

SPRING FRAMEWORK

Eclipse를 이용한 Spring Web Project

주요 강의 내용

2

- Dependency Injection
- Aspect Oriented Programming
- Spring JDBC
- MyBatis를 이용한 Database Programming
- Spring MVC
- Tiles2를 이용한 Template page

목차

3

1. 개발환경 설정
2. Spring Framework 개요
3. Dependency Injection
4. Aspect Oriented Programming
5. Database 연동
6. Spring Web MVC
7. Spring을 이용한 웹 어플리케이션

1. 개발환경 설정

1. Eclipse IDE
2. Spring Framework
3. MyBatis Framework
4. UML
5. ERD

- 개발 환경 설정
 - ▣ 이클립스 다운로드 및 설치
 - ▣ Version History
 - ▣ Compare package
 - ▣ Popular Projects & Countries
 - ▣ 이클립스 기본화면 구성
 - ▣ 이클립스 Source Code Editor
 - ▣ 줄 바꿈 기능
 - ▣ 폰트 크기 및 라인번호 설정
- Spring IDE 및 Framework 사용 설정
 - ▣ Spring IDE 플러그인 설치
 - ▣ Spring Framework 라이브러리 다운로드
 - ▣ Spring 라이브러리 설정
 - ▣ Simple Spring Application Test

- <http://www.eclipse.org>
- JDK가 설치되어 있어야 합니다.
 - ▣ <http://java.sun.com>
 - ▣ <http://www.oracle.com/technetwork/java/index.html>
 - ▣ JDK7 설치했는데...
 - Unsupported major.minor version 51.0 ← 이런 에러를 보기 싫으시면 JDK를 6.0으로 다시 설치하세요.
 - 이클립스의 런타임환경이 JDK 6 이하 일 경우 위와 같은 에러가 발생합니다.
 - 이클립스를 Indigo(3.7) 버전 이상 설치하세요.
- 다운받아 압축만 풀면 실행이 가능합니다.
 - ▣ C:\java 폴더에 풀어 놓으세요.



JDK 설치하지 않고 이클립스를 실행시킬 때 발생하는 에러메시지

1.1.1 Eclipse Version History

- 2007
 - ▣ Eclipse Europa Packages (v 3.3)
- 2008
 - ▣ Eclipse Ganymede Packages (v 3.4)
 - ▣ Eclipse Ganymede SR1 Packages (v 3.4.1)
 - ▣ Eclipse Ganymede SR2 Packages (v 3.4.2)
- 2009
 - ▣ Eclipse Galileo Packages (v 3.5)
 - ▣ Eclipse Galileo SR1 Packages (v 3.5.1)
 - ▣ Eclipse Galileo SR2 Packages (v 3.5.2)
- 2010
 - ▣ Eclipse Helios Packages (v 3.6)
 - ▣ Eclipse Helios SR1 Packages (v 3.6.1)
 - ▣ Eclipse Helios SR2 Packages (v 3.6.2)
- 2011
 - ▣ Eclipse Indigo Packages (v 3.7)
 - ▣ Archived Eclipse Platform Releases
 - ▣ Archived Webtools Releases
- 2012.6
 - ▣ Eclipse Juno Packages (4.2)

1.1.2 Compare packages

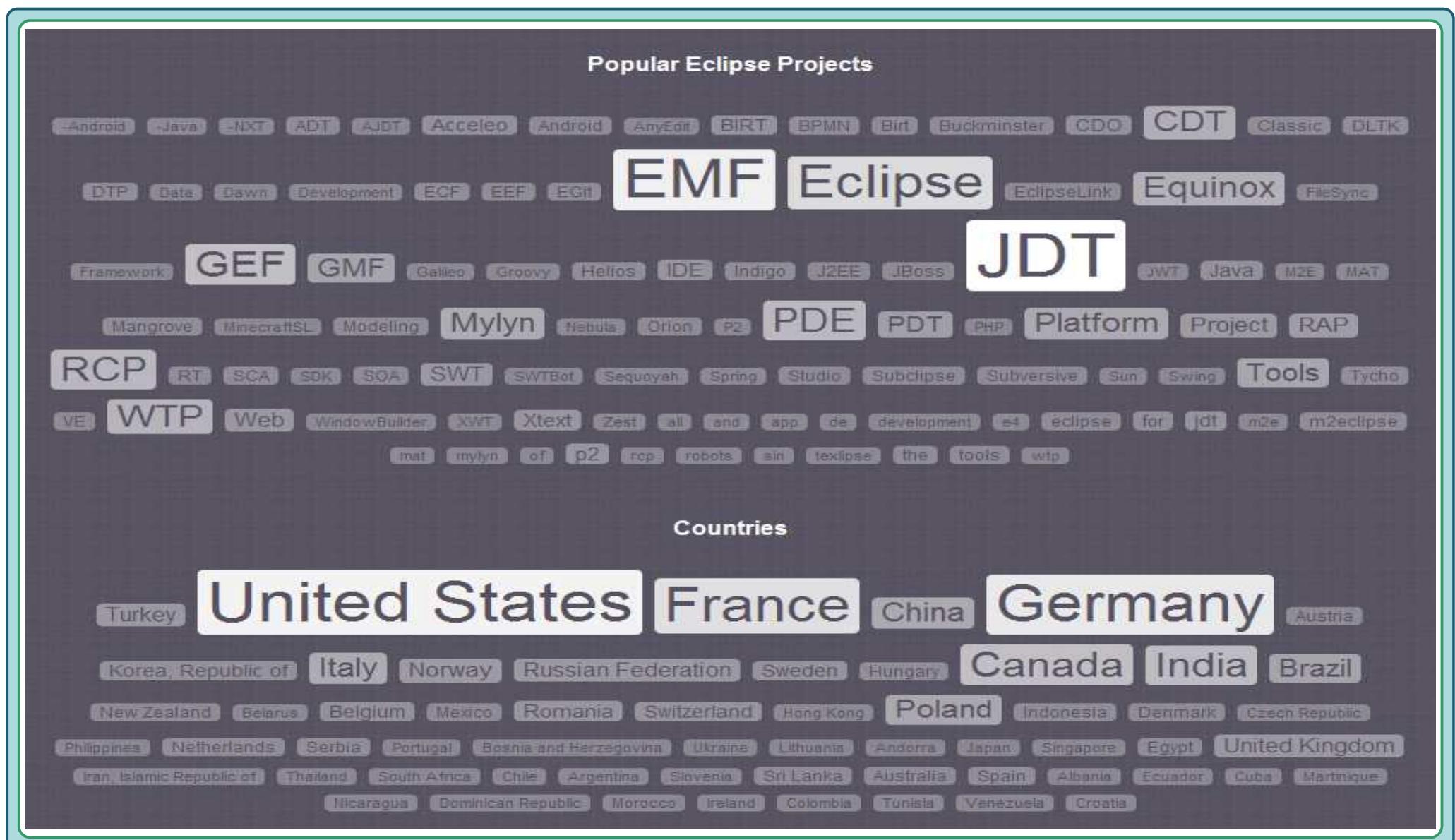
| |  Java |  Java EE |  C/C++ |  C/C++ Linux |  RCP/RAP |  Modeling |  JEE BIRT |  Parallel |  Scout |  Testers |  Javascript |  Classic |
|---------------|--|---|---|---|---|---|--|--|---|---|--|---|
| RCP/Platform | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| CVS | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| EGit | | | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | |
| EMF | ✓ | ✓ | | | | ✓ | ✓ | | | | | |
| GEF | ✓ | ✓ | | | | ✓ | ✓ | | | | | |
| JDT | ✓ | ✓ | | | ✓ | ✓ | ✓ | | ✓ | | | ✓ |
| Mylyn | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Web Tools | ✓ | | | | | | ✓ | | | | | ✓ |
| Linux Tools | | | ✓ | ✓ | | | | | ✓ | | | |
| Java EE Tools | ✓ | | | | | | ✓ | | | | | |
| XML Tools | ✓ | ✓ | | | ✓ | | ✓ | ✓ | ✓ | | | |
| RSE | ✓ | | ✓ | ✓ | | | ✓ | ✓ | ✓ | | | |
| EclipseLink | ✓ | | | | | | ✓ | | | ✓ | | |
| PDE | ✓ | | | | ✓ | ✓ | ✓ | ✓ | | | | ✓ |
| Datatools | ✓ | | | | | | ✓ | | | | | |
| CDT | | | ✓ | ✓ | | | | ✓ | | | | |
| BIRT | | | | | | | | ✓ | | | | |
| GMF | | | | | | ✓ | | | | | | |
| PTP | | | | | | | | | ✓ | | | |
| MDT | | | | | | ✓ | | | | | | |
| Scout | | | | | | | | | | ✓ | | |
| Jubula | | | | | | | | | | | ✓ | |
| RAP | | | | | ✓ | | | | | | | |
| WindowBuilder | ✓ | | | | | | | | | | | |
| Maven | ✓ | | | | | | | | | | | |

Legend:

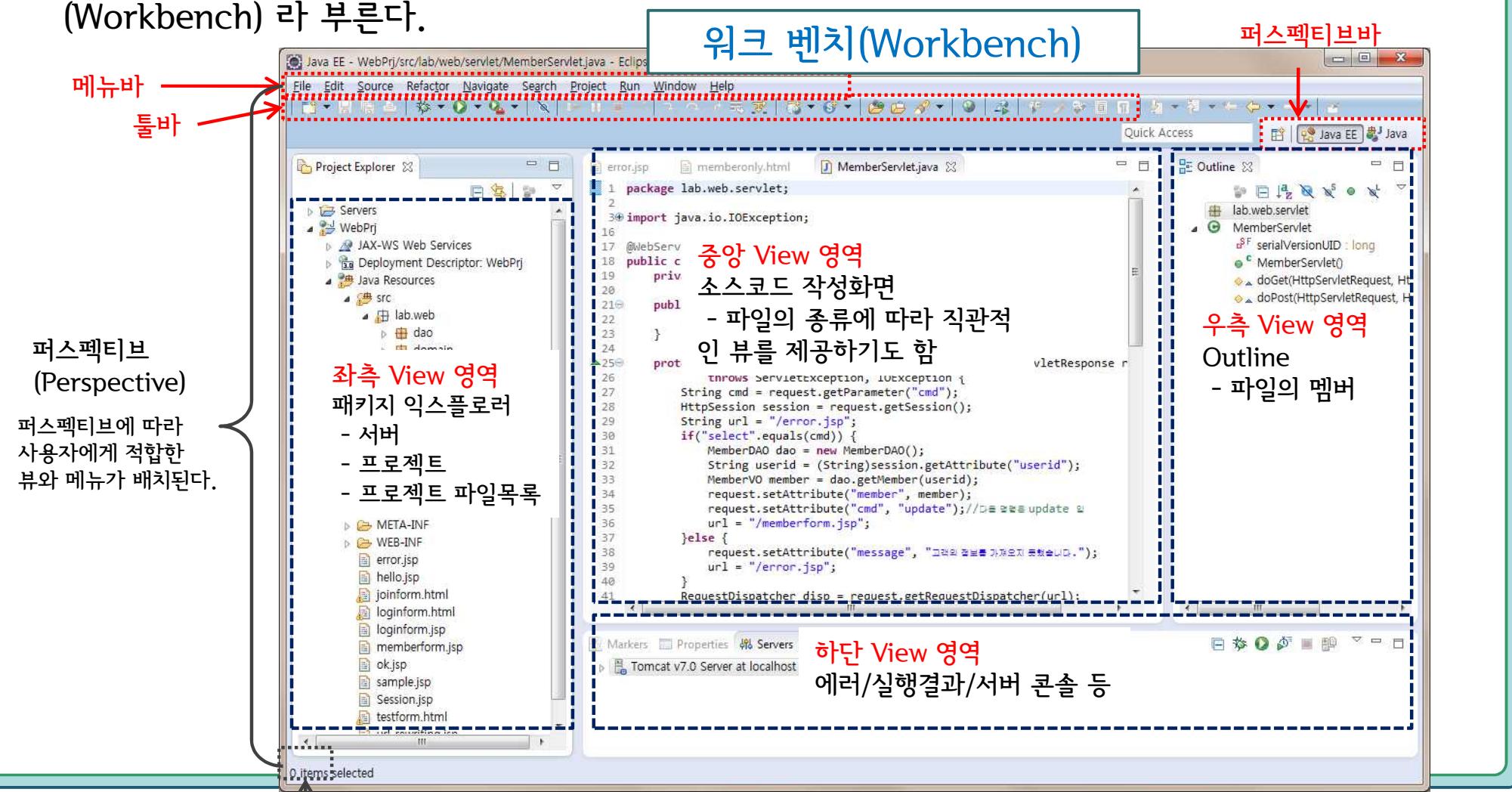
✓ Included (with Source) ✓ Included ✓ Partially Included alist.co.kr

1.1.3 Popular Eclipse Projects & Countries

9



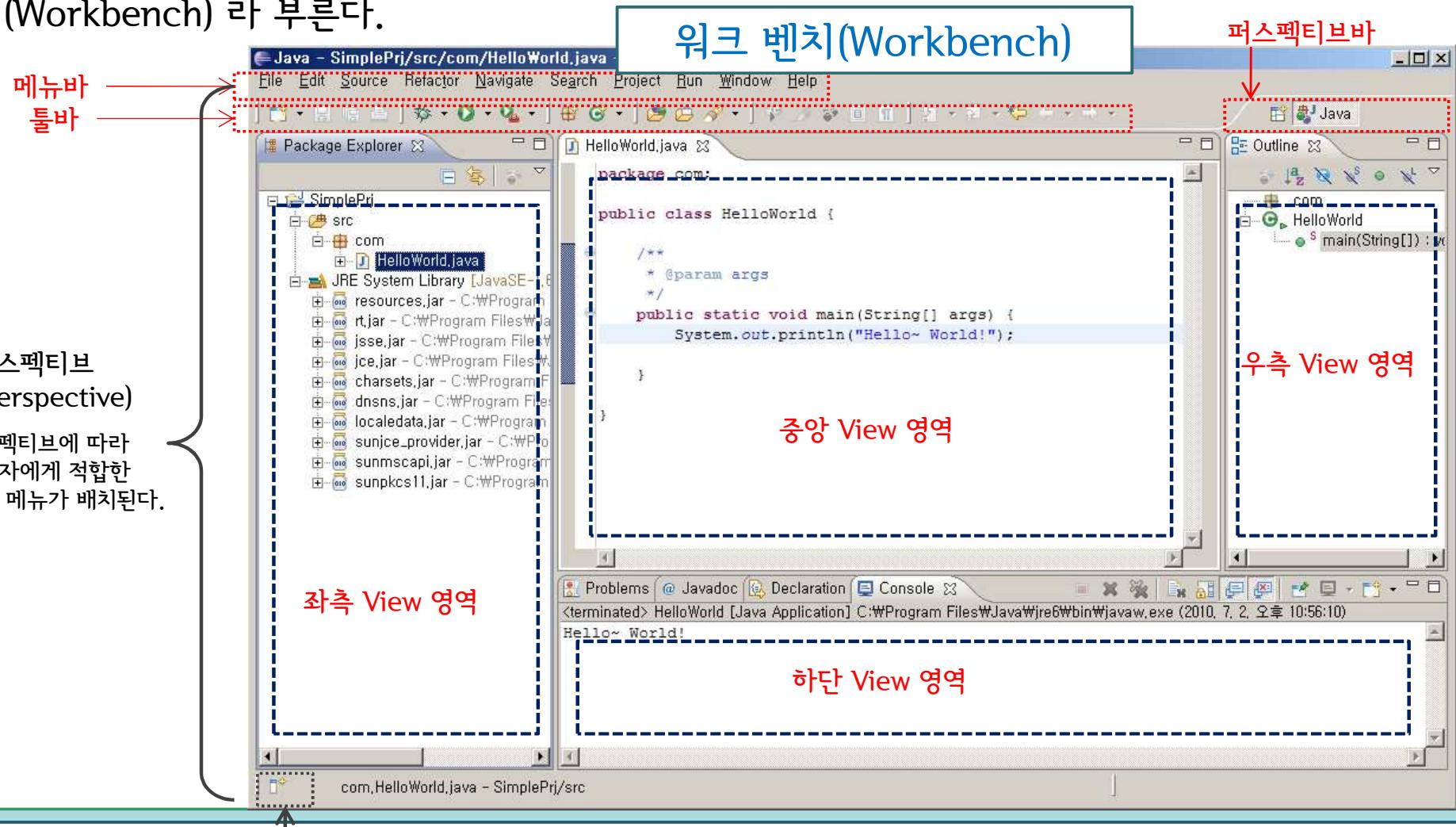
- Eclipse 기본화면은 메뉴바, 툴바, 퍼스펙티브바, 4개의 View 영역으로 크게 구분된다. 이 4개의 뷰를 포함하는 전체 영역을 퍼스펙티브(Perspective)라고 하고, Eclipse Window 전체를 워크벤치(Workbench)라 부른다.



