

# Spring

# Spring framework

# Spring Framework 선수 :

JAVA 기초문법, JSP & Servlet, HTML, CSS, JavaScript, DBMS, etc.

강의진행 : 이론강의(50%)와 실습(50%)

- 프레임워크 : 소프트웨어의 구체적인 부분에 해당하는 설계와 구현을 재사용이 가능하도록 일련의 협업화된 형태로 클래스들을 제공하는 것
- 소스코드
  - <https://github.com/web1p1/studySpring2023>

# 준비

- jdk 1.8(202) & 11
  - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- STS 3
  - <https://spring.io/tools>
  - <https://github.com/spring-attic/toolsuite-distribution/wiki/Spring-Tool-Suite-3>
- Tomcat 9.0
  - <https://tomcat.apache.org/>
- GIT
  - <https://git-scm.com/>

# 준비

- Oracle Express 11g XE
  - <http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html>
- sqldeveloper
  - <https://www.oracle.com/database/sqldeveloper/technologies/download/>
- MySQL 5.7 & Workbench
  - <https://www.mysql.com/downloads/>

# 설치

```
git
<<git-scm.com 다운>>

git init
git config --global user.name "myName"
git config --global user.email "myEmail@gmail.com"


git status
git add --all .
git commit -m "설명"
git branch -M main

<<github 계정 생성>>
github.com


<<github 저장>>
git remote add origin https://github.com/mygitid/myreposit.git
git push -u origin main
```

# 설치

C:\java

\sts-3.9.17.RELEASE (eclipse 4.19)

\apache-tomcat-9

\workSpace

# 1. 스프링이란?

- 프레임워크
- 스프링(SPRING)
- 설치

## 1-2. 스프링(SPRING)

---

### SW 재사용방안

copy & paste → 수정시 모두를 바꿔야

함수 호출 → 함수(function)만 바꾸면 OK

클래스 상속 → 기능의 메서드(함수)와 정보의 필드(변수)를 함께

AOP (Aspect Oriented Programming) → 관심의 분리\*(N.P.)

## 1-2. 스프링(SPRING)

```
#define DEBUG 1  
...  
#ifdef DEBUG  
printf("imformations...")  
#endif
```

## 1-2. 스프링(SPRING)

---

### 디자인패턴

프로그램 개발에서 자주 나타나는 과제를 해결하기 위한 방법 중 하나로, 소프트웨어 개발과정에서 발견된 Know-How를 축적하여 이름을 붙여 이후에 재사용하기 좋은 형태로 특정 규약을 묶어서 정리한 것.

## 1-2. 스프링(SPRING)

---

### 디자인패턴 사용이유

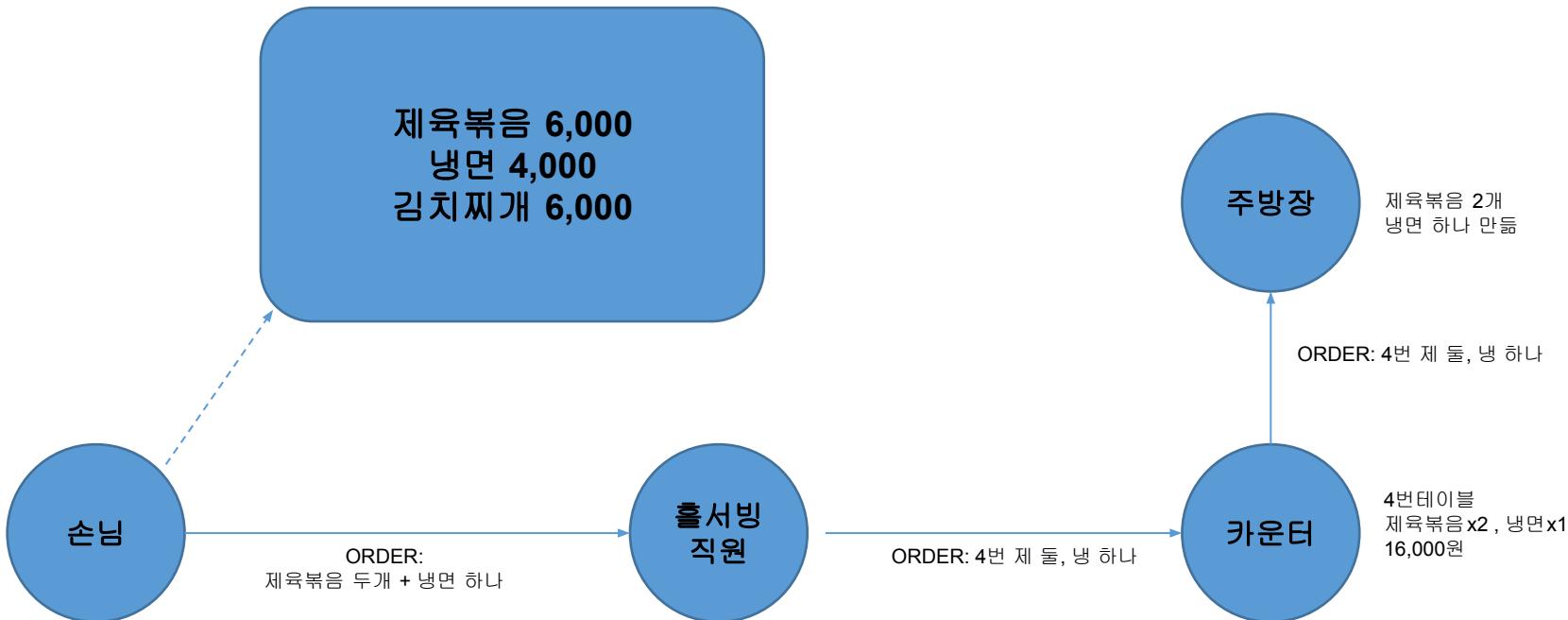
- 요구사항 수시 변경 → 소스코드 변경
  - 소스코드 변경 최소화/구조화
- 팀프로젝트 → 범용코딩스타일 필요
- 인수인계 → 직관적 코드 필요

### 프레임워크 정의

- 비기능적(**Non-Functional**) 요구사항(성능, 보안, 확장성, 안정성 등)을 만족하는 구조와 구현된 기능을 안정적으로 실행하도록 제어해주는 잘 만들어진 구조의 라이브러리의 덩어리
- 프레임워크는 애플리케이션들의 최소한의 공통점을 찾아 하부 구조를 제공함으로써 개발자들로 하여금 시스템의 하부 구조를 구현하는데 들어가는 노력을 절감하게 해줌

## 1-2. 프레임워크

특정한 목적에 맞게 프로그래밍을 쉽게 하기 위한 약속



## 1-2. 스프링(SPRING)

---

- 자바(JAVA)언어를 기반
  - 다양한 어플리케이션을 제작하기 위해 약속된 프로그래밍 툴
  - 개발 중에 테스트 용이
  - EJB\* 대신 톰캣 사용 가능
  - 개발환경 구축에 비용 절감(EJB 비교 WAS\* 필요없음)
    - 경량 컨테이너
- 국내 자바개발자들 사이에서 표준프레임워크

## 1-2. 스프링(SPRING)

---

### EJB : Enterprise JavaBeans

- 미국 Sun Microsystems사가 제창한 규약
- 비지니스로직과 시스템 서비스를 이용하는 로직을 분산
- 비지니스 로직을 탑제한 부품 - "Enterprise Bean"
- 시스템 서비스 로직을 담고 있는 부분 - "컨테이너"
- 개발자는 비지니스 로직 개발에만 전념
- 4개 요소 구성 : Enterprise Bean, Container, EJB Server, Client Application

## 1-2. 스프링(SPRING)

---

### Web Server

- 클라이언트의 요청을 HTML이나 Object를 http 프로토콜을 이용해 전송
- 정적 페이지 생성
- Apache, IIS(Internet Information Server)

### Web Container

- 시스템로직(JSP, 서블릿 실행, 데이터베이스처리 등)을 담고 있는 코드모음
- 동적 페이지 생성
- Servlet 컨테이너, JSP 컨테이너, EJB 컨테이너

### Web Application Server / WAS

- 웹서버 + 웹컨테이너
- EJB 같은 Bean들이 올라가며, 웹서비스에 필요한 기능을 포함
- 웹서버는 웹문서(HTML), 웹컨테이너는 JSP페이지를 처리, 분할처리
- BEA의 WebLogic, IBM의 WebSphere, T-max의 Jeus, Tomcat, Redhot의 JBOSS

## 1-2. 스프링(SPRING) - 위키 자료

---

스프링 프레임워크(Spring Framework)는 자바 플랫폼을 위한 오픈소스 애플리케이션 프레임워크로서 간단히 스프링(Spring)이라고도 불린다. 동적인 웹 사이트를 개발하기 위한 여러 가지 서비스를 제공하고 있다. 대한민국 공공기관의 웹 서비스 개발 시 사용을 권장하고 있는 **전자정부 표준프레임워크**의 기반 기술로서 쓰이고 있다.

로드존슨 2002년 저서 Expert One-on-One J2EE Design and Developement 에 첫 공개

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 : 2017년 10월

6.0 : 2022년 11월 – Java 17 이상 (이전 8 지원 중단)

## 1-2. 스프링(SPRING)

---

### 선행학습

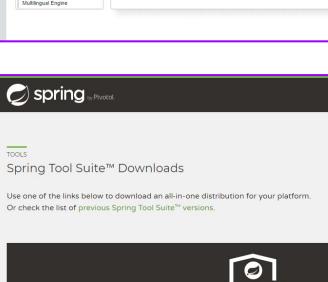
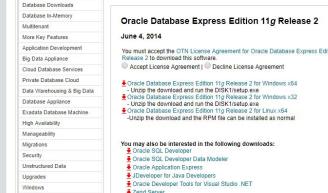
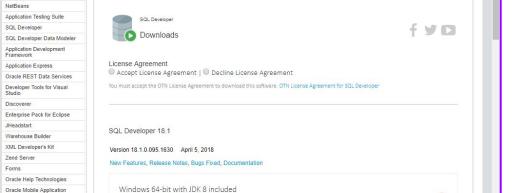
- JAVA 기본문법
- JSP & Servlet
- HTML
- CSS
- Javascript , jquery

### 연계학습

- DBMS (MySQL, Oracle)
- 서버관리 (tomcat)
- 클라우드 AWS, GCP, AZURE
- Git
- Blog

### 1-3. 설치

JDK , Tomcat, STS, MySQL, Oracle Express edition 11g xe, git 설치



# 준비

- jdk 1.8(202) & 11
  - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- STS
  - <https://spring.io/tools>
- Tomcat 9.0
  - <http://tomcat.apache.org/>
- GIT
  - <https://git-scm.com/>

# 준비

- Oracle Express 11g XE
  - <http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html>
- sqldeveloper
  - <http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html>
- MySQL 5.7 & Workbench
  - <https://www.mysql.com/downloads/>

# 설치

C:\java

\sts-3.9.17.RELEASE

\apache-tomcat-9

\workSpace

# Top 10 DOS Commands & Pages

- |                      |                  |
|----------------------|------------------|
| 1. <b>cd</b>         | 1. <b>xcopy</b>  |
| 2. <b>dir</b>        | 2. <b>copy</b>   |
| 3. <b>copy</b>       | 3. <b>dir</b>    |
| 4. <b>del</b>        | 4. <b>net</b>    |
| 5. <b>edit</b> (이지원) | 5. <b>format</b> |
| 6. <b>move</b>       | 6. <b>del</b>    |
| 7. <b>ren</b>        | 7. <b>attrib</b> |
| 8. <b>deltree</b>    | 8. <b>cd</b>     |
| 9. <b>cls</b>        | 9. <b>ping</b>   |
| 10. <b>format</b>    | 10. <b>set</b>   |

## JAVA 환경변수 설정

```
>set JAVA_HOME="C:\...\jdk1.8..."
```

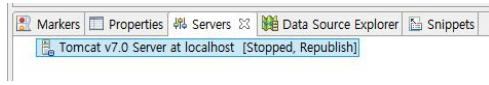
```
>path=JAVA_HOME\bin;%path%
```

자바 설치된 경로

실행파일이 있는 경로

## 1-3. 설치

### 1. 서버 더블클릭



### 2. 설정

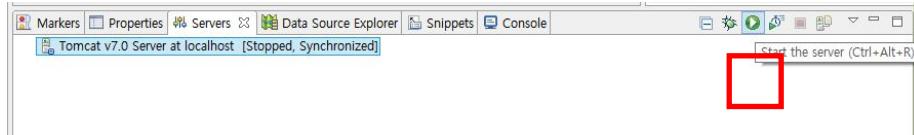
The screenshot shows the 'Tomcat v7.0 Server at localhost' configuration page in the Eclipse IDE. Several settings are highlighted with red boxes:

- Server Locations:** The 'Server path' field (containing 'C:\java\alec\apache-tomcat-7.0.57\apache-tomcat-7') is highlighted.
- Server Options:** The 'Publish module contexts to separate XML files' checkbox is checked and highlighted.
- Ports:** The 'HTTP/1.1' port (8181) is highlighted.

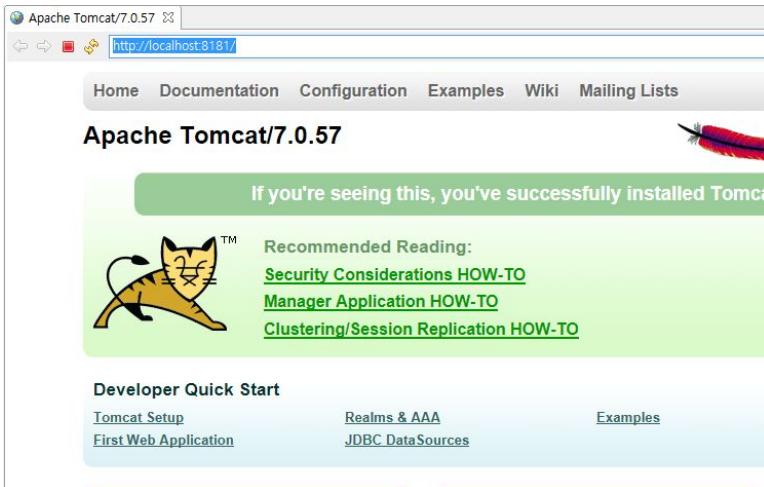
The configuration page also includes sections for General Information, Publishing, Timeouts, and MIME Mappings.

## 1-3. 설치

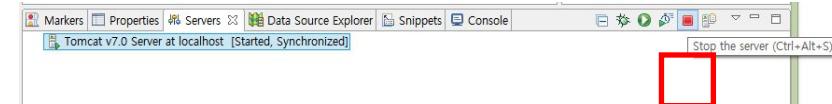
### 3. 서버 start



### 4. 서버 구동 확인



### 5. 서버 stop



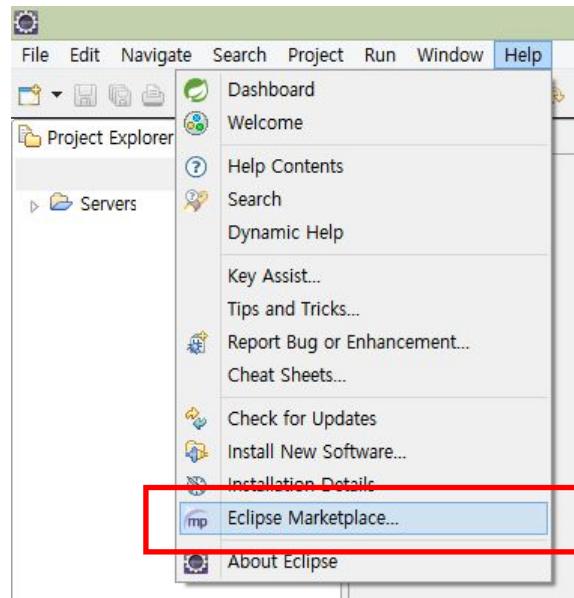
## 2. 스프링 프로젝트 만들기

- 이클립스에 스프링 플러그인 설치(STS)
- 처음 만들어 보는 스프링 프로젝트
- DI(Dependency Injection)와 IOC컨테이너

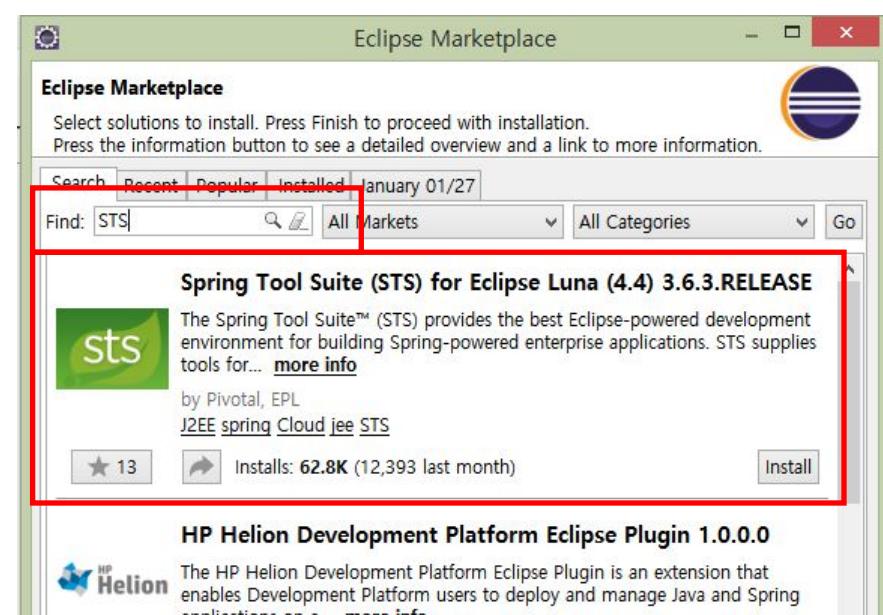
## 2-1. 이클립스에 스프링 플로그인 설치(Spring Tool Suite) - 이클립스에서만 필요

이클립스에서 스프링 프로젝트를 만들기 위해서는 스프링 플러그인이 필요 합니다.

이클립스 실행 및 Marketplace.... 진입



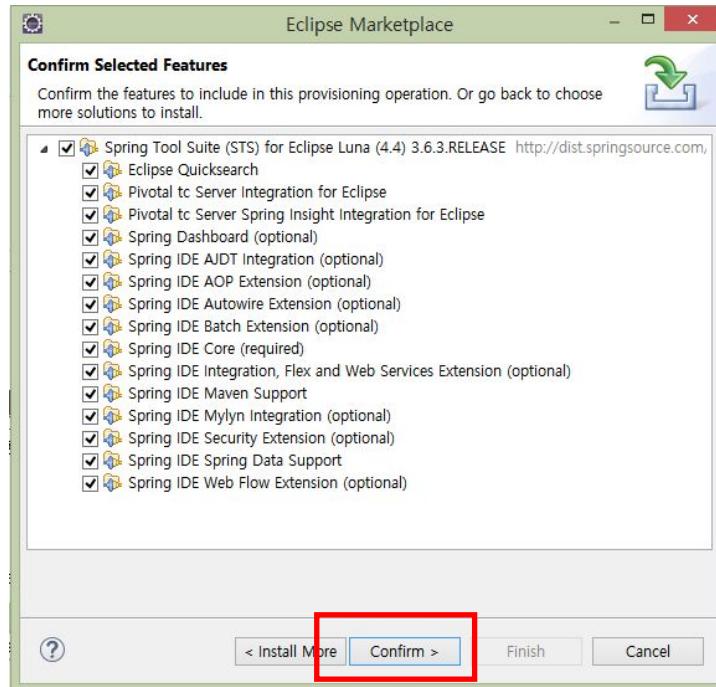
STS 플러그인 검색 하기



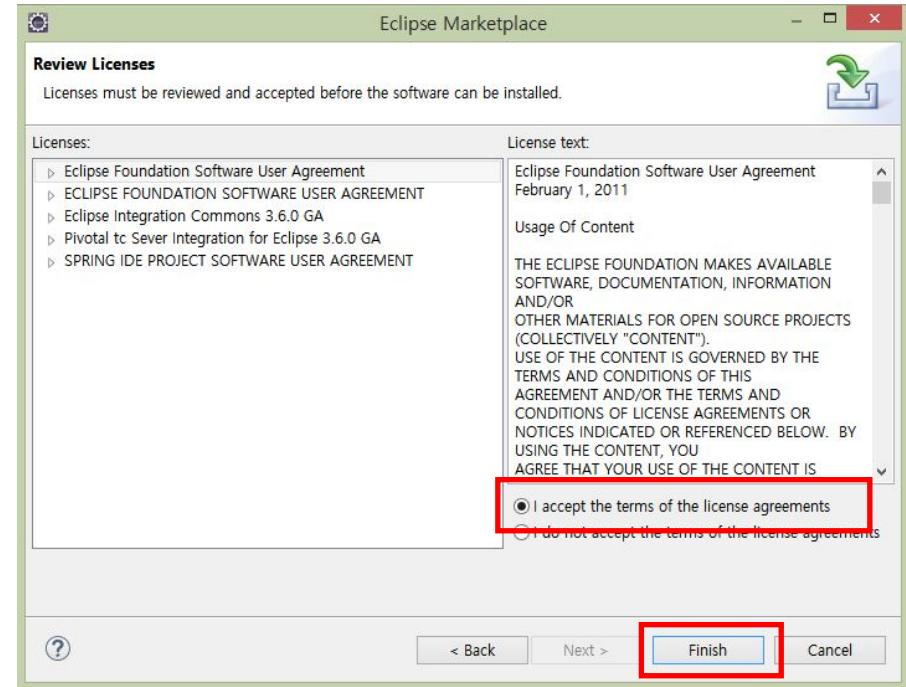
## 2-1. 이클립스에 스프링 플로그인 설치(Spring Tool Suite)

이클립스에서 스프링 프로젝트를 만들기 위해서는 스프링 플러그인이 필요 합니다.

모두 선택하고 Confirm 버튼 클릭



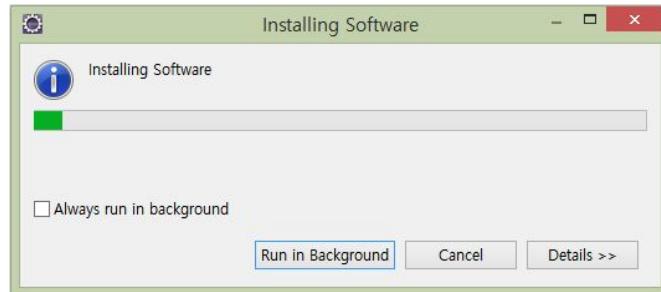
약관에 동의 후 진행



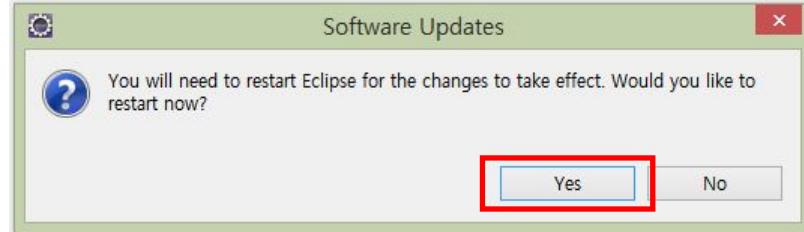
## 2-1. 이클립스에 스프링 플로그인 설치(Spring Tool Suite)

이클립스에서 스프링 프로젝트를 만들기 위해서는 스프링 플러그인이 필요 합니다.

설치 진행중



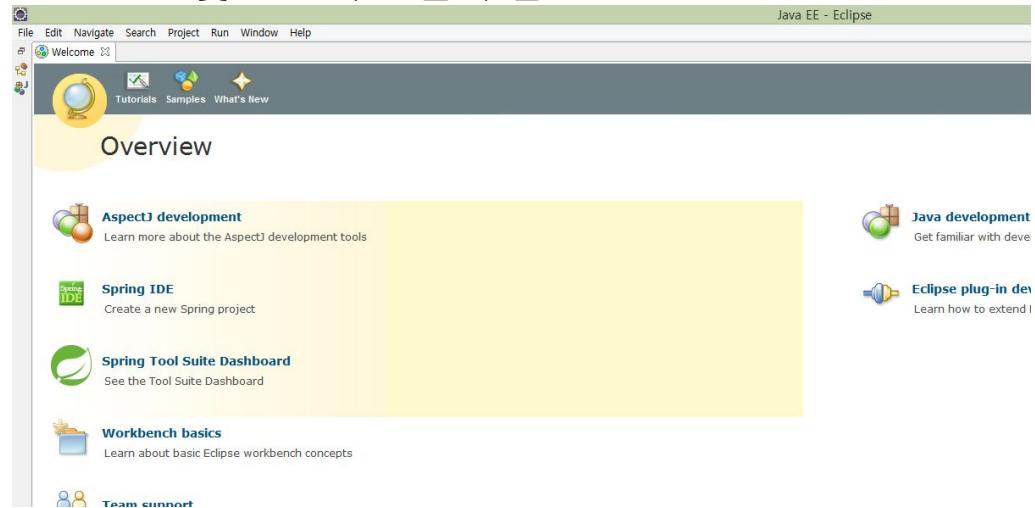
설치 완료 후 재부팅



## 2-1. 이클립스에 스프링 플로그인 설치(Spring Tool Suite)

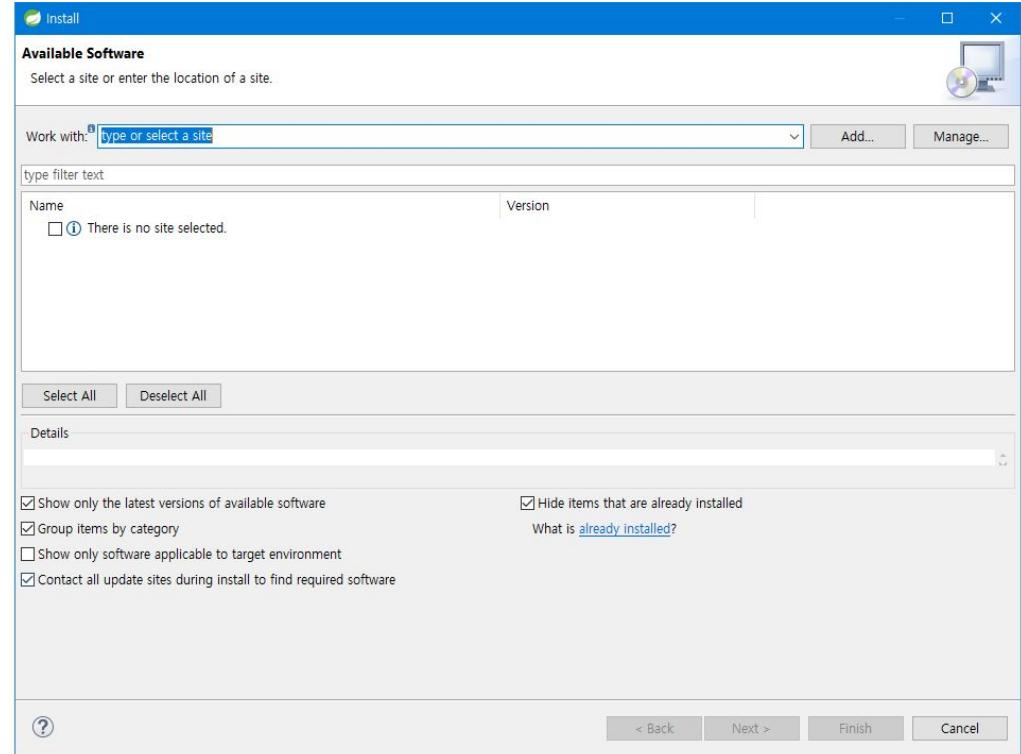
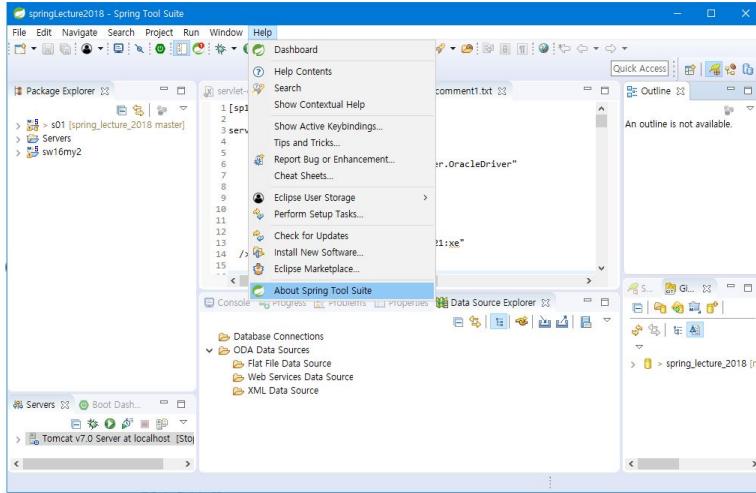
이클립스에서 스프링 프로젝트를 만들기 위해서는 스프링 플러그인이 필요 합니다.

