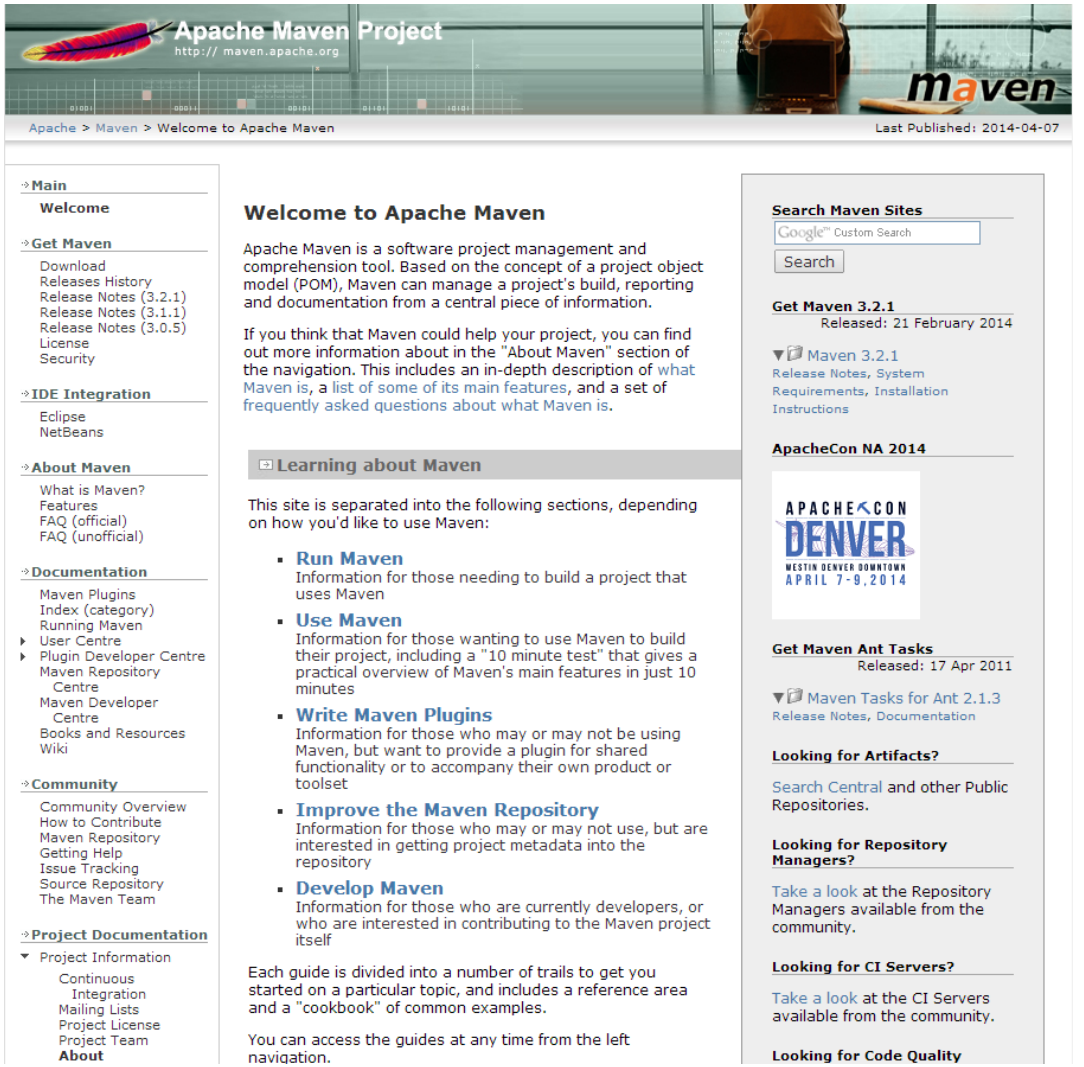
- ✓ Maven by google trends



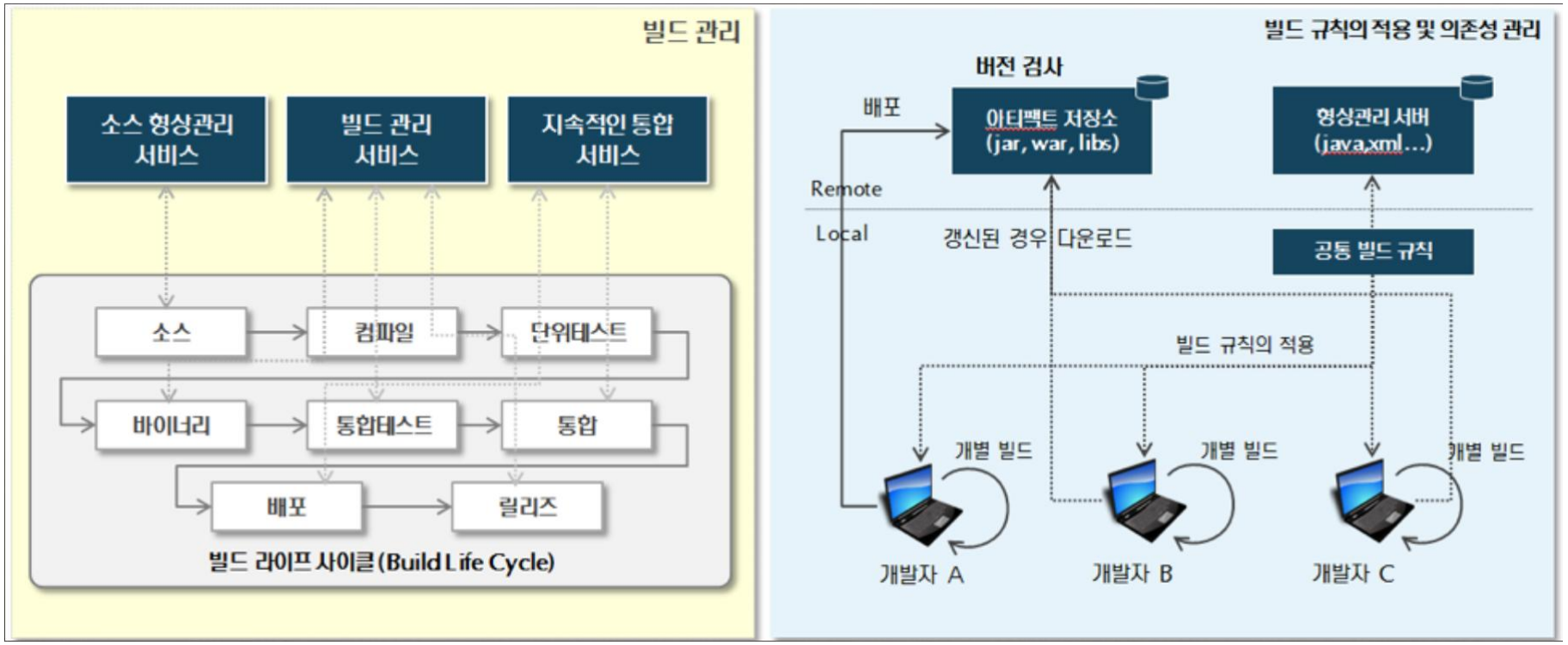
maven
Search term

ant build
Search term

gradle
Search term

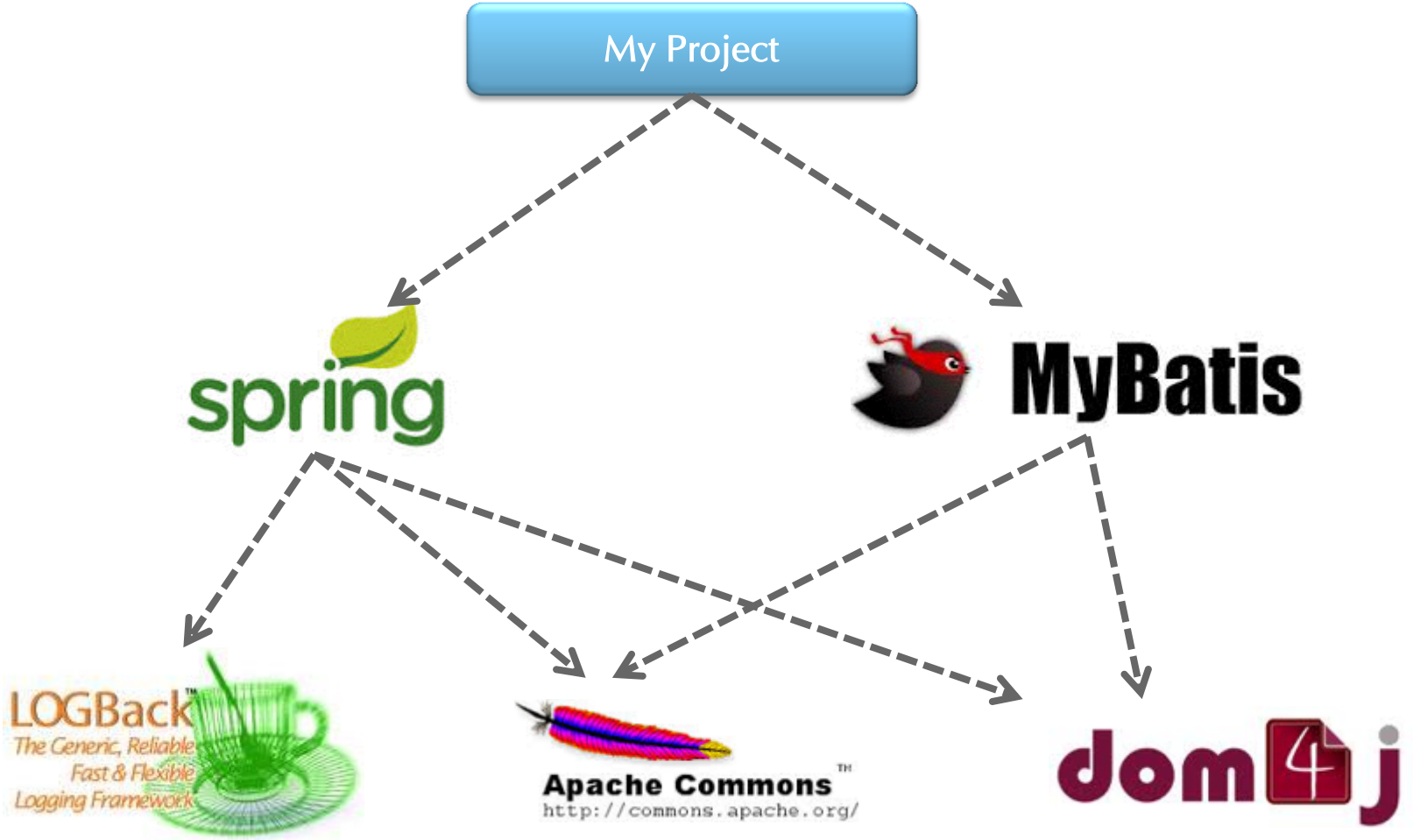


Maven


빌드 관리
소스 형상관리 서비스
빌드 관리 서비스
지속적인 통합 서비스
소스
컴파일
단위테스트
바이너리
통합테스트
통합
배포
릴리즈
빌드 라이프 사이클 (Build Life Cycle)
빌드 규칙의 적용 및 의존성 관리
버전 검사
배포
아티팩트 저장소 (jar, war, libs)
형상관리 서버 (java,xml...)
Remote
Local
갱신된 경우 다운로드
공통 빌드 규칙
빌드 규칙의 적용
개별 빌드
개별 빌드
개별 빌드
개발자 A
개발자 B
개발자 C

Maven 의존관계 관리

✓ Maven의 pom.xml에서 프로젝트간 의존관계를 관리할 수 있다.