Fast 뷰 : 잘 사용하지 않는 화면들을 최소화 시켜놓고 필요한 시점에만 잠깐 활성화 시킬 때 사용된다.

1.1.4 이클립스 기본화면 구성(Indigo)

- Eclipse 기본화면은 메뉴바, 툴바, 퍼스펙티브바, 4개의 View 영역으로 크게 구분된다. 이 4개의 뷰를 포함하는 전체 영역을 퍼스펙티브(Perspective)라고 하고, Eclipse Window 전체를 워크벤치(Workbench)라 부른다.

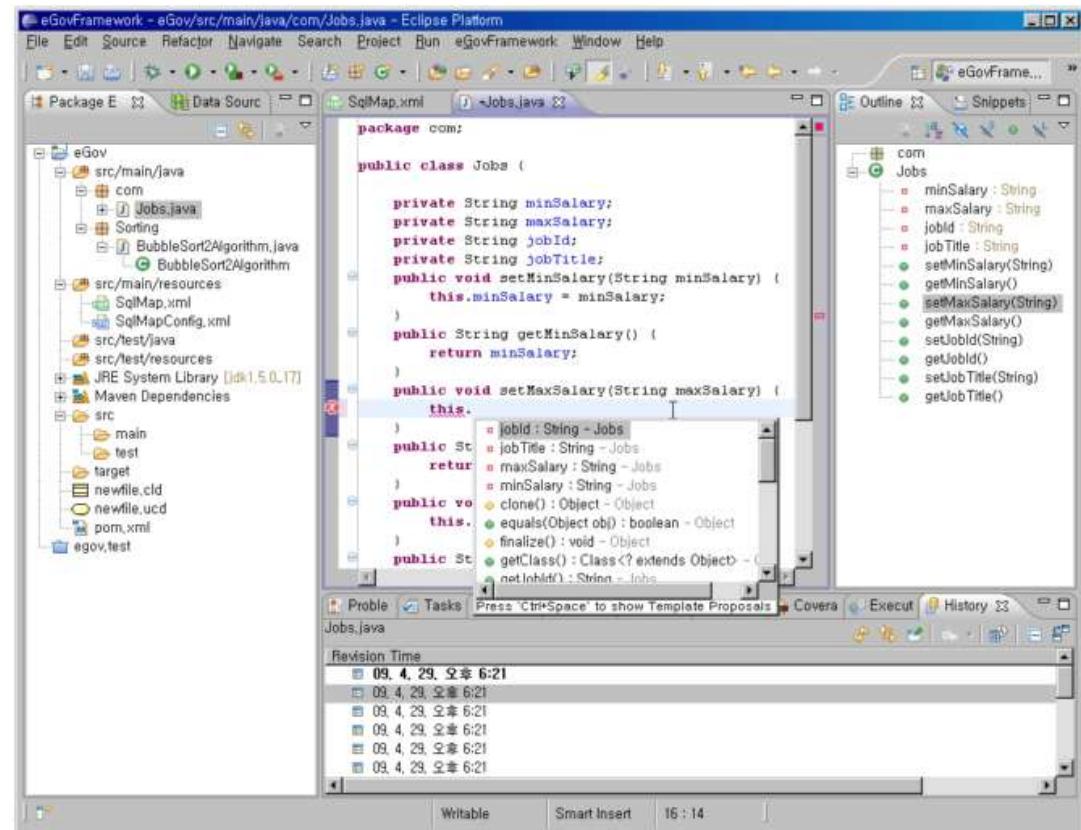


Fast 뷰 : 잘 사용하지 않는 화면들을 최소화 시켜놓고 필요한 시점에만 잠깐 활성화 시킬 때 사용된다.

1.1.5 이클립스 Source Code Editor

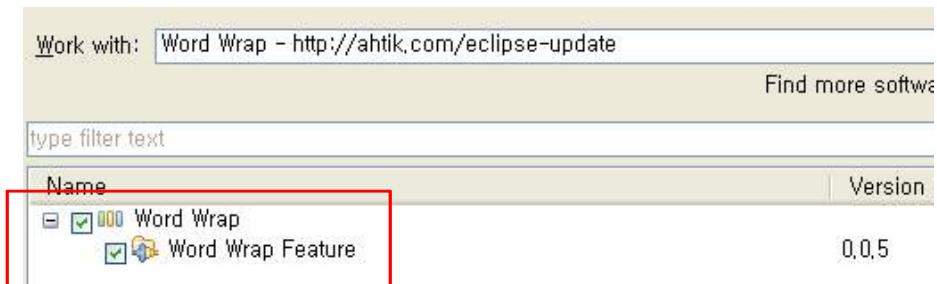
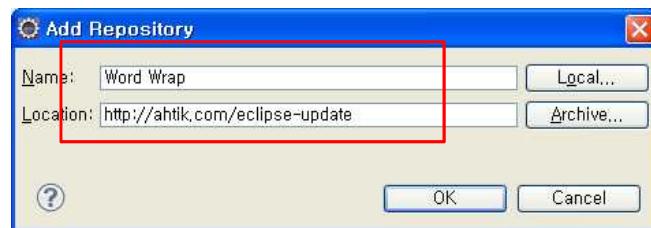
- Source Code Editor는 Eclipse내 Eclipse Java development tools(JDT)를 통해 지원

- Code Assist : Code 도움말 기능
- Quick Fix : Code에러 빠른 수정 가능
- 코드 스타일 적용
- 코드 템플릿 처리
- Quick Type Hierarchy : 상속 구조 표시
- Quick Outline : 코드 개요 표시
- 소스코드 네비게이션
- Mark Occurrences
 - 커서가 위치한 지역변수, 상수, 필드, 메서드, 클래스 등의 코드 요소가 나타나는 곳이 모두 하이라이트 된다
- 로컬 히스토리: file 수정 이력 비교/복구

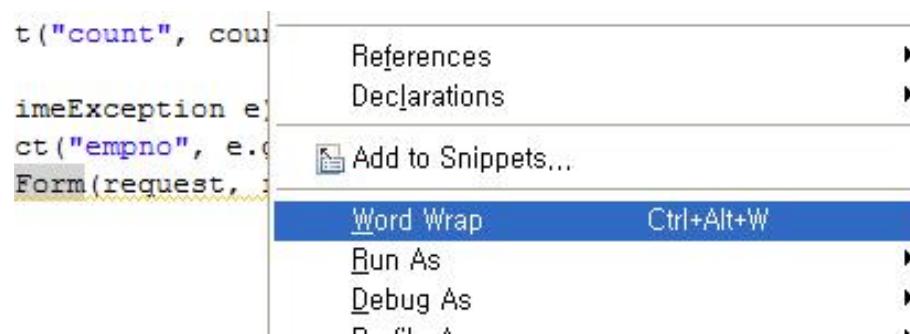


Ctrl + Space 를 자주 이용하세요.

- [Help] -> [Install New Software]
 - ▣ <http://ahtik.com/eclipse-update/>



- 플러그인 설치 후에 줄 바꿈 기능 사용하기 위해서는
 - ▣ 소스코드에서 오른쪽 클릭 후 [Word Wrap] 메뉴 선택



1. 개발 환경 설정

1.2 Spring Framework 설치

- Spring IDE 플러그인 설치
- Spring Framework 라이브러리 다운로드
- Spring 라이브러리 설정
- Simple Spring Application Test

□ Help -> Eclipse Marketplace

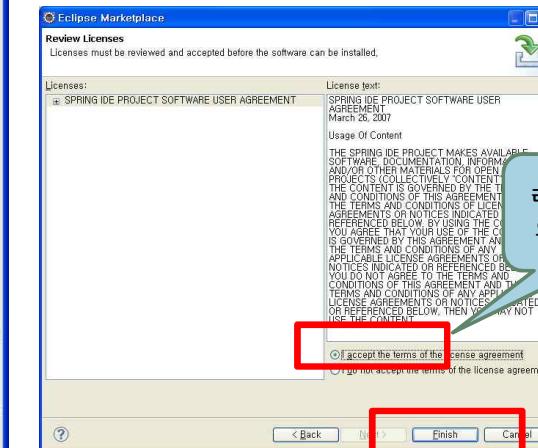
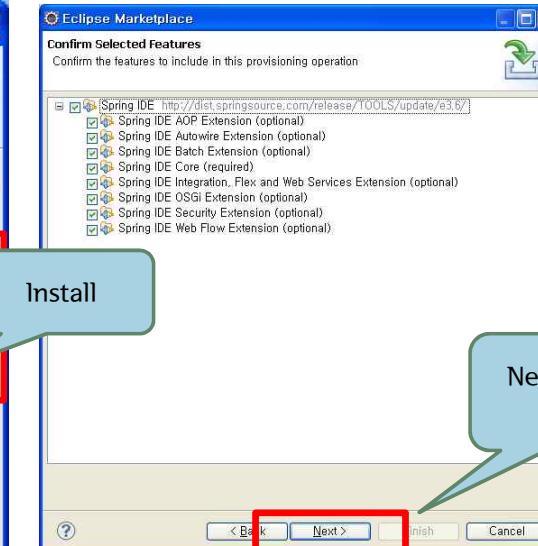
Spring으로
검색

이클립스 버전에
맞는 STS 선택

Install

Next 버튼
클릭

라이센스 동
의 후 Next



1.2.1 Spring IDE plug-in 설치(Install New Software)

17

□ [Help] -> [Install New Software]

□ Add...

- Name: Spring IDE
- Location: http://springide.org/updatesite

□ 다음 항목 체크 후 Next >

- Core / Spring IDE
- Extensions (Incubation) / Spring IDE
- Extensions / Spring IDE
- Resources / Spring IDE

□ Install Details창에서 설치될 항목 다시 확인 후 Next >

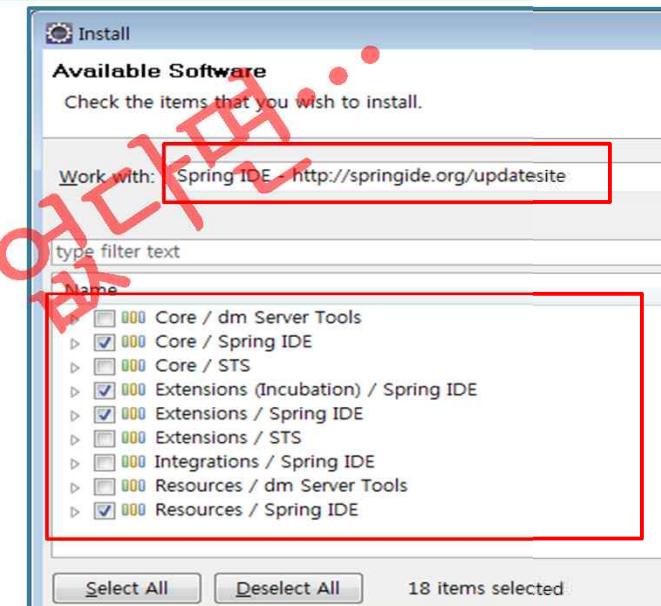
□ Licenses 동의 후 Finish

□ [Run in Background] 버튼을 클릭하면 다른 작업을 진행하면서 설치 작업을 진행할 수 있습니다.

- 진행 상황을 다시 보고 싶으면 이클립스 우측 하단의 [Show background...]버튼 클릭
- 설치 도중 unsigned content를 포함한다는 Security Warning 창이 뜨면 [OK] 눌러 무시

□ 설치 확인은 Help -> About Eclipse 에서 [Spring IDE]버튼 클릭

- Eclipse IDE for Java EE Developers Indigo(3.7) 버전에 Spring IDE 2.6.0설치(11년 11월)



[Show background...]



[Spring IDE]

□ Spring Framework 다운

- ▣ <http://www.springsource.org/download>

- Spring Framework 3.1.2
 - requires Java 1.5+
- Spring Framework 2.5.6
 - compatible with Java 1.4+
- Spring Framework 2.0.8
 - compatible with Java 1.3+

- ▣ 다운로드 받은 파일을 <C:/java/workspace/libraries/spring> 폴더에 압축 풀어 놓으세요.

1

SPRING DOWNLOADS

The Spring projects are all available from the SpringSource [Download Center](#).

GET THE LATEST SPRING RELEASES HERE

2

- Spring Framework **3.1.0.RELEASE** is the current production release.
 - [Download](#) | [Changelog](#)
- Spring Framework **2.5.6.SEC02** is the latest Spring 2.5.x release.
 - [Download](#) | [Changelog](#)
- Spring Framework **2.0.8** is the latest Spring 2.0.x release (current stable release).
 - [Download](#) | [Changelog](#) | [Announcement](#)
- Spring Framework **sample projects** are available for checking out.
 - [Browse spring-samples repository](#)
- Spring Framework **nightly snapshots** are available for testing.
 - [Download](#)

COMMUNITY DOWNLOADS

Please register below so we can notify you of community newsletters and roadmap information.

All fields are required.

| | |
|------------------|--------------------------------|
| First Name: | <input type="text"/> |
| Last Name: | <input type="text"/> |
| Company: | <input type="text"/> |
| Title: | <input type="text"/> |
| Role: | <input type="text"/> select... |
| Email: | <input type="text"/> |
| Phone: | <input type="text"/> |
| Zip/Postal Code: | <input type="text"/> |

[Submit](#)

(I'd rather not fill in the form. Just take me to the download page)

SpringSource respects your [privacy](#).