### Overview 및 프로젝트 준비 완료



## 2-1. 이클립스에 추가소프트웨어 설치(Spring Tool Suite)

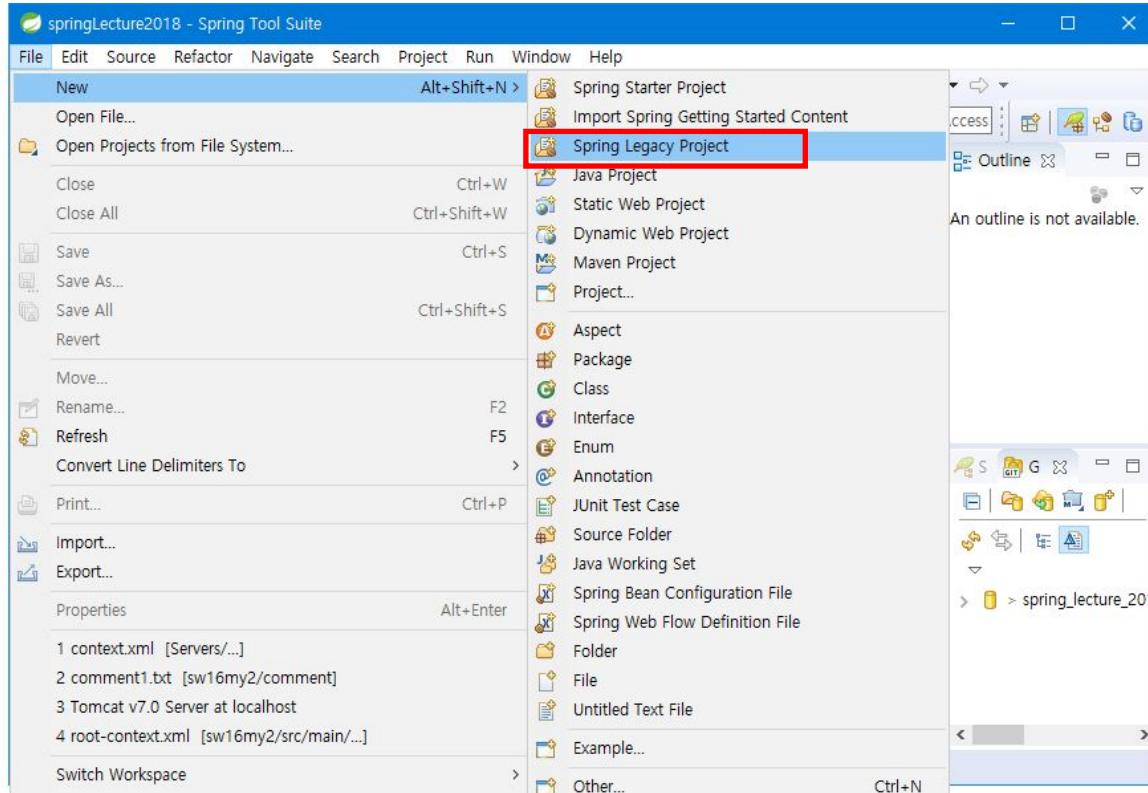
이클립스에서 다양한 3rd party 프로그램을 설치할 수 있습니다.



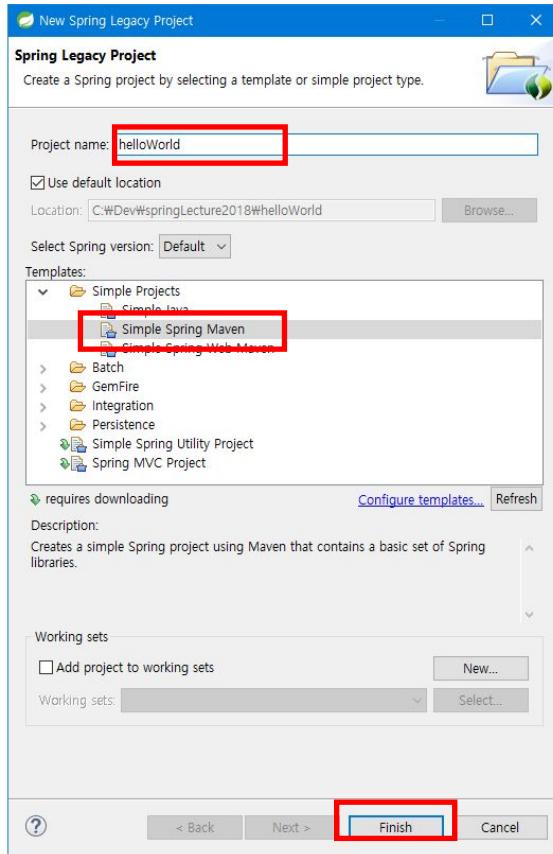
## 2-2. 처음 만들어 보는 스프링 프로젝트

- 쉽게 시작 & 기초를 단단히

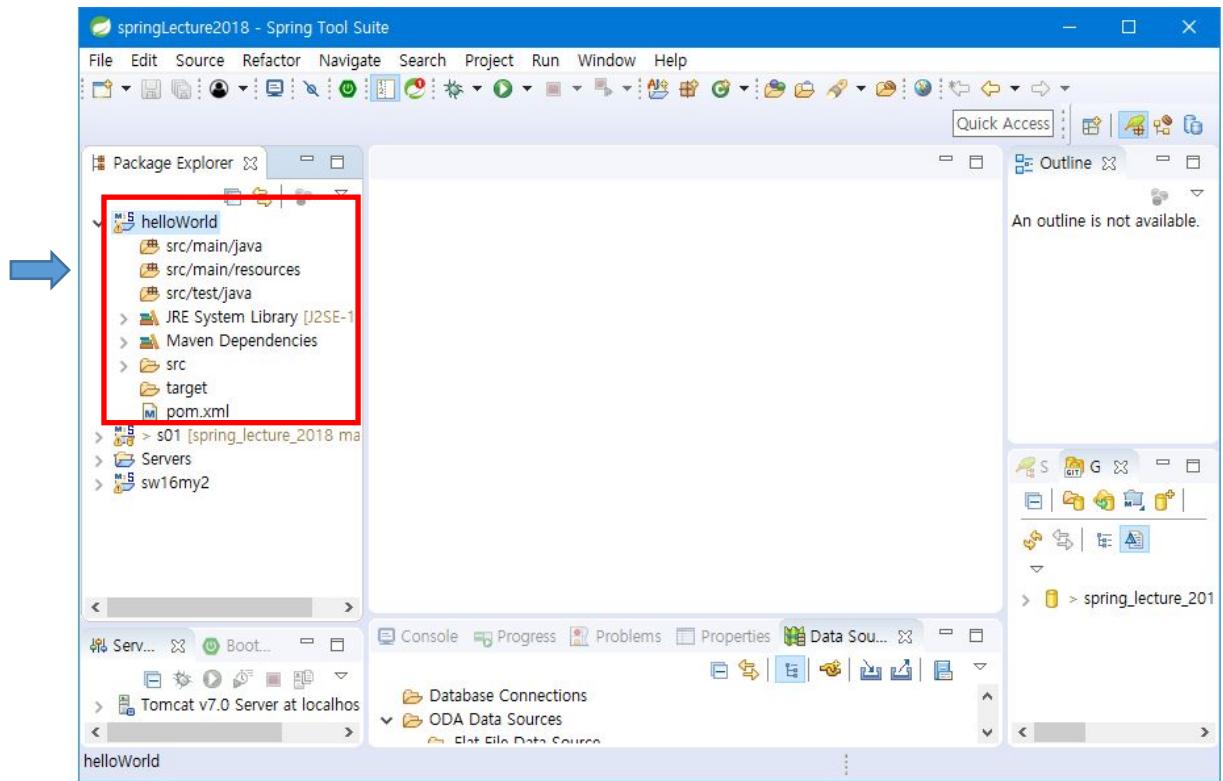
스프링 프로젝트 생성하기



## 2-2. 처음 만들어 보는 스프링 프로젝트

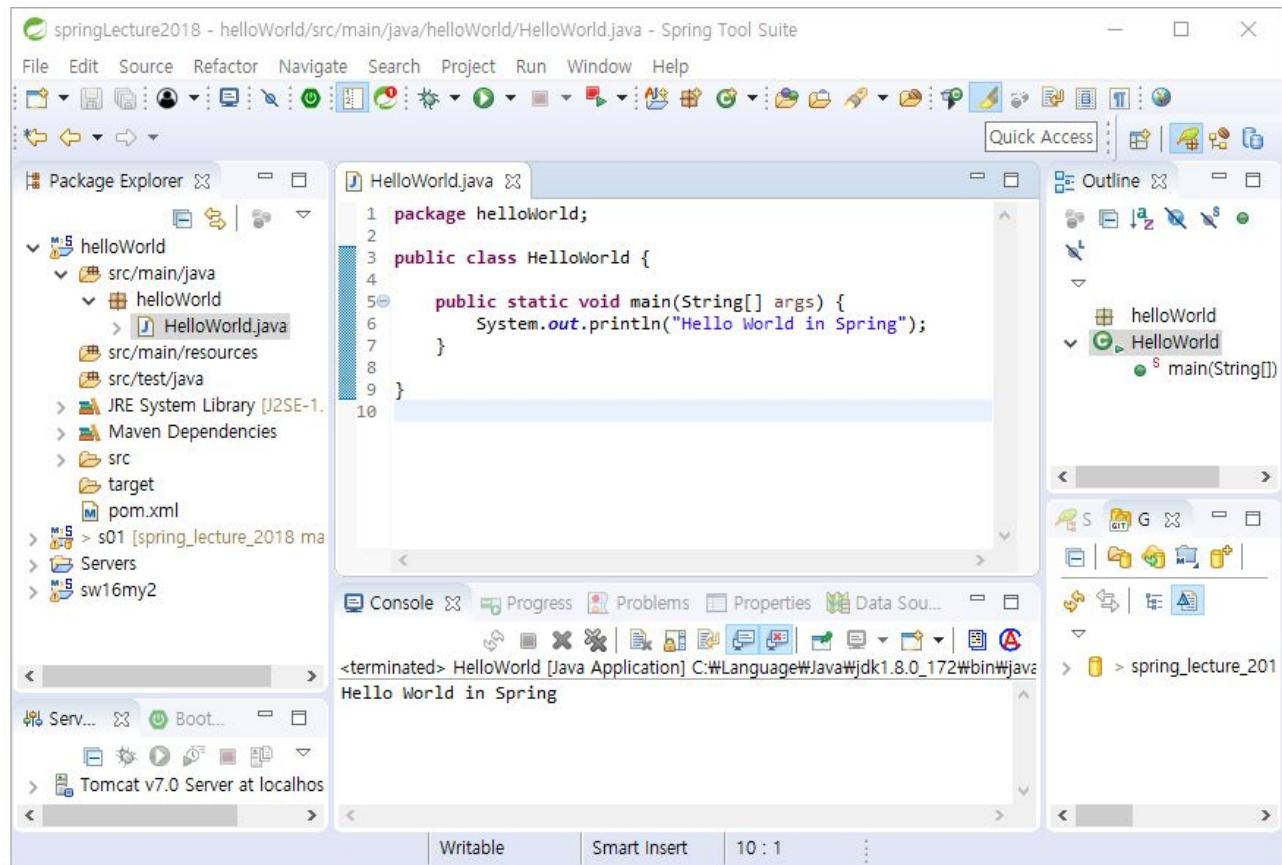


스프링 프로젝트 생성 완료



## 2-2. 처음 만들어 보는 스프링 프로젝트

### “Hello World in Spring”



## 2-2. 처음 만들어 보는 스프링 프로젝트

---

“Hello World in Spring”

```
package helloworld;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World in Spring");
    }

}
```

project s01

## 2-2. 처음 만들어 보는 스프링 프로젝트

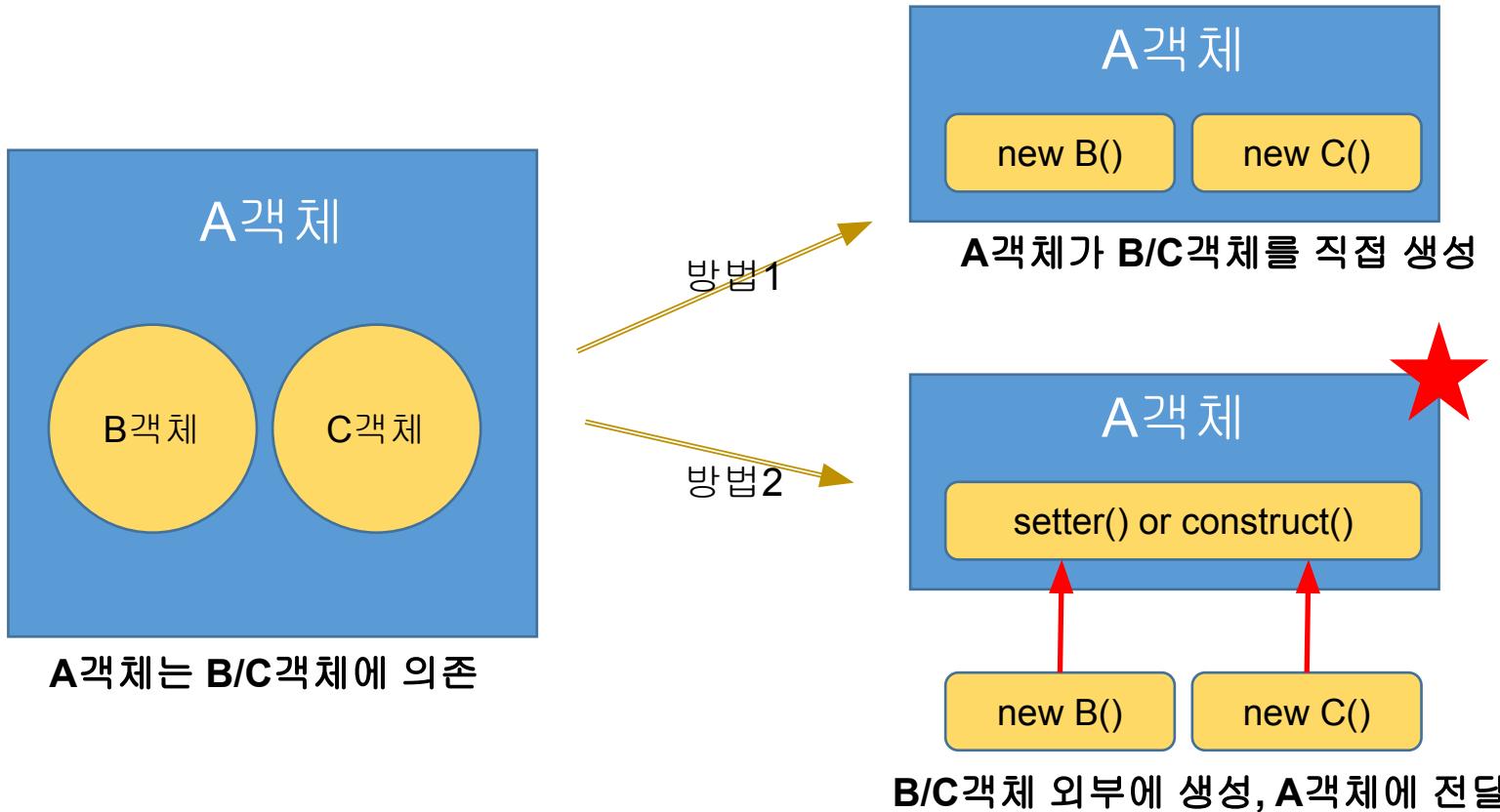
---

### Framework 종류

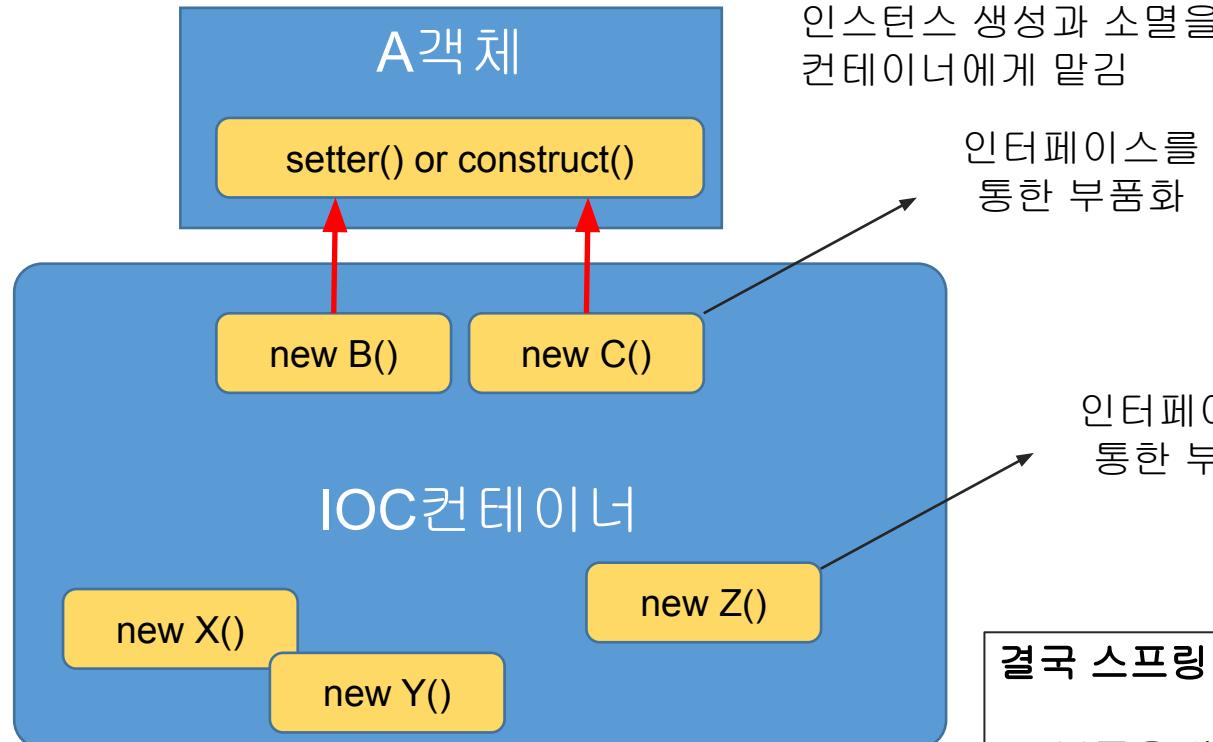
아키텍쳐 = 사용하는 프레임워크의 종류 + 사용전략

기능	Framework 종류
웹 (MVC)	<b>Spring MVC</b> , Struts2, Webwork, PlayFramework
OR 맵핑	<b>MyBatis</b> , Hibernate, JPA, Spring JDBC
AOP	Spring AOP, AspectJ, JBoss AOP
DI	Spring DI, Google Guice
Build / Lib 관리	Ant+Ivy, <b>Maven</b> , Gradle
단위테스트	jUnit, TestNG, Cactus
JavaScript	jQuery, AngularJS, Node.js

## 2-3. DI(Dependency Injection)와 IOC컨테이너



## 2-3. DI(Dependency Injection)와 IOC컨테이너



IOC : inversion Of Control, 제어의 반전(역전),  
인스턴스 생성과 소멸을 개발자가 아닌  
컨테이너에게 맡김

인터페이스를  
통한 부품화

인터페이스를  
통한 부품화

결국 스프링이란?

- 부품을 생성하고 조립하는 라이브러리  
집합체

## 2-3. DI(Dependency Injection)와 IOC컨테이너

---

**IoC** Inversion of Control

제어의 역전

개발자가 아닌 컨테이너가 제어

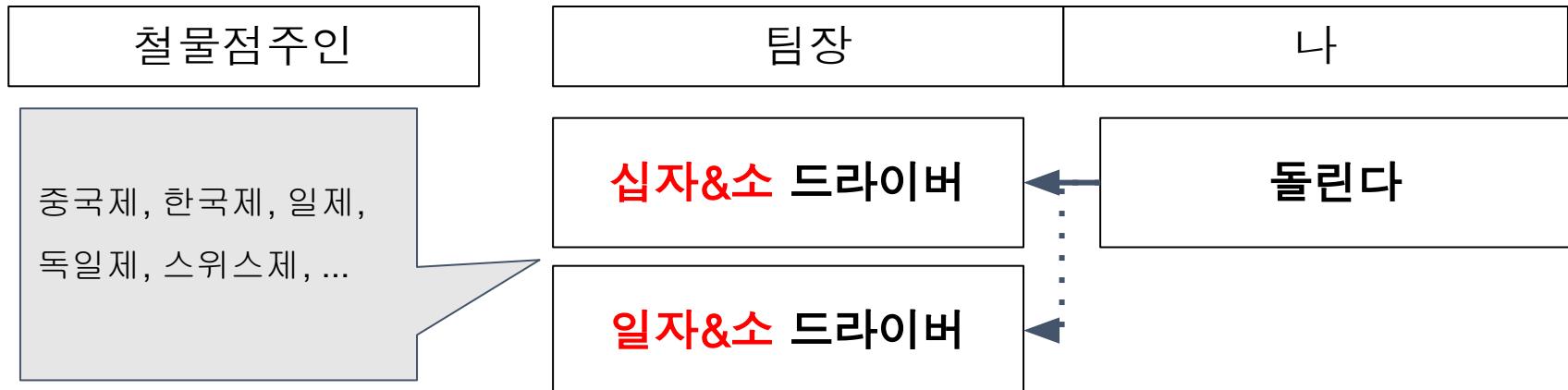
개발자가 신경 쓸 부분을 감소

## 2-3. DI(Dependency Injection)와 IOC컨테이너

### Screw Driver Important Informations



1. 머리타입 : 십자(+), 일자(-)
2. 크기 : 소, 중, 대



## 2-3. DI(Dependency Injection)와 IOC컨테이너

```
public class MainClass {  
    public static void main(String[] args) {  
        Calculator calculator = new Calculator();  
        calculator.setFirstNum(10);  
        calculator.setSecondNum(2);  
        calculator.addition();  
    }  
}
```

```
public class Calculator {  
    private int firstNum;  
    private int secondNum;  
    public int getFirstNum() {...}  
    public void addition() { System.out.println(firstNum + secondNum); }  
}
```

project s02

# git 명령어

- git add --all .
- git commit -m “First Commit”
- git push

### Calculator

com.wind.sp02

- firstNum: int
- secondNum: int
- getFirstNum(): int
- setFirstNum(firstNum: int): void
- getSecondNum(): int
- setSecondNum(secondNum: int): void
- addition(): void
- subtraction(): void
- multiplication(): void
- division(): void

### MainClass

com.wind.sp02

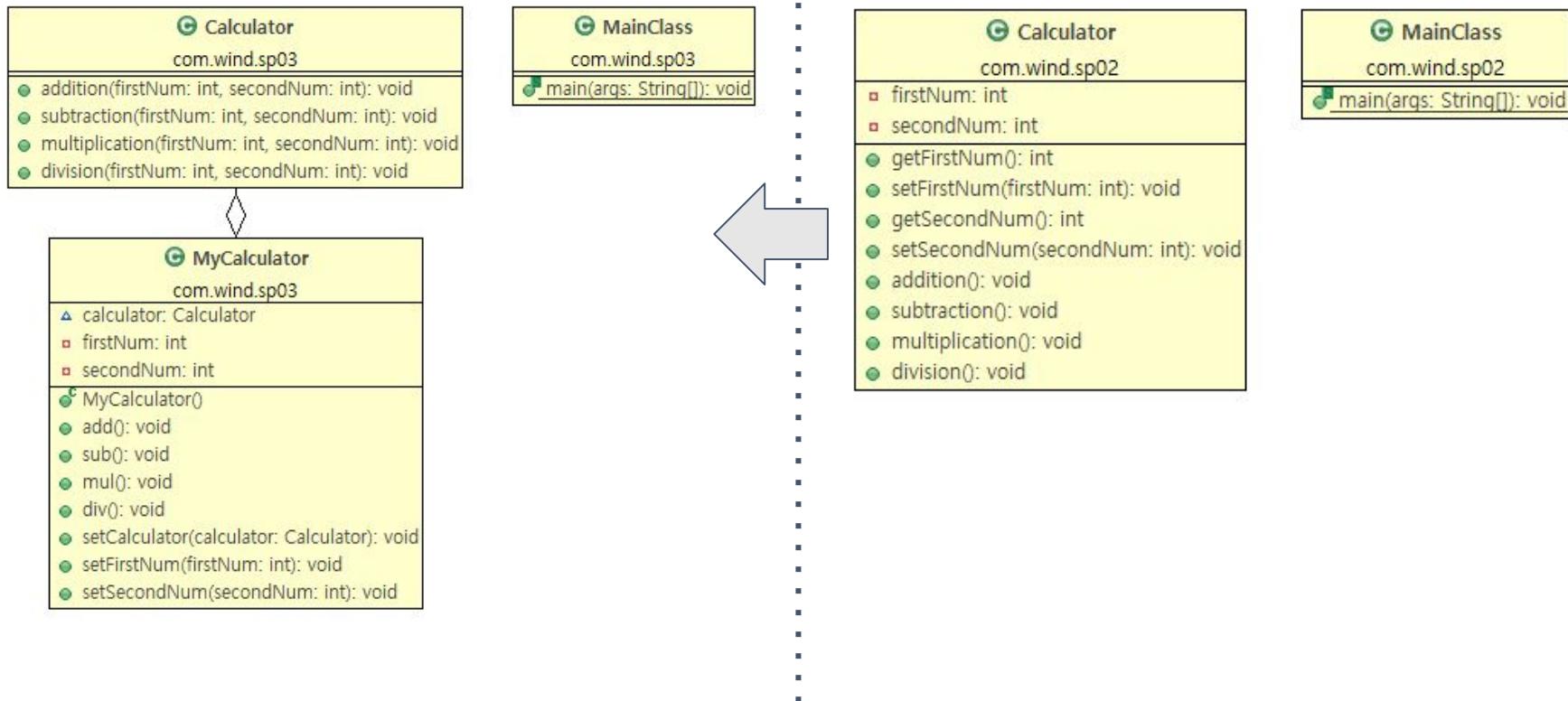
- main(args: String[]): void

### 3. DI(Dependency Injection)-I

- 스프링을 이용한 객체 생성과 조립
- 스프링 설정 파일의 이해

project s03a

### 3-1. 스프링을 이용한 객체 생성과 조립



### 3-1. 스프링을 이용한 객체 생성과 조립

---

지난 시간 예제는 스프링의 특징(사용법)이 적용되지 않은 프로젝트 예제

```
public static void main(String[] args) {  
  
    MyCalculator myCalculator = new MyCalculator();  
    myCalculator.setCalculator(new Calculator());  
  
    myCalculator.setFirstNum(10);  
    myCalculator.setSecondNum(2);  
  
    myCalculator.add();  
    myCalculator.sub();  
    myCalculator.mul();  
    myCalculator.div();  
  
}
```

스프링을 사용하지 않은 프로젝트

```
public static void main(String[] args) {  
  
    String configLocation = "classpath:applicationCTX.xml";  
    AbstractApplicationContext ctx = new GenericXmlApplicationContext(configLocation);  
    MyCalculator myCalculator = ctx.getBean("myCalculator", MyCalculator.class);  
  
    myCalculator.add();  
    myCalculator.sub();  
    myCalculator.mul();  
    myCalculator.div();  
  
}
```

스프링을 적용한 프로젝트

### 3-2. 스프링 설정 파일의 이해

```
main(){

String configLocation = "classpath:applicationCTX.xml";
AbstractApplicationContext ctx = new GenericXmlApplicationContext(configLocation);
MyCalculator myCalculator = ctx.getBean("myCalculator", MyCalculator.class);
ctx.close();
...
}
```

**classpath** = src/main/java + src/main/resource

**GenericXmlApplicationContext** : xml로부터 정보를 얻어옴

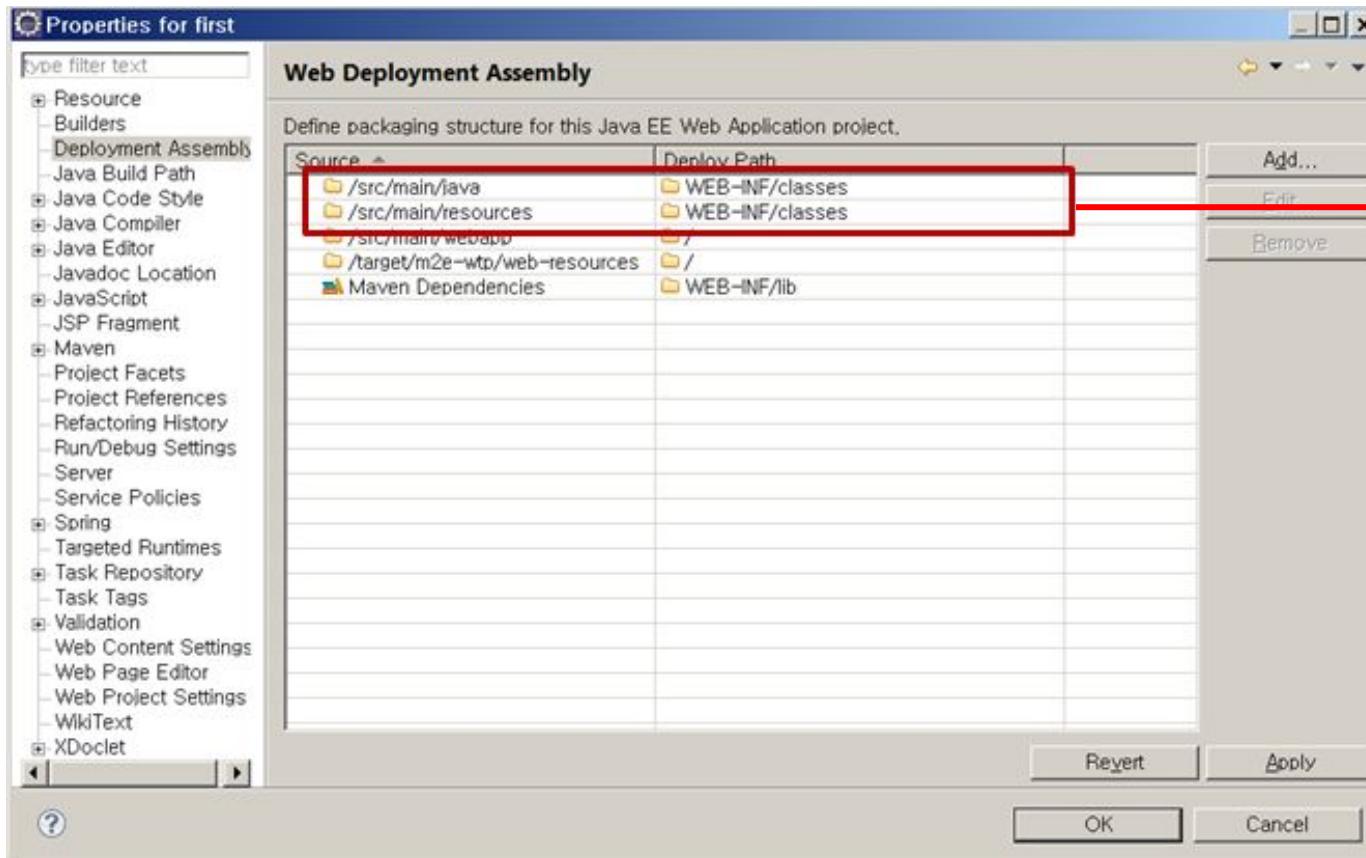
**AbstractApplicationContext** : GenericXmlApplicationContext 의 부모클래스(추상)