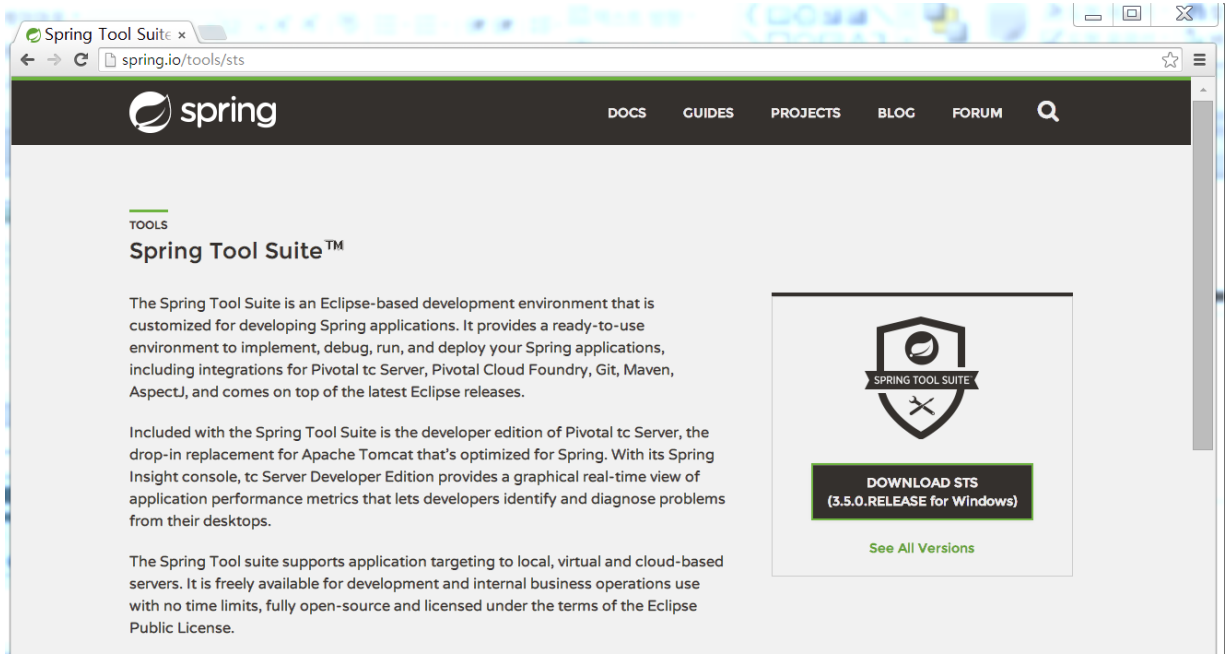
STS

✓ Spring Tool Suite

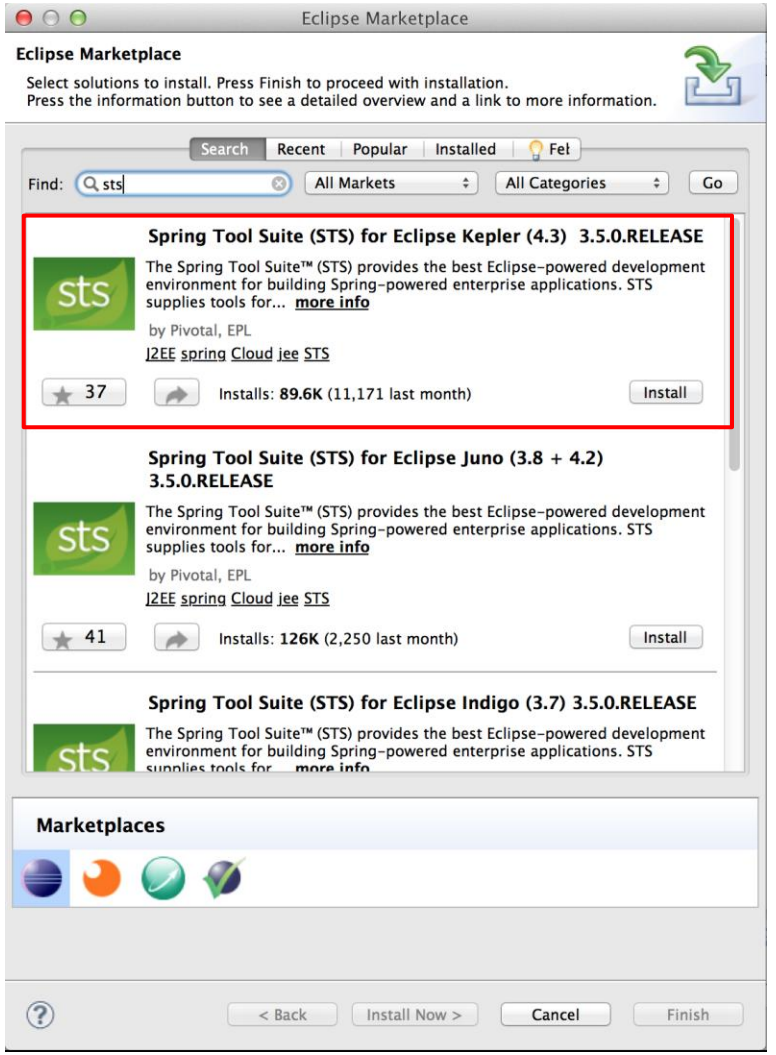
- 이클립스 기반의 스프링용 통합개발환경

✓ STS를 설치하는 두 가지 방법

1. STS IDE 설치하기
2. 기존 이클립스에 설치하기



Spring.io 사이트에서 STS 다운로드



Eclipse Marketplace에서 STS 설치


2. 스프링 프레임워크 이해와 활용

2.1 스프링 프레임워크의 이해

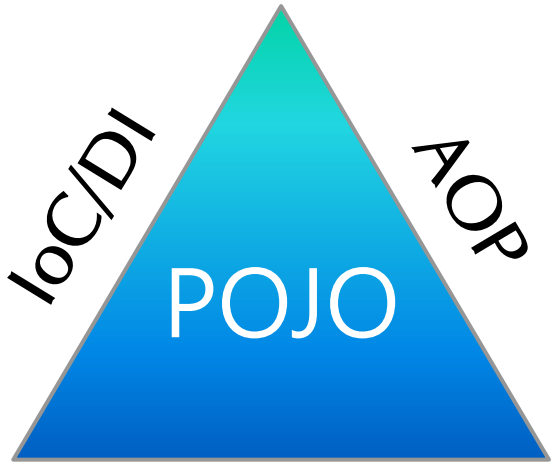
2.2 IoC

- Spring Framework의 구조
- POJO
- 공통 프로그래밍 모델 - IoC/AOP/PSA
- 경량 컨테이너
- SOLID 원칙

Spring Framework 구조

✓ **스프링 삼각형**

- 스프링 애플리케이션은 POJO를 이용해서 만든 애플리케이션 코드와 POJO가 어떻게 관계를 맺고 동작하는지를 정의해놓은 설계정보로 구분된다.



PSA

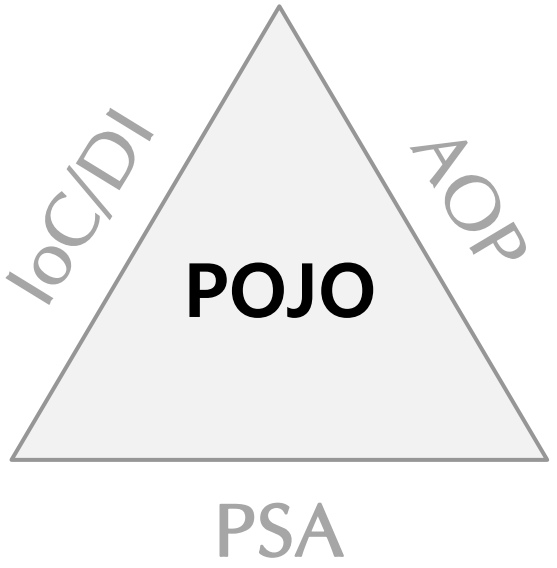
설계정보

Plain Old Java Object

✓ POJO - Plain Old Java Object

- 특별한 기술에 종속되지 않은 순수한 자바 객체
- 테스트하기 용이하며, 객체 지향 설계를 자유롭게 적용할 수 있음
  - Martin Fowler, Rebecca Parson, Josh Mackenzie가 2000년에 만든 용어