4

COMMUNITY DOWNLOADS

Spring Framework

- Latest Development release: 3.2.0
- Latest GA release: 3.1.2.RELEASE

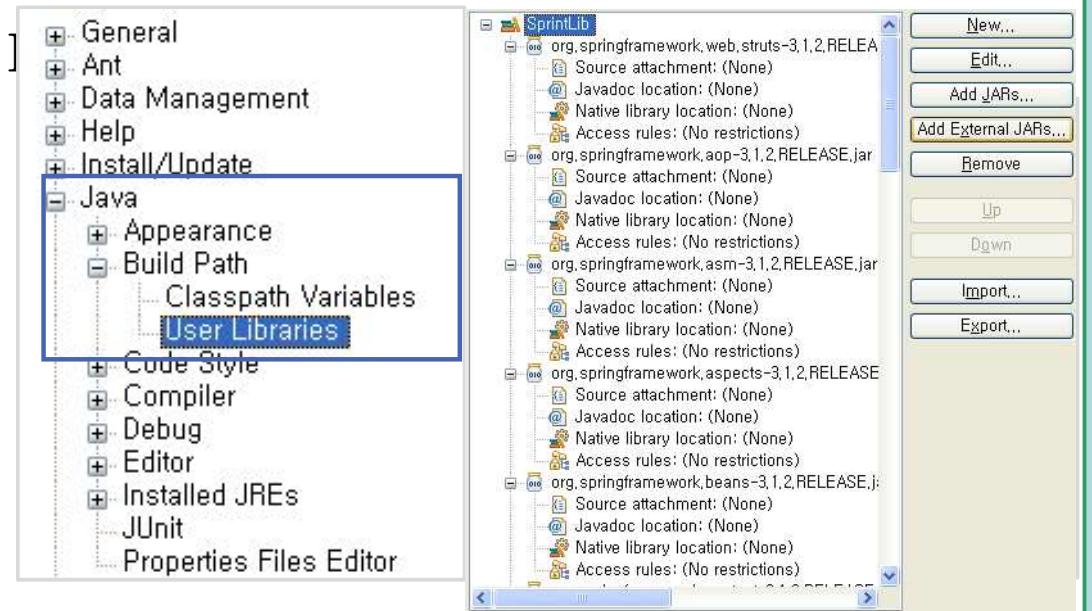
[spring-framework-3.1.2.RELEASE-with-docs.zip \(sha1\) 51.9 MB](#)
[spring-framework-3.1.2.RELEASE.zip \(sha1\) 27.4 MB](#)

◦ [More >>](#)

3

5

- [Window] -> [Preferences]
 - ▣ [Java] - [Build Path] - [User Libraries]
 - [New]
 - Spring Lib 입력 후 [OK]
 - 생성된 라이브러리 클릭 후
 - [Add External JARs...]
 - dist 폴더 안의 jar 파일들 선택 후 OK



- 위에서 생성된 라이브러리 사용하기 위해서...
 - ▣ 프로젝트의 컨텍스트 메뉴(우클릭) -> [Build Path] -> [Configure Build Path]
 - ▣ Libraries 탭에서 [Add Library...]
 - User Library 선택 후 [Next >]
 - 앞에서 만든 Spring Lib 체크 후 [Finish]
 - ▣ [OK]

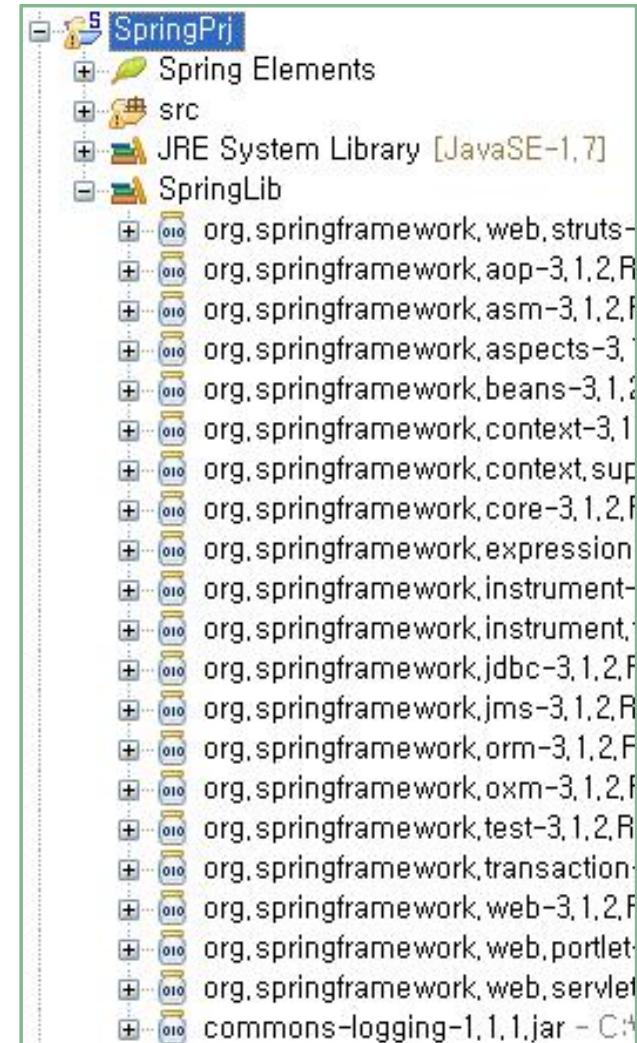


<http://commons.apache.org>에서
Logging API 다운로드 해서 User Library에 추가해야 함

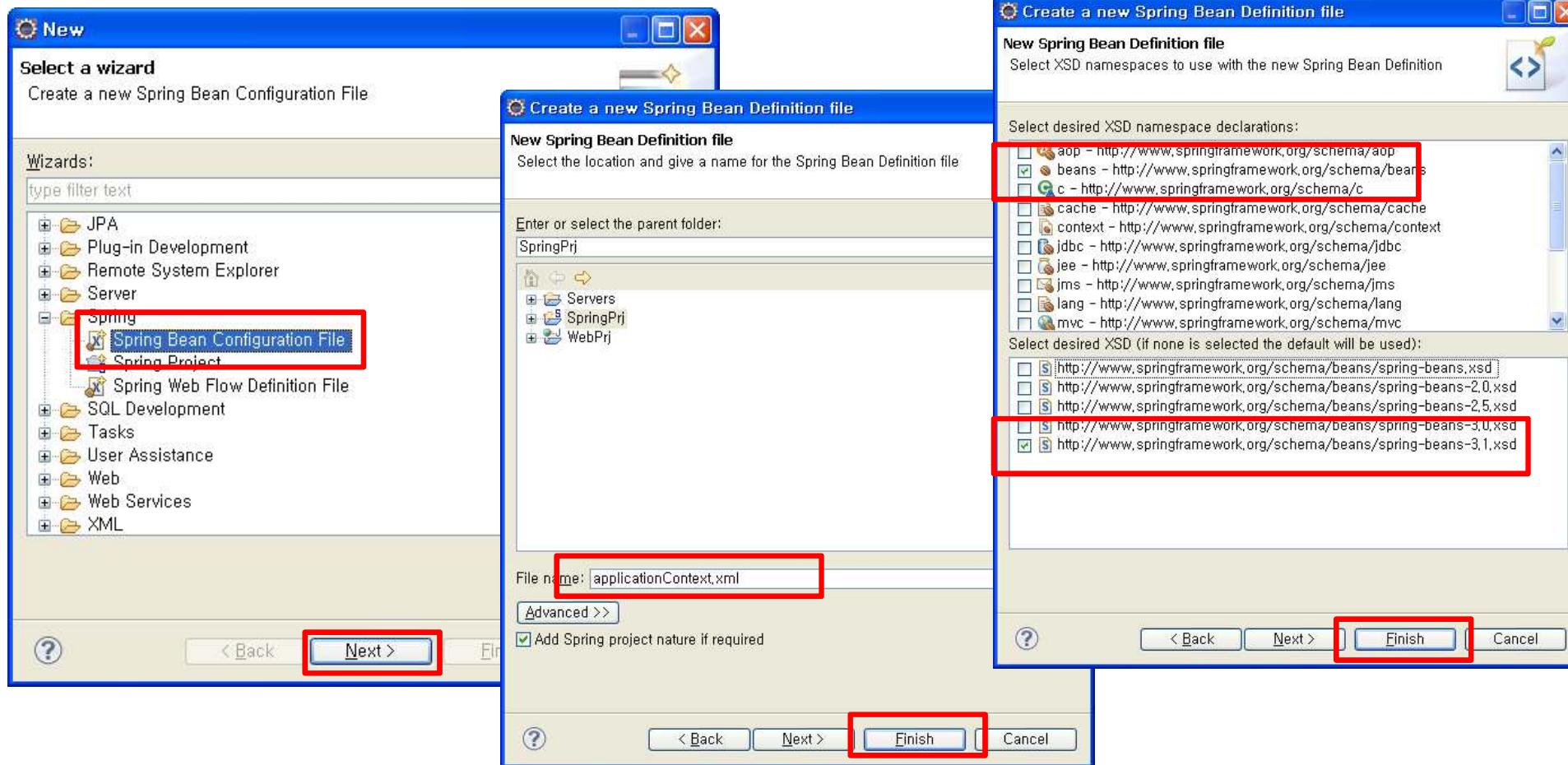
- New -> Other
 - ▣ Spring -> Spring Project

- 프로젝트 생성 후
 - ▣ 빌드패스에 라이브러리 추가해줘야 함
 - ▣ 프로젝트에서 마우스 오른쪽버튼 메뉴 중에서
 - > Build Path
 - > Configure Build Path
 - > Libraries 탭에서 Add Library 버튼 클릭

 - ▣ * 웹 프로젝트의 경우 WEB-INF/lib 폴더에 복사해 두면 자동으로 빌드패스에 추가됨



- New -> Other
 - ▣ Spring -> Spring Bean Configuration File



- HelloBean.java(파키지 선언해야 함)
 - interface 임

```
void sayHello(String name);
```
- HelloBeanImpl.java (파키지 선언해야 함)

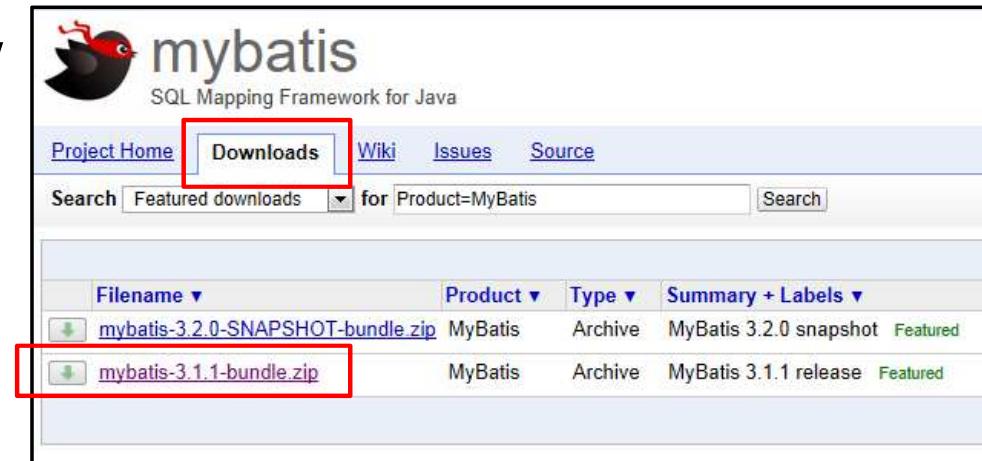

```
public void sayHello(String name) {
    System.out.println("안녕하세요. " + name + "씨!");
```
- applicationContext.xml(src 폴더에 작성한다)


```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">
  <bean id="helloBean" class="x.y.HelloBeanImpl"/>
</beans>
```
- HelloSpringApp.java


```
public static void main(String[] args) {
    Resource resource = new ClassPathResource("applicationContext.xml");
    BeanFactory factory = new XmlBeanFactory(resource);
    HelloBean bean = (HelloBean)factory.getBean("helloBean");
    bean.sayHello("Spring");
}
```

□ 다운로드

- ▣ <http://code.google.com/p/mybatis/>
 - http://ibatis.apache.org 에서 옮겨짐
 - MyBatis로 이름이 바뀜
- ▣ Download
 - MyBatis Core Framework 클릭
 - mybatis-3.1.1-bundle.zip 다운로드



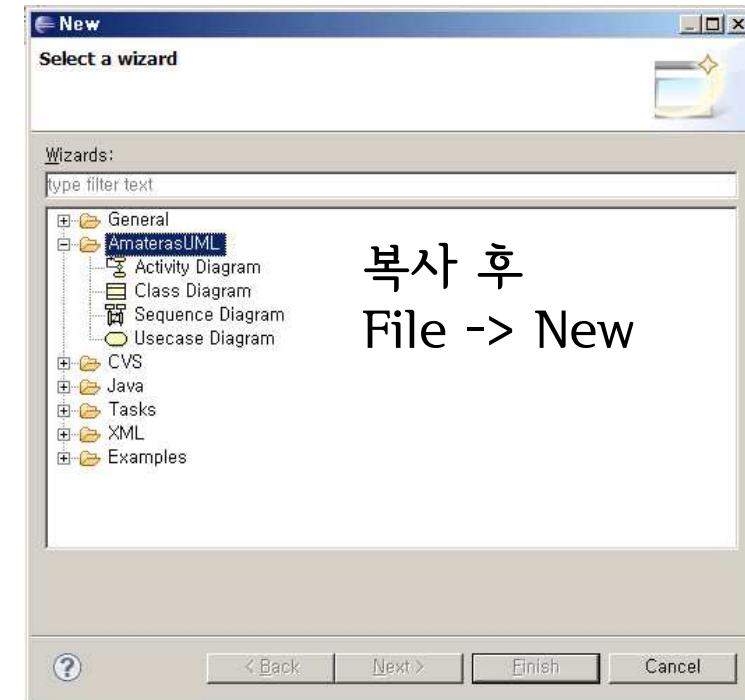
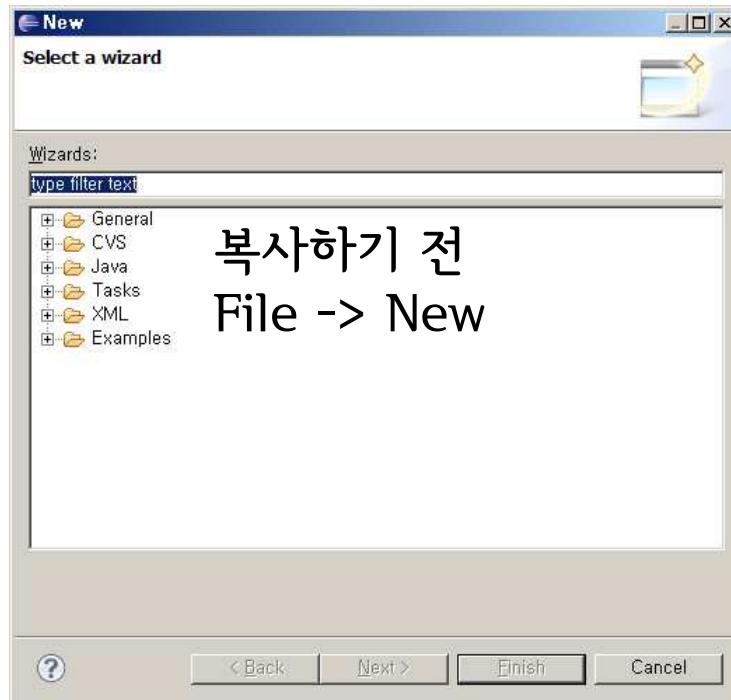
□ 설치