**getBean(name, Class)** : Class로부터 name 의 bean 생성

### 3-2. 스프링 설정 파일의 이해 - DI (Dependency Injection) 용어정리

빈(Bean)	<ul style="list-style-type: none"><li>스프링이 IoC 방식으로 관리하는 객체</li><li>스프링이 직접 생성과 제어를 담당하는 객체</li></ul>
빈 팩토리 (BeanFactory)	<ul style="list-style-type: none"><li>스프링의 IoC를 담당하는 핵심 컨테이너</li><li>Bean을 등록, 생성, 조회, 반환하는 기능</li><li>BeanFactory 보다 ApplicationContext를 주로 이용</li></ul>
애플리케이션 컨텍스트 (Application Context)	<ul style="list-style-type: none"><li>BeanFactory를 확장한 IoC 컨테이너</li><li>Bean을 등록하고 관리하는 기능은 BeanFactory와 동일</li><li>스프링이 제공하는 각종 부가 서비스를 추가로 제공</li><li>BeanFactory 보다 ApplicationContext가 주로 사용됨</li></ul>
설정 메타정보 (Configuration metadata)	<ul style="list-style-type: none"><li>ApplicationContext / BeanFactory가 IoC 적용을 위해 사용</li><li>IoC컨테이너에 의해 관리되는 Bean 객체 생성에 사용</li></ul>

### 3-2. 스프링 설정 파일의 이해



클래스패스 : classpath

### 3-2. 스프링 설정 파일의 이해

#### 스프링을 학습하는 첫 단계는 스프링 설정 파일의 이해

```
<?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="calculator" class="com.javalec.ex.Calculator" /> 변수설정

    <bean id="myCalculator" class="com.javalec.ex.MyCalculator">
        <property name="calculator">
            <ref bean="calculator"/>
        </property>
        <property name="firstNum" value="10"/>
        <property name="secondNum" value="2"/>
    </bean> 필드설정

```

객체생성

### 3-2. 스프링 설정 파일의 이해

스프링 설정

```
<bean id="myCalculator" class="com.javalec.ex.MyCalculator">
    <property name="calculator">
        <ref bean="calculator"/>
    </property>
    <property name="firstNum" value="10"/>
    <property name="secondNum" value="2"/>
</bean>
```

Java 소스

```
private int firstNum;
private int secondNum;
public void setFirstNum(int firstNum) {
    this.firstNum = firstNum;
}
public void setSecondNum(int secondNum) {
    this.secondNum = secondNum;
}
```

xml : Java  
firstNum : firstNum & setFirstNum()

project s03b

### 3-2. 스프링 설정 파일의 이해

```
private int firstNum;  
  
public setFirstNum(int firstNum) {  
    this.firstNum = firstNum;  
}
```

변수(필드)이름 - 영문소문자 : **firstNum**

setter 이름 영문소문자시작, 카멜규칙 : **setFirstNum()**

```
<bean id="myCalculator" class="com.wind.sp03.MyCalculator">  
    .....  
    <property name="firstNum" value="10" />  
</bean>
```

### 3-2. 스프링 설정 파일의 이해

```
private int firstNum;  
  
public setFirstNum(int firstNum) {  
    this.firstNum = firstNum;  
}
```

변수(필드)이름 - 영문소문자 : **firstNum**

setter 이름 영문소문자시작, 카멜규칙 : **setFirstNum()**

```
<bean id="myCalculator" class="com.wind.sp03.MyCalculator">  
    .....  
    <property name="firstNum" value="10" />  
</bean>
```

### 3-2. 스프링 설정 파일의 이해

```
private int firstNum;  
  
public setFirstNum(int firstNum)  
    this.firstNum = firstNum;  
}
```

변수(필드)이름 - 영문소문자 : **firstNum**

setter 이름 영문소문자시작, 카멜규칙 : **setFirstNum()**

```
<bean id="myCalculator" class="com.wind.sp03.MyCalculator">  
    .....  
    <property name="firstNum" value="10" />  
</bean>
```

### 3-2. 스프링 설정 파일의 이해

#### xml에서 bean 사용(등록)

```
<bean id="calculator" class="com.wind.sp03.Calculator" />

<bean id="myCalculator" class="com.wind.sp03.MyCalculator">
    <property name="calculator" ref="calculator"/>
    <property name="firstNum" value="10" />
    <property name="secondNum" value="4" />
</bean>
```



value 사용 : 단순값(문자열, 숫자 등)

## 4. DI(Dependency Injection)- II

- 스프링 프로퍼티 설정
- 스프링 컨테이너의 이해

## 4-1. 스프링 프로퍼티 설정

### 스프링 프로퍼티 설정에 대한 기본적인 사항 학습

```
<bean id="myInfo" class="com.javalec.ex.MyInfo">
    <property name="name">
        <value>홍길동</value>
    </property>
    <property name="height">
        <value>187</value>
    </property>
    <property name="weight">
        <value>84</value>
    </property>
    <property name="hobbys">
        <list>
            <value>수영</value>
            <value>요리</value>
            <value>독서</value>
        </list>
    </property>
    <property name="bmiCalculator">
        <ref bean="bmiCalcaulator"/>
    </property>
</bean>
```

기초데이터

List 타입

다른 빈 객체 참조

```
private String name;
private double height;
private double weight;
private ArrayList<String> hobbys;
private BMIcalculator bmiCalculator;

public void setBmiCalculator(BMIcalculator bmiCalculator) {
    this.bmiCalculator = bmiCalculator;
}

public void setName(String name) {
    this.name = name;
}

public void setHeight(double height) {
    this.height = height;
}

public void setWeight(double weight) {
    this.weight = weight;
}

public void setHobbys(ArrayList<String> hobbys) {
    this.hobbys = hobbys;
}
```

#### 4-1. 스프링 프로퍼티 설정

```
<bean>
    <property>
        <value>...</value>
    </property>
    <property>
        <list>
            <value>...</value>
            <value>...</value>
        </list>
    </property>
    <property name="var_name">
        <ref bean="out_name"/>
    </property>
    <property name="###" ref="#" />
</bean>
```

기초데이터 : int, long, double, etc.

<value> 이용

List 타입 : Array<>

<list> 사용

다른 Bean 객체 :

ref 사용

## 4-2. 스프링 컨테이너의 이해

---

### 스프링 컨테이너를 생성하고 컴포넌트를 사용하는 방법

```
String configLocation = "classpath:applicationCTX.xml";
AbstractApplicationContext ctx = new GenericXmlApplicationContext(configLocation); ← 스프링 컨테이너 생성
MyInfo myInfo = ctx.getBean("myInfo", MyInfo.class); ← 스프링 컨테이너에서 컴포넌트 가져옴
myInfo.getInfo();
ctx.close();
```

project s04

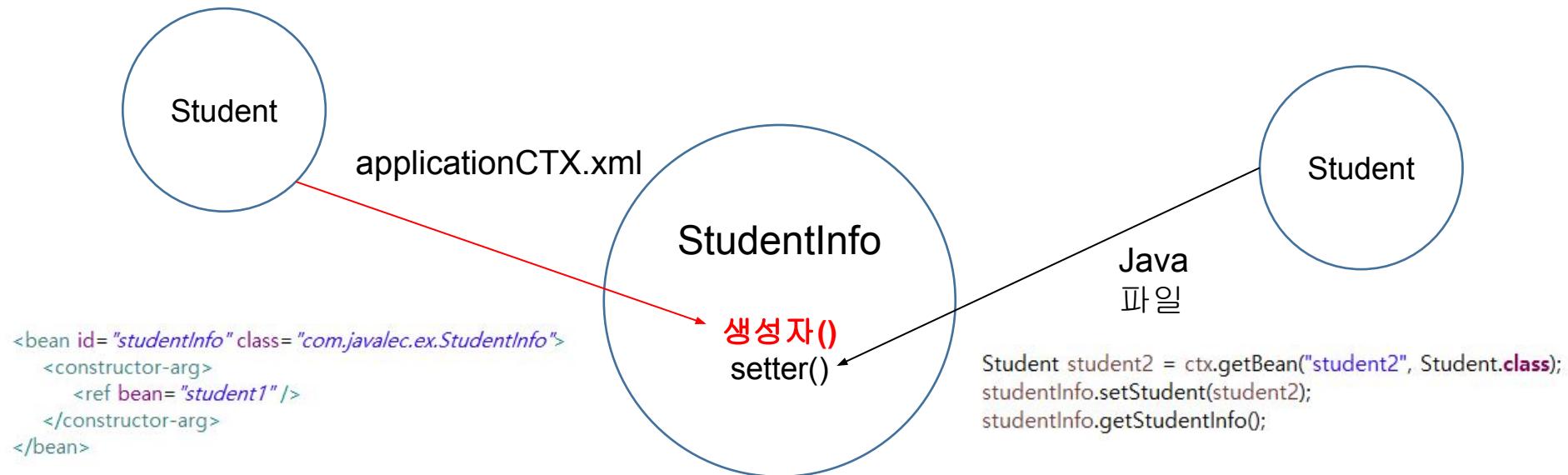
# 5. DI 활용

-의존 관계

-DI 사용에 따른 장점

## 5-1. 의존 관계

DI : Dependency Injection, ‘의존주입’



**StudentInfo 객체는 Student 객체에 의존**

## 5-1. 의존 관계 - 생성자(Constructor Injection) 이용

```
<bean id="###" class="####">
    <constructor-arg>
        <value>...</value>
    </constructor-arg>

    <constructor-arg value="#" />

    <constructor-arg index="1" value="#" />
    <constructor-arg index="0" ref="#" />
</bean>
```

차이점

Setter      <-> 생성자

**<property>**      **<constructor-arg>**

## 5-1. 의존 관계

---

**Setter Injection** : <property> 태그 -

단순 값 주입(기초데이터 : int, double, etc.)

Collection 타입의 값 주입(List, Set, Map, Properties, etc.)

list -> <list> <value>##</value> </list>

set -> <set> <value>##</value> </set>

map -> <map> <entry key="##" value="###" />...</map>

**Constructor Injection** : <constructor-arg> 태그

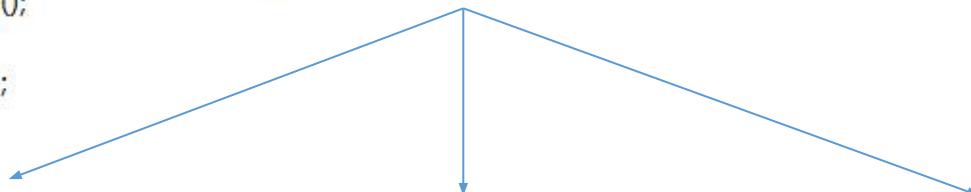
project s05a

## 5-2. DI 사용에 따른 장점

- 작은 규모의 프로젝트에서는 스프링의 DI 사용을 하는 것 보다 일반적인 방법을 사용
- 규모가 어느 정도 커지고, 유지보수 업무가 발생시 DI 사용

```
AbstractApplicationContext ctx = new GenericXmlApplicationContext("classpath:applicationCTX.xml");
Pencil pencil = ctx.getBean("pencil", Pencil.class);
pencil.use();

ctx.close();
```



<bean id="pencil" class="com.javalec.ex.Pencil4B" />

정보: Pre-instantiating

4B 굵기로 쓰입니다.

2월 09 2015 4:28:10

<bean id="pencil" class="com.javalec.ex.Pencil6B" />

정보: Pre-instantiating

6B 굵기로 쓰입니다.

2월 09 2015 4:28:15

<bean id="pencil" class="com.javalec.ex.Pencil6BWithEraser" />

정보: Pre-instantiating singletons in C

6B 굵기로 쓰이고, 지우개가 있습니다.

2월 09 2015 4:28:19 오후 org.springframework

Java파일의 수정 없이  
스프링 설정 파일만을 수정  
부품들을 생성/조립

project s05b

# 6강. DI설정 방법

- XML파일을 이용한 DI설정 방법
- JAVA를 이용한 DI설정 방법
- XML과 JAVA를 같이 사용

## 6-1. XML파일을 이용한 DI 설정 방법

### 기본 사항 학습

```
<bean id="student1" class="com.javalec.ex.Student">
    <constructor-arg value="홍길동"/> ← 생성자 설정(기초데이터)
    <constructor-arg value="10"/>
    <constructor-arg>
        <list>
            <value>수영</value> ← 생성자 설정(객체데이터)
            <value>요리</value>
        </list>
    </constructor-arg>

    <property name="height">
        <value>187</value> ← setter() 설정(property)
    </property>

    <property name="weight" value="84" />
</bean>
```

## 6-1. XML파일을 이용한 DI 설정 방법

```
<bean id="studentInfo1" class="com.javalec.ex.StudentInfo">
    <property name="student">
        <ref bean="student1"/>
    </property>
</bean>
```



다른 빈객체 설정

```
String configLocation1 = "classpath:applicationCTX.xml";
String configLocation2 = "classpath:applicationCTX1.xml";
AbstractApplicationContext ctx = new GenericXmlApplicationContext(configLocation1, configLocation2);
Student student1 = ctx.getBean("student1", Student.class);
```

스프링 컨테이너 생성

스프링 설정 파일이  
다수인 경우

스프링 컨테이너에서  
객체 생성\*

## 6-1. XML파일을 이용한 DI 설정 방법

---

2개 이상의 XML 설정파일을 사용할 때

MainClass -- main() --

```
String configLocation1 = "classpath:applicationCTX.xml";
String configLocation2 = "classpath:applicationCTX1.xml";
AbstractApplicationContext ctx =
    new GenericXmlApplicationContext(configLocation1, configLocation2);
```

## 6-1. XML파일을 이용한 DI 설정 방법

```
xmlns:c="http://www.springframework.org/schema/c"  
xmlns:p="http://www.springframework.org/schema/p"
```

c네임스페이스  
constructor

```
<bean id="family" class="com.wind.sp06.Family">  
    <constructor-arg value="더스틴 호프먼" />  
    <constructor-arg value="메릴 스트립" />  
    <property name="brotherName" value="빌리" />  
</bean>
```

p네임스페이스  
property

## 6-1. XML파일을 이용한 DI 설정 방법

```
xmlns:c="http://www.springframework.org/schema/c"  
xmlns:p="http://www.springframework.org/schema/p"
```

c네임스페이스  
constructor

```
<bean id="family" class="com.wind.sp06.Family">  
    <constructor-arg value="더스틴 호프먼" />  
    <constructor-arg value="메릴 스트립" />  
    <property name="brotherName" value="빌리" />  
</bean>
```

```
<bean id="family" class="com.wind.sp06.Family"  
    c:fatherName="더스틴 호프먼"  
    c:motherName="메릴 스트립"  
    p:brotherName="빌리" >  
</bean>
```

p네임스페이스  
property

project s06a

## 전략1 – XML 단독 사용

### 모든 Bean을 XML에 등록

- Bean의 개수가 많아지면 XML 파일을 관리 불편
- 여러 개발자가 같은 설정파일을 공유해서 개발하다 보면 설정파일을 동시에 수정하다가 충돌
- DI에 필요한 setter 또는 constructor가 필수
- 개발 중에는 어노테이션 설정방법, 운영에는 XML설정

## 전략2 – XML 과 Bean Scanning의 혼용

어노테이션이 부여된 클래스를 찾아 자동으로 Bean 등록

- 빈 스캐닝(Bean Scanning)으로 자동인식 Bean 등록기능
- XML 문서 생성과 관리 수고를 덜고 개발 속도를 향상
- 등록 Bean, Bean들 사이의 의존관계 파악 곤란

### 어노테이션 (Annotation)

- 이란?           JDK 1.5 부터 새롭게 추가된 문법
- 사전적 주석의 의미
  - 자바 코드에 주석처럼 달아 특수한 의미를 부여
  - 특별한 의미는 컴파일 또는 런타임에 해석
  - @기호가 붙은 심벌을 클래스, 메소드, 프로퍼티, 변수에 사용

## 6-2. JAVA를 이용한 DI 설정 방법 - 빈등록 메타정보 구성 전략

<b>@Component</b>	컴포넌트를 나타냄, <bean> 태그 역할
<b>@Repository</b>	퍼시스턴스(persistence) 레이어, 영속성을 가지고 파일, 데이터베이스등을 다룰때 주로 사용하는 클래스
<b>@Service</b>	서비스 레이어, 비즈니스 로직 클래스
<b>@Controller</b>	view 담당, 웹 어플리케이션에서 웹 요청과 응답을 처리

## 6-2. JAVA를 이용한 DI 설정 방법 - 의존 객체를 자동으로 주입 : **@Autowired, @Resource**

---

### 의존 객체를 자동으로 주입 **@Autowired, @Resource**

#### **@Autowired**

- 정밀한 의존관계 주입 (Dependency Injection)이 필요한 경우
- 프로퍼티, setter 메서드, 생성자, 일반메서드에 적용
- 의존하는 객체를 주입할 때 주로 **Type**을 이용
- <property>, <constructor-arg> 태그 역할

## 6-2. JAVA를 이용한 DI 설정 방법 - 의존 객체를 자동으로 주입 : **@Autowired, @Resource**

---

**@Autowired** 는 타입으로,  
**@Resource** 는 이름으로 연결

### **@Resource**

- 어플리케이션에서 필요로 하는 자원을 자동 연결할 때 사용
- **@Resource**은 프로퍼티, setter 메서드에 적용
- 의존 객체를 주입할 때 주로 Name을 이용

## 6-2. JAVA를 이용한 DI 설정 방법 - 의존 객체를 자동으로 주입 : **@Autowired, @Resource**

---

### @Value

- 단순한 값(int, double, String, etc.)을 주입할 때 사용
- @Value("student")은 <property .. value="student" />

### @Qualifier

- @Qualifier는 @Autowired 어노테이션과 함께 사용
- @Autowired는 타입으로 찾아서 주입하므로, 다수의 동일 타입 Bean 객체 중 원하는 Bean을 특정하기 위해 @Qualifier를 함께 사용

# Spring - Registering beans in component classes

```
@Configuration  
@ComponentScan("com.example")  
public class MyConfig{  
    .....  
}
```

1 Registering a new bean definition. The method will not be CGLIB proxied like @Configuration classes.

```
package com.example;  
  
@Component  
public class TestBean {  
  
    @Bean  
    public AnotherBean anotherBean(){  
        return new AnotherBean();  
    }  
    .....  
}
```

Scan

2 We can even use different package here because this bean is not scanned but a direct instance is created by TestBean.

```
package com.example  
public class AnotherBean {  
    .....  
}
```

3 We can even use @Configuration instead of @Component because @Configuration is annotated with @Component itself. Any class annotated with @Configuration will be CGLIB proxied.

4 Calling this method directly will give a new instance every time. But if enclosing class is annotated with @Configuration, same instance will be returned every time because of CGLIB enhancement.

## 6-2. JAVA를 이용한 DI 설정 방법 - 의존 객체를 자동으로 주입 : **@Autowired, @Resource**

---

# Spring 3.0 추가 - **@Configuration & @Bean**

## **@Configuration**

```
public class ApplicationConfig {  
    @Bean  
    public Student student1() { .. . . .; }  
}
```

```
AnnotationConfigApplicationContext ctx =  
    new AnnotationConfigApplicationContext(ApplicationConfig.class);  
  
Student student1 = ctx.getBean("student1", Student.class);
```

## 6-2. JAVA를 이용한 DI 설정 방법

```
@Configuration  
public class ApplicationConfig {
```

**@Configuration**

'이 클래스는 스프링 설정에 사용되는 클래스입니다.' 라고 명시하는 어노테이션

```
@Bean  
public Student student10{
```

**@Bean** - 객체 생성, 메소드 위에 선언

```
ArrayList<String> hobbys = new ArrayList<String>();  
hobbys.add("수영");  
hobbys.add("요리");
```

```
Student student = new Student("홍길동", 20, hobbys);  
student.setHeight(180);  
student.setWeight(80);
```

생성자에 설정  
프로퍼티에 설정

```
return student;
```

```
}
```

project s06b

## 6-2. JAVA를 이용한 DI 설정 방법 - 의존 객체를 자동으로 주입 : **@Autowired, @Resource**

---

### <context:component-scan> 태그

- **@Component**를 통해 자동으로 Bean을 등록
- **@Autowired** 어노테이션을 클래스에 선언하여 사용
- 해당 클래스가 위치한 특정 패키지를 Scan하기 위한 설정

```
<context:component-scan base-package="com.wind.ctx" />
```

```
<context:component-scan base-package="com.wind.ctx2" />
```

```
<context:component-scan base-package="com.wind.ctx3, com.wind.ctx4"/>
```

```
AbstractApplicationContext ctx = new  
GenericXmlApplicationContext("classpath:appCTX.xml");  
Student student1 = ctx.getBean("student1", Student.class);
```

main()

com.wind.sp06 --- ApplicationConfig.java

```
@Configuration/@Component  
public class ApplicationConfig {  
    @Bean  
    public Student student1() {
```

java

appCtx.xml

```
<context:component-scan base-package="com.wind.sp06" />
```

메타설정  
XML

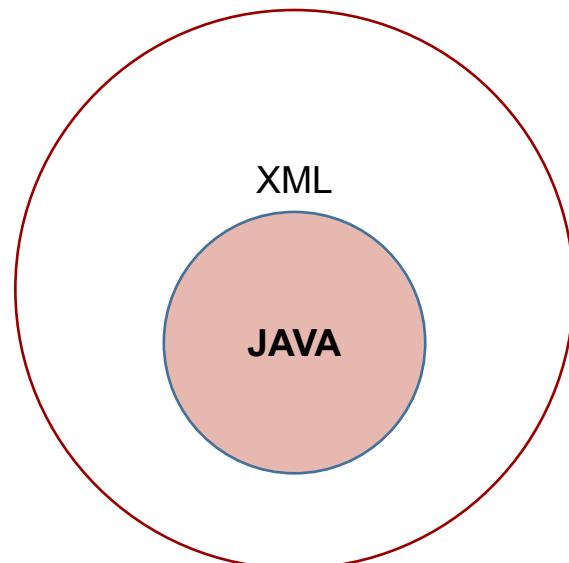
project  
s06b-di-noXml-annot-**component-scan-2**

### 6-3. XML과 JAVA를 같이 사용

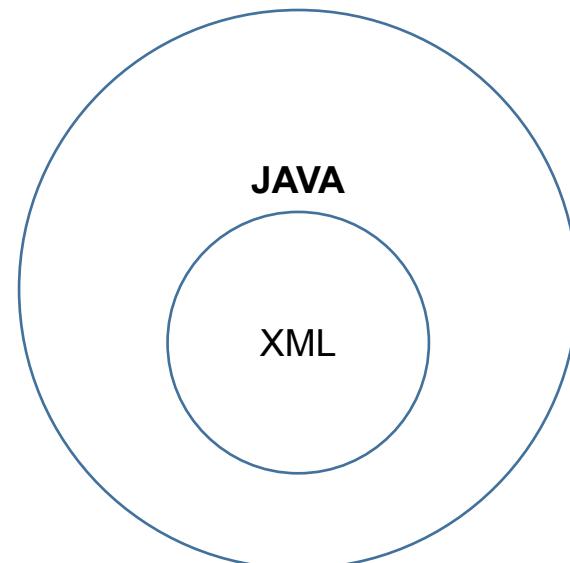
---

XML파일과 JAVA파일 같이 사용 하여 스프링 설정을 하고  
컨테이너를 만들고 컴포넌트들을 생성

Xml파일에 JAVA파일을  
포함시켜 사용 하는 방법



JAVA파일에 XML파일을  
포함시켜 사용 하는 방법

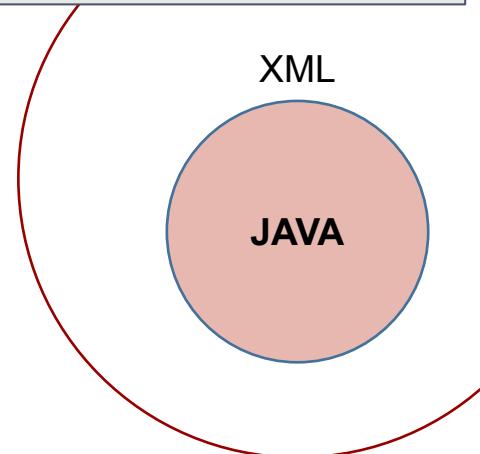


### 6-3. XML과 JAVA를 같이 사용

```
<<MAIN>>  
AbstractApplicationContext ctx =  
new GenericXmlApplicationContext(  
"classpath:applicationCTX.xml");
```

```
<<XML>>  
<context:annotation-config />  
<bean class="com...06.ApplicationConfig" />  
  
<bean .... >
```

```
<<JAVA>>  
@Configuration  
public class ApplicationConfig {  
    @Bean  
    public Student student1(){...}
```



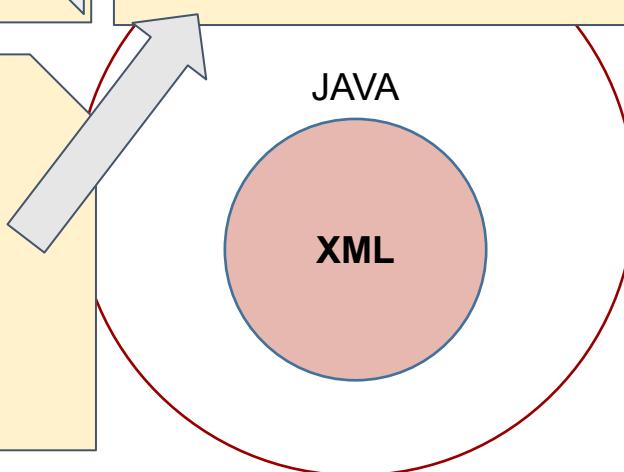
project s06c

## 6-4. XML과 JAVA를 같이 사용

```
<<MAIN>>  
AnnotationConfigApplicationContext  
ctx = new  
AnnotationConfigApplicationContext  
(ApplicationConfig.class);
```

```
<<JAVA>>  
@Configuration  
@ImportResource("classpath:appCTX.xml")  
public class ApplicationConfig {  
  
    @Bean  
    public Student student1(){
```

```
<<XML>>  
<bean id="##" class="###">  
...  
...
```



## 6-4. XML과 JAVA를 같이 사용

<<MAIN>>

```
AbstractApplicationContext ctx = new GenericXmlApplicationContext("classpath:appCTX.xml");
```

<<XML>>

```
<context:annotation-config />
<bean class="com.pack.ApplicationConfig" />

<bean id="##" name="##" >
```

<<JAVA>>

```
@Configuration
public class ApplicationConfig {
    @Bean
    public Student student1(){...}
```

<<XML>>

```
<bean id="##" class="##">
    ...