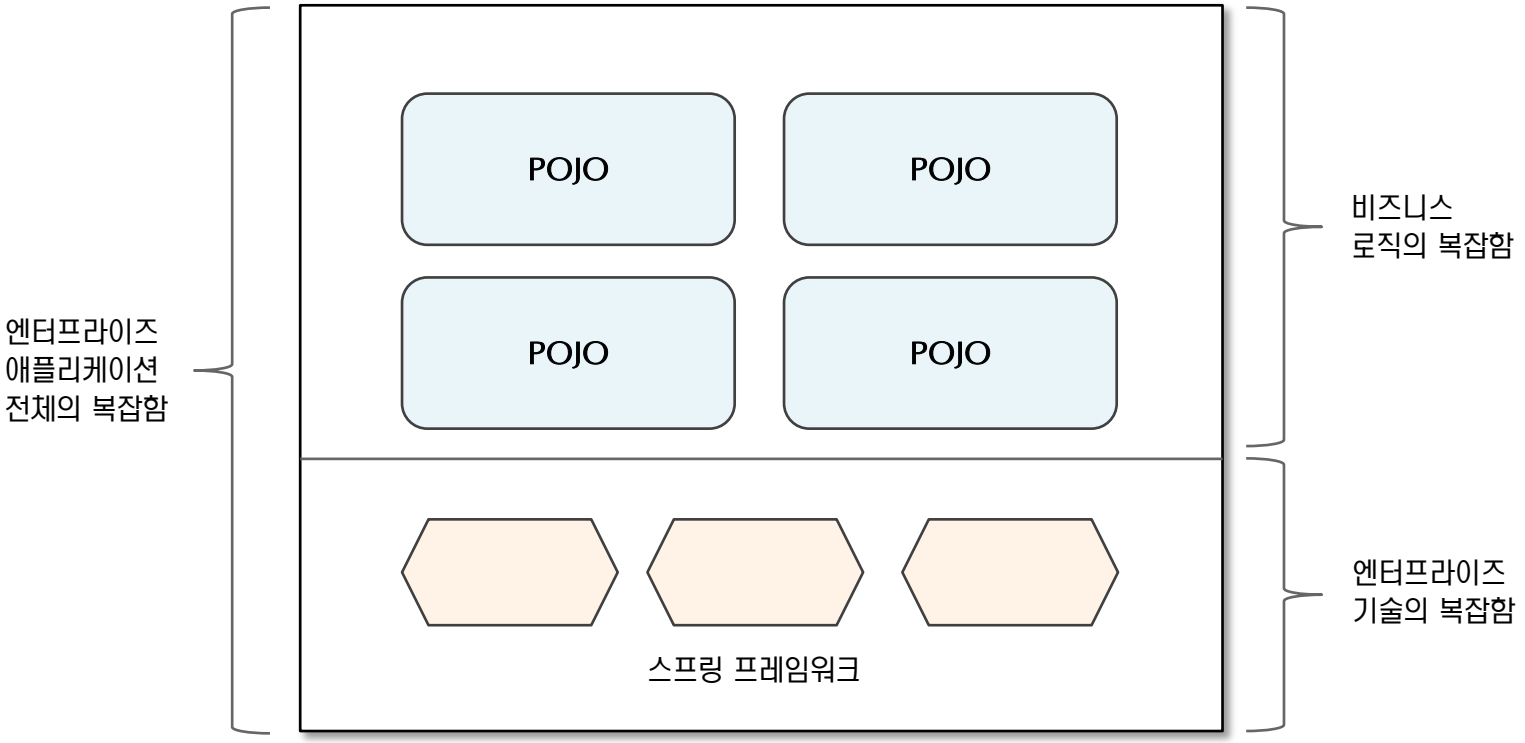
> "We wondered why people were so against using regular objects in their systems and concluded that it was because simple objects lacked a fancy name. So we gave them one, and it's caught on very nicely"
>
> *http://en.wikipedia.org/wiki/POJO*



Plain Old Java Object

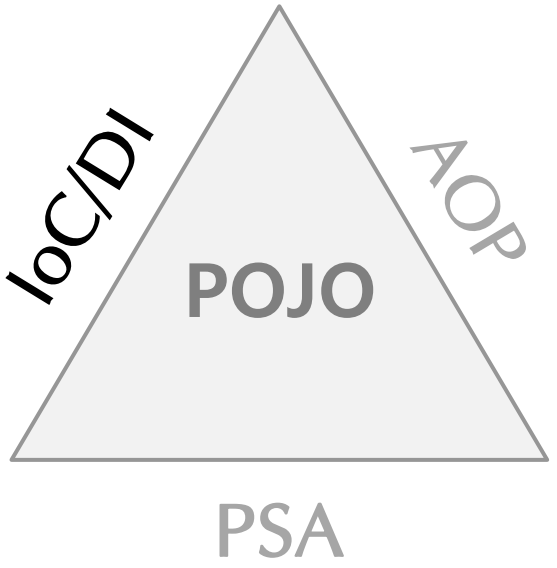
✓ **스프링이 엔터프라이즈 시스템의 복잡함을 다루는 방법**

- 스프링은 비즈니스 로직의 복잡함과 엔터프라이즈 기술의 복잡함을 분리해서 구성할 수 있도록 도와준다.



Inversion Of Control

- ✓ IoC
  - 객체지향 언어에서 Object간의 연결 관계를 런타임에 결정하게 하는 방법
  - 객체 간의 관계가 느슨하게 연결됨(loose coupling)
  - IoC의 구현 방법 중 하나가 DI(Dependency Injection)
  - http://en.wikipedia.org/wiki/Inversion_of_control 참조



Inversion Of Control

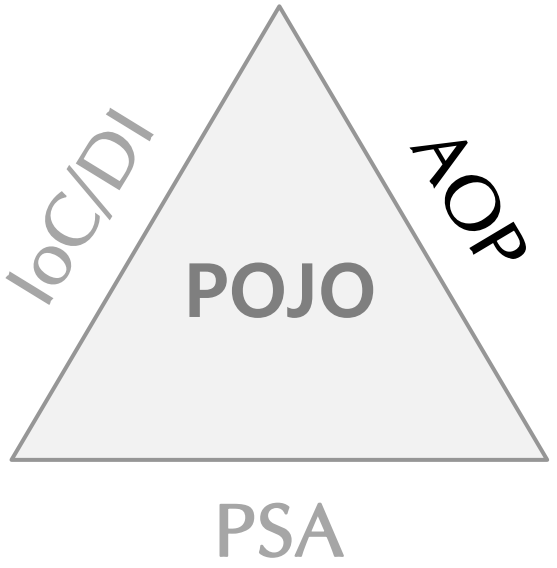
✓ 헐리우드 원칙

- Hollywood Principle - Don’t call us, we will call you!

Aspect Oriented Programming

✓ AOP

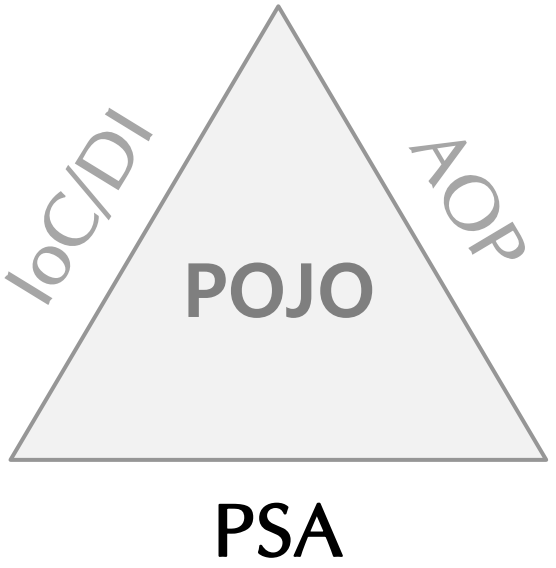
- 관심사의 분리(SoC)를 통해서 소프트웨어의 모듈성을 향상시키고자 하는 프로그래밍 패러다임
- 소스코드 레벨에서 관심사의 모듈화를 지함하는 프로그래밍 방법이나 도구를 포함
- http://en.wikipedia.org/wiki/Aspect-oriented_programming 참조



Portable Service Abstraction

✓ PSA

- 환경과 세부 기술의 변화에 관계없이 일관된 방식으로 기술에 접근할 수 있게 해주는 설계 원칙
- 예를 들어, 데이터베이스에 관계 없이 동일하게 적용할 수 있는 트랜잭션 처리 방식