- ▣ MyBatis 데이터 매퍼 프레임워크 설치는 클래스패스에 JAR 파일을 넣어주는 간단한 작업이다. 이 클래스 패스는 JVM 시작시 지정된 클래스패스(java -cp 인자로 지정된)나 웹애플리케이션의 /WEB-INF/lib 디렉토리가 될 수도 있다.
 - 이클립스 프로젝트의 Build Path에 /lib 폴더 안의 mybatis-3.0.5.jar 파일 추가
 - optional 폴더 안의 commons-logging-1.1.1.jar 파일도 추가해야 함

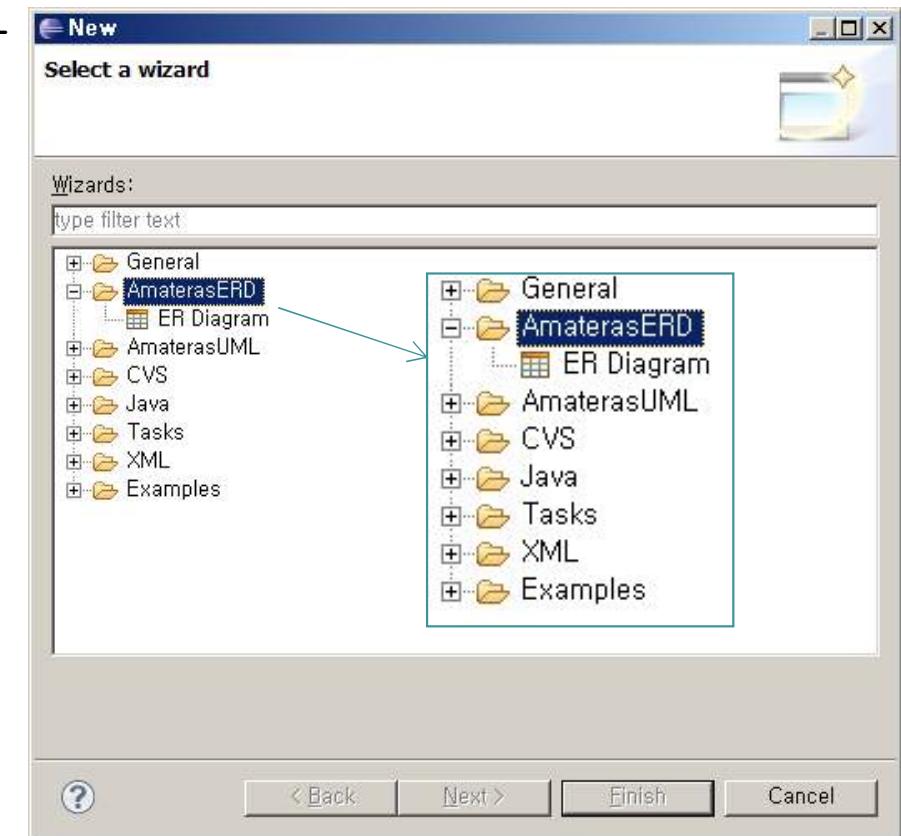
이거 뒷부분에서 필요할 때 다시 설명됩니다.

1.4 UML 작성 Plug-in - AmaterasUML 설치하기

- UseCase Diagram, Class Diagram 작성 기능을 제공
- <http://amateras.sourceforge.jp/>
- AmaterasUML_1.3.3.zip 파일을 압축을 풀어 jar파일 3개를 C:/eclipse/plugins 폴더에 복사



- 논리모델과 물리모델 작성을 지원
- <http://amateras.sourceforge.jp/>
- net.java.amateras.db_1.0.8.jar 파일다운로드 후 C:/eclipse/plugins 폴더에 복사
- UseCase Diagram 작성
 - ▣ Use Case Diagram 작성할 수 있는 기능을 제공
- Class Diagram 작성
 - ▣ Class Diagram 작성할 수 있는 기능을 제공
- 제약사항
 - ▣ UML 1.4 기반으로 설계작업 가능
 - ▣ UML 2.0 지원하지 않음



26

Maven

이클립스에 Maven 설치

27

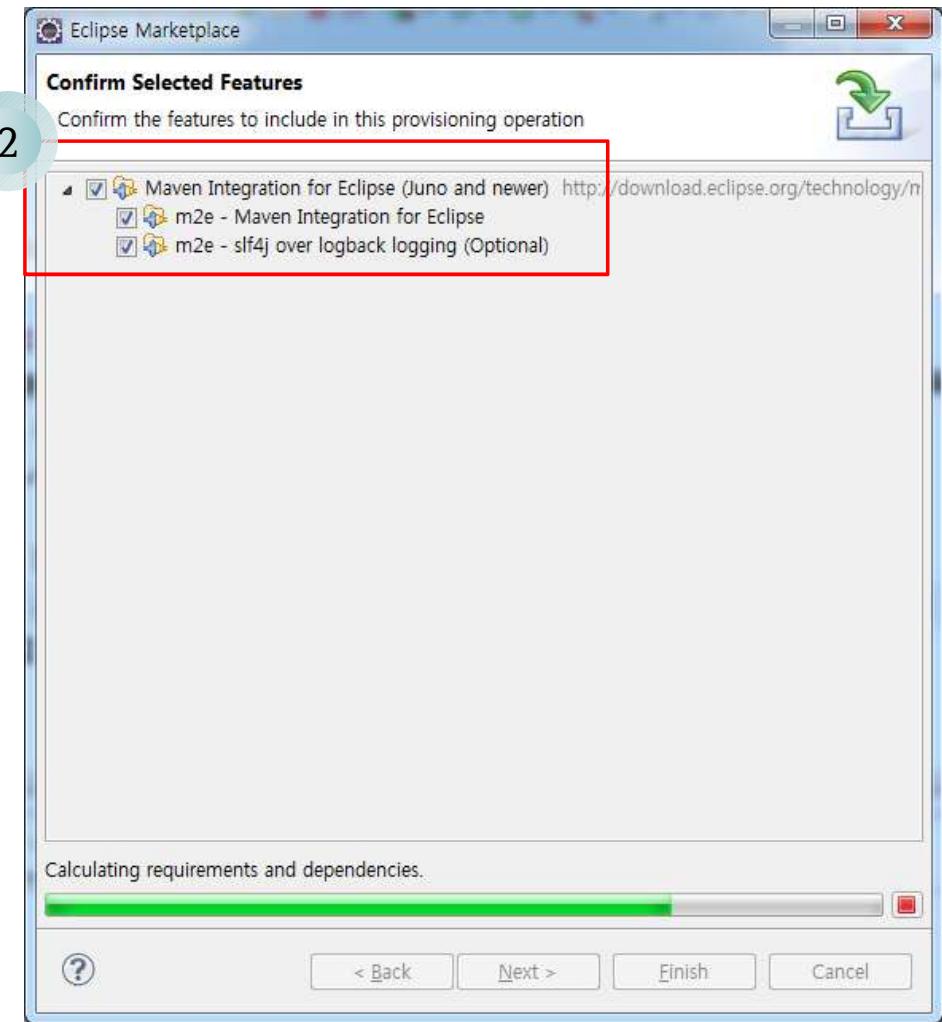
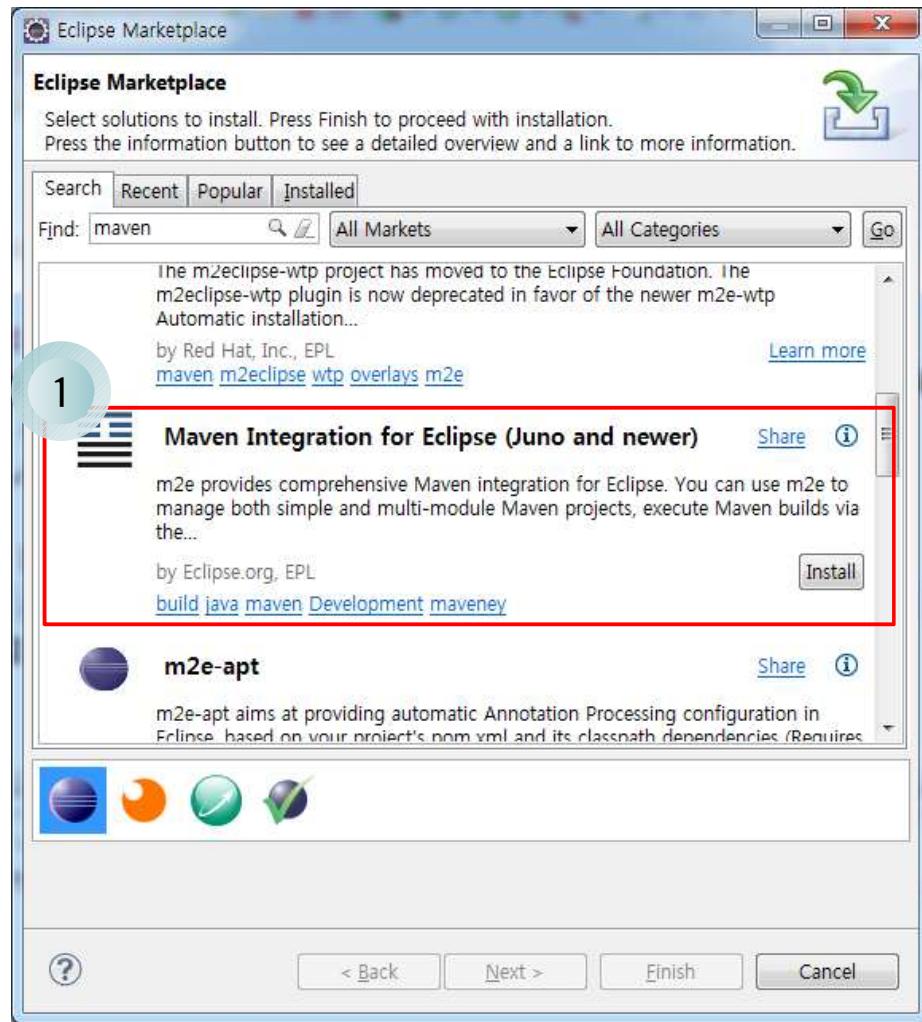
- 1. 이클립스 상단에 Help 메뉴의 Eclipse Marketplace를 클릭합니다.
- 2. Find 입력란에 maven이라고 입력 후 엔터를 누릅니다.
검색 결과 중 Maven Integration for Eclipse라는 플러그인이 출력됩니다.
- Install 버튼을 클릭하여 설치를 진행합니다.
- 3. 설치할 Maven 요소를 체크 후 Next 버튼을 클릭합니다.
- 4. 약관을 확인한 후 동의(accept)를 체크하고 Finish 버튼을 클릭합니다.
- 5. 설치가 진행됩니다.
- 6. 설치가 완료되면 Yes 버튼을 클릭하여 이클립스를 재시작(restart)하면 메이븐 플러그인 설치가 완료됩니다.

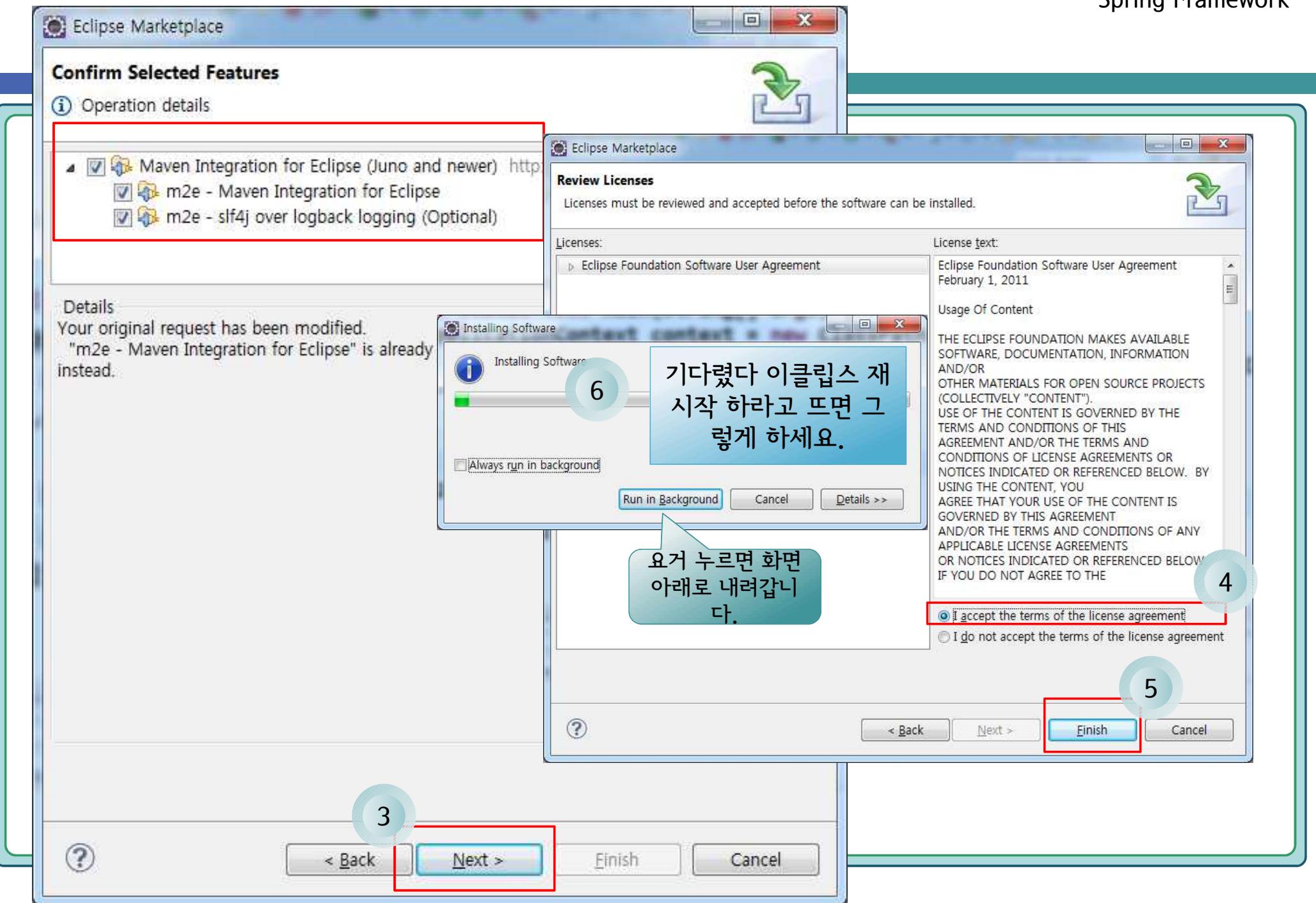
Maven(메이븐)

- 프로젝트 의존성 관리 도구
- 라이브러리들을 쉽게 관리할 수 있습니다.
- 빌드패스에 필요한 jar 파일들을 일일이 추가할 필요 없습니다.
- jar 파일들을 설정파일(pom.xml)로 관리합니다.

maven 플러그인 설치

□ Help -> Eclipse Marketplace





메이븐 설치 기다리는 동안...