```

<<JAVA>>

```
@Configuration
@ImportResource("classpath:appCTX.xml")
public class AppConfig {
    @Bean
    public Student student1(){...}
```

<<MAIN>>

```
AnnotationConfigApplicationContext ctx =
    new AnnotationConfigApplicationContext (AppConfig.class);
```

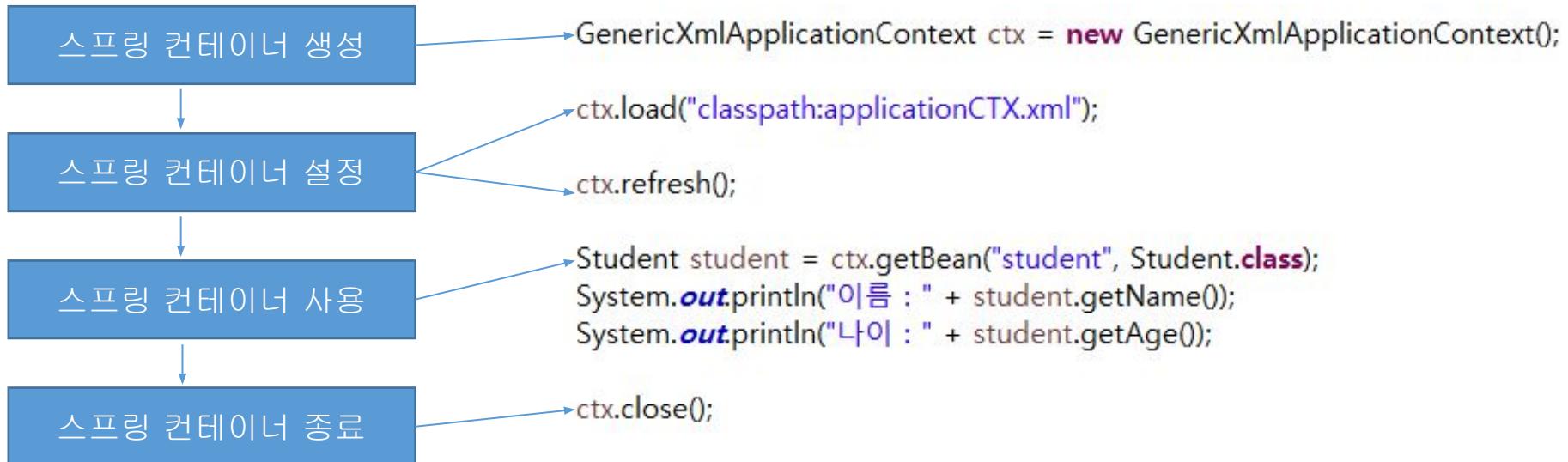
project s06d

## 7. 생명 주기(life cycle) 와 범위 (scope)

- 스프링 컨테이너 생명 주기
  - 스프링 빈 생명 주기
  - 스프링 빈 범위

## 7-1. 스프링 컨테이너 생명 주기

---



project s07a

## 7-2. 스프링 빈 생명 주기

### 1) implements InitializingBean, DisposableBean

```
@Override  
public void afterPropertiesSet() throws Exception {  
    // TODO Auto-generated method stub  
    System.out.println("afterPropertiesSet()");  
}
```

```
@Override  
public void destroy() throws Exception {  
    // TODO Auto-generated method stub  
    System.out.println("destroy()");  
}
```

```
GenericXmlApplicationContext ctx = new GenericXmlApplicationContext();  
  
ctx.load("classpath:applicationCTX.xml");  
  
ctx.refresh();  
  
ctx.close();
```

빈 초기화  
과정에서 호출

빈 소멸  
과정에서  
생성

#### [참고]

- ctx.close() - 컨테이너가 소멸 하는 단계
- 컨테이너가 소멸 하면, 빈은 자동 소멸
- 빈만 소멸하려면, student.destroy() API를 이용

## 7-2. 스프링 빈 생명 주기

### 2) @PostConstruct, @PreDestroy

```
@PostConstruct  
public void initMethod0 {  
    System.out.println("initMethod0");  
}
```

빈 초기화  
과정에서 호출

```
@PreDestroy  
public void destroyMethod0 {  
    System.out.println("destroyMethod0");  
}
```

빈 소멸 과정에서  
생성

```
GenericXmlApplicationContext ctx = new GenericXmlApplicationContext();  
  
ctx.load("classpath:applicationCTX.xml");  
  
ctx.refresh();  
  
ctx.close();
```

project s07b

### 7-3. 스프링 빈 범위(scope)

스프링컨테이너 생성 -> 스프링빈 생성 -> scope

Scope , 범위 : 객체가 영향을 미치는 범위

```
<bean id="student" class="com.javalec.ex.Student" scope="singleton">
    <constructor-arg value="홍길순"></constructor-arg>
    <constructor-arg value="30"></constructor-arg>
</bean>
```

```
Student student1 = ctx.getBean("student", Student.class);
System.out.println("이름 : " + student1.getName());
System.out.println("나이 : " + student1.getAge());

System.out.println("=====");
```

```
Student student2 = ctx.getBean("student", Student.class);
student2.setName("홍길자");
student2.setAge(100);
```

```
System.out.println("이름 : " + student1.getName());
System.out.println("나이 : " + student1.getAge());

System.out.println("=====");
```

```
if(student1.equals(student2)) {
    System.out.println("student1 == student2");
} else {
    System.out.println("student1 != student2");
}
```

```
이름 : 홍길순
나이 : 30
=====
이름 : 홍길자
나이 : 100
=====
```

```
student1 == student2
```

### 7-3. 스프링 빈 범위(scope)

singleton

(기본) 하나의 Bean 정의로 하나의 빈 객체만 생성

prototype

Bean 정의로 다수의 객체 생성

request

Bean 정의가 하나의 HTTP 요청의 리사이클 범위

session

Bean 정의가 하나의 HTTP session 의 리사이클 범위

global session

Bean 정의가 전역적인 HTTP session 의 리사이클 범위



web 기반 스프링 ApplicationContext 구현에서만 사용 가능

project s07c

project s07c2

# 8. 외부 파일을 이용한 설정

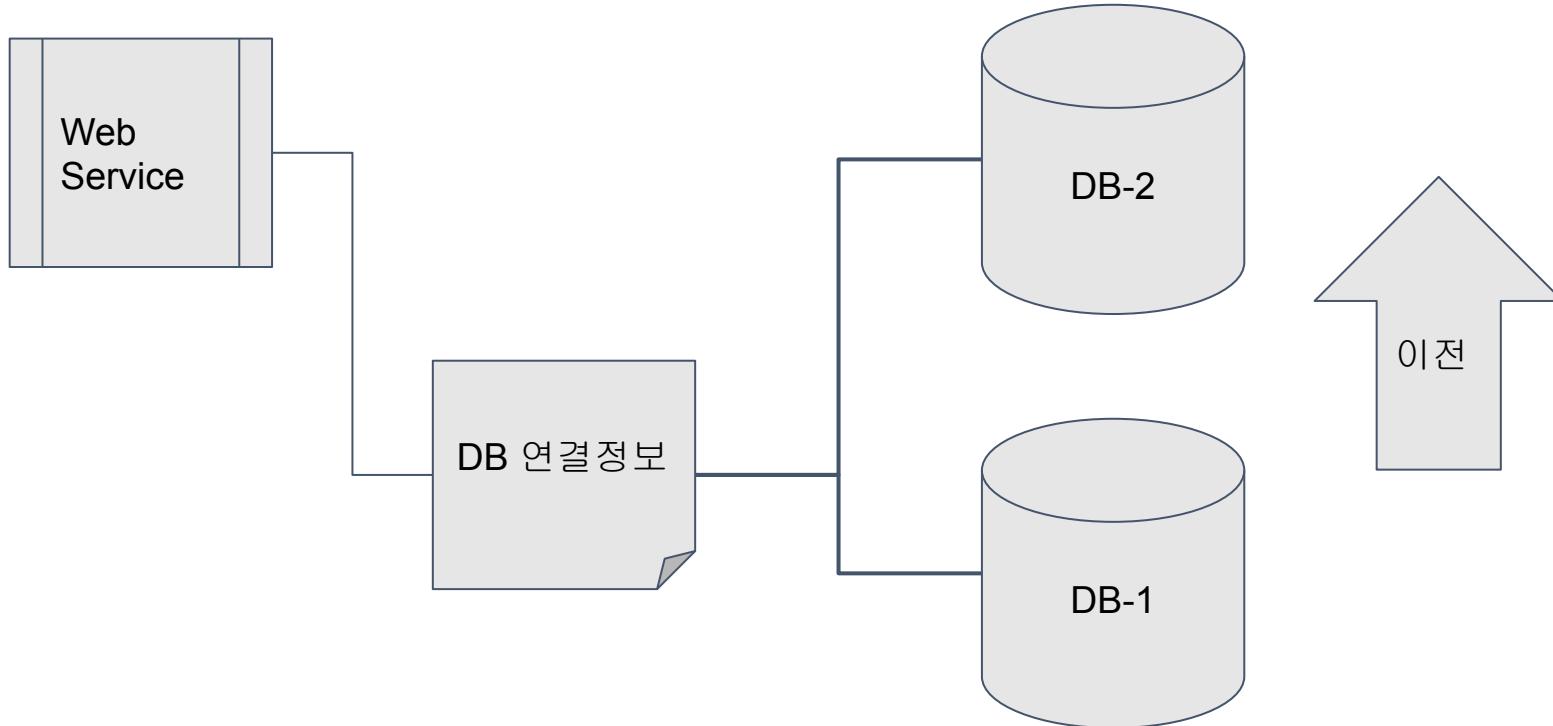
-Environment 객체

-프로퍼티 파일을 이용한 설정

-프로파일(profile) 속성을 이용한 설정

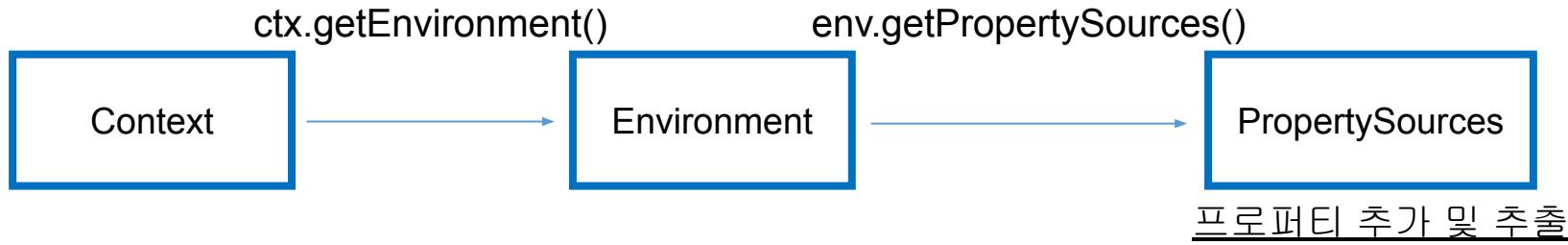
## 8-1. Environment 객체

---



## 8-1. Environment 객체

---

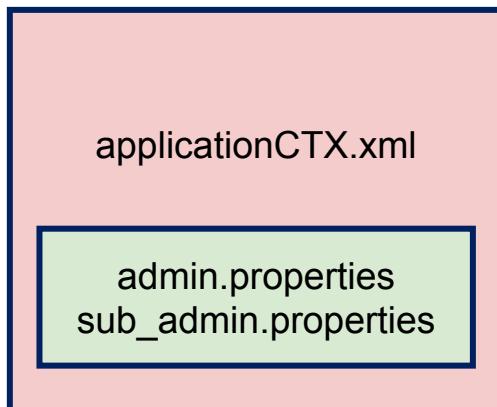


추가 : `propertySources.addLast()`  
추출 : `env.getProperty()`

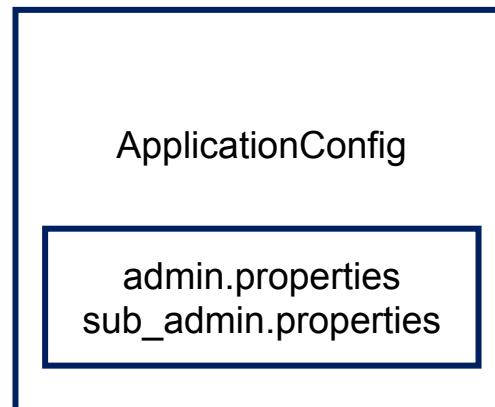
## 8-2. 프로퍼티 파일을 이용한 설정

Environment 객체를 사용하지 않고 프로퍼티 파일을 직접 이용하여 스프링 빈을 설정

스프링 설정 XML파일에  
프로퍼티 파일을 명시 합니다.

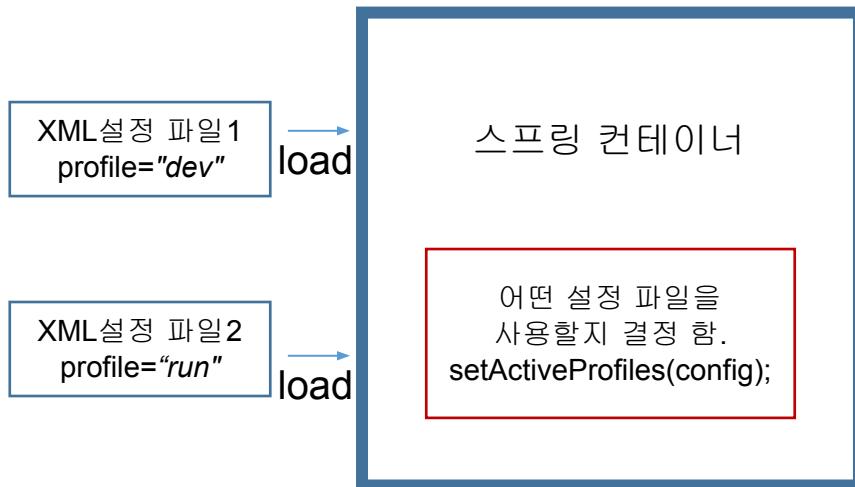


스프링 설정 JAVA파일에  
프로퍼티 파일을 명시 합니다.

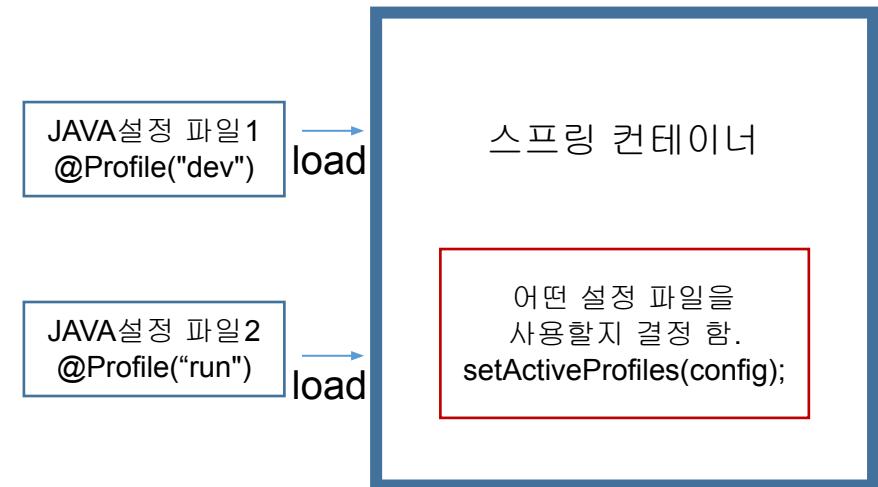


### 8-3. 프로파일(profile) 속성을 이용한 설정

동일한 스프링 빈을 여러 개 만들어 놓고 상황(환경)에 따라서 적절한 스프링 빈을 사용  
profile 속성을 사용



Xml 설정 파일을 이용하는 경우



JAVA 설정 파일을 이용하는 경우

### 8-3. 프로파일(profile) 속성을 이용한 설정

`{} (프로퍼티 치환자) 값의 치환 : <context:property-placeholder> 태그로  
PropertyPlaceHolderConfigurer Bean이 담당`

```
<database.properties>

db.driverClass=com.mysql.jdbc.Driver
db.url=jdbc:mysql://local...
db.username=studyid
db.password=stu<databaseBeans.xml>

<context:property-placeholder
location="classpath:config/database.properties" />

<bean id="dataSource" class="org.springframework.jdbc...">
    <property name="driverClass" value="${db.driverClass}" />
    <property name="url" value="${db.url}" />
    ...
</bean>
```

# 9. AOP(Aspect Oriented Programming)-I

-AOP란?

-XML기반의 AOP구현

## 9-1. AOP란?

---

- 함수, 클래스 - 공통적인 기능이 발생할 때 처리하는 방식
- 상속 - 클래스와 메소드를 이용
- JAVA 다중 상속 불가
- 다양한 모듈에 상속기법을 통한 공통 기능 부여 한계
- 기능 구현부분에 핵심 기능 코드와 공통 기능 코드가 섞여 있어 효율성이 저하

## AOP :

핵심 기능과 공통 기능을 **분리**,

공통 기능을 필요로 하는 핵심 기능들에서 사용

## 9-1. AOP란?

---

아침에 밥을 짓는 상황을 AOP로

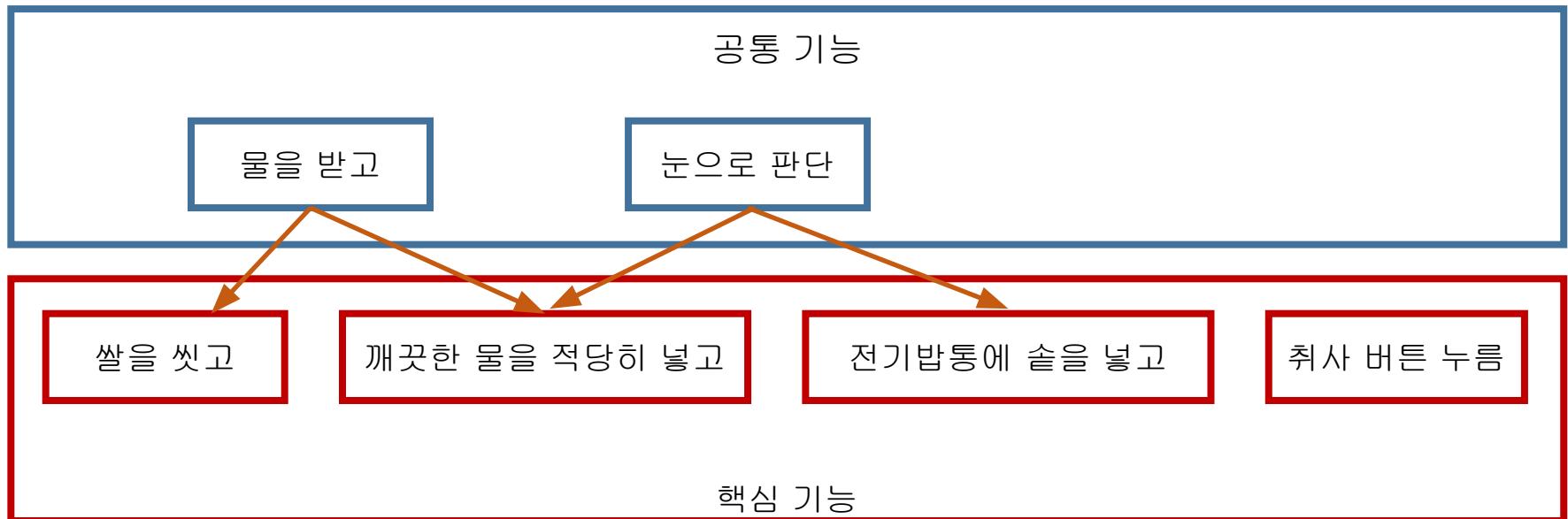
### 핵심 기능

- 쌀을 씻고,
- 깨끗한 물을 적당히 넣고,
- 전기밥통에 속을 넣고,
- 취사 버튼을 누름

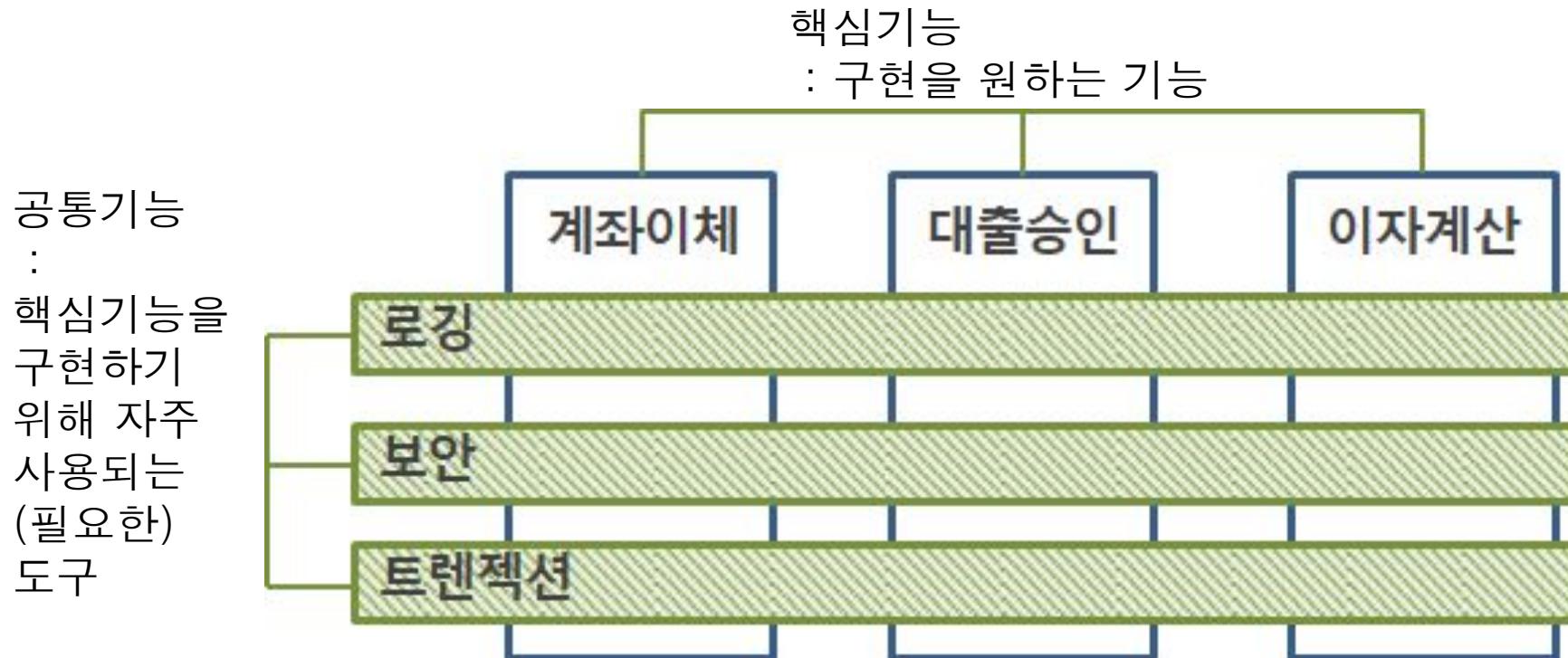
### 공통 기능

- 수도꼭지를 열어 물을 받고, → (밥할때, 찌게 끓일때, 감자 삶을때, 요리할 때..)
- 쌀이 깨끗이 씻겼는지 물이 적당한지 눈으로 판단 → (모니터링, 로그기록)

## 9-1. AOP란?



## 9-1. AOP란?



## 9-1. AOP란?

- 업무(Biz) 로직을 포함하는 기능을 **핵심 기능**(Core Concerns)
- 핵심기능을 도와주는 부가적인 기능(로깅, 보안 등)을 **부가기능** (Cross-cutting Concerns) 이라고 부른다.
- 객체지향의 기본 원칙을 적용하여도 **핵심기능에서 부가기능을 분리해서 모듈화하는 것은 매우 어렵다.**

```
#define DEBUG 1
void main(){
    ..
    ..
#define DEBUG
    print(x);
#endif
    a();
    ..
    ..
#define DEBUG
    print(x);
#endif
}

a(){  
    ..  
    ..  
    #ifdef DEBUG  
        print(x);  
    #endif  
}  
}
```

## 9-1. AOP란?

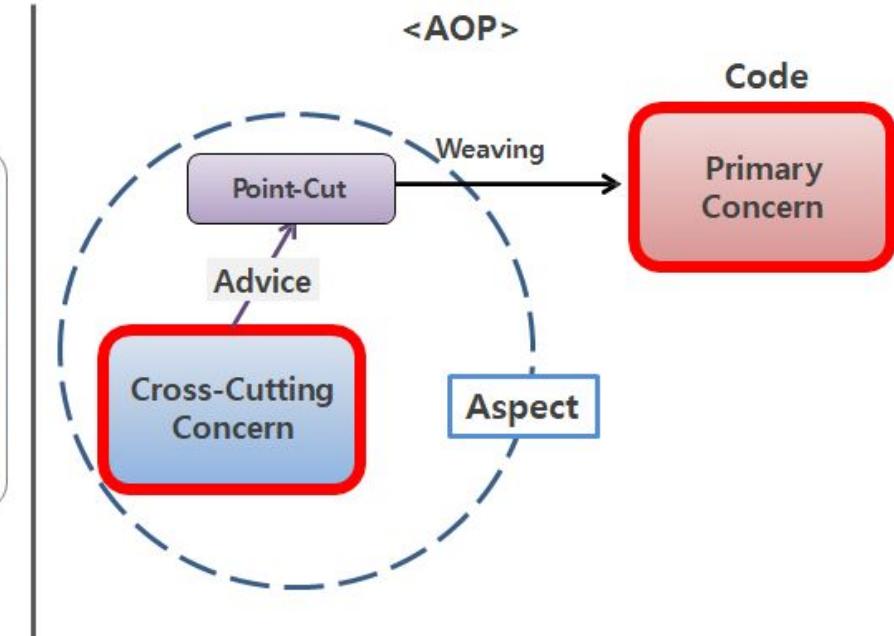
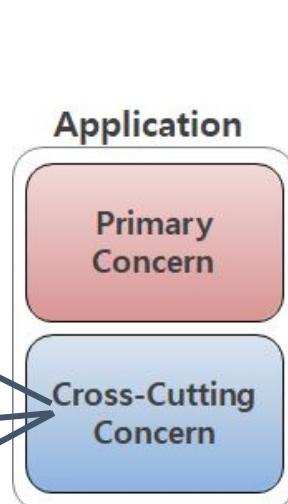
---

### 용어

- Aspect      어드바이스(Advice) + 포인트컷(PointCut)
  - AOP 기본모듈
  - 싱글톤형태 객체로 존재
- Target      핵심기능을 담고 있는 모듈, 부가기능을 부여할 대상
- Advice      Aspect의 기능 자체, 타겟에 제공할 부가기능을 담고 있는 모듈
- Joinpoint    핵심기능, Advice를 적용해야 되는 부분, 위치 (모든 함수&메서드)
- Pointcut     Joinpoint의 부분으로 실제로 Advice가 적용될 부분에 대한 정규표현식
  - execution 으로 시작되고, Signature 비교하는 방법을 주로 사용
- Weaving     Advice를 핵심 기능에 적용 하는 행위

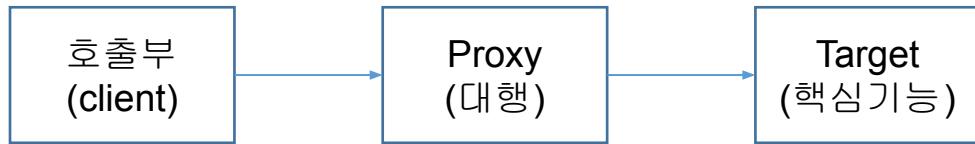
## 9-1. AOP란?

```
#define DEBUG 1
void main(){
    ...
    ...
    #ifdef DEBUG
        print(x);
    #endif
    a();
    ...
    ...
    #ifdef DEBUG
        print(x);
    #endif
    ...
    ...
}
```



## 9-1. AOP란?

스프링에서 AOP 구현 : proxy \*\* 를 이용



스프링에서 AOP 구현 방식

- XML 스키마 기반의 AOP구현
- @Aspect 어노테이션 기반의 AOP 구현

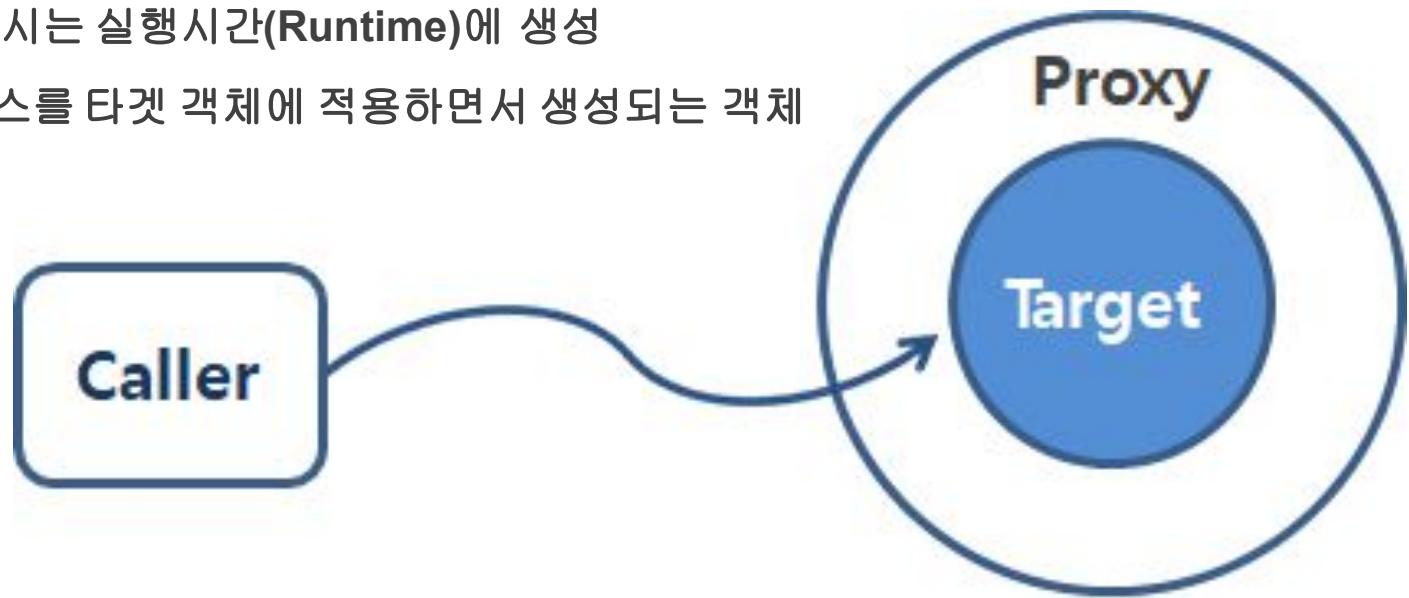
# Spring AOP의 특징

## (1) 프록시(Proxy) 기반

타겟(target) 객체에 대한 프록시를 만들어 제공

타겟을 감싸는 프록시는 실행시간(Runtime)에 생성

프록시는 어드바이스를 타겟 객체에 적용하면서 생성되는 객체



## (2) 프록시(Proxy)가 호출을 가로챈다(Intercept).

프록시는 타겟 객체에 대한 호출을 가로챈 다음 어드바이스의 부가기능 로직을 수행하고 난 후에 타겟의 핵심기능 로직을 호출한다. (전처리 어드바이스)

또는 타겟의 핵심기능 로직 메서드를 호출한 후에 부가기능(어드바이스)을 수행하는 경우도 있다.(후처리 어드바이스)



### (3) Spring AOP는 메서드 조인 포인트만 지원한다.

- Spring은 동적 프록시를 기반으로 AOP를 구현하므로 메서드 조인 포인트만 지원한다. 즉, 핵심기능(타겟)의 메서드가 호출되는 런타임 시점에만 부가기능(어드바이스)을 적용할 수 있다.
- 반면에 **AspectJ** 같은 고급 AOP 프레임워크를 사용하면 객체의 생성, 필드값의 조회와 조작, static 메서드 호출 및 초기화 등의 다양한 작업에 부가기능을 적용할 수 있다.