경량 컨테이너

✓ 가벼움 (Lightweight)

- 20여개의 모듈로 세분화되는 수십만 코드 라인
- 불필요하게 무겁지 않음
- Spring이 필요로 하는 처리 능력(성능)은 무시할 정도

✓ 경량 컨테이너(Lightweight Container)

- 애플리케이션 객체의 생명주기(Lifecycle)와 설정(Configuration)을 담고 관리함
- 애플리케이션 객체를 생성하고, 설정하고, 객체 간의 관계를 지정하는 방법을 제공

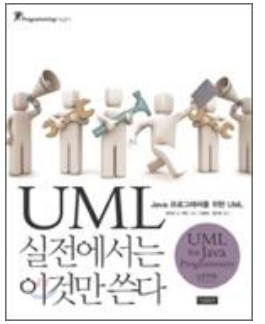
객체지향 설계원칙 (SOLID)

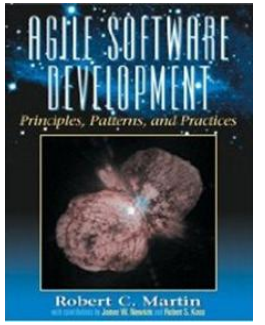
✓ SOLID 원칙

- 객체지향 프로그래밍 및 설계의 5가지 기본원칙
- 이 원칙들을 적용하면, 상대적으로 유지보수하기 쉽고 확장성 있는 시스템을 만들 수 있음
- 리팩토링을 위한 가이드라인으로 사용할 수 있음
- agile, adaptive programming의 주요 전략
- http://www.butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod 참조

Robert C. Martin
(Uncle Bob)

OOP, C++, Java, Patterns,
UML, 컨설턴트
애자일 소프트웨어 개발 선언문
Object Mentor 설립
(www.objectmentor.com)
cleancoders.com





객체지향 설계원칙 (SOLID)

✓ SOLID 원칙

약자	FullName	설명
SRP	The Single Responsibility Principle	한 클래스(모듈)은 하나의 책임을 가져야 함. 한 클래스는 변경할 이유를 하나만 가져야 함
OCP	The Open Closed Principle	클래스를 변경하는 것 없이 클래스의 행위를 확장할 수 있어야 함
LSP	The Liskov Substitution Principle	상속된 클래스는 기반 클래스 타입으로 대체할 수 있어야 함
ISP	The Interface Segregation Principle	클라이언트가 명확하도록 상세한 인터페이스를 만들어라.
DIP	The Dependency Inversion Principle	구체클래스가 아닌 추상화된 클래스에 의존해라.


2. 스프링 프레임워크 이해와 활용

2.1 스프링 프레임워크의 이해

2.2 스프링 IoC

2.3 스프링 빈 설정

- IoC 개념
- 의존관계 검색과 주입
- IoC 컨테이너
- 애플리케이션 컨테이너

IoC 개념 (1/5)

✓ **객체 제어 방식**
- 기존 : 필요한 위치에서 개발자가 필요한 객체 생성 로직 구현
- IoC : 객체 생성을 Container에게 위임하여 처리

✓ **IoC 사용에 따른 장점**
- 객체간의 결합도를 떨어뜨릴 수 있음 (loose coupling)

✓ **객체간 결합도가 높으면?**
- 해당 클래스가 유지보수 될 때 그 클래스와 결합된 다른 클래스도 같이 유지보수 되어야 할 가능성이 높음

IoC 개념 (2/5)

- ✓ Open/Closed Principle
  - OOP에서의 엔티티(클래스, 모듈, 함수…)는 확장에는 열려있고 변경에는 닫혀 있어야 함
- ✓ High cohesion/Loose Coupling
  - 하나의 클래스는 하나의 기능만을 담당해야 함
  - 클래스간의 관계는 책임들이 얽혀 있어서는 안되며 쉽게 다른 관계로 대치될 수 있어야 함
- ✓ Hollywood Principle
  - Don't call us, we will call you!

IoC 개념 (3/5)

✓ 객체간의 강한 결합의 예

- BookService 구현체와 UserService 구현체를 BookShelfService에서 직접 생성하여 사용.
- BookService 또는 UserService가 교체되거나 내부 코드가 변경되면 BookShelfService 까지 수정해야 할 필요성이 있음

```
public class BookShelfServiceImpl implements BookShelfService {

    private BookService bookService = new BookServiceImpl();
    private UserService userService = new UserServiceImpl();

    @Override
    public void addBookShelf() {
        User user = new User();
        bookService.addBook();
        userService.addUser(user);
    }

}
```

BookShelfService에서 BookService의 구현체와 UserService의 구현체를 직접 생성함

IoC 개념 (4/5)

✓ 객체간의 강한 결합을 다형성을 이용해 결합도를 낮춤

- BookService 와 UserService 는 CommonService를 상속함
- BookShelfService에서 각 서비스를 이용시 BookService 와 UserService 모두 CommonService 타입으로 이용 가능

```java
public interface BookService extends CommonService {

    public void addBook();
}
```

```java
public interface UserService extends CommonService {

    public void addUser(User user);
```

```java
public class BookShelfServiceImpl implements BookShelfService {

    private CommonService bookService = new BookServiceImpl();
    private CommonService userService = new UserServiceImpl();

    @Override
    public void addBookShelf() {
        bookService.init();
        userService.init();
    }

}
```

BookService와 UserService 모두 CommonService 형태로 이용 가능함

IoC 개념 (5/5)

✓ **객체간의 강한 결합을 Factory를 이용해 결합도를 낮춤**

- 각 Service를 생성하여 반환하는 Factory 이용
- Service를 이용하는 쪽에서는 인터페이스만 알고 있으면 어떤 구현체가 어떻게 생성되는지에 대해서는 알 필요 없음
- 이 Factory 패턴이 적용된 것이 Container의 기능이며 이 Container의 기능을 제공해주고자 하는 것이 IoC모듈

```java
public class ServiceFactory {

    public static CommonService getBookService() {
        return new BookServiceImpl();
    }

    public static CommonService getUserService() {
        return new UserServiceImpl();
    }
}
```