- <http://projects.spring.io/spring-framework/> 에서 스프링 api 다운로드 받아야 하는데...
- 다운로드 하는 곳에는 Maven 설정파일에 포함될 XML 태그가 있다.
- 태그 내용을 복사한다.

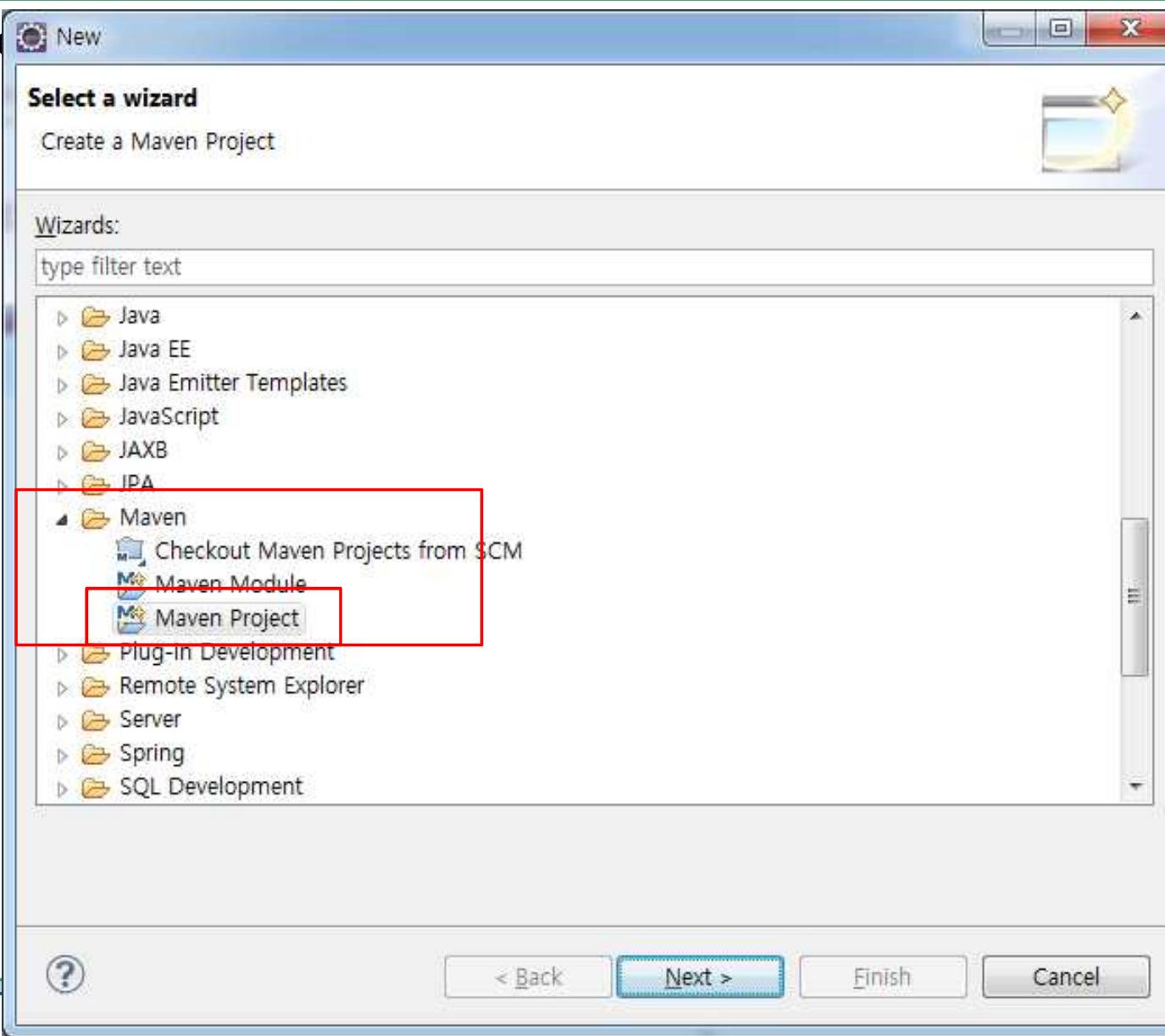
Quick Start

The recommended way to get started using `spring-framework` in your project is with a dependency management system – the snippet below can be copied and pasted into your build. Need help? See our getting started guides on building with [Maven](#) and [Gradle](#).

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>3.2.4.RELEASE</version>
    </dependency>
</dependencies>
```

메이븐 프로젝트 생성

File -> N



The screenshot shows the Eclipse IDE interface with the 'New Maven Project' dialog open. The dialog is divided into two main sections: 'New Maven project' (configuration) and 'Configure project' (artifact details).

New Maven project

Select project name and location

Create a simple project (skip archetype selection)

Use default Workspace location

Location: [empty input field]

Add project(s) to working set

Working set: [empty input field]

▶ Advanced

Configure project

Artifact

| | |
|--------------|------------|
| Group Id: | com.test |
| Artifact Id: | helloworld |

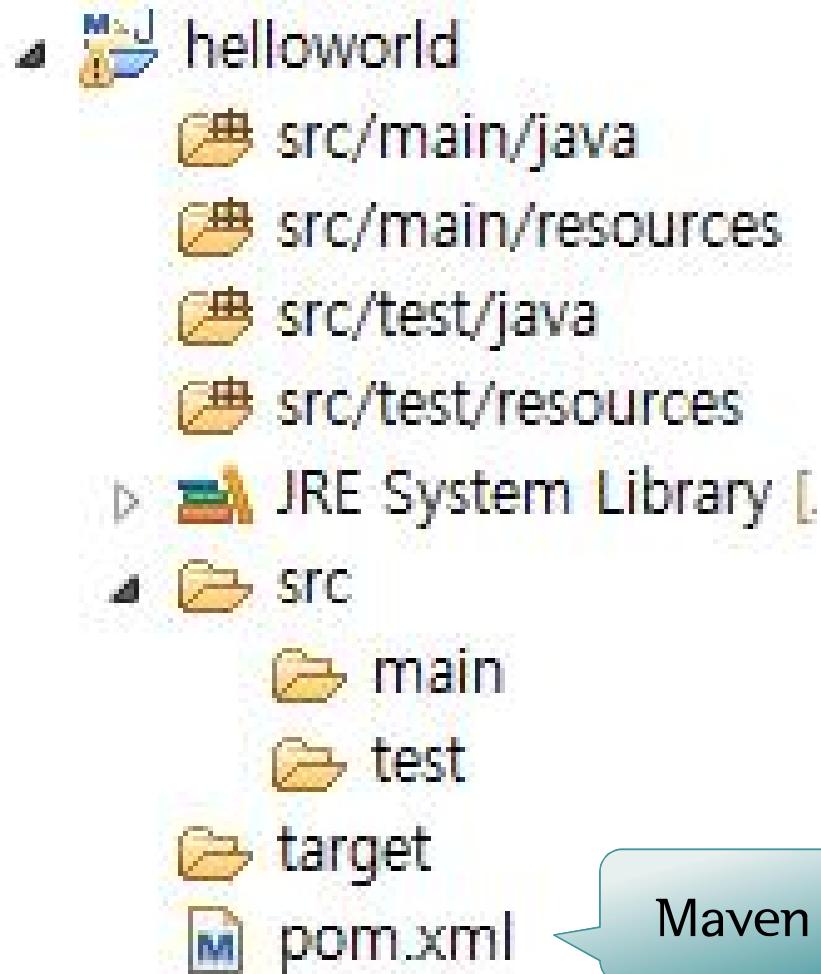
Version: 0.0.1-SNAPSHOT

Packaging: jar

Name: [empty input field]

Description: [empty input field]

Parent Project: [empty input field]



pom.xml - 의존성 설정파일

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.test</groupId>
  <artifactId>helloworld</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>3.2.4.RELEASE</version>
    </dependency>
  </dependencies>

</project>
```

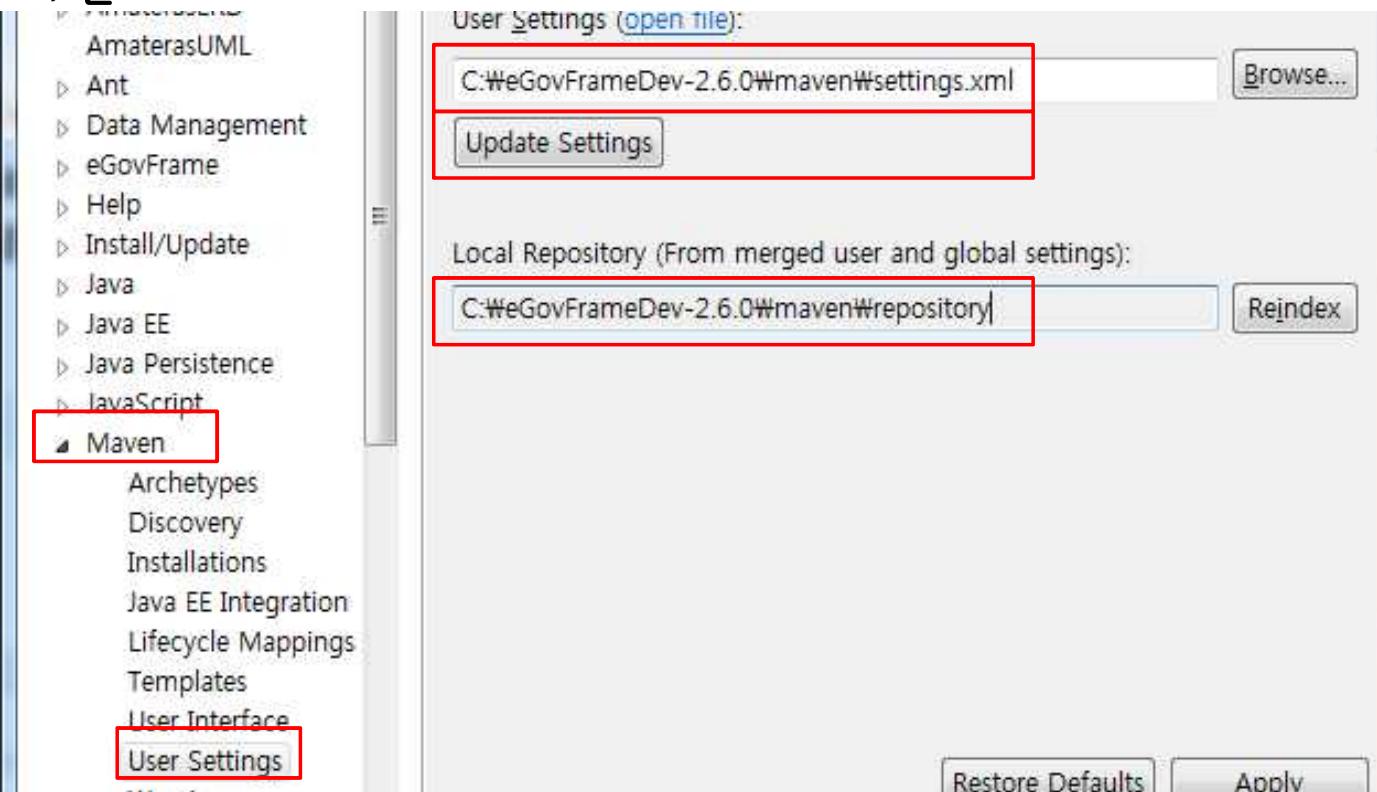
메이븐 설정파일에 의존성을 추가하면... 해당 API는 메이븐 도구에 의해 다운로드 되고... 빌드 된다.

settings.xml - 메이븐설정파일

- 내문서/.m2/settings.xml 에 있는 디폴트 설정파일을 사용
- 이 파일에는 repository 디렉토리를 설정하는 내용이 있다.
- repository 디렉토리는 pom.xml 파일에 설정한 의존성에 따라 해당 라이브러리(API)를 다운로드 하여 저장해 놓을 디렉토리이다.
- 디폴트 리파지토리 디렉토리는
 - 내문서/.m2/repository
- 리파지토리 디렉토리를 C:\eGovFrameDev-2.6.0\maven\repository 로 변경하여 다운로드 받은 라이브러리 파일을 별도로 관리하자.
- settings.xml 파일이 필요하다.

C:\eGovFrameDev-2.6.0\maven\settings.xml

- settings.xml 파일의 내용을 수정한다.
- 리파지토리 디렉토리의 경로를 <localRepository>태그를 이용해 설정한다.
- <localRepository>C:\eGovFrameDev-2.6.0\maven\repository</localRepository>
- 그리고 이클립스 Window -> Preferences -> Maven -> User Settings에서 settings.xml 파일