## 9-2. XML 기반의 AOP구현

### 작업 순서

- 1) 의존 설정(pom.xml)
- 2) 공통 기능의 클래스 제작 – Advice 역할 클래스
- 3) XML설정 파일에 Aspect 설정

<http://mvnrepository.com/>

#### 의존 설정

```
<!-- AOP -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.7.4</version>
</dependency>
```

## 9-2. XML 기반의 AOP구현

### 작업 순서

1) 의존 설정(pom.xml)

2) 공통 기능의 클래스 제작 – Advice 역할 클래스

3) XML설정 파일에 Aspect 설정

공통 기능 클래스

```
public class LogAop {  
  
    public Object loggerAop(ProceedingJoinPoint joinpoint) throws Throwable {  
        String signatureStr = joinpoint.getSignature().toShortString();  
        System.out.println(signatureStr + " is start.");  
        long st = System.currentTimeMillis();  
  
        try {  
            Object obj = joinpoint.proceed();  
            return obj;  
        } finally {  
            long et = System.currentTimeMillis();  
            System.out.println(signatureStr + " is finished.");  
            System.out.println(signatureStr + " 경과시간 : " + (et - st));  
        }  
    }  
}
```

## 9-2. XML 기반의 AOP구현

### 작업 순서

- 1) 의존 설정(pom.xml)
- 2) 공통 기능의 클래스 제작 – Advice 역할 클래스
- 3) XML설정 파일에 Aspect 설정

### XML파일 설정

```
<bean id="logAop" class="com.javalec.ex.LogAop"/>

<aop:config>
    <aop:aspect id="logger" ref="logAop">
        <aop:pointcut id="publicM" expression="within(com.javalec.ex.*)" />
        <aop:around pointcut-ref="publicM" method="loggerAop"/>
    </aop:aspect>
</aop:config>
```

[pom.xml] - mvnrepository.com에서 검색하여 설치

1. aspectj runtime 1.7.4 설치
2. aspectj weaver 1.7.4 설치
3. spring aop 3.2.17 설치[삭제] => 충돌일어남
4. spring 버전을 5.0.4. 와 3.2.3.[기본버전]으로 각각 해볼것

### Advice 종류

<aop:before> : 메소드 실행 전에 advice 실행

<aop:after-returning> : 정상적으로 메소드 실행 후에 advice 실행

<aop:after-throwing> : 메소드 실행 중 exception 발생 시 advice 실행

<aop:after> : 메소드 실행 중 exception 이 발생하여도 advice 실행

**<aop:around>** : 메서드 실행 전/후 및 exception 발생 시 advice 실행

## 10강. AOP(Aspect Oriented Programming)-II

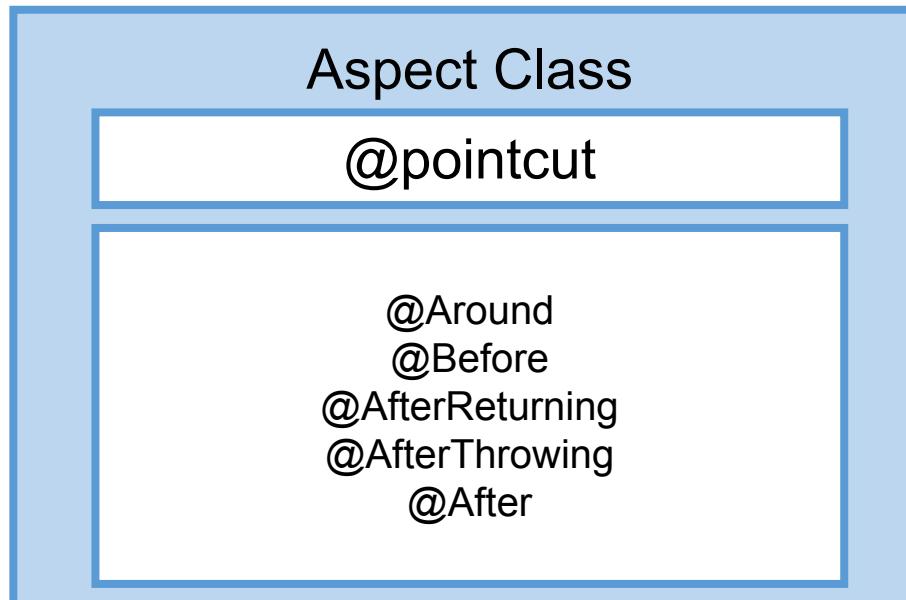
- @Aspect를 이용한 AOP 구현
- AspectJ Pointcut 표현식

## 10-1. @Aspect를 이용한 AOP구현

---

### 작업 순서

- 1) 의존 설정(pom.xml)
- 2) @Aspect 어노테이션을 이용한 Aspect클래스 제작
- 3) XML파일에 <aop:aspectj-autoproxy /> 설정



## 10-2. AspectJ Pointcut 표현식

Pointcut을 지정할 때 사용하는 표현식으로 AspectJ 문법을 사용 합니다.

\* : 모든

. : 현재

.. : 0개 이상

### Execution

```
// @Pointcut("execution(public void get*(..))") // public void인 모든 get메소드  
// @Pointcut("execution(* com.javalec.ex.*.*())") // com.javalec.ex 패키지에 파라미터가 없는 모든 메소드  
// @Pointcut("execution(* com.javalec.ex..*.*())") // com.javalec.ex 패키지 & com.javalec.ex 하위 패키지에 파라미터가 없는 모든 메소드  
// @Pointcut("execution(* com.javalec.ex.Worker.*())") // com.javalec.ex.Worker 안의 모든 메소드
```

### within

```
// @Pointcut("within(com.javalec.ex.*)") //com.javalec.ex 패키지 안에 있는 모든 메소드  
// @Pointcut("within(com.javalec.ex..*)") //com.javalec.ex 패키지 및 하위 패키지 안에 있는 모든 메소드  
// @Pointcut("within(com.javalec.ex.Worker)") //com.javalec.ex.Worker 모든 메소드
```

### bean

```
// @Pointcut("bean(student)") //student 빈에만 적용  
@Pointcut("bean(*ker)") //~ker로 끝나는 빈에만 적용
```

## 10-2. AspectJ Pointcut 표현식

---

\* 모든, .. 0개 이상,

```
// @Pointcut("execution(public void get(..))")      // public void인 모든 get메소드
// @Pointcut("execution(* com.wind.s10.*.*())")    // com.wind.s10 패키지에 파라미터가 없는 모든 메소드
// @Pointcut("execution(* com.wind.s10..*.*())")    // com.wind.s10 패키지 & com.wind.sp10 하위 패키지에
파라미터가 없는 모든 메소드
// @Pointcut("execution(* com.wind.s10.Worker.*())") // com.wind.s10.Worker 안의 모든 메소드

// @Pointcut("within(com.wind.s10.*)")   //com.wind.s10 패키지 안에 있는 모든 메소드
// @Pointcut("within(com.wind.s10..*)") //com.wind.s10 패키지 및 하위 패키지 안에 있는 모든 메소드
@Pointcut("within(com.wind.s10.Worker)") //com.wind.s10.Worker 모든 메소드

// @Pointcut("bean(student)")          //student 빈에만 적용
// @Pointcut("bean(*ker)")           //~ker로 끝나는 빈에만 적용
```

## 10-2. AspectJ Pointcut 표현식

---

```
@Pointcut("execution(public void get*(..))")  
@Pointcut("execution(* com.wind.s10.*.*())")  
@Pointcut("execution(* com.wind.s10..*.*())")  
@Pointcut("execution(* com.wind.s10.Worker.*())")
```

```
@Pointcut("within(com.wind.s10.*)")  
@Pointcut("within(com.wind.s10..*)")  
@Pointcut("within(com.wind.s10.Worker)")
```

```
@Pointcut("bean(student)")  
@Pointcut("bean(*ker)")
```

\* : 모든  
.. : 0개 이상

# 11. 스프링 MVC 기초

- 스프링 MVC 개요
- 스프링 MVC 구조 살펴보기
- resources 폴더

# MVC(Model–View–Controller) 패턴의 개념

모델-뷰-컨트롤러(Model–View–Controller, MVC)

- 소프트웨어 공학에서 사용되는 **아키텍쳐 패턴**
- Business logic과 Presentation logic을 분리

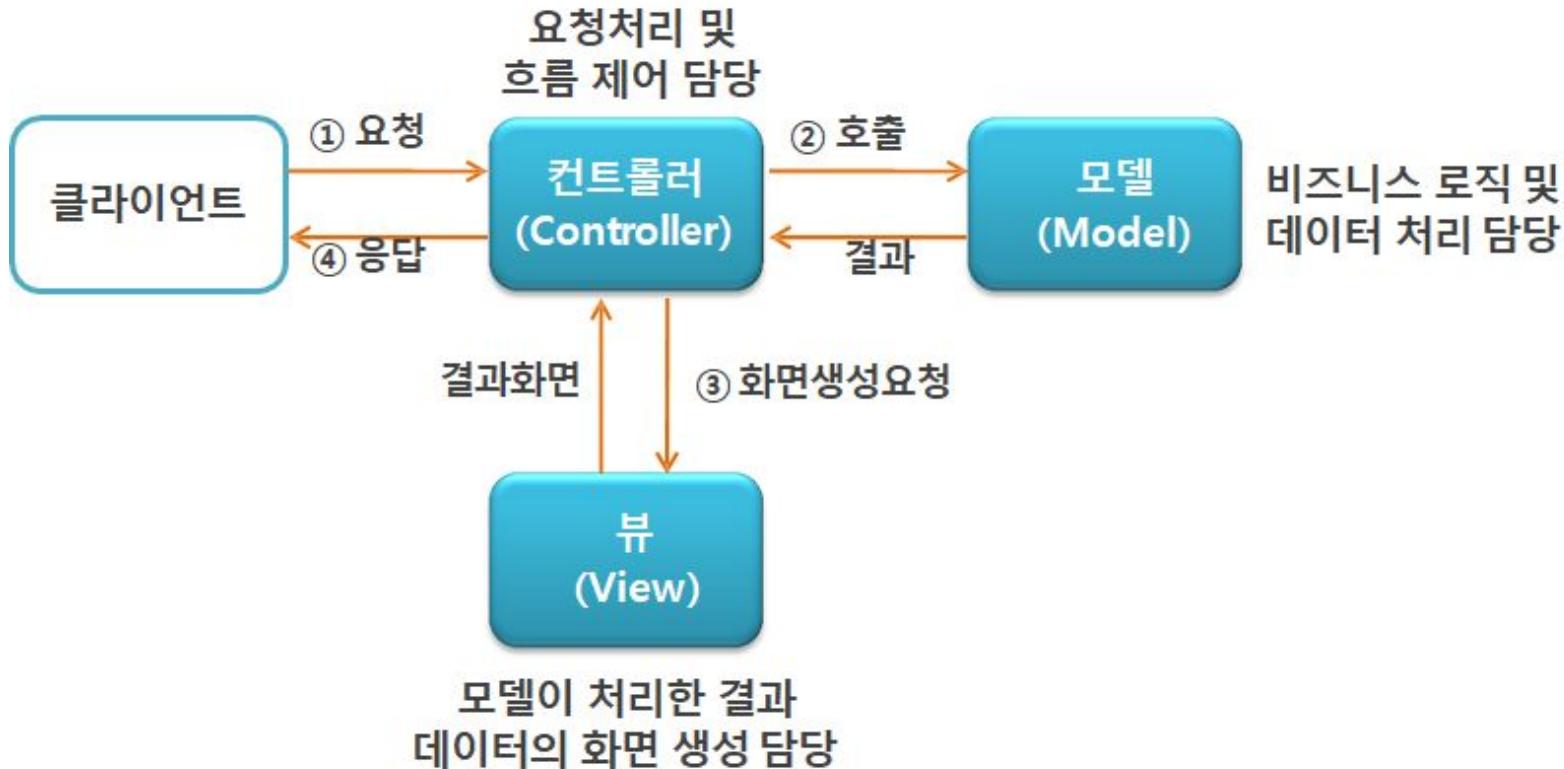
MVC 패턴을 사용하면, 사용자 인터페이스로부터 비지니스 로직을 분리하여 애플리케이션의 시각적 요소나 그 이면에서 실행되는 비지니스 로직을 쉽게 고칠 수 있는 애플리케이션을 만들 수 있음

-**Model** : 애플리케이션의 정보(데이터, Business Logic 포함)

-**View** : 사용자에게 제공할 화면(Presentation Logic)

-**Controller** : Model과 View 사이의 상호 작용을 관리

# MVC(Model–View–Controller) 패턴의 개념



## 11-1. 스프링 MVC 개요

---

### 각각의 MVC 컴포넌트의 역할

#### 모델(Model) 컴포넌트

- 데이터 저장소(ex : 데이터베이스 등)와 연동하여 사용자가 입력한 데이터나 사용자에게 출력할 데이터를 다루는 일
- 여러 개의 데이터 변경 작업(추가, 변경, 삭제)을 하나의 작업으로 묶는 트랜잭션을 다루는 일
- DAO 클래스, Service 클래스에 해당

## 11-1. 스프링 MVC 개요

---

### 각각의 MVC 컴포넌트의 역할

#### 뷰(View) 컴포넌트

- 모델이 처리한 데이터나 그 작업 결과를 가지고 사용자에게 출력할 화면을 만드는 일을 함
- 생성된 화면은 웹 브라우저가 출력하고, 뷰 컴포넌트는 HTML과 CSS, Java Script를 사용하여 웹 브라우저가 출력할 UI를 만듦
- Html과 JSP를 사용하여 작성

## 11-1. 스프링 MVC 개요

---

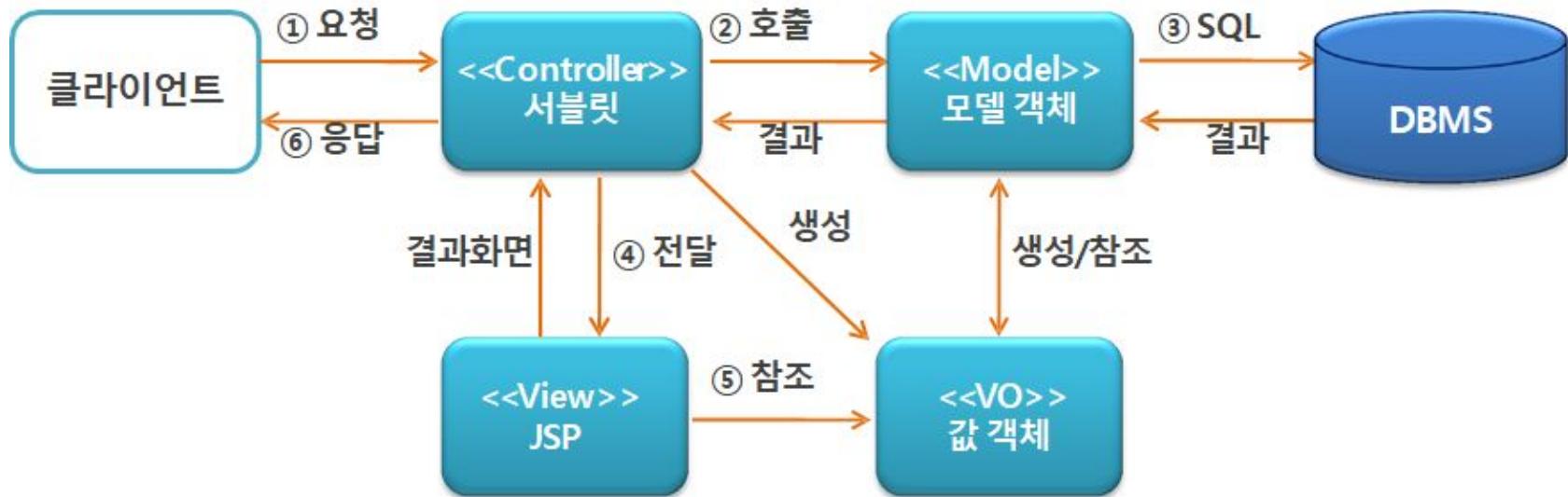
### 각각의 MVC 컴포넌트의 역할

#### 컨트롤러(Controller) 컴포넌트

- 클라이언트의 요청을 받으며 그 요청에 대해 실제 업무를 수행하는 모델을 호출
- 클라이언트가 보낸 데이터를 적절히 가공, 모델 호출시 활용
- 모델이 업무 수행을 완료하면, 그 결과로 화면 생성하여 뷰에게 전달  
(클라이언트 요청에 대해 모델과 뷰를 결정하여 전달)
- Servlet과 JSP를 사용

## 모델2 아키텍쳐 개념

- 모델 1 아키텍쳐 : Controller 역할을 JSP가 담당함
- 모델 2 아키텍쳐 : Controller 역할을 Servlet이 담당함



## 모델2 아키텍쳐 호출순서



① 웹 브라우저가 웹 애플리케이션 실행을 요청하면, 웹 서버가 그 요청을 받아서 서블릿 컨테이너(ex : 톰캣서버)에 넘겨준다.  
서블릿 컨테이너는 URL을 확인하여 그 요청을 처리할 서블릿을 찾아서 실행한다.

## 모델2 아키텍쳐 호출순서



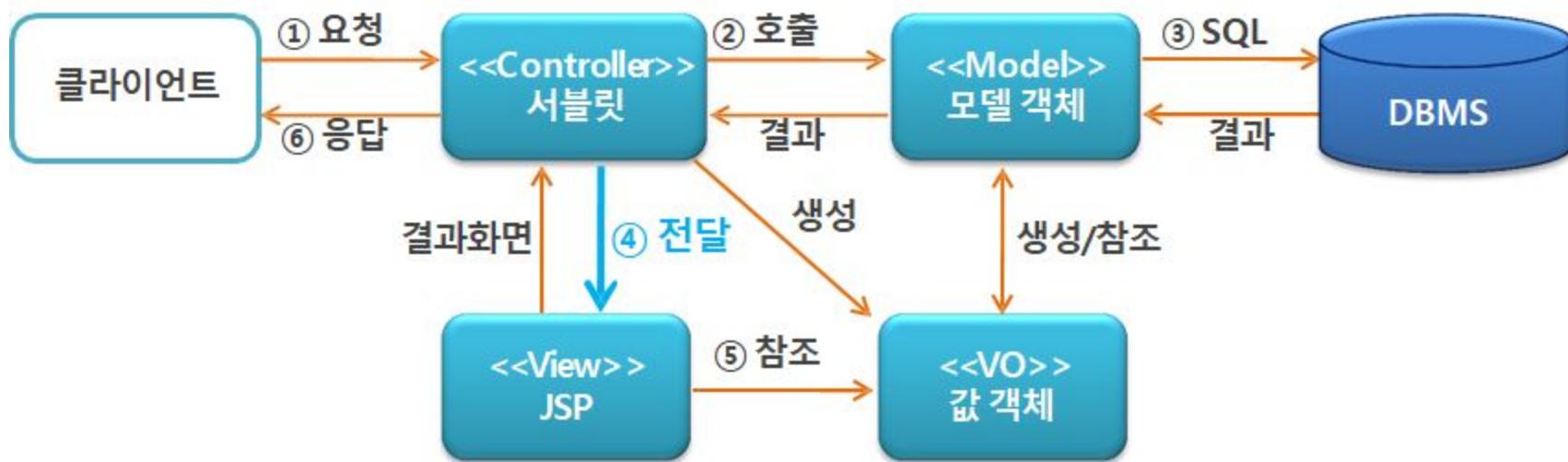
② 서블릿은 처리할 모델 자바 객체의 메서드를 호출한다.  
만약 웹 브라우저가 보낸 데이터를 저장하거나 변경해야 한다면  
그 데이터를 가공하여 VO 객체(Value Object)를 생성하고, 모델  
객체의 메서드를 호출할 때 인자 값으로 넘긴다.  
모델 객체는 엔터프라이즈 자바빈(EJB; Enterprise JavaBeans),  
일반 자바 객체(POJO로 된 Service, DAO object) 모두 가능하다.

## 모델2 아키텍쳐 호출순서



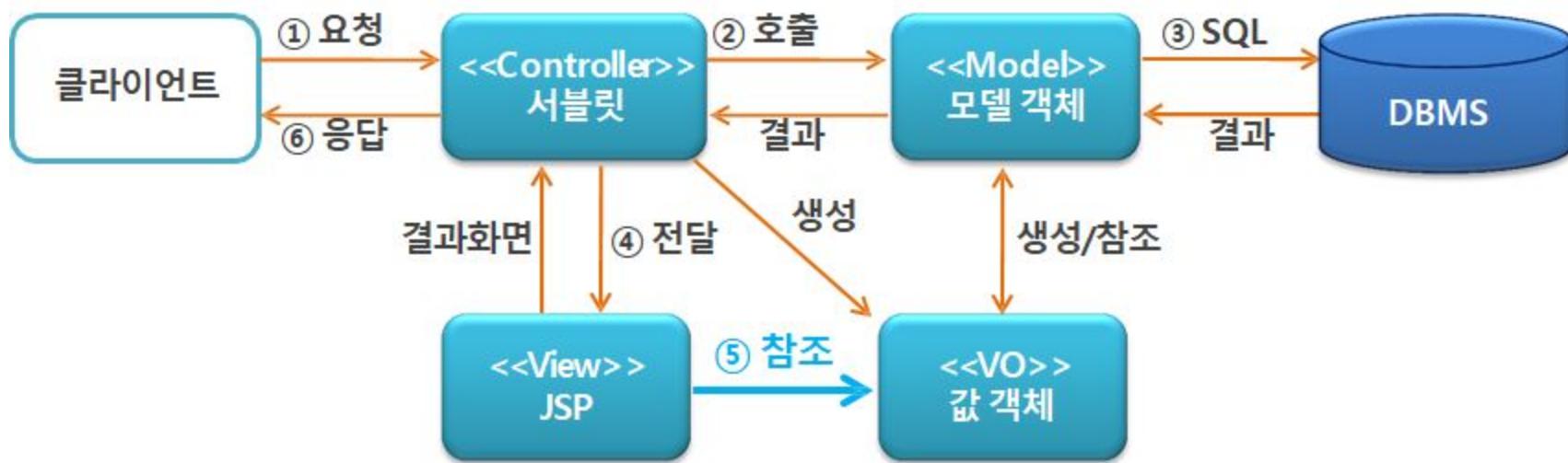
③ 모델 객체는 JDBC를 사용하여 매개변수로 넘어온 값 객체를 데이터베이스에 저장하거나, 데이터베이스로부터 질의 결과를 가져와서 VO 객체로 만들어 반환한다.

## 모델2 아키텍쳐 호출순서



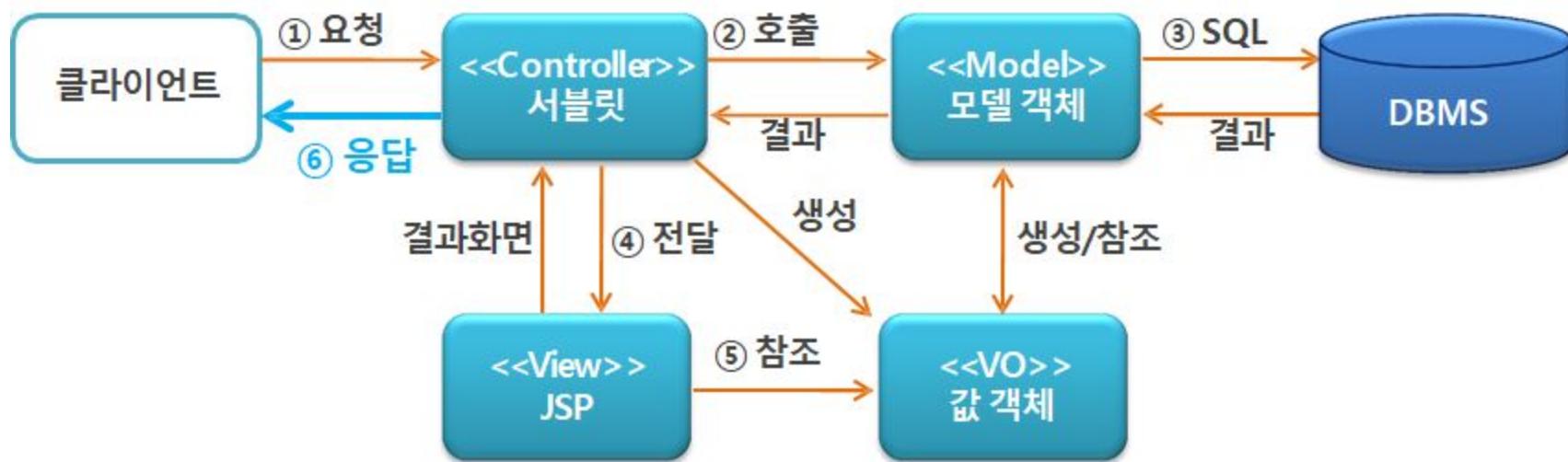
④ 서블릿은 모델 객체로부터 반환 받은 값을 JSP에 전달한다.

## 모델2 아키텍쳐 호출순서



⑤ JSP는 서블릿으로 부터 전달받은 값 객체를 참조하여 웹 브라우저가 출력할 결과 화면을 만들고, 웹 브라우저에 출력함으로써 요청 처리를 완료한다.

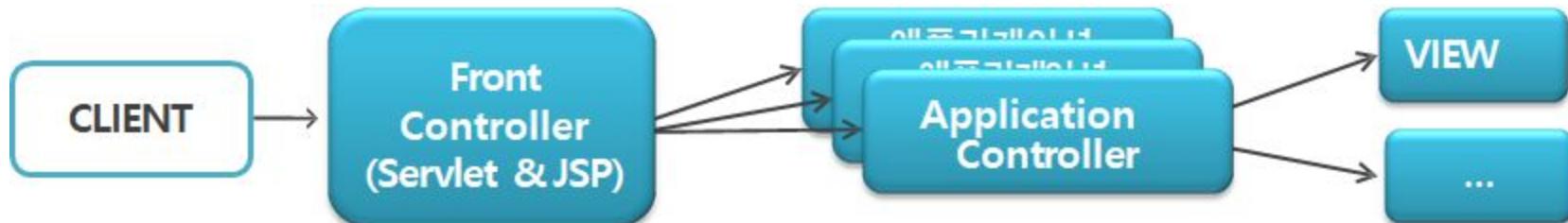
## 모델2 아키텍쳐 호출순서



⑥ 웹 브라우저는 서버로부터 받은 응답 내용을 화면에 출력한다.

# Front Controller 패턴 아키텍쳐

## Front Controller 프로세스



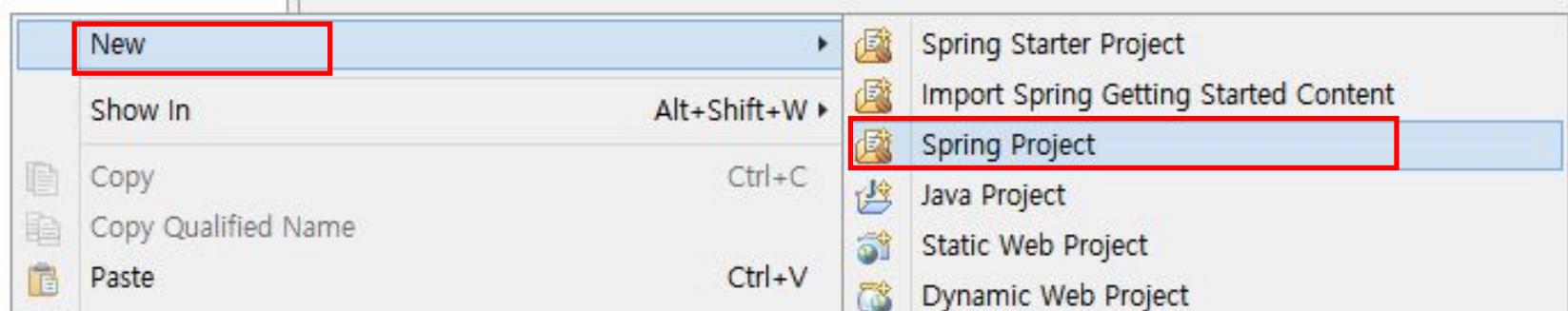
- Front Controller는 클라이언트가 보낸 요청을 받아서 공통적인 작업을 먼저 수행
- Front Controller는 적절한 세부 Controller에게 작업을 위임
- 각각의 애플리케이션 Controller는 클라이언트에게 보낼 뷰를 선택해서 결과를 생성
- Front Controller 패턴은 인증, 권한 체크처럼 모든 요청에 대하여 공통적으로 처리할 로직이 있을 경우 클라이언트의 요청을 중앙에 집중하여 관리하기 용이



## 11-2. 스프링 MVC 구조 살펴보기

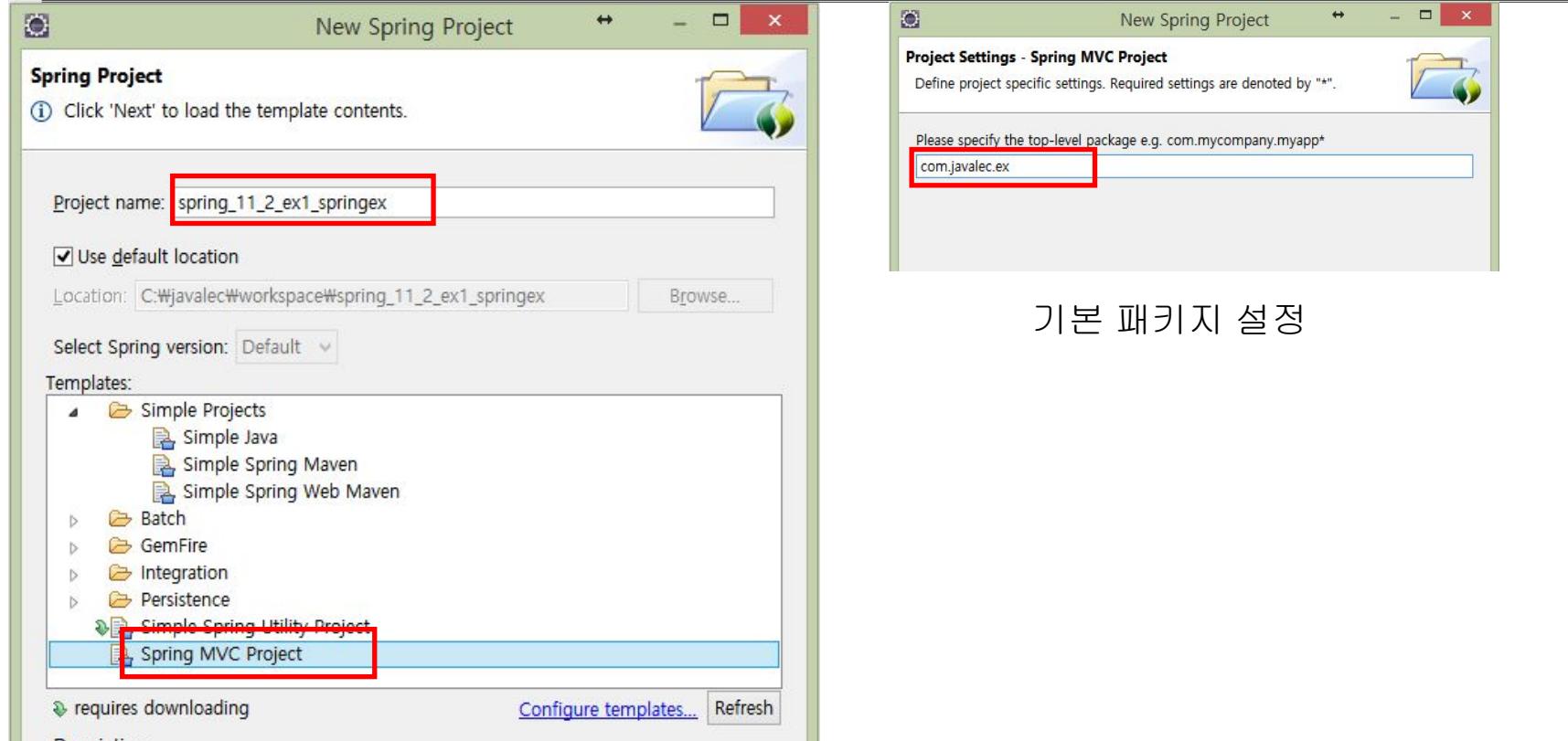
MVC의 전체적인 구조를 이해하자 → 매우 중요 !!!

우선 스프링 MVC 프로젝트를 이클립스에서 만들어 보고,  
전체적인 구조를 익힌다.