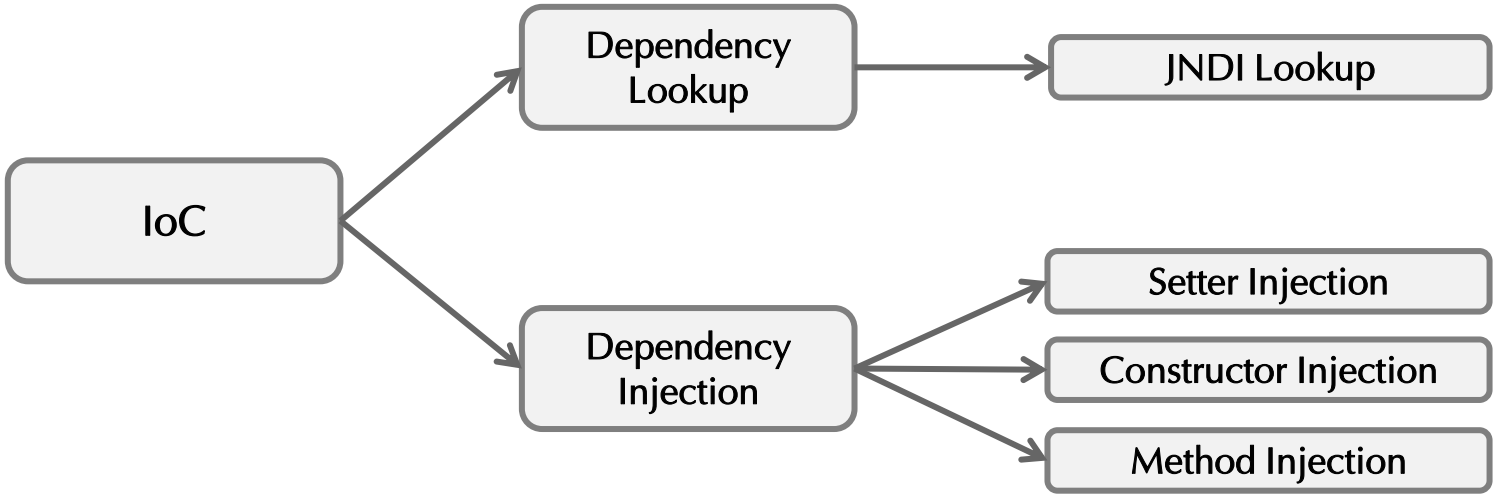
각 서비스를 생성하여 반환하는 책임을 가진 Factory

```java
public class BookShelfServiceImpl implements BookShelfService {

    private CommonService bookService = ServiceFactory.getBookService();
    private CommonService userService = ServiceFactory.getUserService();

    @Override
    public void addBookShelf() {
        bookService.init();
        userService.init();
    }
}
```

의존관계 검색과 주입

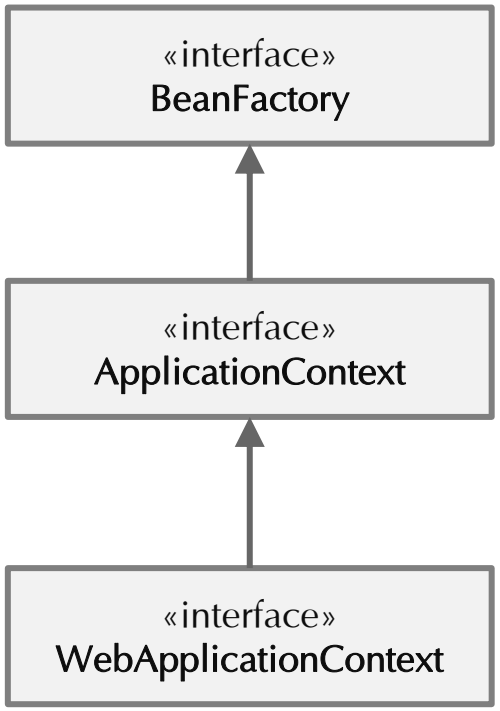
✓ Dependency Lookup
- 컨테이너가 lookup context를 통해서 필요한 리소스나 오브젝트를 얻는 방식
- JNDI 이외의 방법을 사용한다면 JNDI관련 코드를 오브젝트 내에서 일일이 변경해주어야 함
- lookup 한 오브젝트를 필요한 타입으로 Casting 해주어야 함
- Naming Exception을 처리하기 위한 로직 필요

✓ Dependency Injection
- 오브젝트에 lookup 코드를 사용하지 않고 컨테이너가 직접 의존구조를 오브젝트에 설정할 수 있도록 지정해주는 방식
- 오브젝트가 컨테이너의 존재 여부를 알 필요가 없음
- lookup 관련된 코드들이 오브젝트 내에서 사라짐
- Setter Injection
- Constructor Injection



IoC 컨테이너

- ✓ 오브젝트의 생성과 관계설정, 사용, 제거 등의 작업을 애플리케이션 코드 대신 독립된 컨테이너가 담당
- ✓ 컨테이너가 코드 대신 오브젝트에 대한 제어권을 갖고 있어 IoC라고 부름
- ✓ 이런 이유로, 스프링 컨테이너를 IoC 컨테이너라고 부르기도 함
- ✓ 스프링에서 IoC를 담당하는 컨테이너에는 BeanFactory,ApplicationContext가 있음



- 빈(bean) 객체에 대한 생성과 제공을 담당
- 단일 유형의 객체를 생성하는 것이 아니라, 여러 유형의 빈(bean)을 생성, 제공
- 객체 간의 연관 관계를 설정, 클라이언트의 요청 시 빈을 생성
- 빈의 라이프 사이클을 관리

- BeanFactory가 제공하는 모든 기능 제공
- 엔터프라이즈 애플리케이션을 개발하는데 필요한 여러 기능을 추가함
- I18N, 리소스 로딩, 이벤트 발생 및 통지
- 컨테이너 생성 시 모든 빈 정보를 메모리에 로딩함

- 웹 환경에서 사용할 때 필요한 기능이 추가된 애플리케이션 컨텍스트
- 가장 많이 사용하며, 특히 XmlWebApplicationContext를 가장 많이 사용

애플리케이션 컨텍스트

✓ ApplicationContext

- 싱글톤 레지스트리로서의 애플리케이션 컨텍스트
- 애플리케이션 컨텍스트는 우리가 만들었던 오브젝트 팩토리와 비슷한 방식으로 동작하는 IoC 컨테이너이다.
- 스프링은 기본적으로 별다른 설정을 하지 않으면 내부에서 생성하는 빈 오브젝트를 모두 싱글톤으로 만든다.

매번 클라이언트에서 요청이 올 때마다 각 로직을 담당하는 오브젝트를 새로 만들어서 사용한다고 생각해보자. 요청 한번에 5개의 오브젝트가 새로 만들어지고 초당 500개의 요청이 들어오면, 초당 2500개의 새로운 오브젝트가 생성된다. 1분이면 십오만 개, 한 시간이면 9백만 개의 새로운 오브젝트가 만들어진다. 아무리 자바의 오브젝트 생성과 가비지 컬렉션 의 성능이 좋아졌다고 한들 이렇게 부하가 걸리면 서버가 감당하기 힘들다.

그래서 엔터프라이즈 분야에서는 서비스 오브젝트라는 개념을 일찍부터 사용해왔다. 서블릿은 자바 엔터프라즈 기술의 가장 기본이 되는 서비스 오브젝트라고 할 수 있다. 스펙에서 강제하진 않지만, 서블릿은 대부분 멀티스레드 환경에서 싱글톤으로 동작한다. 서블릿 클래스당 하나의 오브젝트만 만들어두고, 사용자의 요청을 담당하는 여러 스레드에서 하나의 오브젝트를 공유해 동시에 사용한다.

스프링 IoC 주요 용어

✓ 빈(bean)
- 스프링이 IoC 방식으로 관리하는 오브젝트를 말한다. 관리 되는 오브젝트라고 부르기도 한다.
- 스프링이 직접 그 생성과 제어를 담당하는 오브젝트만을 빈이라고 부른다.