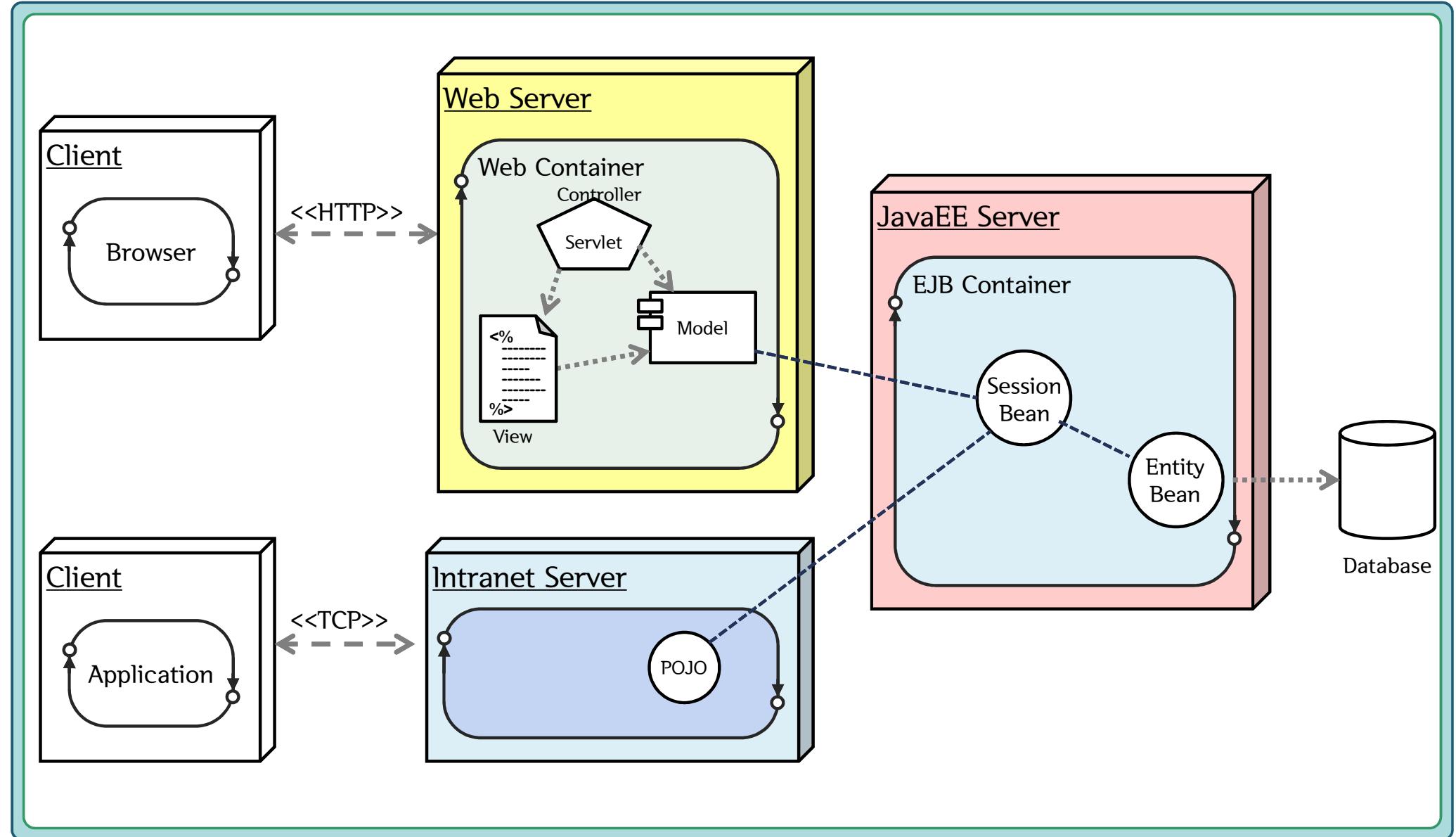
2. Spring Framework 개요

1. 스프링 프레임워크 소개
2. 스프링 프레임워크 특징
3. 스프링 프레임워크 모듈
4. 스프링 프레임워크 라이브러리

2.1 Spring Framework 소개

- Rod Johnson이 “Expert One-on-One J2EE Development without EJB”라는 책을 통해서 EJB를 사용하지 않고 엔터프라이즈 어플리케이션을 개발하는 방법을 소개하였고 이것이 스프링 프레임워크의 모태가 됨.
- 엔터프라이즈 어플리케이션에서 필요로 하는 기능을 제공하는 프레임워크로 복잡한 Enterprise Application 개발을 겨냥해 만들어짐.
- 2003년 2월부터 오픈 소스로 시작된 프로젝트로 경량의 제어역행(IoC)과 관점지향(AOP)의 컨테이너 프레임워크.

Java EE Architecure



2.2 Spring Framework 특징

- 1) AOP(기능별 모듈화, 진정한 OOP 제공)
 - ▣ 컨테이너는 일관성을 유지시켜 주고 투명한 환경 내에서 느슨한 컴포넌트(POJO)의 집합에서 복잡한 시스템을 조립할 수 있는 능력을 제공하며 조직을 해치지 않음.
 - ▣ AOP지원을 통해 주요 비즈니스 로직과 시스템 전반에 걸친 기능 모듈을 완벽히 분리해내도록 도와준다.
- 2) IoC(Inversion of Control: 역제어) 컨테이너
 - ▣ 애플리케이션 객체를 연결해 주고 자동화된 설정 및 집중화된 설정을 제공하는 가장 완전한 경량 컨테이너.
 - ▣ 개발자가 직접 객체를 생성을 하지 않고, 객체의 생성에서 소멸까지 컨테이너가 관리.
 - ▣ Dependency Injection을 통해 객체간의 의존성 주입.
 - ▣ EJB 컨테이너에 비해 가벼운 IoC 컨테이너.(Lightweight 컨테이너)
- 3) Test Unit(편리한 테스트) 제공
 - ▣ 컨테이너는 민첩함을 제공하고 지렛대 역할을 하며 소프트웨어 컴포넌트를 먼저 개발하고 고립시켜 테스트할 수 있게 함으로써 테스트와 확장성을 향상시킨다.
 - ▣ 작성된 코드에 대한 단위테스트를 쉽게 할 수 있도록 도와준다
- 4) 트랜잭션
 - ▣ 트랜잭션 관리를 위한 공통의 추상화된 레이어, 트랜잭션 관리자를 플러그인할 수 있어서 저 수준 트랜잭션을 문제없이 처리한다.
 - ▣ 선언적인 트랜잭션을 지원하여 코드를 수정하지 않고도 트랜잭션을 적용 및 변경 가능하도록 한다.
- 5) JDBC 추상화 레이어
 - ▣ 중요한 예외 계층을 제공하며 예외처리를 단순화시켜 코드의 양을 덜어준다.

2.2 Spring Framework 특징

- 6) ORM 프레임워크 연동 제공
 - ▣ Hibernate, MyBatis, JDO 등과 같은 ORM 프레임워크와 통합되어 있다.
- 7) 좀더 쉬운 J2EE 개발 지향(저비용 유연한 코드 유지)
 - ▣ 계층화된 아키텍처를 갖고 있으며, 그 중 어떤 부분도 독립적으로 사용될 수 있도록 모듈화 되어있다.
 - ▣ EJB를 사용하든 하지 않든 관계없이 비즈니스 객체들을 효과적으로 구성하고 관리할 수 있도록 한다.
 - ▣ 컨테이너의 API에 의존적이지 않은 POJO 관리.
 - ▣ 자바 이외에 빈쉘, 제이루비, 그루비과 같은 스크립트 언어를 지원한다.
 - ▣ 다른 여러 프레임워크와의 연동을 지원한다.
- 8) 다양한 프리젠테이션 계층 제공(jsp, velocity, excel, pdf ...)
 - ▣ 프리젠테이션 계층을 위해 각종 뷰 기술을 지원한다.
- 9) 좋은 설계(아키텍처) 제공
 - ▣ 서블릿 기반의 MVC 프레임워크를 지원한다.
- 10) 분산(원격) 서비스
 - ▣ RMI, JAX-RPC 등과 같은 기술을 이용하여 쉽게 원격 서비스 구현이 가능하다.
- 11) 보안
- 12) 기타
 - ▣ Timer클래스나 큐초 스케줄링 엔진을 이용하여 Job 스케줄링을 지원한다.
 - ▣ JavaMail이나 제이슨 헌터의 MailMessage를 이용하여 메일>Email)을 지원한다.
 - ▣ 다국어를 통한 국제화를 지원한다.

2.2.1 경량 컨테이너

43

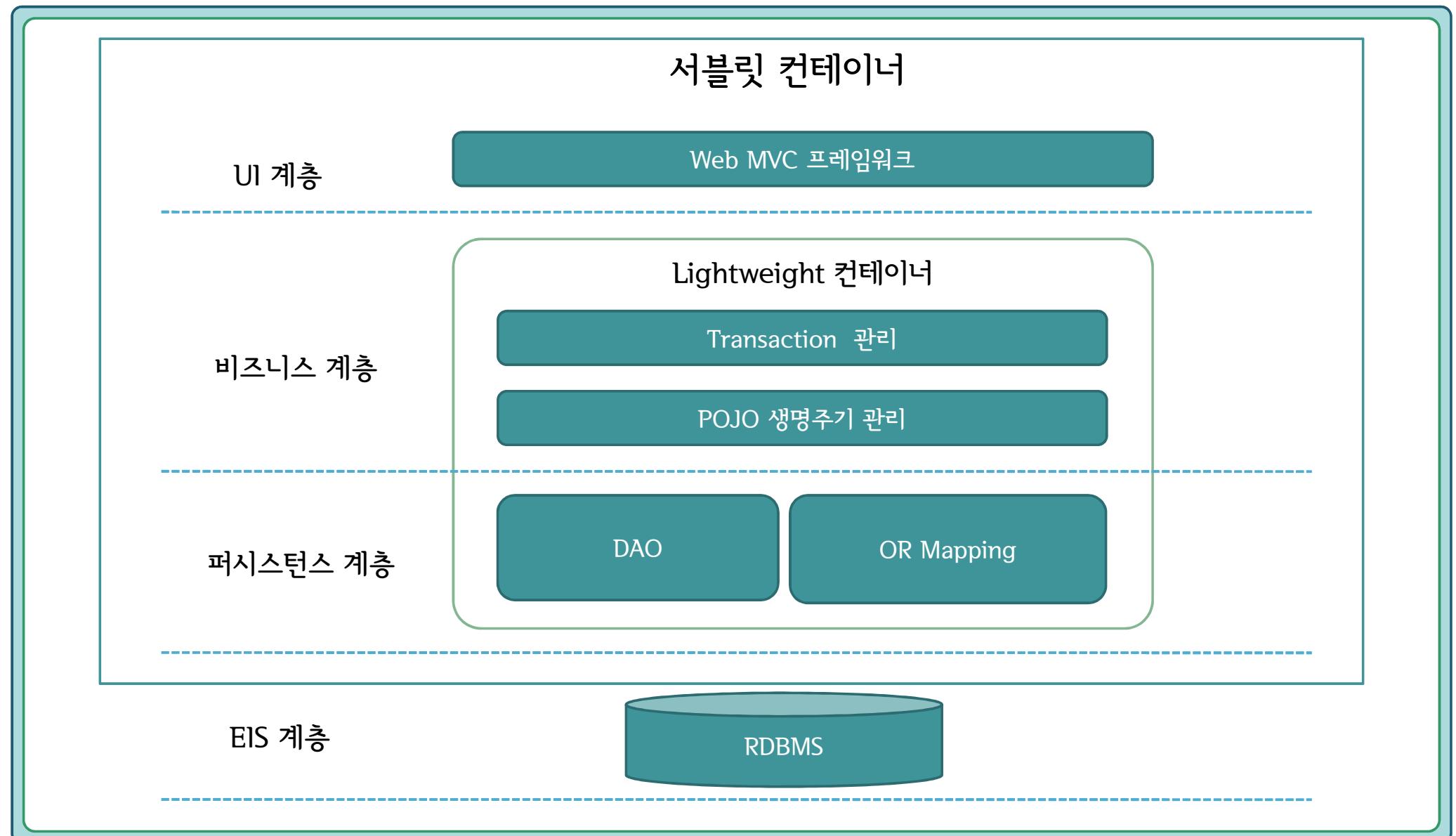
- IoC(Inversion of Control: 역제어) 컨테이너
 - ▣ 개발자가 직접 객체를 생성을 하지 않고, 객체의 생성에서 소멸까지 컨테이너가 관리.
 - ▣ Dependency Injection을 통해 객체간의 의존성 주입.
- Lightweight 컨테이너
 - ▣ EJB 컨테이너에 비해 가벼운 IoC 컨테이너.
 - ▣ 컨테이너의 API에 의존적이지 않은 POJO 관리.

POJO(Plain Old Java Object)

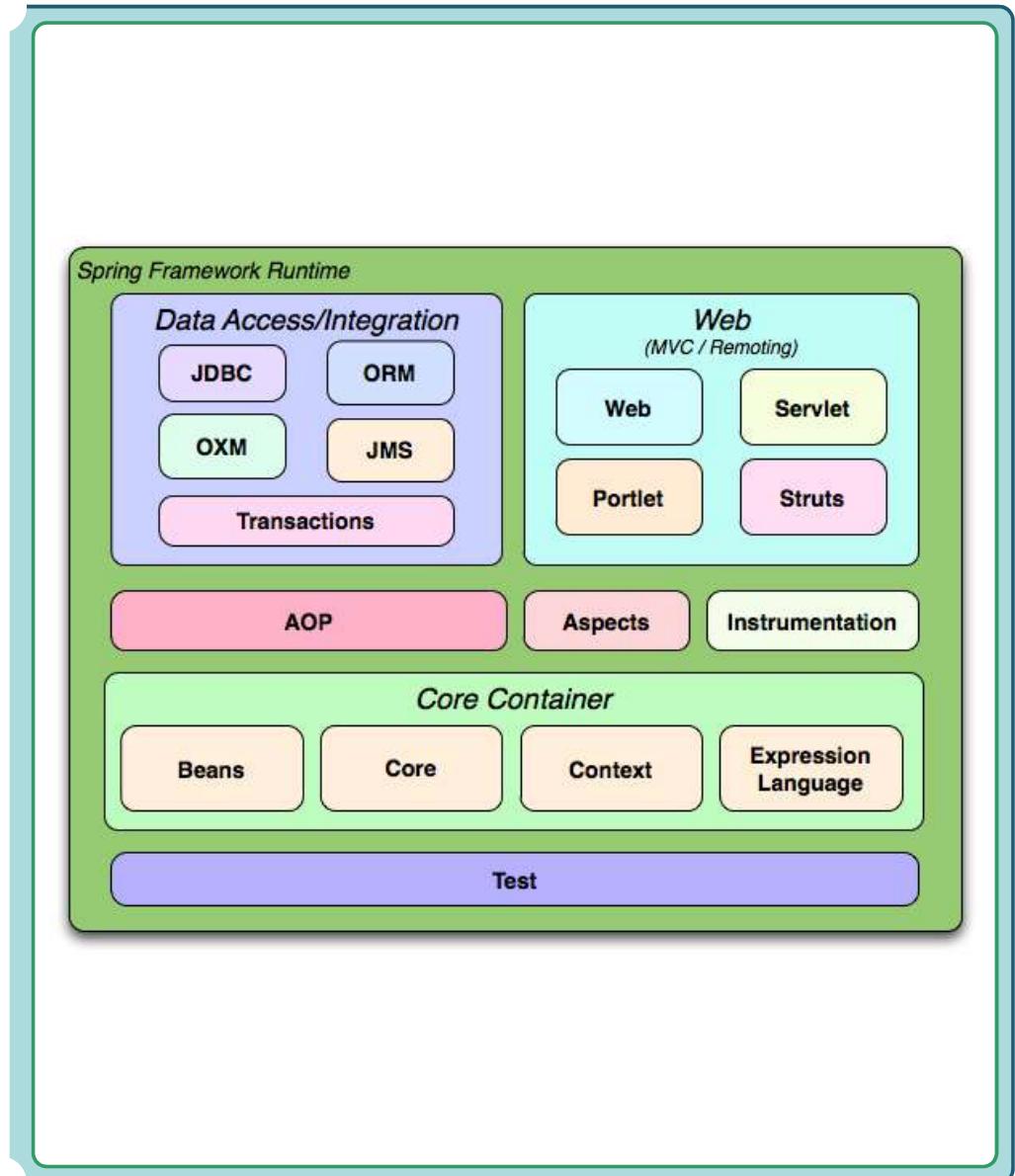
특정 인터페이스 또는 클래스를 상속하지 않는 일반 자바 객체를 의미합니다.