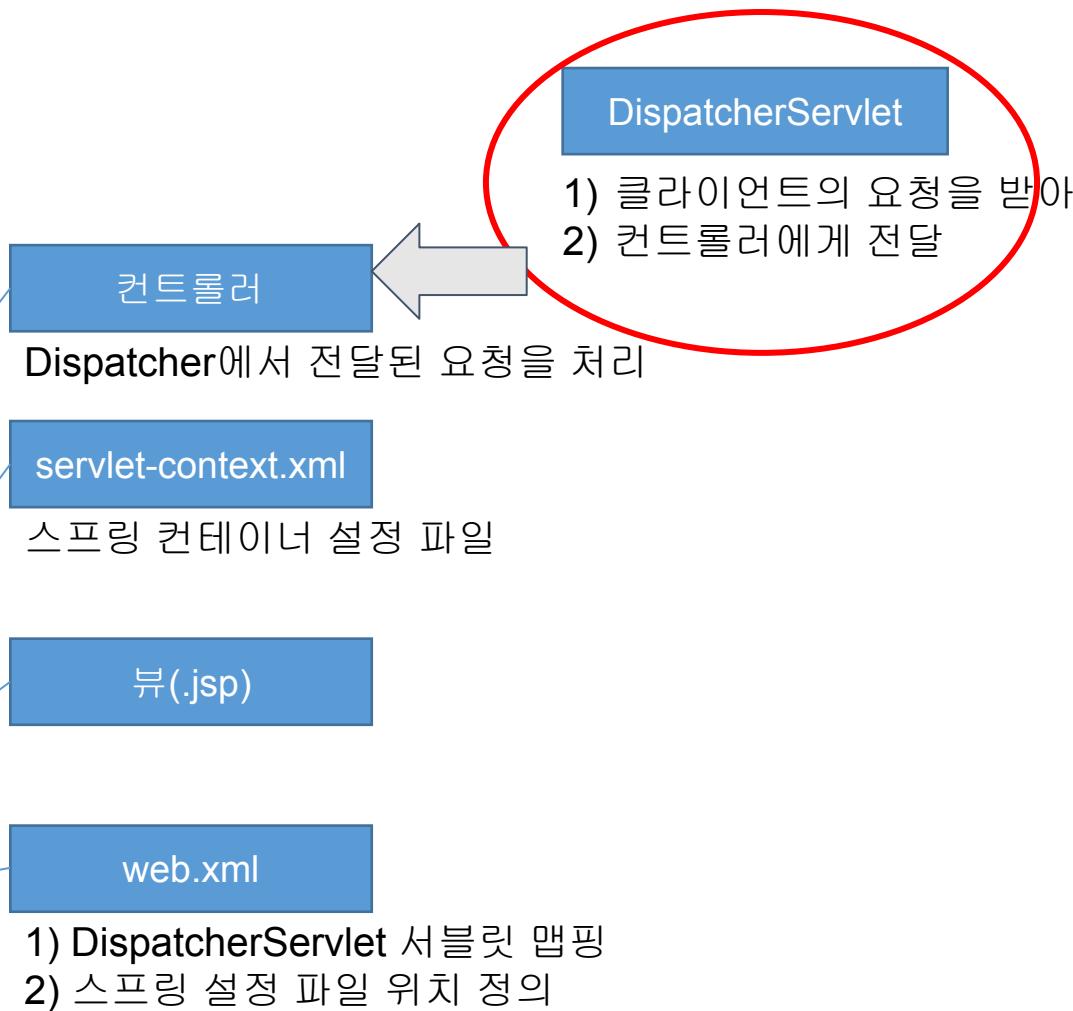
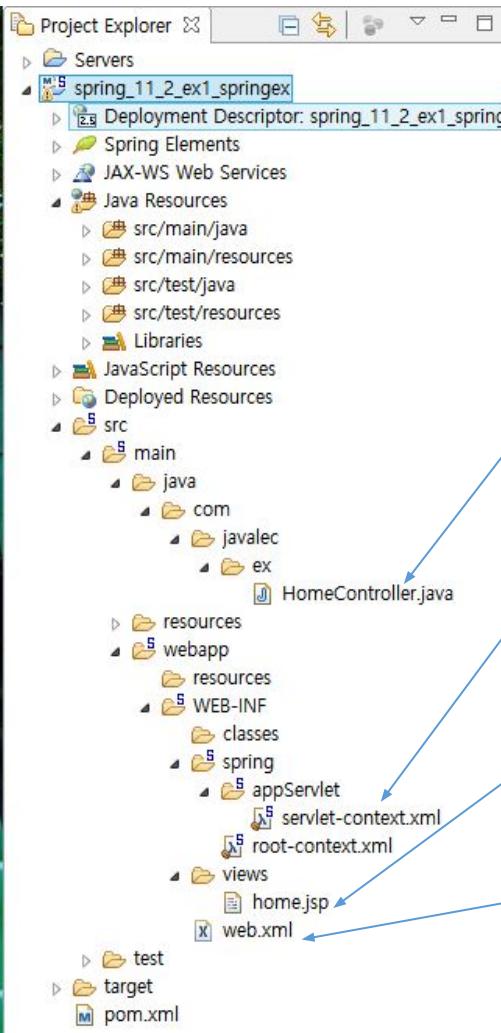
Spring Project 메뉴 진입

## 11-2. 스프링 MVC 구조 살펴보기



기본 패키지 설정

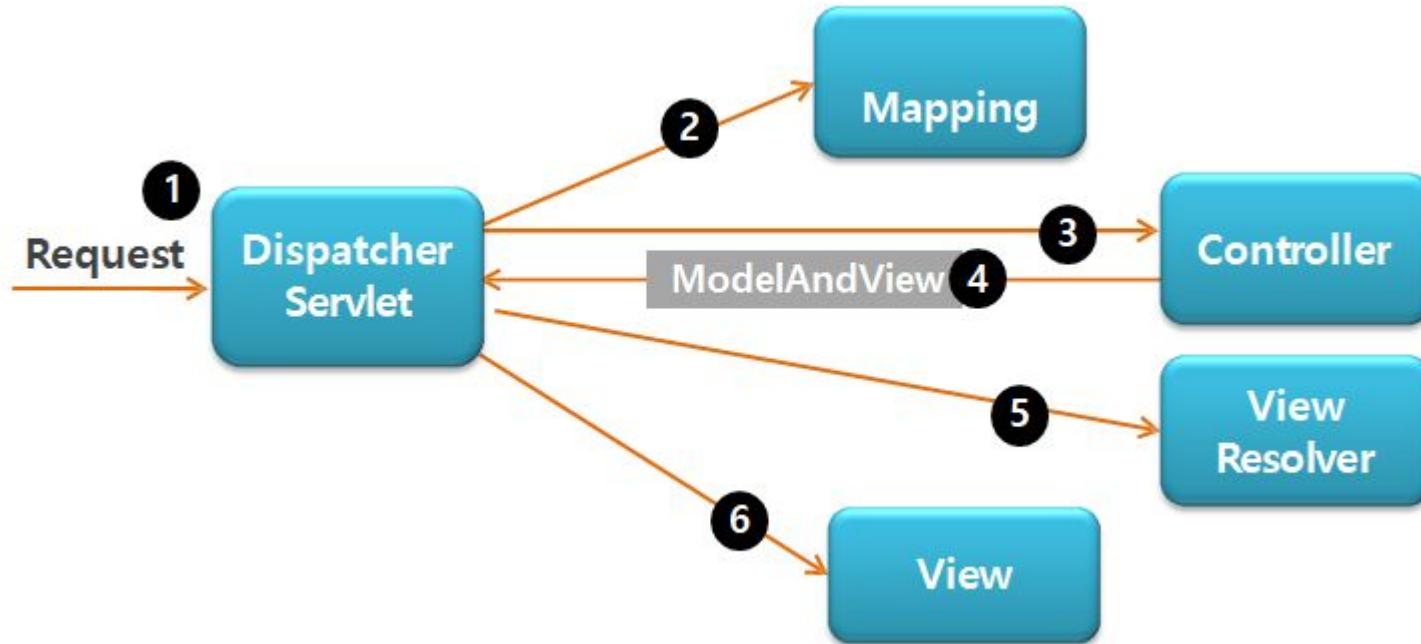
Project 이름 및 Spring MVC Project 설정



# Spring MVC의 주요 구성 요소

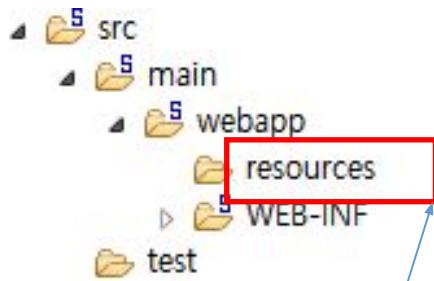
구성요소	설명
DispatcherServlet	클라이언트의 요청을 받아서 Controller에게 클라이언트의 요청을 전달하고, 리턴한 결과값을 View에게 전달하여 알맞은 응답을 생성
HandlerMapping	URL과 요청 정보를 기준으로 어떤 핸들러 객체를 사용할지 결정하는 객체이며, DispatcherServlet은 하나 이상의 핸들러 매핑을 가질 수 있음
Controller	클라이언트의 요청을 처리한 뒤, Model을 호출하고 그 결과를 DispatcherServlet에게 알려줌
ModelAndView	Controller가 처리한 데이터 및 화면에 대한 정보를 보유한 객체
View	Controller의 처리 결과 화면에 대한 정보를 보유한 객체
ViewResolver	Controller가 리턴한 뷰 이름을 기반으로 Controller 처리 결과를 생성할 뷰를 결정

# Spring MVC의 주요 구성 요소의 요청 처리 과정



### 11-3. resources 폴더

webapp/resources 폴더에 대해서 살펴 봅니다.



```
<!-- Processes application requests -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

web.xml

```
<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the ${webappRoot}/resources directory -->
<resources mapping="/resources/**" location="/" />
```

spring\_11\_3\_ex1\_springex

### 11-3. resources 풀더

---

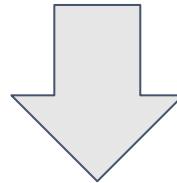
```
<!-- Processes application requests -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

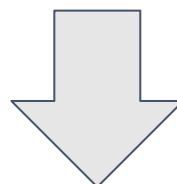
### 11-3. resources 풀더

---

```
<param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
```



```
<param-value>
    /WEB-INF/spring/appServlet/servlet-context.xml
    /WEB-INF/spring/appServlet/servlet-context2.xml
</param-value>
```



```
<param-value>/WEB-INF/spring/appServlet/servlet-context*.xml</param-value>
```

# 12. 컨트롤러

- 컨트롤러 클래스 제작
- 요청 처리 메소드 제작
  - 뷰에 데이터 전달
- 클래스에 @RequestMapping 적용

## 12-1. 컨트롤러 클래스 제작

클라이언트에서 요청 → 컨트롤러로 진입 → 뷰쪽으로 데이터 전달

## 컨트롤러 클래스 제작 순서

### @Controller를 이용한 클래스 생성

@RequestMapping을 이용한 요청 경로 지정

요청 처리 메소드 구현

뷰 이름 리턴

```
3 @Controller  
3 public class HomeController {
```

## 12-2. 요청 처리 메소드 제작

클라이언트의 요청을 처리할 메소드 제작

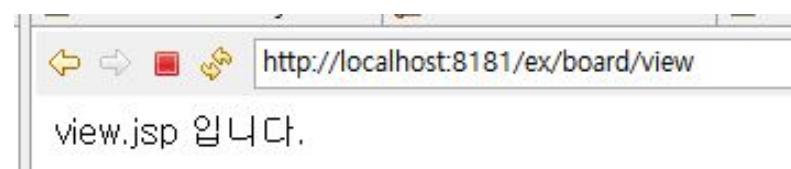
```
@RequestMapping("/board/view")  
public String view() {
```

요청 경로(path)

```
    return "board/view";  
}
```

뷰페이지 이름

결과 화면



## 12-2. 요청 처리 메소드 제작

뷰페이지 이름 생성(조합) 방법

뷰페이지 이름 = prefix + 요청처리 메소드 반환값 + suffix

The screenshot shows a file tree on the left and an XML configuration file on the right.

**File Tree:**

- WEB-INF
  - classes
  - spring
  - appServlet
    - servlet-context.xml
    - root-context.xml
  - views
    - board
    - home.jsp
  - web.xml

**XML Configuration (servlet-context.xml):**

```
<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>
```

The XML code defines a bean for view resolution. It uses the InternalResourceViewResolver class. The 'prefix' property is set to '/WEB-INF/views/' and the 'suffix' property is set to '.jsp'. These properties define the template for generating view names from controller method names.

### 12-3. 뷰에 데이터 전달

컨트롤러에서 로직 수행 후 뷰페이지 반환

이때 뷰에서 사용하게 될 데이터를 객체로 전달 가능

```
@RequestMapping("/board/content")
public String content(Model model) {
    model.addAttribute("id", 30);
    return "board/content";
}
```

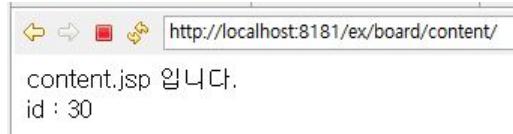
```
</head>
<body>
    content.jsp 입니다. <br />
    id : ${id}
</body>
</html>
```

Model 클래스를 이용한 데이터 전달

Model 객체를 파라미터로 받음

Model 객체에 데이터를 담음

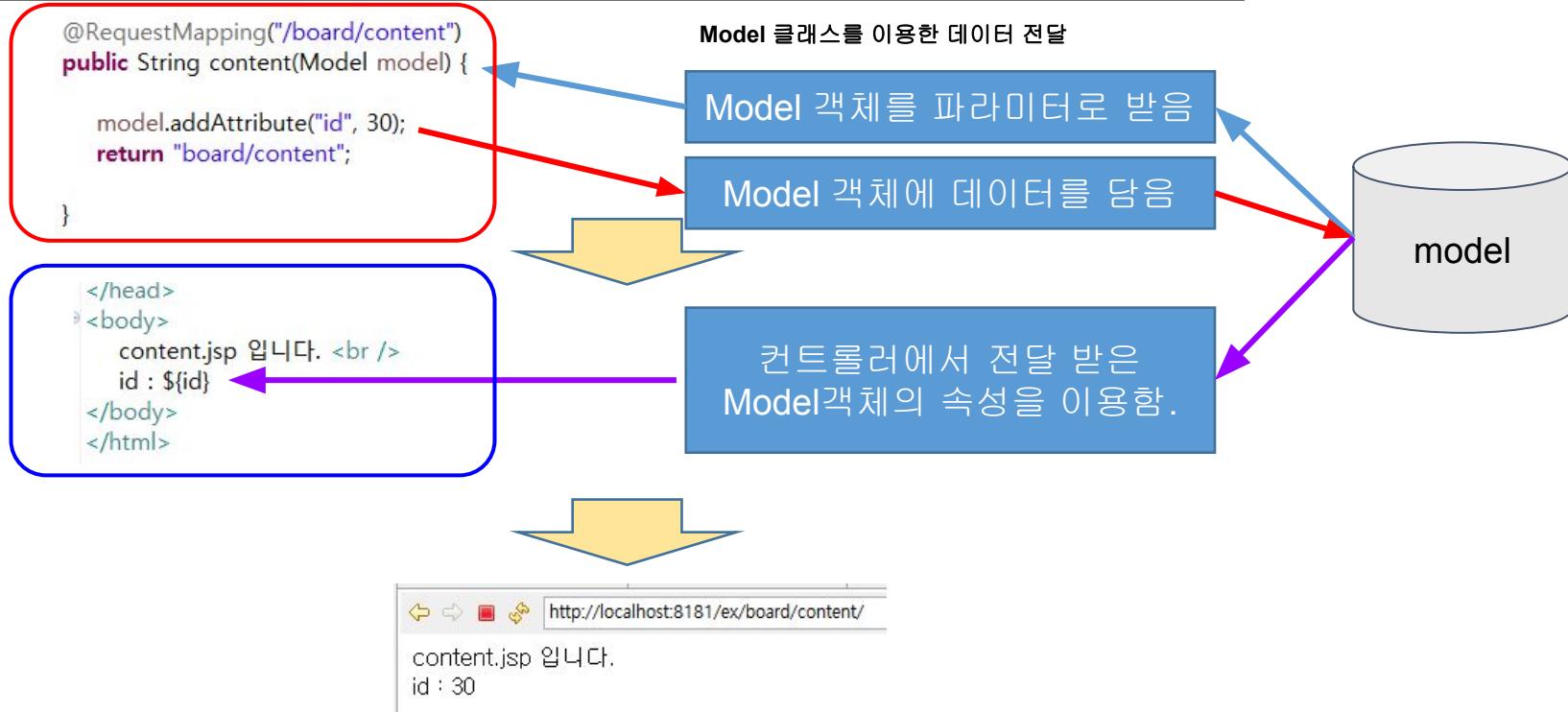
컨트롤러에서 전달 받은  
Model 객체의 속성을 이용함.



### 12-3. 뷰에 데이터 전달

컨트롤러에서 로직 수행 후 뷰페이지 반환

이때 뷰에서 사용하게 될 데이터를 객체로 전달 가능



## 12-3. 뷰에 데이터 전달

### ModelAndView 클래스를 이용한 데이터 전달

```
@RequestMapping("/board/reply")
public ModelAndView reply0 {
    ModelAndView mv = new ModelAndView();
    mv.addObject("id", 30);
    mv.setViewName("board/reply");
    return mv;
}
```



## 12-4. 클래스에 @RequestMapping 적용

메서드에 @RequestMapping 어노테이션을 적용하여 요청경로를 얻습니다.

```
@Controller  
@RequestMapping("/board")  
public class HomeController {
```

클래스에 @RequestMapping 적용



/board

```
@RequestMapping("/write")  
public String write(Model model) {
```

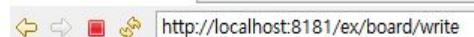
메소드에 @RequestMapping 적용



/write

```
model.addAttribute("id", 30);  
  
return "board/write";  
}
```

조합된 요청 경로 : /board/write



write.jsp 파일입니다.

# 13. Form 데이터

- HttpServletRequest 클래스
- @RequestParam 어노테이션
  - 데이터(커맨드) 객체
  - @PathVariable

### 13-1. HttpServletRequest 클래스

HttpServletRequest 클래스를 이용해서 데이터를 전송하는 방법에 대해서 살펴 봅니다.

```
@RequestMapping("board/confirmId")
public String confirmId(HttpServletRequest httpServletRequest, Model model) {
    String id = httpServletRequest.getParameter("id");
    String pw = httpServletRequest.getParameter("pw");
    model.addAttribute("id", id);
    model.addAttribute("pw", pw);
    return "board/confirmId";
}
```



ID : abcd  
PW : 1234

http://localhost:8181/ex/board/confirmId?id=abcd&pw=1234

```
<body>
ID : ${id} <br />
PW : ${pw}
</body>
</html>
```

## 13-2. @RequestParam 어노테이션

@RequestParam 어노테이션을 이용해서 데이터를 전송하는 방법에 대해서 살펴 봅니다.

```
@RequestMapping("board/checkId")
public String checkId(@RequestParam("id") String id, @RequestParam("pw") int pw, Model model) {
    model.addAttribute("identify", id);
    model.addAttribute("password", pw);
    return "board/checkId";
}
```



```
</head>
<body>
    ID : ${identify} <br />
    PW : ${password}
</body>
</html>
```

### 13-3. 데이터(커맨드) 객체

데이터(커맨드) 객체를 이용하여 데이터 많을 경우 사용

**기존 방법 : 다소 코드양이 많다.**

```
@RequestMapping("/member/join")
public String joinData(@RequestParam("name") String name, @RequestParam("id") String id,
    @RequestParam("pw") String pw, @RequestParam("email") String email, Model model) {

    Member member = new Member();
    member.setName(name);
    member.setId(id);
    member.setPw(pw);
    member.setEmail(email);

    model.addAttribute("memberInfo", member);

    return "member/join";
}
```



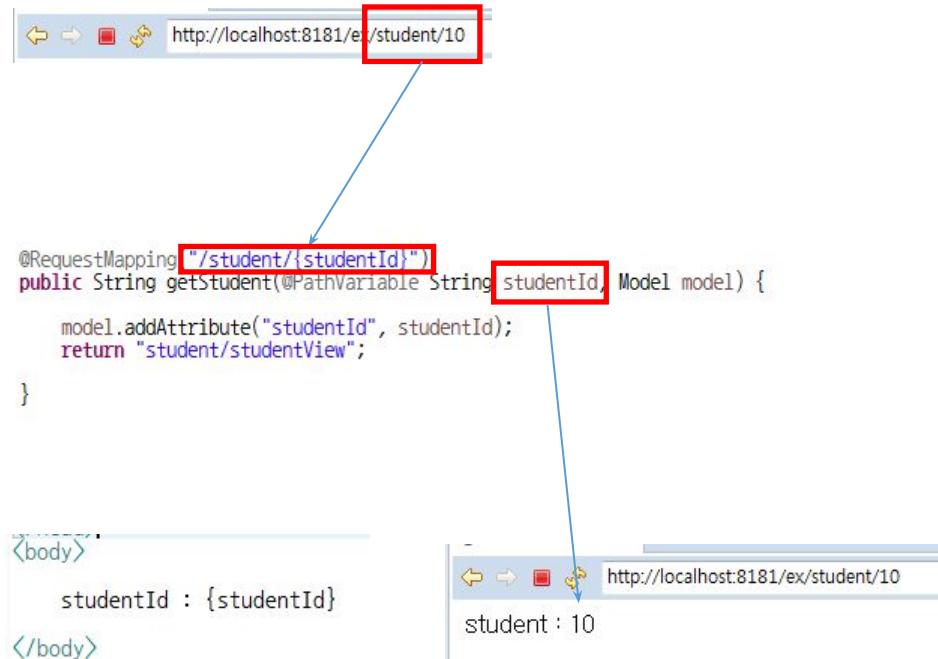
**개선 방법 : 코드양이 적다.**

```
@RequestMapping("/member/join")
public String joinData(Member member) {
    return "member/join";
}
```



## 13-4. @PathVariable

경로(path)에 변수를 넣어 요청메소드에서 파라미터로 이용



## 14-1. @RequestMapping에서 Get방식과 Post방식

---

# Charles Proxy

## GET 요청

1. 웹 브라우저의 주소창에 주소를 입력
2. 링크를 클릭
3. 입력폼의 method 속성이 GET 일때

....?name1=value1&name2=value2

**GET 서비스주소 ?name1=value1&name2=value2**

- ? - 서비스와 데이터 분리
- name1 - 파라미터이름(key)
- value1 - 파라미터값(value)
- & 데이터구분자
- = 파라미터명과 값을 구분자

## GET 요청 특징

- URL에 데이터 포함 → 데이터 조회에 적합
- 바이너리 및 대용량 데이터 전송 불가
- 요청라인과 헤더필드의 최대 크기 제한
- 대용량 URL로 인한 문제 발생 - 웹 서버별 최대 크기 제한
  - IIS : 16kB
  - Apache : 8kB

## POST 요청

1. URL에 데이터가 포함되지 않음
2. 보안 용이
3. 메시지 본문에 데이터 포함 - 실행결과공유불가
4. 바이너리 및 대용량 데이터 전송 가능

## 14. @RequestMapping 파라미터

- Get방식과 Post방식
  - @ModelAttribute
  - 리다이렉트(redirect:) 키워드

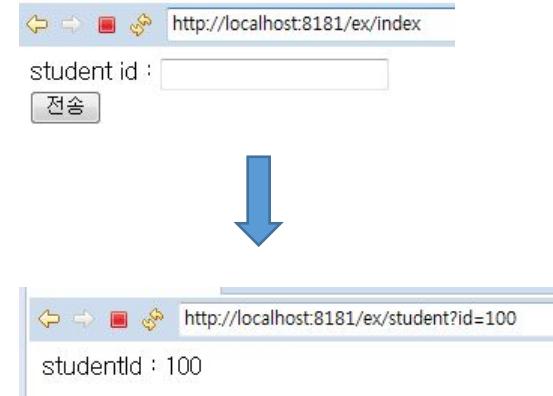
## 14-1. @RequestMapping에서 Get방식과 Post방식

@RequestMapping에서 요청을 받을 때 Get방식과 Post방식을 구분 할 수 있습니다.  
(spring\_14\_1\_ex1\_springex)

```
<form action="student" method="get">
    student id : <input type="text" name="id" > <br />
    <input type="submit" value="전송">
</form>

@RequestMapping(method = RequestMethod.GET, value = "/student")
public String goStudent(HttpServletRequest httpServletRequest, Model model) {
    System.out.println("RequestMethod.GET");
    String id = httpServletRequest.getParameter("id");
    System.out.println("id : " + id);
    model.addAttribute("studentId", id);
    return "student/studentId";
}
```

```
<form action="student" method="post">
    student id : <input type="text" name="id" > <br />
    <input type="submit" value="전송">
</form>
```



### HTTP Status 405 - Request method 'POST' not supported

**type** Status report

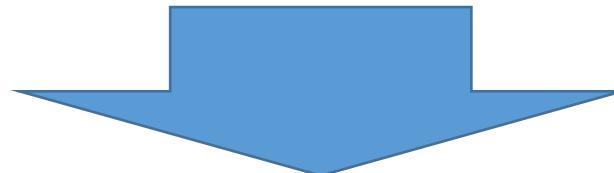
**message** Request method 'POST' not supported

**description** The specified HTTP method is not allowed

## 14-2. @ModelAttribute

@ModelAttribute 어노테이션을 이용하면 커맨드 객체의 이름을 개발자가 변경 가능  
(spring\_14\_2\_ex1\_springex)

```
@RequestMapping("/studentView")
public String studentView(StudentInformation studentInformation){
    return "studentView";
}
```



```
<body>
이름 : ${studentInformation.name} <br />
나이 : ${studentInformation.age} <br />
학년 : ${studentInformation.classNum} <br />
반 : ${studentInformation.gradeNum}
</body>
```

```
@RequestMapping("/studentView")
public String studentView(@ModelAttribute("studentInfo") StudentInformation studentInformation){
    return "studentView";
}
```

```
<body>
이름 : ${studentInfo.name} <br />
나이 : ${studentInfo.age} <br />
학년 : ${studentInfo.classNum} <br />
반 : ${studentInfo.gradeNum}
</body>
```

### 14-3. 리다이렉트(redirect: 키워드)

다른 페이지로 이동할 때 사용 합니다. (spring\_14\_3\_ex1\_springex)

```
@RequestMapping("/studentConfirm")
public String studentRedirect(HttpServletRequest httpServletRequest, Model model){
    String id = httpServletRequest.getParameter("id");
    if(id.equals("abc")){
        return "redirect:studentOk";
    }
    return "redirect:studentNg";
}
```

A diagram illustrating a redirect. On the left, the code for the `studentRedirect` method is shown. An arrow points from the `return "redirect:studentOk";` line to a browser window on the right. The browser window shows the URL `http://localhost:8181/spring_14_3_ex1_springex/studentConfirm?id=abc`. The browser interface includes back and forward buttons, a search bar, and a title bar.

```
@RequestMapping("/studentOk")
public String studentOk(Model model){
    return "student/studentOk";
}
```

A diagram illustrating the result of the redirect. An arrow points from the `studentOk` method call in the code to a browser window on the right. The browser window shows the URL `http://localhost:8181/spring_14_3_ex1_springex/studentOk`. The browser interface includes back and forward buttons, a search bar, and a title bar. The page content is labeled "studentOk.jsp".

```
@RequestMapping("/studentNg")
public String studentNg(Model model){
    return "student/studentNg";
}
```

A diagram illustrating the result of the redirect. An arrow points from the `studentNg` method call in the code to a browser window on the right. The browser window shows the URL `http://localhost:8181/spring_14_3_ex1_springex/studentNg`. The browser interface includes back and forward buttons, a search bar, and a title bar. The page content is labeled "studentNg.jsp".