✓ 빈 팩토리(bean factory)
- 스프링이 IoC를 담당하는 핵심 컨테이너를 가리키다. 빈을 등록하고. 생성하고, 조회하고 돌려주고, 그 외에 부가적인 빈을 관리하는 기능을 담당한다.
- 보통은 이 빈 팩토리를 바로 사용하지 않고 이를 확장한 애플리 케이션 컨텍스트를 이용한다.

✓ 애플리케이션 컨텍스트(application context)
- 빈 팩토리를 확장한 IoC 컨테이너이다. 빈을 등록하고 관리하는 기본적인 기능은 빈 팩토리와 동일하다.
- 스프링이 제공하는 각종 부가 서비스를 추가로 제공한다.
- 빈 팩토리라고 부를 때는 주로 빈의 생성과 제어의 관점에서 이야기하는 것이고, 애플리케이션 컨텍스트라고 할 때는 스프링이 제공하는 애플리케이션 지원 기능을 모두 포함해서 이야기하는 것이라고 보면 된다.

✓ 설정정보/설정 메타정보(configuration metadata)
- 스프링의 설정정보란 애플리케이션 컨텍스트 또는 빈 팩토리가 IoC를 적용하기 위해 사용하는 메타정보를 말한다. 이는 구성정보 내지는 형상정보라는 의미이다.

✓ 스프링 프레임워크
- 스프링 프레임워크는 IoC 컨테이너, 애플리케이션 컨텍스트를 포함해서 스프링이 제공하는 모든 기능을 통틀어 말할 때 주로 사용한다.


2. 스프링 프레임워크 이해와 활용

2.1 스프링 프레임워크의 이해

2.2 스프링 IoC

2.3 스프링 빈 설정

- 스프링 빈 Scope
- 스프링 빈 설정
- 스프링 빈 생명주기

스프링 빈 Scope

✓ 스프링 빈 생성범위

범위	설명
singleton	스프링 컨테이너 당 하나의 인스턴스 빈만 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

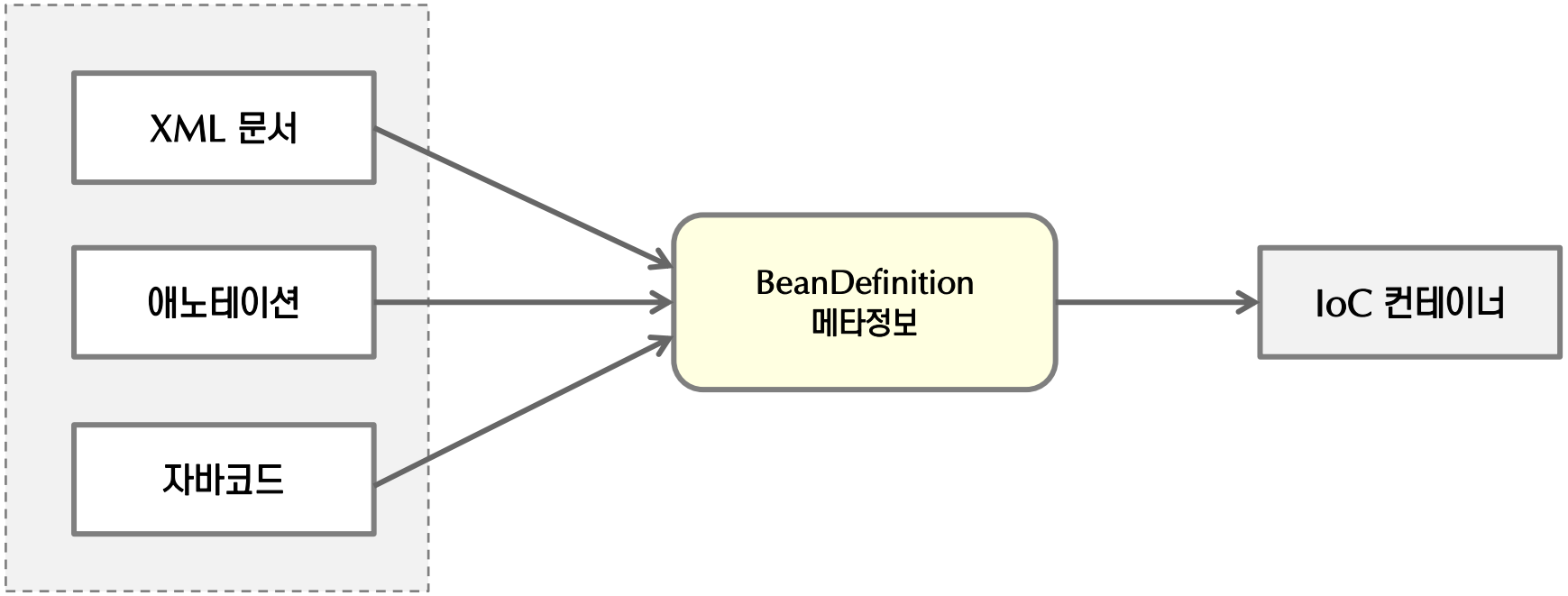
✓ 싱글톤 빈(Singleton Bean)

- 스프링 빈은 기본적으로 싱글톤으로 만들어짐
- 따라서, 컨테이너가 제공하는 모든 빈의 인스턴스는 항상 동일함
- 컨테이너가 항상 새로운 인스턴스를 반환하게 만들고 싶다면 scope을 prototype으로 설정해야 함

```
@Component("beanScopeCookBookDatasource")
@Scope("singleton")
public class BeanScopeBookDatasource {

    private String driverClass;
    private String url;
    private String username;
    private String password;
    …
}
```

스프링 빈 설정

✓ 스프링 빈 설정 메타정보



메타정보의 표현방식

스프링 빈 설정 – XML

✓ XML 방식의 스프링 빈 설정

- XML문서 형태로 빈의 설정 메타 정보를 기술
- 단순하며 사용하기 쉬움, 가장 많이 사용하는 방식
- <bean> 태그를 통해 세밀한 제어가 가능
- XML 방식부터 익히는 것이 쉬움

```
<?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.xsd">

        <bean id="chef" class="kr.or.kosta.spring.cookbook.Chef">
            <property name="recipe" ref="recipe"/>
         </bean>

        <bean id="recipe" class="kr.or.kosta.spring.cookbook.recipe.GimChi"/>
</beans>
```

스프링 빈 설정 – Annotation

✓ 애노테이션 방식의 스프링 빈 설정

- 애플리케이션의 규모가 커지고 빈의 갯수가 많아질 경우 XML 파일을 관리하는 것이 번거로움
- 빈으로 사용될 클래스에 특별한 애노테이션을 부여해 주면 자동으로 빈 등록 가능
- “오브젝트 빈 스캐너”로 “빈 스캐닝”을 통해 자동 등록
- 빈 스캐너는 기본적으로 클래스 이름을 빈의 아이디로 사용한다. 정확히는 클래스 이름의 첫 글자만 소문자로 바군 것을 사용한다.

```
@Component
public class AnnotatedChef {
        @Autowired
        Recipe recipe;

        public void sayIngredients() {
                StringBuffer sb = new StringBuffer();
                String[] ingredients = recipe.ingredients();
                for (String ingredient : ingredients) {
                        sb.append(ingredient);
                        sb.append("\n");
                }
                System.out.println(sb.toString());
        }

…
```

스프링 빈 설정 – Annotation

✓ 애노테이션으로 빈 설정시 꼭 component-scan 을 설정 해주어야 한다.