Cf) Servlet 개체는 HttpServlet를 반드시 상속, EJB 개체는 SessionBean을 반드시 구현

2.2.1 경량(Lightweight) 컨테이너



- Spring Core
 - 프레임워크의 가장 기본적인 부분
 - 컨테이너 기능을 수행하기 위해 의존성 주입 기능을 제공
- Spring Context
 - Email, JNDI접근, EJB 연계등과 같은 다수의 엔터프라이즈 서비스 제공
- Spring DAO
 - JDBC 추상화 API 제공
- Spring AOP
 - AOP 구현 API 제공
- Spring ORM
 - MyBatis, Hibernate, JPA등 지원
- Spring Web
 - 웹 기반 기능을 제공
- Spring Web MVC
 - MVC 구현 API를 제공



2.4 Spring Framework 라이브러리

- Spring Framework 3.1.2 다운로드.
 - ▣ <http://www.springsource.org/download>
- 다운로드 받은 spring-framework-3.1.2.RELEASE-with-docs.zip 파일을 압축을 풀고 그 중 필수라이브러리들을 라이브러리로 등록하여 사용한다.
- 필수라이브러리
 - ▣ org.springframework.xxx.jar
 - ▣ commons-logging.jar
 - <http://commons.apache.org>에서 logging 다운로드
- 다른 라이브러리들은 필요할 때마다 추가해서 사용한다.

* 여러분은 이러한 여러 라이브러리들을 Maven을 이용해 관리할 수 있습니다.

3. Dependency Injection

1. DI 개요
2. 스프링에서 DI
3. Constructor Injection
4. Setter Injection
5. 컬렉션 타입의 설정
6. 의존관계 자동 설정
7. 빈 객체 범위
8. 스프링 컨테이너
9. 빈 라이프 사이클

3.1 Dependency Injection 개요

- 의존성 주입이라고 한다.
- 스프링 프레임워크가 지원하는 핵심 기능 중 하나이다.
- 객체 사이의 **의존관계가** 객체 자신이 아닌 **외부에 의해 설정된다**는 개념이다.
- 컨테이너는 어떤 객체(A)가 필요로 하는 의존관계에 있는 다른 객체(B)를 직접 생성하여 어떤 객체(A)로 주입(설정)해주는 역할을 담당하게 된다.

- DI는 의존관계의 구현을 어떻게 하느냐에 대한 내용이다.
- 의존관계는 B의 인스턴스를 A의 biz() 메서드에서 사용함을 의미한다.



3.1.1 의존하는 객체를 지정하는 방법

- 직접 의존하는 객체를 코드에 명시하는 방법
 - ▣ 단위 테스트가 어렵다.
 - ▣ 의존 객체 변경 시 코드 수정이 불가피하다.
 - ▣ ex) Foo f = new Foo();
Bar b = new Bar(f);
- Factory나 JNDI를 이용하여 검색하는 방법
 - ▣ 단위 테스트가 어렵다.
 - ▣ 실제 의존 객체와의 느슨한 의존성 대신 Factory나 JNDI와의 의존성이 생긴다.
- 외부의 조립기(Assembler)를 이용하는 방법
 - ▣ 단위 테스트가 용이하다.
 - ▣ 느슨한 의존성.

3.2 Spring에서의 DI

- 객체간의 의존성을 설정 파일로 손쉽게 관리한다.
- 스프링은 각 클래스 간의 의존 관계를 관리하기 위한 두 가지 방법을 제공한다.

1. Constructor-based Injection

생성자를 이용한 의존관계 설정 방식

클래스에 생성자를 만들어 줘야 함

의존하는 객체를 생성자를 통해 전달 함

2. Setter-based Injection

setXxx() 메서드를 이용한 의존관계 설정 방식

클래스에 set 메서드가 있어야 함

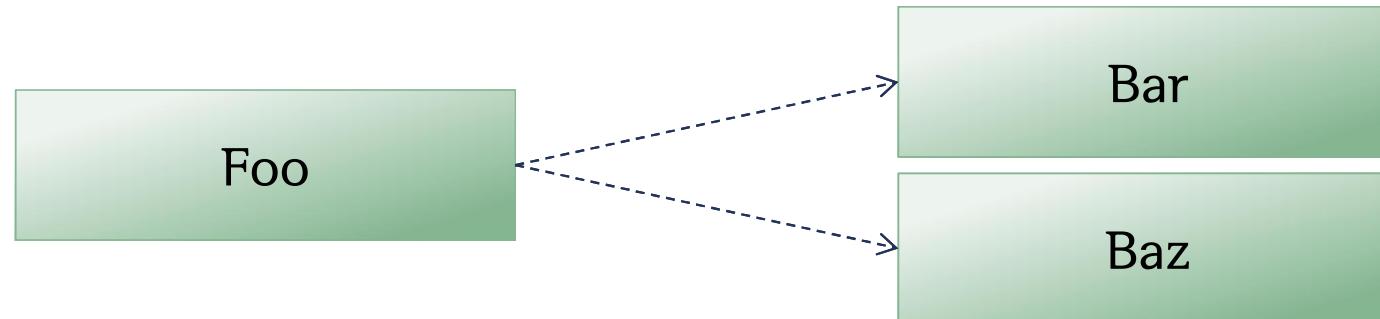
의존하는 객체를 set 메서드를 통해 전달함

3.2.1 DI - Constructor-based Injection

- 의존하는 객체를 생성자를 통해서 전달 받는 방법.
 1. 의존하는 객체를 전달받을 생성자를 작성한다.
 2. 설정 파일에 <constructor-arg> 태그를 이용한다.
 - ▣ 객체인 경우 <ref> 태그 이용
 - ▣ 문자열이나 기본 데이터 타입이라면 <value> 태그 이용

3.2.1 DI - Constructor-based Injection

52



Foo.java

```
package x.y;  
public class Foo{  
    private Bar bar;  
    private Baz baz;  
    public Foo(Bar bar, Baz baz){  
        this.bar = bar;  
        this.baz = baz;  
    }  
}
```

applicationContext.xml

```
...  
<bean id="bar" class="x.y.Bar" />  
<bean id="baz" class="x.y.Baz" />  
<bean id="foo" class="x.y.Foo">  
    <constructor-arg>  
        <ref bean="bar" />  
    </constructor-arg>  
    <constructor-arg ref="baz"/>  
</bean>  
...
```

3.2.1 DI - Constructor-based Injection

- 생성자로 전달할 객체나 값이 여러 개인 경우에는 <constructor-arg>태그를 반복하여 선언한다.
- 이때 선언한 순서대로 생성자의 매개변수로 전달된다.
- 순서를 명시하려면 index 속성을 이용한다.
 - ▣ index는 0부터 시작한다.
- type을 명시하려면 type 속성을 이용한다.

3.2.1 DI - Constructor-based Injection

GreetingServiceImpl.java

```
private String greeting;
private int loopCount;
public GreetingServiceImpl(String greeting, int loopCount){
    this.greeting = greeting;
    this.loopCount = loopCount;
}
...
```

applicationContext.xml

```
<bean id="greeting1"
      class="myspring.sample2.constructor.GreetingServiceImpl">
    <constructor-arg index="1" type="int" value="3" />
    <constructor-arg index="0">
        <value>안녕</value>
    </constructor-arg>
</bean>
...
```

생성자의 인자의 타입

생성자의 인자 순서(0부터 시작)

3.2.2 DI - Setter-based Injection

- `setXxx()` 형태의 설정 메서드를 통해서 전달받는 방법으로 프로퍼티 설정 방식이라고도 한다.
 1. 의존하는 객체를 전달받을 setter 메서드를 작성한다.
 2. 설정파일에 `<property>`태그를 이용한다.
 - 객체인 경우 `<ref>`태그 이용
 - 문자열이나 기본데이터 타입이라면 `<value>`태그 이용
- 생성자 오버로딩 시 기본 생성자가 반드시 필요하다.

3.2.2 DI - Setter-based Injection



Foo.java

```
public class Foo{  
    private Bar bar;  
    public void setBar(Bar bar){  
        this.bar = bar;  
    }  
}
```

applicationContext.xml

```
...  
<bean id="bar" class="Bar" />  
<bean id="foo" class="Foo">  
    <property name="bar">  
        <ref bean="bar" />  
    </property>  
</bean>  
...
```

3.2.3 XML Namespace 이용

- <property> 태그를 사용하지 않고 프로퍼티의 값을 설정하는 방법
- 네임스페이스를 지정해야 함
 - ▣ <beans xmlns="http://www.springframework.org/schema/beans"
 - ▣ xmlns:p="http://www.springframework.org/schema/p"
 - ▣ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 - ▣ xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

Foo.java

```
public class Foo{
    private Bar bar;
    public void setBar(Bar bar){
        this.bar = bar;
    }
}
```

applicationContext.xml

```
...
<bean id="bar" class="Bar" />
<bean id="foo" class="Foo"
      p:bar-ref="bar" />
```

Spring3.1부터는 c-namespace도 사용할 수 있습니다.
물론 Constructor-based injection에 사용됩니다.

3.2.4 Look-up 메서드 인젝션

필요한 객체를 제공하는 루업 메서드를 구현

루업 메서드 규칙

- ▢ 접근제한자는 public 또는 protected
- ▢ 리턴타입이 void가 아니다.
- ▢ 인자를 갖지 않음
- ▢ 추상메서드도 가능
- ▢ final이 되면 안됨(스프링이 해당 클래스를 상속받아 루업메서드를 재정의 하기 때문)

- CGLIB 모듈이 있어야 함
- CGLIB는 코드 생성 라이브러리로서(Code Generator Library) 런타임에 동적으로 자바 클래스의 프록시를 생성해주는 기능을 제공함
- <http://cgleb.sourceforge.net/>에서 다운로드
- ASM 모듈을 포함하는 cglib-nodep-x.x.x.jar 파일이 필요함
- CGLIB 2.2.3은 ASM3.3.1 사용해야 함
- ASM 모듈은 <http://forge.ow2.org/projects/asm>에서 다운로드

Foo.java

```
public class Foo{
    private Bar bar;
    public void setBar(Bar bar){
        this.bar = bar;
    }
    public void doBar() {
        Bar bar = getBar();
        bar.dolt();
    }
    protected Bar getBar() {
        return null;
    }
}
```

applicationContext.xml

```
...
<bean id="bar" class="Bar" />
<bean id="foo" class="Foo" >
    <lookup-method name="getBar"
        bean="bar"/>
</bean>
```

3.2.5 임의 빈 객체 사용

- <constructor-arg> 태그나 <property> 태그는 <ref> 태그 또는 ref 속성을 이용하여 다른 빈 객체를 전달하지만 식별값(변수명)을 갖지 않는 빈 객체를 생성해서 전달할 수도 있음

Foo.java

```
public class Foo{
    private Bar bar;
    private String email;
    public void setBar(Bar bar){
        this.bar = bar;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

applicationContext.xml

```
...
<bean id="foo" class="Foo"
      p:email="heojk@daum.net" >
    <property name="bar">
      <bean class="Bar"/>
    </property>
</bean>
...
```

3.2.6 Constructor-based or setter-based DI?

- 둘 다 사용 가능
- setter 메서드에 @Required 어노테이션 사용시 setter의 존성을 사용하도록 함
- 스프링 팀은 setter injection을 추천
 - ▣ 많은 수의 생성자 인자는 다루기 힘들다.
 - ▣ setter 메서드는 나중에 재구성 하기 쉬움

3.3 컬렉션 타입의 프로퍼티 설정

| 태그 | 컬렉션 타입 |
|---------|----------------------|
| <list> | java.util.List나 배열 |
| <set> | java.util.Set |
| <map> | java.util.Map |
| <props> | java.util.Properties |

- 컬렉션 원소가 객체인 경우 <ref> 태그 이용
- 컬렉션 원소가 기본타입인 경우 <value> 태그 이용
 - ▣ type 속성 이용하여 타입 지정 가능하다.
 - ▣ 제네릭을 이용하면 타입 지정하지 않아도 된다.

3.3.1 List 타입의 프로퍼티 설정

CalculatorServiceImpl.java

```
private List valueList;
public void setValueList(List valueList) {
    this.valueList = valueList;
}
...
```

applicationContext.xml

```
<bean id="calculator" class="myspring.sample4.collection.CalculatorServiceImpl">
    <property name="valueList">
        <list>
            <value type="java.lang.Integer">10</value>
            <value type="java.lang.Integer">20</value>
            <ref bean="bar"/>
        </list>
    </property>
</bean>
...
```

자바클래스에 제네릭을 사용하면 type속성 생략 가능
<value>태그는 <list>태그의 value-type 속성으로 지정 가능

3.3.2 Map 타입의 프로퍼티 설정

63

```
<property name="...">
  <map>
    <entry>
      <key><value>hello</value></key>
      <ref bean="bar"/>
    </entry>
    <entry>
      <key><value>hi</value></key>
      <ref bean="bar2"/>
    </entry>
  </map>
</property>
```

3.3.3 Properties 타입

64

```
<property name=...>
<props>
    <prop key="server">192.168.1.100</prop>
    <prop key="timeout">5000</prop>
</props>
</property>
```

3.4 의존관계 자동 설정

- 의존하는 빈 객체의 타입이나 이름을 이용하여 의존객체를 자동으로 설정할 수 있는 기능으로 4가지 방식 제공한다.
- autowire 속성을 이용한다.
- 자동설정과 직접설정의 혼합도 가능하다.

| 방식 | 설명 |
|-------------|--|
| byName | 프로퍼티의 이름과 같은 이름을 갖는 빈 객체를 설정한다. |
| byType | 프로퍼티의 타입과 같은 타입을 갖는 빈 객체를 설정한다. |
| constructor | 생성자 파라미터 타입과 같은 타입을 갖는 빈 객체를 생성자에 전달한다. |
| autodetect | Constructor 방식을 먼저 적용하고, 적용할 수 없을 경우 byType 방식을 적용하여 빈 객체를 설정한다. |

3.4.1 의존관계 자동 설정 - byName

GreetingServiceImpl.java

```
...
private OutputService outputter;
public void setOutputter(OutputService outputter) {
    this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting"
      class="myspring.sample5.autosetting.GreetingServiceImpl"
      autowire="byName"/>

<bean id="outputter"
      class="myspring.sample5.autosetting.OutputServiceImplConsole"/>
...
```

3.4.2 의존관계 자동 설정 – byType

GreetingServiceImpl.java

```
...
private OutputService outputter;
public void setOutputter(OutputService outputter) {
    this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting"
      class="myspring.sample5.autosetting.GreetingServiceImpl"
      autowire="byType"/>

<bean id="consoleOutput"
      class="myspring.sample5.autosetting.OutputServiceImplConsole"/>
...
```

3.4.3 의존관계 자동 설정 – constructor

GreetingServiceImpl2.java

```
...
private OutputService outputter;
public GreetingServiceImpl2(OutputService outputter){
    this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting2"
      class="myspring.sample5.autosetting.GreetingServiceImpl2"
      autowire="constructor" />

<bean id="consoleOutput"
      class="myspring.sample5.autosetting.OutputServiceImplConsole"/>
...
```

3.4.4 의존관계 자동 설정 – autodetect

GreetingServiceImpl2.java

```
private OutputService outputter;  
public GreetingServiceImpl2(OutputService outputter){  
    this.outputter = outputter;  
}  
public void setOutputter(OutputService outputter) {  
    this.outputter = outputter;  
}  
...
```

applicationContext.xml