# 15강. 폼 데이터 값 검증

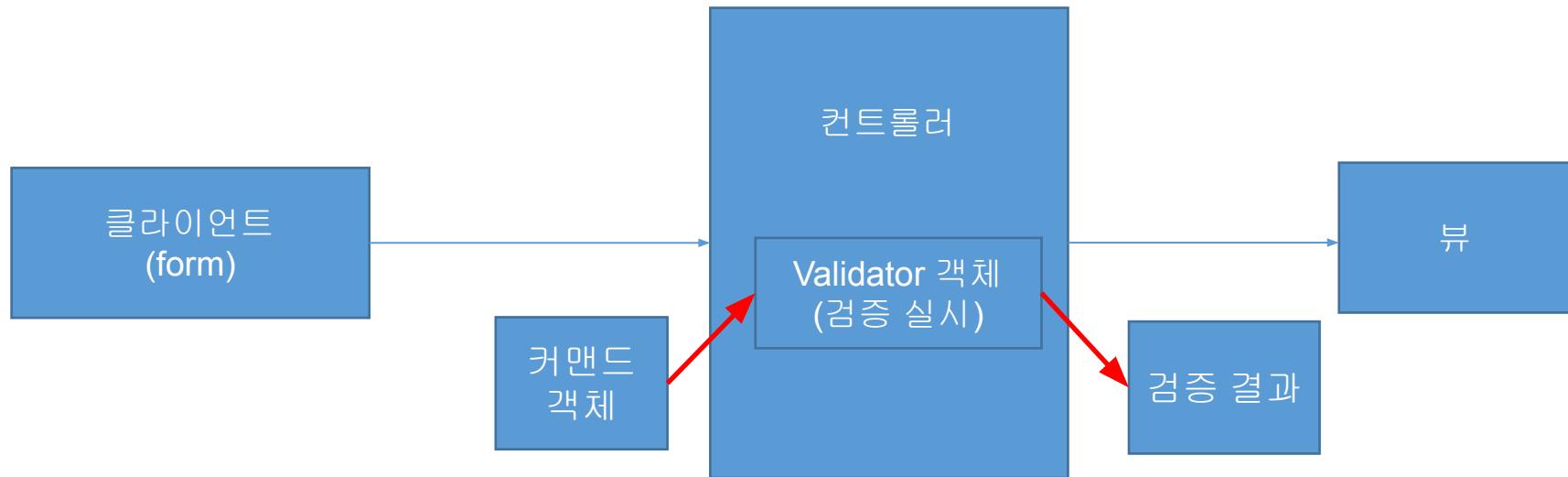
- Validator를 이용한 검증
- ValidationUtils 클래스
- @Valid와 @InitBinder

## 15-1. Validator를 이용한 검증

폼에서 전달 되는 데이터를 커맨드 객체에 담아 컨트롤러 객체에 전달

이때 커맨드 객체의 유효성 검사

참고로 javascript을 이용하는 것은 클라이언트에서 검사하는 방법이고,  
Validator 인터페이스를 이용하는 방법은 서버에서 검사하는 방법  
(spring\_15\_1\_ex1\_springex)



## 15-1. Validator를 이용한 검증

```
@RequestMapping("/student/create")
public String studentCreate(@ModelAttribute("student") Student student, BindingResult result) {
    String page = "createDonePage";

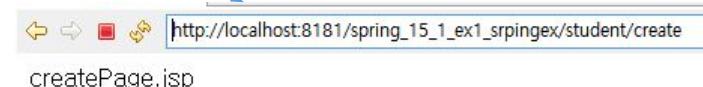
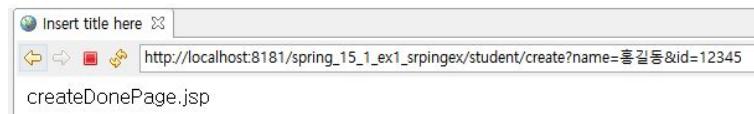
    StudentValidator validator = new StudentValidator();
    validator.validate(student, result);
    if(result.hasErrors()) {
        page = "createPage";
    }

    return page;
}
```

```
public void validate(Object obj, Errors errors) {
    System.out.println("validate()");
    Student student = (Student)obj;

    String studentName = student.getName();
    if(studentName == null || studentName.trim().isEmpty()) {
        System.out.println("studentName is null or empty");
        errors.rejectValue("name", "trouble");
    }

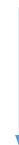
    int studentId = student.getId();
    if(studentId == 0) {
        System.out.println("studentId is 0");
        errors.rejectValue("id", "trouble");
    }
}
```



## 15-2. ValidationUtils 클래스

데이터 검증을 위해서 Validator 인터페이스의 validate() 메소드를 사용  
ValidationUtils 클래스는 validate()를 좀 더 편리하게 사용 할 수 있도록 고안된 클래스  
(spring\_15\_2\_ex1\_springex)

```
String studentName = student.getName();
if(studentName == null || studentName.trim().isEmpty()) {
    System.out.println("studentName is null or empty");
    errors.rejectValue("name", "trouble");
}
```



```
ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "trouble");
```

### 15-3. @Valid와 @InitBinder

데이터 검증을 하기 위해서 Validator 인터페이스를 구현한 클래스를 만들고, validate() 메소드를 직접 호출하여 사용.

직접 호출하지 않고, 스프링 프레임워크에서 호출하는 방법(spring\_15\_3\_ex1\_springex)

#### 의존 추가

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>4.2.0.Final</version>
</dependency>
```

#### @InitBinder 추가

```
@InitBinder
protected void initBinder(WebDataBinder binder){
    binder.setValidator(new StudentValidator());
}
```

#### @Valid 추가

```
@RequestMapping("/student/create")
public String studentCreate(@ModelAttribute("student") Student student, BindingResult result) {
```

```
    String page = "createDonePage";
```

```
    StudentValidator validator = new StudentValidator();
    validator.validate(student, result);
```

```
    if(result.hasErrors()) {
        page = "createPage";
    }
```

```
}
```

```
@RequestMapping("/student/create")
public String studentCreate(@ModelAttribute("student") @Valid Student student, BindingResult result) {
```

```
    String page = "createDonePage";
```

```
    StudentValidator validator = new StudentValidator();
    validator.validate(student, result);
```

```
    if(result.hasErrors()) {
        page = "createPage";
    }
```

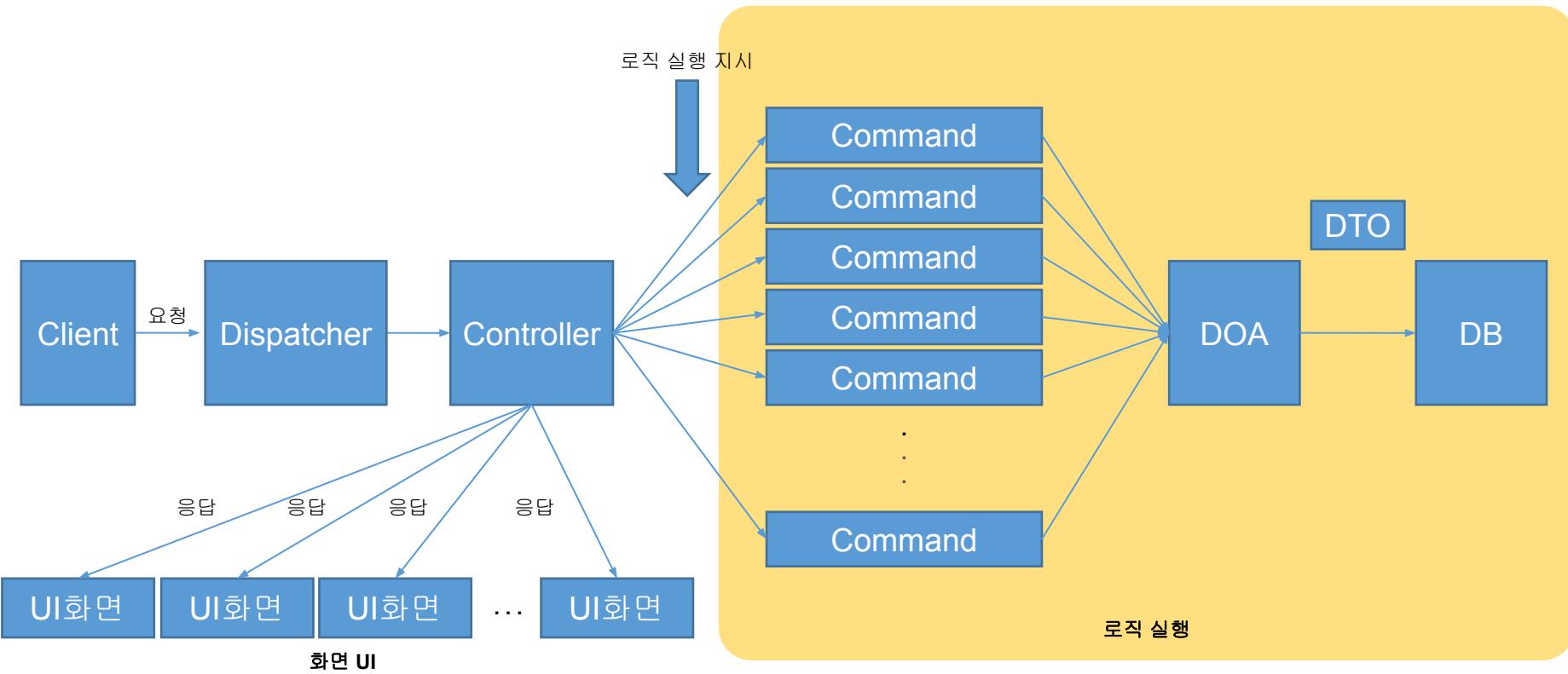
```
}
```

# 16강. 스프링MVC 게시판-I

- 프로젝트 설계
- DataBase 구축
- 프로젝트 생성

## 16-1. 프로젝트 설계

### 프로젝트의 전체적인 설계



## 16-2. DataBase 구축

### 데이터 베이스 구축

```
create table mvc_board(
    bId NUMBER(4) PRIMARY KEY,
    bName VARCHAR2(20),
    bTitle VARCHAR2(100),
    bContent VARCHAR2(300),
    bDate DATE DEFAULT SYSDATE,
    bHit NUMBER(4) DEFAULT 0,
    bGroup NUMBER(4),
    bStep NUMBER(4),
    bIndent NUMBER(4)
);
```

```
create sequence mvc_board_seq;
```

The screenshot shows the 'MVC\_BOARD' table structure in Oracle SQL Developer. The table has 9 columns:

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 BID	NUMBER(4, 0)	No	(null)	1 (null)	
2 BNAME	VARCHAR2(20 BYTE)	Yes	(null)	2 (null)	
3 BTITLE	VARCHAR2(100 BYTE)	Yes	(null)	3 (null)	
4 BCONTENT	VARCHAR2(300 BYTE)	Yes	(null)	4 (null)	
5 BDATE	DATE	Yes	SYSDATE	5 (null)	
6 BHIT	NUMBER(4, 0)	Yes	0	6 (null)	
7 BGROUP	NUMBER(4, 0)	Yes	(null)	7 (null)	
8 BSTEP	NUMBER(4, 0)	Yes	(null)	8 (null)	
9 BINDENT	NUMBER(4, 0)	Yes	(null)	9 (null)	

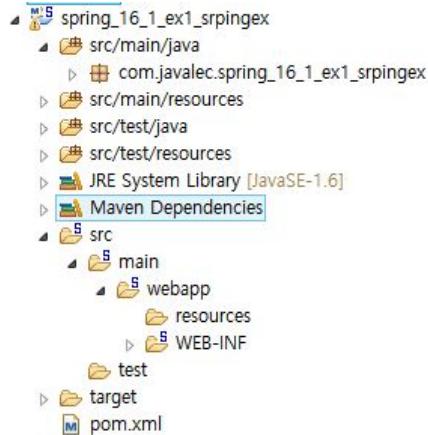
### 테스트를 위한 Dummy Data 입력

```
insert into mvc_board (bId, bName, bTitle, bContent, bHit, bGroup, bStep, bIndent)
values (mvc_board_seq.nextval, 'abcd', 'is title', 'is content', 0, mvc_board_seq.currval, 0, 0);
```

### 16-3. 프로젝트 생성

이클립스에서 프로젝트를 생성 합니다.

#### 프로젝트 생성



#### 한글처리 : web.xml

```
<filter>
<filter-name>encodingFilter</filter-name>
<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>
<param-name>encoding</param-name>
<param-value>UTF-8</param-value>
</init-param>
</filter>

<filter-mapping>
<filter-name>encodingFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

# 17. 스프링MVC 게시판-II

- 패키지, 인터페이스, 클래스 제작

- Controller 제작

- 리스트 페이지 만들기

## 17-1. 패키지, 인터페이스, 클래스 제작

프로젝트 설계에 따른 전체적인 패키지, 인터페이스, 기본 클래스들을 만들어 봅니다.  
(spring\_16\_1\_ex1\_springex)

- ▲ com.javalec.spring\_16\_1\_ex1\_springex.command
  - ▷ J BCommand.java
  - ▷ J BContentCommand.java
  - ▷ J BDeleteCommand.java
  - ▷ J BListCommand.java
  - ▷ J BModifyCommand.java
  - ▷ J BReplyCommand.java
  - ▷ J BReplyViewCommand.java
  - ▷ J BWriteCommand.java
- ▲ com.javalec.spring\_16\_1\_ex1\_springex.controller
  - ▷ BController.java
- ▲ com.javalec.spring\_16\_1\_ex1\_springex.dao
  - ▷ BDao.java
- ▲ com.javalec.spring\_16\_1\_ex1\_springex.dto
  - ▷ BDto.java

BDto

BDao

BController

BCommand

BContentCommand

BDeleteCommand

BListCommand

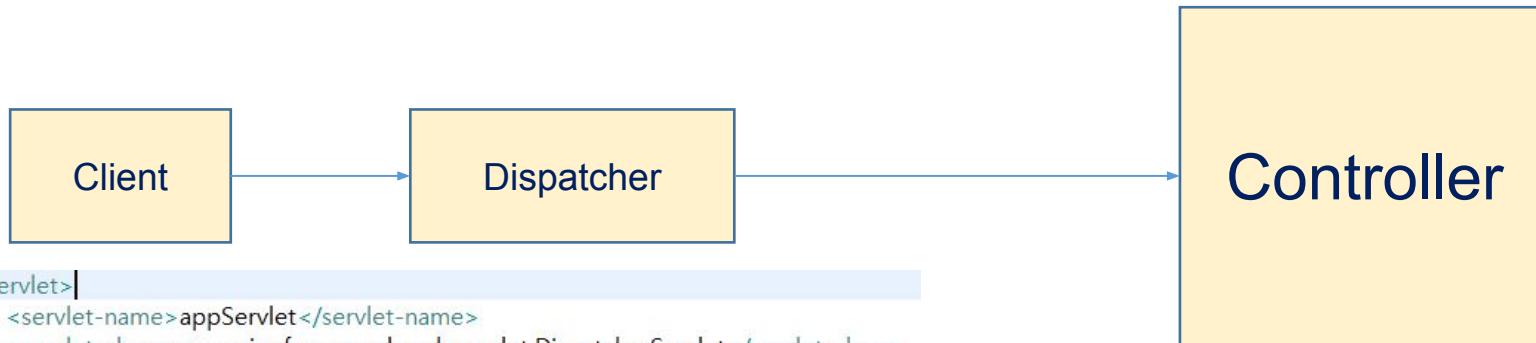
BModifyCommand

## 17-1. Controller 제작

클라이언트의 요청에 따른 전체적인 작업을 지휘하는 Controller

JSP의 MVC model2 방식으로 작업 진행

(spring\_16\_1\_ex1\_springex)



```
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

servlet-context.xml

```
<context:component-scan base-package= "com.javalec.springMVCBoard" />
```

### 17-3. 리스트 페이지 만들기

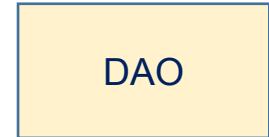
```
@RequestMapping("/list")
public String list(Model model) {
    System.out.println("list()");
    command = new BListCommand();
    command.execute(model);

    return "list";
}
```

Bcontroller.class

```
@Override
public void execute(Model model) {
    // TODO Auto-generated method stub
    BDao dao = new BDao();
    ArrayList<BDto> dtos = dao.list();
    model.addAttribute("list", dtos);
}
```

BListCommand.class



# 18. 스프링MVC 게시판-III

- 글 작성 페이지 만들기
- 글 내용 페이지 만들기

## 18-1. 글 작성 페이지 만들기

---

Bcontroller.class

```
@RequestMapping("/write_view")
public String write_view(Model model) {
    System.out.println("write_view0");

    return "write_view";
}

@RequestMapping("/write")
public String write(HttpServletRequest request, Model model) {
    System.out.println("write0");

    model.addAttribute("request", request);
    command = new BWriteCommand();
    command.execute(model);

    return "redirect:list";
}
```



## 18-1. 글 작성 페이지 만들기

```
@Override  
public void execute(Model model) {  
    // TODO Auto-generated method stub  
  
    Map<String, Object> map = model.asMap();  
    HttpServletRequest request = (HttpServletRequest) map.get("request");  
    String bName = request.getParameter("bName");  
    String bTitle = request.getParameter("bTitle");  
    String bContent = request.getParameter("bContent");  
  
    BDao dao = new BDao();  
    dao.write(bName, bTitle, bContent);  
}
```

DAO

BWriteCommand.class

## 18-2. 글 내용 페이지 만들기

---

```
@RequestMapping("/content_view")
public String content_view(HttpServletRequest request, Model model){
    System.out.println("content_view0");

    model.addAttribute("request", request);
    command = new BContentCommand();
    command.execute(model);

    return "content_view";
}
```

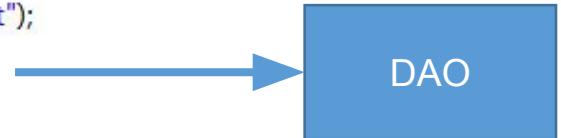


Bcontroller.class

## 18-2. 글 내용 페이지 만들기

```
@Override  
public void execute(Model model) {  
    // TODO Auto-generated method stub  
  
    Map<String, Object> map = model.asMap();  
    HttpServletRequest request = (HttpServletRequest) map.get("request");  
    String bld = request.getParameter("bld");  
  
    BDao dao = new BDao();  
    BDto dto = dao.contentView(bld);  
  
    model.addAttribute("content_view", dto);  
}
```

BContentCommand.class



# 19. 스프링MVC 게시판-IV

- 글 수정 페이지 만들기
- 글 삭제 페이지 만들기

## 19-1. 글 수정 페이지 만들기

---

```
@RequestMapping(value="/modify", method=RequestMethod.POST)
public String modify(HttpServletRequest request, Model model){
    System.out.println("modify0");

    model.addAttribute("request", request);
    command = new BModifyCommand();
    command.execute(model);

    return "redirect:list";
}
```

Bcontroller.class



## 19-1. 글 수정 페이지 만들기

```
@Override  
public void execute(Model model) {  
    // TODO Auto-generated method stub  
  
    Map<String, Object> map = model.asMap();  
    HttpServletRequest request = (HttpServletRequest) map.get("request");  
    String bld = request.getParameter("bld");  
    String bName = request.getParameter("bName");  
    String bTitle = request.getParameter("bTitle");  
    String bContent = request.getParameter("bContent");  
  
    BDao dao = new BDao0;  
    dao.modify(bld, bName, bTitle, bContent);  
}
```

BModifyCommand.class



## 19-2. 글 삭제 페이지 만들기

---

```
@RequestMapping("/delete")
public String delete(HttpServletRequest request, Model model) {
    System.out.println("delete0");

    model.addAttribute("request", request);
    command = new BDeleteCommand();
    command.execute(model);

    return "redirect:list";
}
```

Bcontroller.class



## 19-2. 글 삭제 페이지 만들기

```
@Override  
public void execute(Model model) {  
    // TODO Auto-generated method stub  
  
    Map<String, Object> map = model.asMap();  
    HttpServletRequest request = (HttpServletRequest) map.get("request");  
  
    String bld = request.getParameter("bld");  
    BDao dao = new BDao();  
    dao.delete(bld);  
  
}
```

BDeleteCommand.class



# 20. 스프링MVC 게시판-V

글 답변 페이지 만들기

## 20-1. 글 답변 페이지 만들기

---

```
@RequestMapping("/reply_view")
public String reply_view(HttpServletRequest request, Model model){
    System.out.println("reply_view0");

    model.addAttribute("request", request);
    command = new BReplyViewCommand();
    command.execute(model);

    return "reply_view";
}

@Controller.class
```



```
@RequestMapping("/reply")
public String reply(HttpServletRequest request, Model model) {
    System.out.println("reply0");

    model.addAttribute("request", request);
    command = new BReplyCommand();
    command.execute(model);

    return "redirect:list";
}
```

## 20-1. 글 답변 페이지 만들기

```
@Override  
public void execute(Model model) {  
    // TODO Auto-generated method stub  
  
    Map<String, Object> map = model.asMap();  
    HttpServletRequest request = (HttpServletRequest) map.get("request");  
  
    String bld = request.getParameter("bld");  
    String bName = request.getParameter("bName");  
    String bTitle = request.getParameter("bTitle");  
    String bContent = request.getParameter("bContent");  
    String bGroup = request.getParameter("bGroup");  
    String bStep = request.getParameter("bStep");  
    String blndent = request.getParameter("blndent");  
  
    BDao dao = new BDao();  
    dao.reply(bld, bName, bTitle, bContent, bGroup, bStep, blndent);  
}
```

DAO

BReplyCommand.class

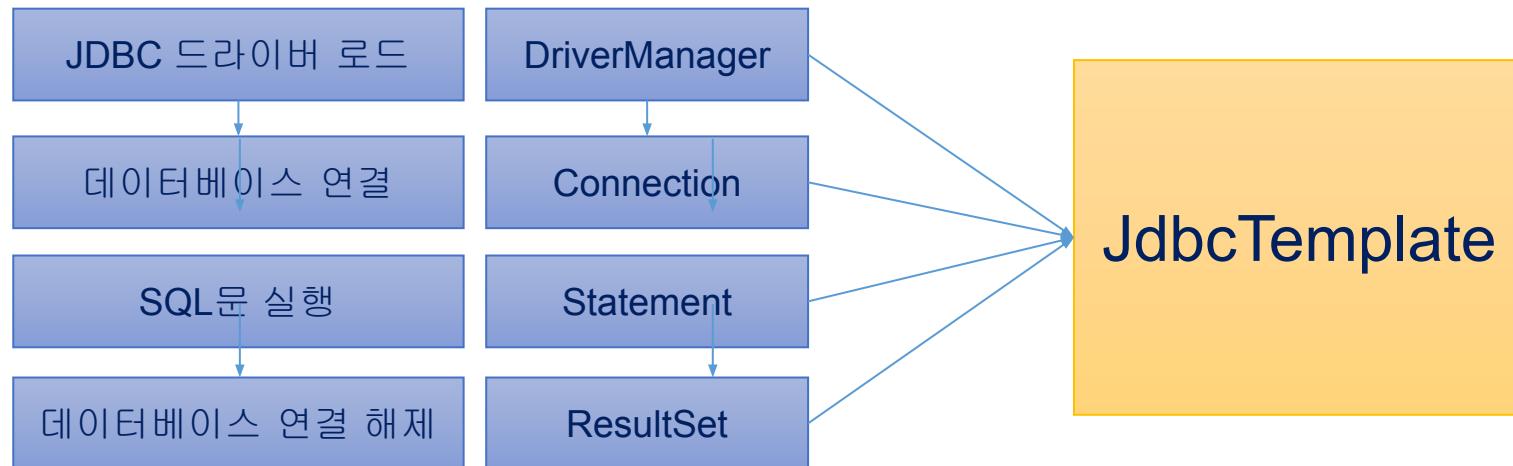
# 21. 스프링 JDBC

- JDBC를 이용한 반복코드 줄이기
- Spring빈을 이용한 코드 간소화
- JDBC를 이용한 리스트 목록 만들기
- insert, update, delete 처리하기

## 21-1. JDBC를 이용한 반복코드 줄이기

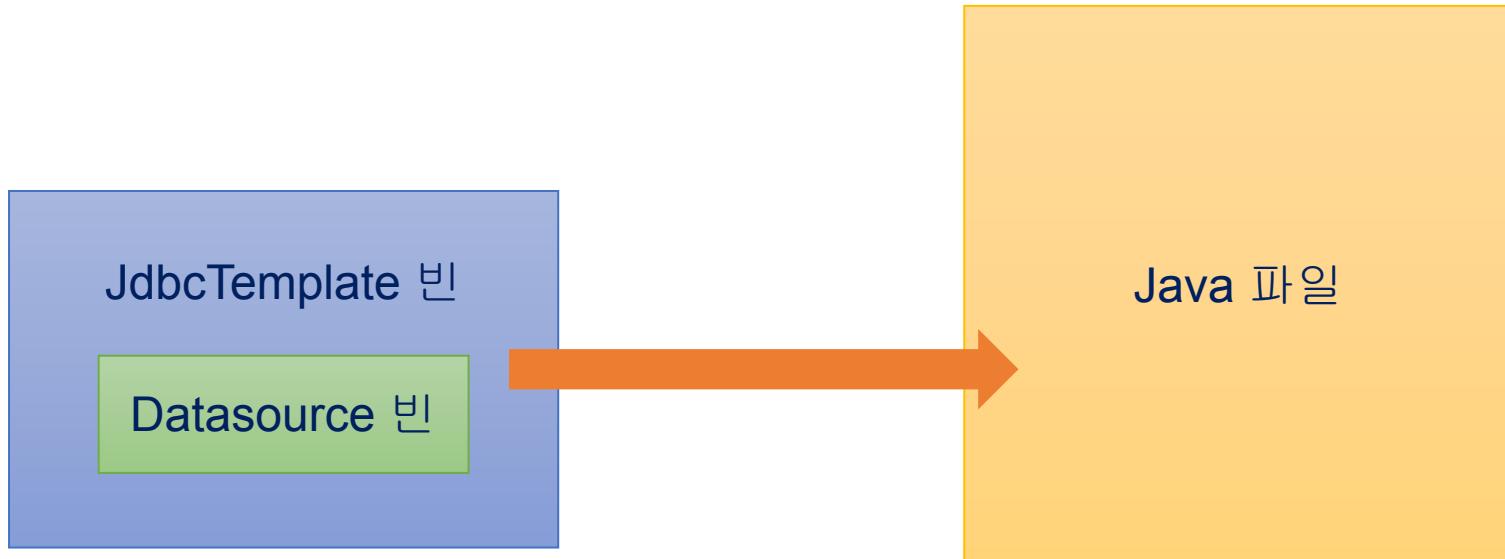
DAO객체를 이용해서 DataBase의 데이터를 이용

드라이버 로드, 커넥션 생성 및 DB연결, SQL실행, 자원해제 - 같은 동작을 반복  
이때의 처리방법



## 21-2. Spring 빈을 이용한 코드 간소화

Spring 빈을 이용 DataBase 관련 클래스들을 생성, 조립



## 21-3. JDBC를 이용한 리스트 목록 만들기

```
public ArrayList<BDto> list() {  
  
    ArrayList<BDto> dtos = new ArrayList<BDto>();  
    Connection connection = null;  
    PreparedStatement preparedStatement = null;  
    ResultSet resultSet = null;  
  
    try {  
        connection = dataSource.getConnection();  
  
        String query = "select bld, bName, bTitle, bContent"  
        preparedStatement = connection.prepareStatement(query);  
        resultSet = preparedStatement.executeQuery();  
  
        while (resultSet.next()) {  
            int bld = resultSet.getInt("bld");  
            String bName = resultSet.getString("bName");  
            String bTitle = resultSet.getString("bTitle");  
            String bContent = resultSet.getString("bContent");  
            Timestamp bDate = resultSet.getTimestamp("bDate");  
            int bHit = resultSet.getInt("bHit");  
            int bGroup = resultSet.getInt("bGroup");  
            int bStep = resultSet.getInt("bStep");  
            int blndent = resultSet.getInt("blndent");  
  
            BDto dto = new BDto(bld, bName, bTitle, bContent);  
            dtos.add(dto);  
        }  
  
    } catch (Exception e) {  
        // TODO: handle exception  
        e.printStackTrace();  
    } finally {  
        try {  
            if (resultSet != null)  
                resultSet.close();  
            if (preparedStatement != null)  
                preparedStatement.close();  
            if (connection != null)  
                connection.close();  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
public ArrayList<BDto> list()
```

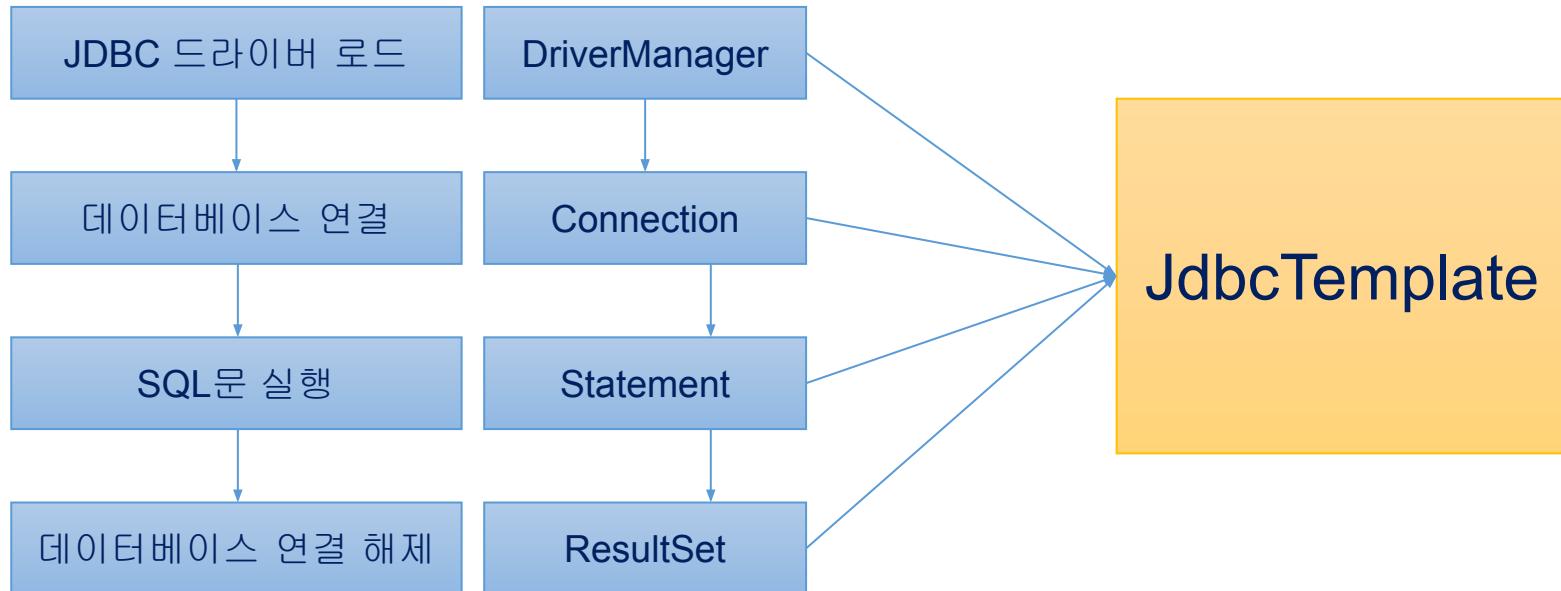
```
String query = "select bld, bName, bTitle, bContent, bDate, bHit, bGroup, bStep, blnder"  
return (ArrayList<BDto>) template.query(query, new BeanPropertyRowMapper<BDto>(B
```