<annotated-root-context.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/springcontext.
xsd">

    <context:component-scan base-package="kr.or.kosta.*"/>

</beans>
```

스테레오타입 애노테이션 목록

✓ **빈 자동등록에 사용할 수 있는 애노테이션 목록**

- 빈 자동인식을 위한 애노테이션을 이렇게 여러가지로 사용하는데는 계층별로 빈의 특성이나 종류를 나타내려는 목적도 있고, AOP 포인트컷 표현식을 사용하면 특정 애노테이션이 달린 클래스만을 선정할 수가 있다. 이를 이용해 특정 계층의 빈에 부가기능을 부여해줄 수도 있다.

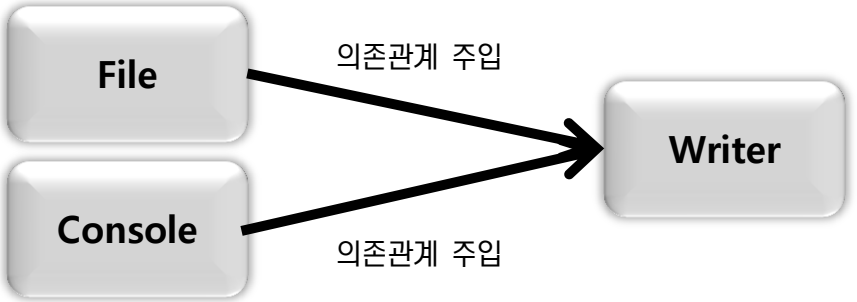
스테레오 타입	적용대상
@Repository	데이터 액세스 계층의 DAO 또는 Repository 클래스에 사용 DataAccessException 자동변환과 같은 AOP의 적용 대상을 선정하기 위해 사용하기도 함
@Service	서비스 계층의 클래스에 사용
@Controller	프레젠테이션 계층의 MVC 컨트롤러에 사용. 스프링 웹 서블릿에 의해 웹 요청을 처리하는 컨트롤러 빈으로 선정
@Component	위의 계층 구분을 적용하기 어려운 일반적인 경우에 사용

의존관계 주입 (Dependency Injection, DI)

✓ 의존 관계 주입 (dependency injection)

- 제어의 역행 (inversion of control, IoC) 이라는 의미로 사용되었음
- Martin Fowler, 2004(http://martinfowler.com/articles/injection.html)
  - 제어의 어떠한 부분이 반전되는가라는 질문에 '의존 관계 주입' 이라는 용어를 사용
  - 복잡한 애플리케이션은 비즈니스 로직을 수행하기 위해서 최소 두 개 이상의 클래스들이 서로 협업
  - 각각의 객체는 협업하고자 하는 객체의 참조를 얻는 것에 책임성이 있음
  - 이 부분은 높은 결합도(highly coupling)와 테스트하기 어려운 코드를 양산함

- DI를 통해 시스템에 있는 각 객체를 조정하는 외부 개체가 객체들에게 생성시에 의존관계를 주어짐
  - 즉, 의존이 객체로 주입됨
  - 객체가 협업하는 객체의 참조를 어떻게 얻어낼 것인가라는 관점에서 책임성의 역행(inversion of responsibility)임
  - 느슨한 결합(loose coupling)이 주요 강점
  - 객체는 인터페이스에 의한 의존관계만을 알고 있으며, 이 의존관계는 구현 클래스에 대한 차이를 모른 채 서로 다른 구현으로 대체 가능



스프링 빈 의존관계 설정 – 애노테이션 사용

스테레오 타입	적용대상
@Resource	JSR-250 표준 Annotation, 스프링 2.5 부터 지원 멤버변수, Setter 메소드에 사용 가능
@Autowired	스프링 2.5부터 지원 스프링에서만 사용가능, required 속성을 통해 DI 여부 조정 멤버변수, Setter, 생성자, 일반 메서드에 사용 가능
@Inject	JSR-330 표준, Spring 3.0부터 사용 가능 프레임워크에 종속적이지 않음, javax.inject-x.x.x.jar 필요 멤버젼수, Setter, 생성자, 일반 메서드에 사용 가능

스프링 빈 의존관계 설정 – 애노테이션 사용

✓ 멤버 변수에 @Resource

```
@Component
public class Chef {

    @Resource(name="gimChi") //동일 한 타입의 빈이 여러 개일 경우 name을 통해서  구분한다.
     Recipe  recipe;


    public void sayIngredients() {
        StringBuffer sb = new StringBuffer();
        String[] ingredients = recipe.ingredients();
        for (String ingredient : ingredients) {
            sb.append(ingredient);
            sb.append("\n");
         }
        System.out.println(sb.toString());
    }

…
```

스프링 빈 의존관계 설정 – 애노테이션 사용

✓ setter에 @Resource

```
@Component
public class Chef {

	Recipe  recipe;


…

	public Recipe getRecipe() {
		return recipe;
	}


	@Resource(name="gimChi")
	public void setRecipe(Recipe  recipe) {
		this.recipe = recipe;
	}
```

스프링 빈 의존관계 설정 – 애노테이션 사용

✓ 생성자에서 @Autowired

```
@Component
public class PrintChef {

        Recipe recipe;
        Printer printer;
        public PrintChef() {}

        @Autowired //생성자가 여러 개 일 경우에는 하나만 사용 가능
        public PrintChef(@Qualifier("bulGoGi") Recipe recipe,
                         @Qualifier("stringPrinter") Printer printer) {
            this.recipe = recipe;
            this.printer = printer;
        }
```

- 동일한 타입의 bean이 여러 개일 경우에는 @Qualifer("이름")으로 식별한다.

스프링 빈 의존관계 설정 – 애노테이션 사용

✓ 맴버변수에서 @Autowired

```
@Component
public class PrintChef {

    @Autowired
    @Qualifier("gimChi")
    Recipe recipe;

    @Autowired
    @Qualifier("stringPrinter")
    Printer printer;

    public PrintChef() {}

    public PrintChef(Recipe recipe, Printer printer) {
        this.recipe = recipe;
        this.printer = printer;
    }
```

- 동일한 타입의 bean이 여러 개일 경우에는 @Qualifer("이름")으로 식별한다.

스프링 빈 의존관계 설정 – 애노테이션 사용

✓ 일반 메서드에서 @Autowired

```
@Component
public class PrintChef {
	Recipe recipe;
	Printer printer;
	public PrintChef(Recipe recipe, Printer printer) {
		this.recipe = recipe;
		this.printer = printer;
	}

	@Autowired
	public void initPrintChef(@Qualifier("bulGoGi") Recipe recipe,
				@Qualifier("stringPrinter") Printer printer) {
		this.recipe = recipe;
		this.printer = printer;
	}
```

- @Qualifer 는 xml 설정 사용시는 불가능 함

스프링 빈 생명주기

토의

- ✓ 질의 응답
- ✓ 토론

감사합니다...

- ❖ 넥스트리소프트(주)
- ❖ 김현오 선임 (hyunohkim@nextree.co.kr)