```
}
```

JdbcTemplate의 메소드를 이용하여 코드를 간단하게 변경

## 21-4. insert, update, delete 처리하기

JdbcTemplate을 이용하여 insert, update, delete 사용하는 메소드를 살펴 봅니다.



# 22. 트랜잭션(Transaction)-I

- 트랜잭션의 개념
- 스프링 트랜잭션 사용방법

## 22-1. 트랜잭션의 개념

---

논리적 단위로 어떤 한 부분의 작업이 완료되었다 하더라도, 다른 부분의 작업이 완료되지 않았을 경우 전체 취소되는 경우

작업이 완료되는 것을 커밋(commit)이라고 하고, 작업이 취소되는 것을 롤백(rollback)

일상생활 중 트랜잭션의 예

- 영화 예매
- 카드 결제
- 마일리지 적립
- 은행 ATM

영화 예매

카드 결제

마일리지 적립

etc.

## 22-2. 스프링 트랜잭션 사용방법

문제 : 트랜잭션 처리를 하지 않아 rollback이 되지 않는 경우  
(spring\_22\_2\_ex1\_springex)

프로야구 한국 시리즈 매표소

카드 결제

- 카드 결제 프로세스입니다.

CONSUMERID	AMOUNT
1 abcde	4
2 abcdef	5
3 abc	1
4 abcd	2

매표소 직원

- 티켓을 고객한테 전달 합니다.
- 티켓은 1인당 4장까지 구매 가능 합니다.

CONSUMERID	COUNTNUM
1 abcde	4
2 abc	1
3 abcd	2

트랜잭션이 깨졌고, 모든 작업이 RollBack되어야 합니다.

## 22-2. 스프링 트랜잭션 사용방법

해결 : 트랜잭션 처리가 되어있는 경우, PlatformTransactionManger 를 이용  
(spring\_22\_2\_ex2\_springex)

프로야구 한국 시리즈 매표소

카드 결제

- 카드 결제 프로세스입니다.

매표소 직원

- 티켓을 고객한테 전달 합니다.
- 티켓은 1인당 4장까지 구매 가능 합니다.

CONSUMERID	AMOUNT
1 abcd	2
2 abcde	3
3 abc	1
4 abcdef	4

CONSUMERID	COUNTNUM
1 abcd	2
2 abc	1
3 abcde	3
4 abcdef	4

트랜잭션이 되었습니다.

# 23. 트랜잭션(Transaction)-II

- TransactionTemplate
- 트랜잭션 전파 속성(1)

## 23-1. TransactionTemplate

---

TransactionTemplate - PlatformTransactionManager 인터페이스 보다 많이 사용  
시간, 비용 대비 효율이 좋음 (spring\_23\_1\_ex1\_springex)

```
// PlatformTransactionManager transactionManager;
TransactionTemplate transactionTemplate;

④ public void setTemplate(JdbcTemplate template) {
    this.template = template;
}

// public void setTransactionManager( PlatformTransactionManager transactionManager) {
//     this.transactionManager = transactionManager;
// }

④ public void setTransactionTemplate(TransactionTemplate transactionTemplate) {
    this.transactionTemplate = transactionTemplate;
}
```

## 23-1. TransactionTemplate

---

TransactionTemplate - PlatformTransactionManager 인터페이스 보다 많이 사용  
시간, 비용 대비 효율이 좋음 (spring\_23\_1\_ex1\_springex)

```
<beans:bean name="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <beans:property name="dataSource" ref="dataSource" />
</beans:bean>
```

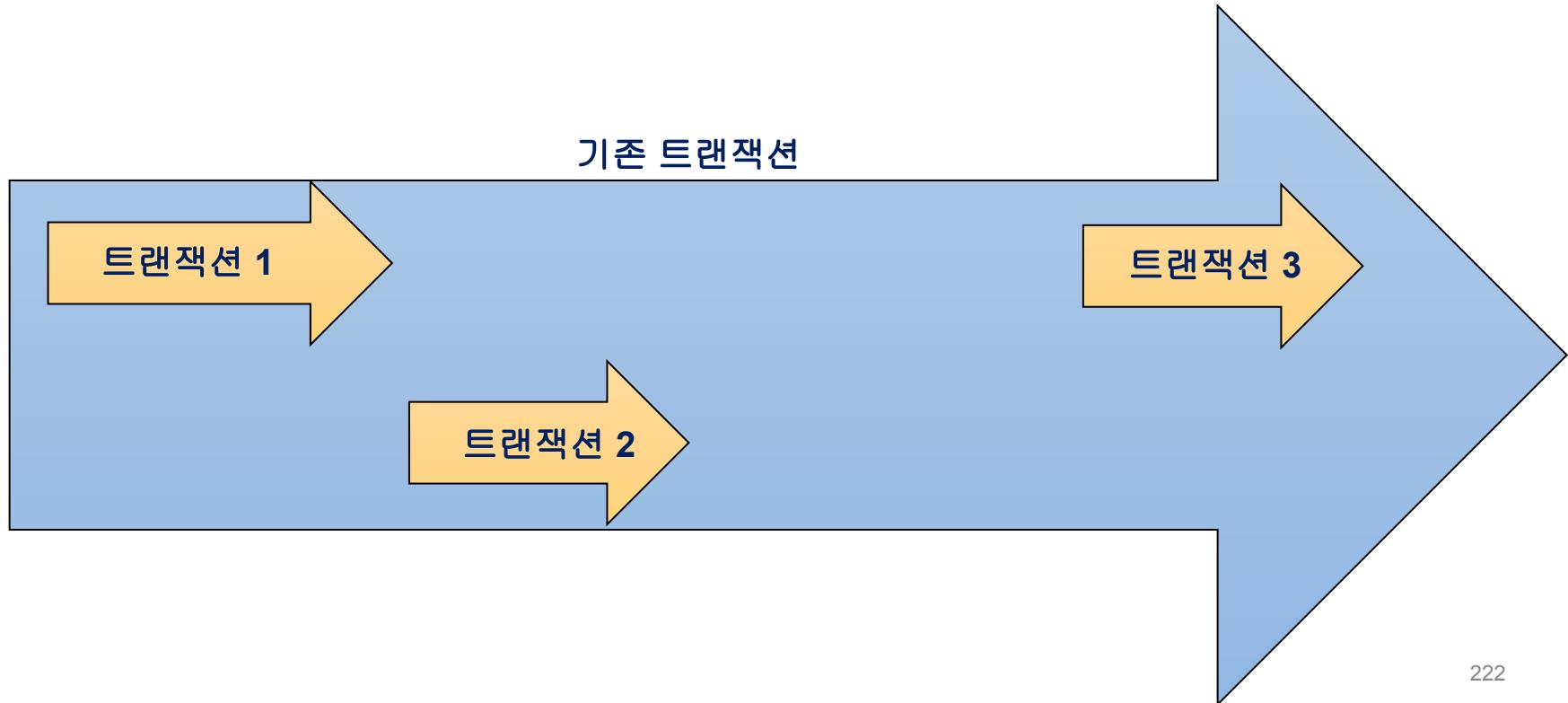


```
<beans:bean name="transactionTemplate" class="org.springframework.transaction.support.TransactionTemplate">
    <beans:property name="transactionManager" ref="transactionManager"></beans:property>
</beans:bean>
```

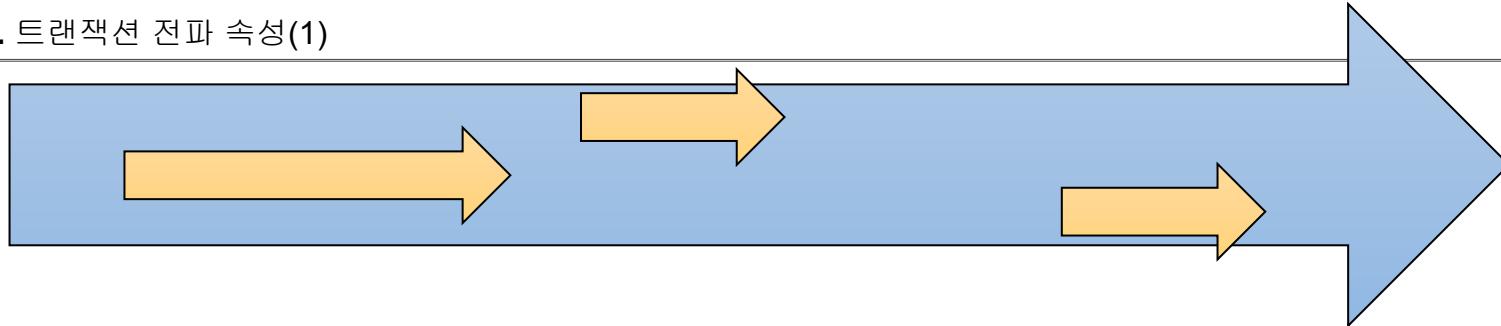
## 23-2. 트랜잭션 전파 속성(1)

2개 이상의 트랜잭션이 작동

기존 트랜잭션에 참여하는 방법 (spring\_23\_2\_ex1\_springex)



## 23-2. 트랜잭션 전파 속성(1)



PROPAGATION\_REQUIRED(0)

DEFAULT : 전체 처리

PROPAGATION\_SUPPORTS(1)

기존 트랜잭션에 의존

PROPAGATION\_MANDATORY(2)

트랜잭션에 포함 – 트랜잭션이 있는 곳에서 호출

PROPAGATION\_REQUIRES\_NEW(3)  
    )

각각 트랜잭션 처리

PROPAGATION\_NOT\_SUPPORTED  
    (4)

트랜잭션에 포함 X – 트랜잭션이 없는 것과 동일

PROPAGATION\_NEVER(5)

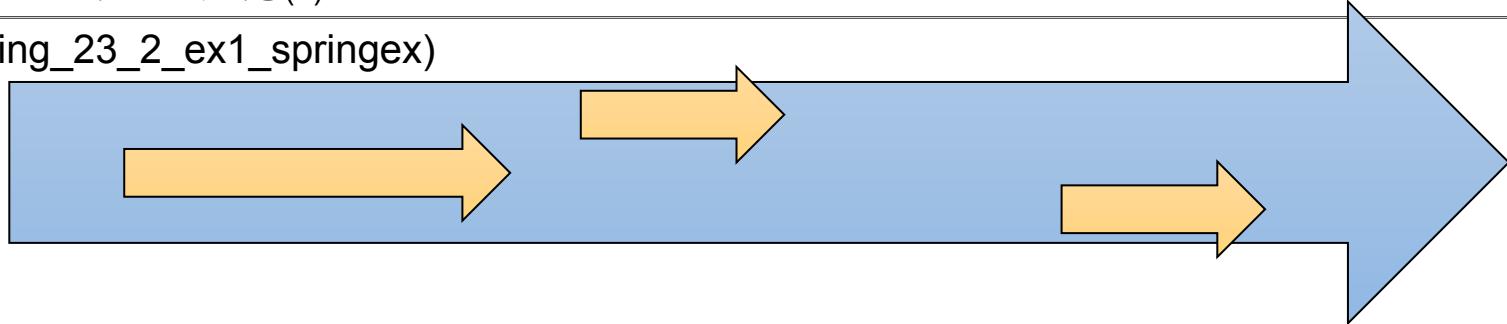
트랜잭션에 포함 X – 트랜잭션이 있는 곳에서 호출하면 에러

# 24. 트랜잭션(Transaction)-III

트랜잭션 전파 속성(2)

## 24-1. 트랜잭션 전파 속성(2)

(spring\_23\_2\_ex1\_springex)



PROPAGATION\_REQUIRED(0)

DEFAULT : 전체 처리

PROPAGATION\_SUPPORTS(1)

기존 트랜잭션에 의존

PROPAGATION\_MANDATORY(2)

트랜잭션에 포함 – 트랜잭션이 있는 곳에서 호출

PROPAGATION\_REQUIRES\_NEW(3)  
    )

각각 트랜잭션 처리

PROPAGATION\_NOT\_SUPPORTED  
    (4)

트랜잭션에 포함 X – 트랜잭션이 없는 것과 동일

PROPAGATION\_NEVER(5)

트랜잭션에 포함 X – 트랜잭션이 있는 곳에서 호출하면 에러

# 25. Security-I

- 보안 관련 프로젝트 생성
- 보안 관련 라이브러리 추가
- 보안 관련 설정 파일 만들기
- In-Memory 인증

## 25-1. 보안 관련 프로젝트 생성

(spring\_25\_1\_ex1\_springex)

1. 프로젝트 생성
2. 한글 처리
3. 보안 관련 라이브러리 추가
4. 스프링 설정 파일 분리
5. 보안 관련 설정 파일 만들기

## 25-2. 보안 관련 라이브러리 추가 (spring\_25\_1\_ex1\_springex)

The screenshot shows the Spring.io website's 'Quick Start' page for Spring Security. On the left, there's a sidebar with a shield icon and the text 'SPRING SECURITY' followed by a brief description: 'Protects your application with comprehensive and extensible authentication and authorization support.' In the center, the 'Quick Start' section features a 'Download' button for version 3.2.5 (CURRENT), with options for 'MAVEN' and 'GRADLE'. Below the download buttons, there's a snippet of XML code for Maven dependencies:

```
<!-- security -->
<dependency>
<groupId>org.springframework.security</groupId>
<artifactId>spring-security-web</artifactId>
<version>3.2.5.RELEASE</version>
</dependency>
```

At the bottom of this snippet is a 'copy to clipboard' button with a clipboard icon.

```
<!-- security -->
<dependency>
<groupId>org.springframework.security</groupId>
<artifactId>spring-security-web</artifactId>
<version>3.2.5.RELEASE</version>
</dependency>
```

```
<dependency>
<groupId>org.springframework.security</groupId>
<artifactId>spring-security-config</artifactId>
<version>3.2.5.RELEASE</version>
</dependency>
```

pom.xml에 추가

### 25-3. 보안 관련 설정 파일 만들기

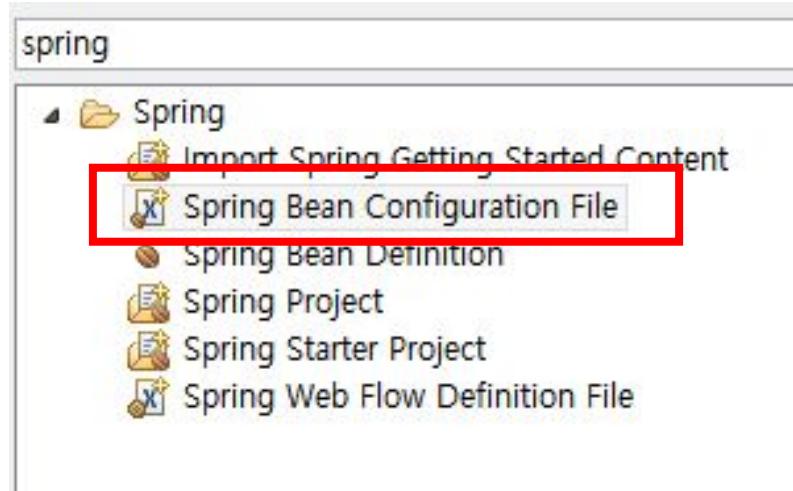
(spring\_25\_1\_ex1\_springex)

```
<!-- Processes application requests -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/spring/appServlet/servlet-context.xml
            /WEB-INF/spring/appServlet/security-context.xml
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

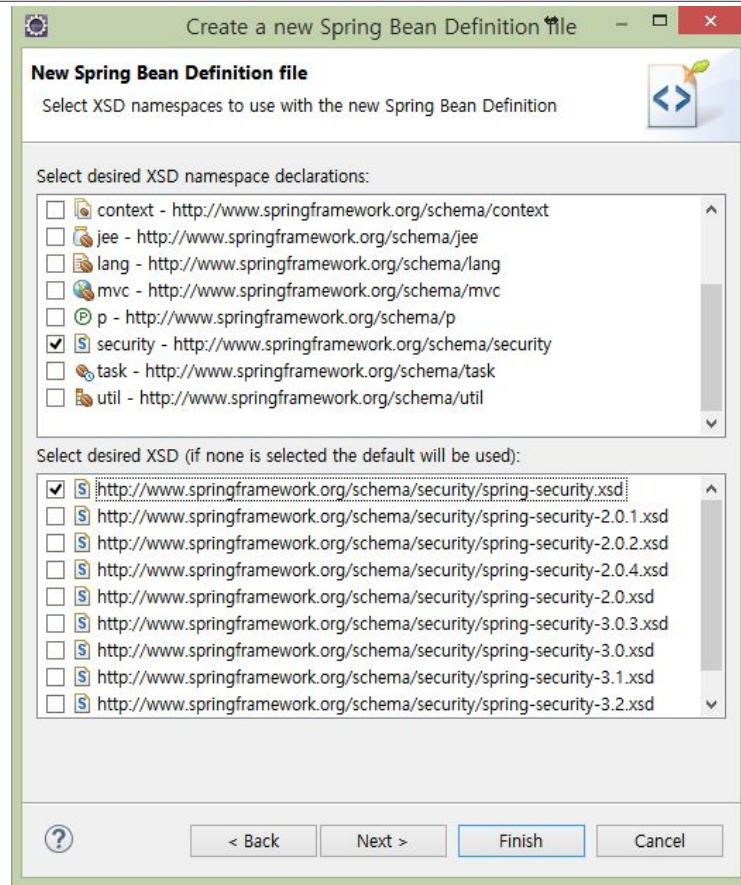
**web.xml**에 추가

## 25-3. 보안 관련 설정 파일 만들기

(spring\_25\_1\_ex1\_springex)



설정 파일 생성



## 25-4. In-Memory 인증

```
<security:http auto-config= "true">
    <security:intercept-url pattern= "/login.html/*" access= "ROLE_USER"/>
    <security:intercept-url pattern= "/welcome.html/*" access= "ROLE_ADMIN"/>
</security:http>

<security:authentication-manager>
    <security:authentication-provider>
        <security:user-service>
            <security:user name= "user" password= "123" authorities= "ROLE_USER"/>
            <security:user name= "admin" password= "123" authorities= "ROLE_ADMIN,ROLE_USER"/>
        </security:user-service>
    </security:authentication-provider>
</security:authentication-manager>

<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

(spring\_25\_1\_ex1\_springex)

1. 설정 파일 작성
2. web.xml에 filter 작성

# 26. Security-II

- 로그인 페이지 만들기
- 로그인, 로그아웃 상태 표시

## 26-1. 로그인 페이지 만들기

(spring\_26\_1\_ex1\_springex)

- 로그인 페이지는 직접 만들어 봅니다.

```
<security:http auto-config="true">
    <security:intercept-url pattern="/login.html/*" access="ROLE_USER"/>
    <security:intercept-url pattern="/welcome.html/*" access="ROLE_ADMIN"/>
</security:http>
```



```
<security:http auto-config="true">
    <security:form-login login-page="/loginForm.html"/>
    <security:intercept-url pattern="/login.html/*" access="ROLE_USER"/>
    <security:intercept-url pattern="/welcome.html/*" access="ROLE_ADMIN"/>
</security:http>
```

## 26-1. 로그인 페이지 만들기

(spring\_26\_1\_ex1\_springex)

- 로그인 페이지는 직접 만들어 봅니다.

```
<form action=" " method="post">
    ID : <input type="text" name="j_username" id="j_username"><br />
    PW : <input type="text" name="j_password" id="j_password"><br />
    <input type="submit" value="전송"><br />
</form>
```

**loginForm.jsp**

## 26-2. 로그인, 로그아웃 상태 표시

```
<form action="" method="post">  
    ID : <input type="text" name="j_username"> <br />  
    PW : <input type="text" name="j_password"> <br />  
    <input type="submit" value="LOGIN"> <br />  
</form>
```



```
<c:url value="j_spring_security_check" var="loginUrl"/>  
<form action="${loginUrl}" method="post">  
    <c:if test="${param.ng != null}">  
        <p>  
            Login NG! <br />  
            <c:if test="${SPRING_SECURITY_LAST_EXCEPTION != NULL}">  
                message : <c:out value="${SPRING_SECURITY_LAST_EXCEPTION.message}" />  
            </c:if>  
        </p>  
    </c:if>  
    ID : <input type="text" name="j_username"> <br />  
    PW : <input type="text" name="j_password"> <br />  
    <input type="submit" value="LOGIN"> <br />  
</form>
```

- 로그인에 실패 했을 경우 실패 메시지를 출력 합니다.

(spring\_26\_2\_ex1\_springex)

**loginForm.jsp**

## 26-2. 로그인, 로그아웃 상태 표시

```
<security:http auto-config= "true">
    <security:form-login login-page= "/loginForm.html"/>
    <security:intercept-url pattern= "/login.html/*" access= "ROLE_USER"/>
    <security:intercept-url pattern= "/welcome.html/*" access= "ROLE_ADMIN"/>
</security:http>
```



```
<security:http auto-config= "true">
    <security:form-login login-page= "/loginForm.html"
        authentication-failure-url= "/loginForm.html?ng"/>
    <security:intercept-url pattern= "/login.html/*" access= "ROLE_USER"/>
    <security:intercept-url pattern= "/welcome.html/*" access= "ROLE_ADMIN"/>
</security:http>
```

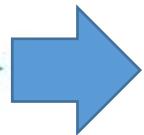
**security-context.xml**

1. 로그인에 실패 했을 경우 실패 메시지를 출력 합니다.

## 26-2. 로그인, 로그아웃 상태 표시

(spring\_26\_2\_ex1\_springex)

```
<h1>login.jsp</h1>
```



```
<h1>login.jsp</h1>

    <c:if test= "${not empty pageContext.request.userPrincipal }">
        <p> is Log-In</p>
    </c:if>

    <c:if test= "${empty pageContext.request.userPrincipal }">
        <p> is Log-Out</p>
    </c:if>

    USER ID : ${pageContext.request.userPrincipal.name}<br/>
    <a href= "${pageContext.request.contextPath}/j_spring_security_logout">Log Out</a> <br />
```

**login.jsp**

**26-2. 로그인, 로그아웃 상태 표시**  
(spring\_26\_2\_ex1\_springex)

**login.jsp**

is Log-In

USER ID : user

[Log Out](#)

**Hello world!**

The time on the server is 2015년 2월 25일 (수) 오후 11시 23분 50초.



# 27. Security-II

보안 관련 taglibs 사용 방법

## 27-1. 보안 관련 taglibs 사용 방법 (spring\_27\_1\_ex1\_springex)

### Dependencies

#### Dependencies

- spring-context : \${org.springframework-version}
- spring-webmvc : \${org.springframework-version}
- aspectjrt : \${org.aspectj-version}
- slf4j-api : \${org.slf4j-version}
- jcl-over-slf4j : \${org.slf4j-version} [runtime]
- slf4j-log4j12 : \${org.slf4j-version} [runtime]
- log4j : 1.2.15 [runtime]
- javax.inject : 1
- jstl : 1.2
- tiles-jsp : 2.2.2
- spring-jdbc : 4.1.4.RELEASE
- junit : 4.7 [test]
- spring-security-config : 3.2.5.RELEASE
- spring-security-core : 3.2.5.RELEASE
- spring-security-web : 3.2.5.RELEASE
- spring-security-taglibs : 3.2.4.RELEASE**

pom.xml

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="s" %>
<%-- USER ID : ${pageContext.request.userPrincipal.name}<br/> --%>
USER ID : <s:authentication property="name"/><br/>
<a href="${pageContext.request.contextPath}/_spring_security_logout">Log Out</a> <br />
```

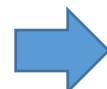
login.jsp

## 27-1. 보안 관련 taglibs 사용 방법

(spring\_27\_1\_ex1\_springex)

```
<c:if test='${not empty pageContext.request.userPrincipal}'>
<p> is Log-In</p>
</c:if>

<c:if test='${empty pageContext.request.userPrincipal}'>
<p> is Log-Out</p>
</c:if>
```



```
<s:authorize ifAnyGranted="ROLE_USER">
<p> is Log-In</p>
</s:authorize>

<s:authorize ifNotGranted="ROLE_USER">
<p> is Log-Out</p>
</s:authorize>
```

login.jsp

# 28. Mybatis-I

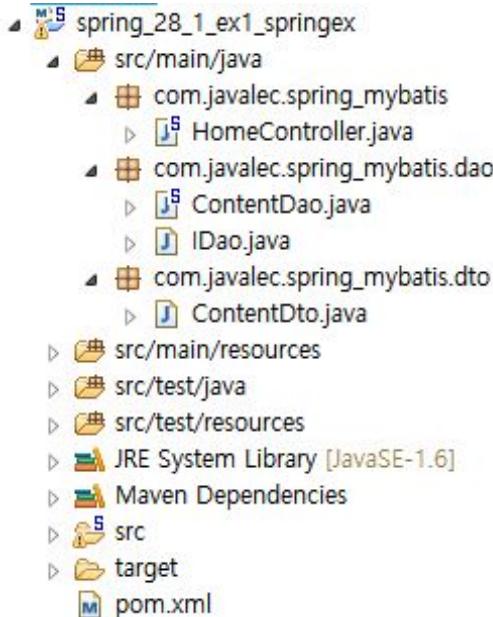
Mybatis를 사용하기 위한 기본 설정

## 28-1. Mybatis를 사용하기 위한 기본 설정 (spring\_28\_1\_ex1\_springex)

```
create table board (
    mId number(4) primary key,
    mWriter varchar2(100),
    mContent varchar2(300)
);

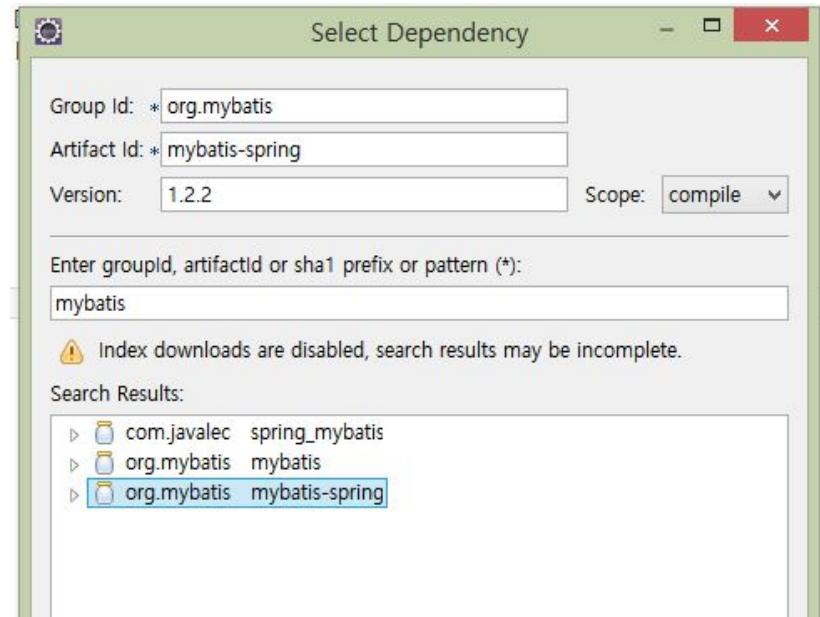
commit;

create sequence board_seq;
```



DataBase 제작

프로젝트 생성

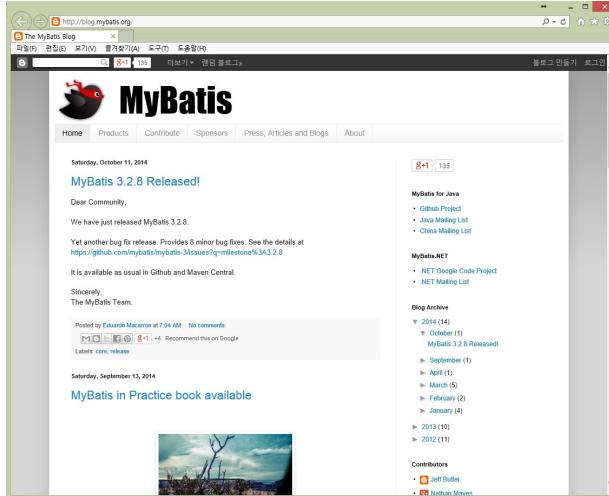


Mybatis libs 다운로드 – pom.xml

# 29. Mybatis-II

Mybatis를 이용한 리스트 출력

## 29-1. Mybatis를 이용한 리스트 출력 (spring\_29\_1\_ex1\_springex)



Mybatis 사이트

```
<beans:bean id= "sqlSessionFactory" class= "org.mybatis.spring.SqlSessionFactoryBean">
    <beans:property name= "dataSource" ref= "dataSource"></beans:property>
    <beans:property name= "mapperLocations" value= "classpath:com/javalec/spring_mybatis/mapper/*Mapper.xml"></beans:property>
</beans:bean>

<beans:bean id= "sqlSession" class= "org.mybatis.spring.SqlSessionTemplate">
    <beans:constructor-arg index= "0" ref= "sqlSessionFactory"></beans:constructor-arg>
</beans:bean>
```

스프링 설정 파일

## 29-1. Mybatis를 이용한 리스트 출력

(spring\_29\_1\_ex1\_springex)

The screenshot shows an IDE interface with two code files:

- ContentDao.xml** (Left Tab): An XML configuration file for MyBatis. It defines a mapper with namespace "com.javalec.spring\_mybatis.dao.IDao". It contains three SQL statements: a select statement with id "listDao", an insert statement with id "writeDao", and a delete statement with id "deleteDao".
- IDao.java** (Right Tab): A Java interface named IDao. It imports java.util.ArrayList. The interface methods correspond to the SQL statements in the XML file: listDao(), writeDao(String mWriter, String mContent), viewDao(String strID), and deleteDao(String bld). The interface ends with a closing brace at line 14.

Blue arrows point from the XML file's select, insert, and delete statements to the corresponding method definitions in the Java interface. The XML file also includes DOCTYPE and PUBLIC declarations.

```
ContentDao.xml (Left):
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.javalec.spring_mybatis.dao.IDao">
  <select id="listDao"> ←
  </select>
  <insert id="writeDao"> ←
  </insert>
  <delete id="deleteDao"> ←
  </delete>
</mapper>
```

```
IDao.java (Right):
```

```
package com.javalec.spring_mybatis.dao;
import java.util.ArrayList;
public interface IDao {
    public ArrayList<ContentDto> listDao();
    public void writeDao(String mWriter, String mContent);
    public ContentDto viewDao(String strID);
    public void deleteDao(String bld);
}
```

## 29-1. Mybatis를 이용한 리스트 출력

(spring\_29\_1\_ex1\_springex)

```
<select id= "listDao" resultType= "com.javalec.spring_mybatis.dto.ContentDto">
    SELECT * FROM BOARD ORDER BY MID DESC
</select>
```

.xml 파일

```
@Autowired
private SqlSession sqlSession;
```

```
@RequestMapping("/list")
public String list(Model model) {
    // ArrayList<ContentDto> dtos = dao.listDao();
    IDao dao = sqlSession.getMapper(IDao.class);
    // ArrayList<ContentDto> dtos = dao.listDao();
    model.addAttribute("list", dao.listDao());
}

return "/list";
}
```

dao 객체 사용처

# 30. Mybatis-III

-Mybatis를 이용한 글작성 및 삭제

-소스 정리

### 30-1. Mybatis를 글작성 및 삭제

(spring\_30\_1\_ex1\_springex)

```
<insert id="writeDao">
    INSERT INTO BOARD (MID, MWRITER, MCONTENT) VALUES (BOARD_SEQ.NEXTVAL, #{param1}, #{param2})
</insert>
```

```
<delete id="deleteDao">
    DELETE FROM BOARD WHERE MID = #{param1}
</delete>
```

END