```
...  
<bean id="greeting2"  
      class="myspring.sample5.autosetting.GreetingServiceImpl2"  
      autowire="autodetect" />  
<bean id="consoleOutput"  
      class="myspring.sample5.autosetting.OutputServiceImplConsole"/>  
...
```

3.5 빈 객체 범위

- 기본적으로 컨테이너에 한 개의 빈 객체를 생성한다.
- 빈의 범위를 설정할 수 있는 방법을 제공한다.
- scope 속성을 이용한다.

| 방식 | 설명 |
|---------------|--|
| singleton | 컨테이너에 한 개의 빈 객체만 생성한다.(기본값) |
| prototype | 빈을 요청할 때마다 빈 객체를 생성한다. |
| thread | 스레드별로 생성되며, 현재 실행중인 스레드에 종속된다. 스레드가 죽으면 빈도 소멸된다. |
| request | HTTP 요청마다 빈 객체를 생성한다. (WebApplicationContext에서만 적용) |
| session | HTTP 세션마다 빈 객체를 생성한다.(WebApplicationContext에서만 적용) |
| application | Singleton 스코프와 유사하다. 다만 이 스코프에 해당하는 빈은 java.servlet.ServletContext에도 등록되는 점만 차이가 있다. |
| globalSession | 글로벌 HTTP 세션에 대한 빈 객체를 생성한다. 포틀릿을 지원하는 컨텍스트에만 적용 가능하다. 글로벌 세션이 없으면 세션 스코프와 기능이 같다. |

3.5.1 빈 객체 범위

GreetingTest.java

```
...
GreetingService bean = (GreetingService)factory.getBean("greeting");
GreetingService bean2 = (GreetingService)factory.getBean("greeting");

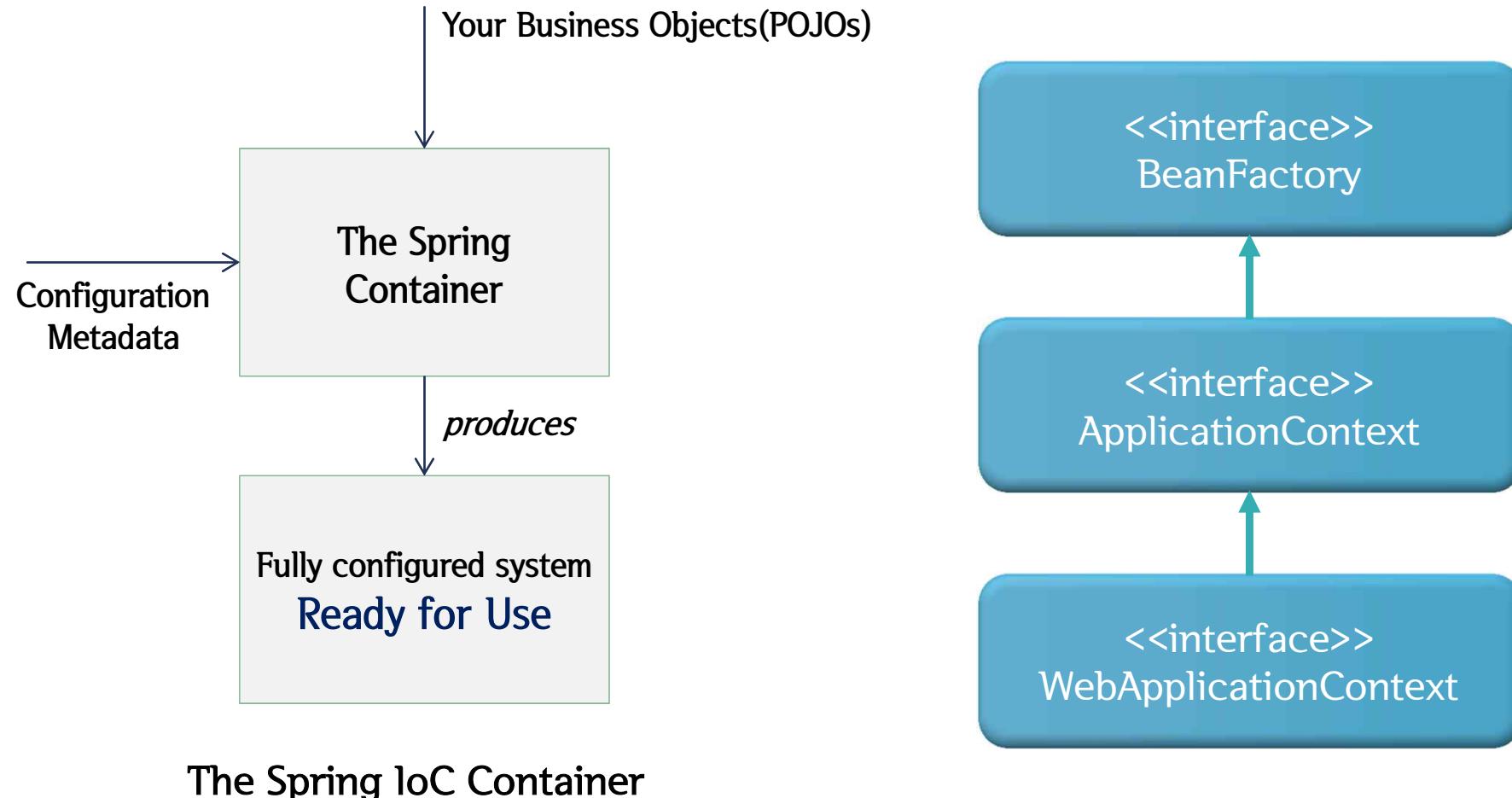
System.out.println(bean == bean2); // true
...
```

applicationContext.xml

```
...
<bean id="greeting" class="myspring.sample6.beanscope.GreetingServiceImpl"
    scope="singleton">
    <property name="greeting">
        <value>Hello</value>
    </property>
</bean>
...
```

prototype으로 바꾸면
bean==bean2 결과는 false가 된다.

3.6 스프링 컨테이너



3.6.1 스프링 컨테이너 - BeanFactory

- 빈 객체를 관리하고 각 빈 객체 간의 의존 관계를 설정해 주는 가장 단순한 컨테이너
- 대표적인 구현 클래스
 - ▣ XmlBeanFactory

GreetingTest.java

...

```
Resource res = new ClassPathResource("applicationContext6.xml");
BeanFactory factory = new XmlBeanFactory(res);
```

```
GreetingService bean = (GreetingService)factory.getBean("greeting");
```

...

3.6.1 org.springframework.beans.factory.BeanFactory

3. Dependency Injection

74

- 빈 객체를 관리하고 각 빈 객체간의 의존 관계를 설정해주는 기능을 제공하는 가장 단순한 컨테이너
- 구현 클래스
 - ▣ org.springframework.beans.factory.XmlBeanFactory
 - ▣ ApplicationContext 인터페이스는 구현하지 않았음
 - ▣ 외부 자원으로부터 설정 정보를 읽어와서 빈 객체를 생성함
 - ▣ 자원의 종류에 따라 Resource 인터페이스를 구현한 클래스들이 있음
- org.springframework.core.io.Resource 인터페이스를 구현한 클래스
 - ▣ FileSystemResource : 파일 시스템의 특정 파일로부터 정보를 읽어옴
 - ▣ InputStreamResource : InputStream으로부터 정보를 읽어옴
 - ▣ ClassPathResource : 클래스 패스에 있는 자원으로부터 정보를 읽어옴
 - ▣ UrlResource : 특정 URL로부터 정보를 읽어옴
 - ▣ ServletContextResource : 웹 어플리케이션의 루트 디렉토리를 기준으로 지정한 경로에 위치한 자원으로부터 정보를 읽어옴

3.6.2 스프링 컨테이너 - ApplicationContext

- BeanFactory 인터페이스를 상속받은 하위 인터페이스
- BeanFactory의 빈 관리 기능 이외에 여러 개의 편리한 기능이 추가되었다.
 - ▣ 메시지의 국제화
 - ▣ 리소스로의 액세스 수단 간편화
 - ▣ 이벤트 처리
- 대표적인 구현 클래스
 - ▣ ClassPathXmlApplicationContext
 - ▣ FileSystemXmlApplicationContext
 - ▣ XmlWebApplicationContext

3.6.2 스프링 컨테이너 - ApplicationContext

GreetingTest.java

...

```
ApplicationContext factory = new  
    ClassPathXmlApplicationContext("applicationContext7.xml");
```

```
GreetingService bean = (GreetingService)factory.getBean("greeting");
```

...

3.6.3 스프링 컨테이너 - WebApplicationContext

- 웹 어플리케이션을 위한 ApplicationContext.
- 하나의 웹 어플리케이션마다 한 개씩 존재한다.
- 웹 어플리케이션을 위해 추가적으로 제공되는 빈 영역(bean scope)을 정의하고 있다.
- 구현한 클래스
 - ▣ XmlWebApplicaitonContext

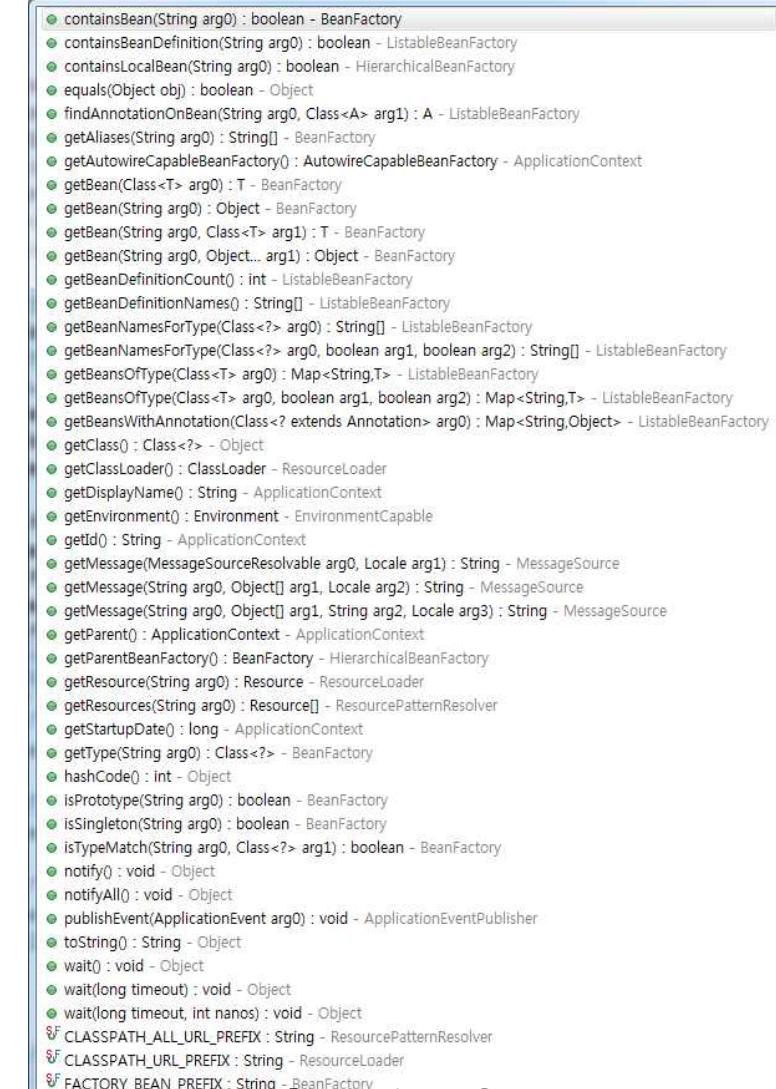
3.6.4 스프링 컨테이너 비교

3. Dependency Injection

78



BeanFactory



ApplicationContext

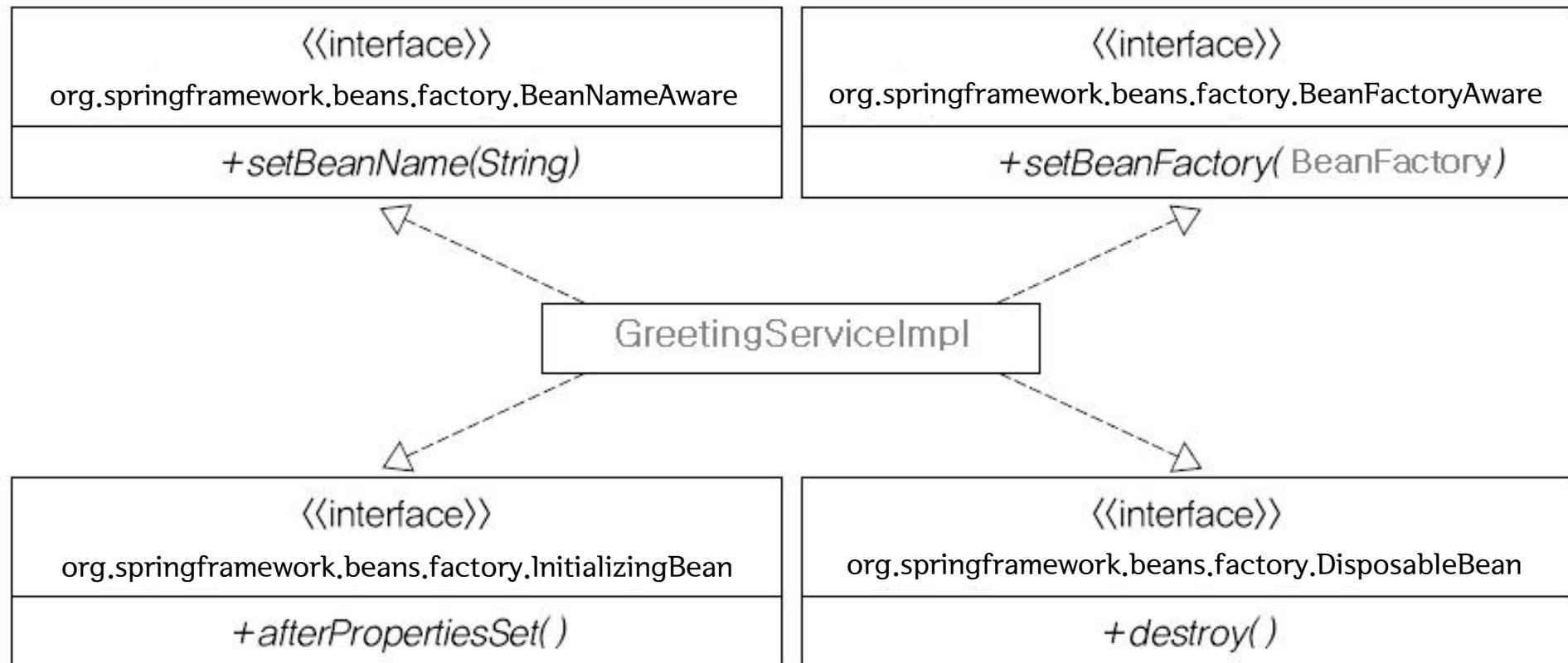
3.7 빈 라이프 사이클

- 스프링 컨테이너에 저장되는 빈 객체는 최소한 생성, 초기화, 소멸의 라이프 사이클을 갖게 된다.
- 스프링은 빈 객체의 생성, 초기화, 소멸뿐만 아니라 추가적인 단계를 제공하고 있으며 이를 통해 라이프 사이클에 따른 빈 객체의 상태를 정교하게 제어할 수 있다.
- 빈 객체의 라이프 사이클은 빈 클래스가 구현한 인터페이스와 관리되는 컨테이너에 따라 달라진다.

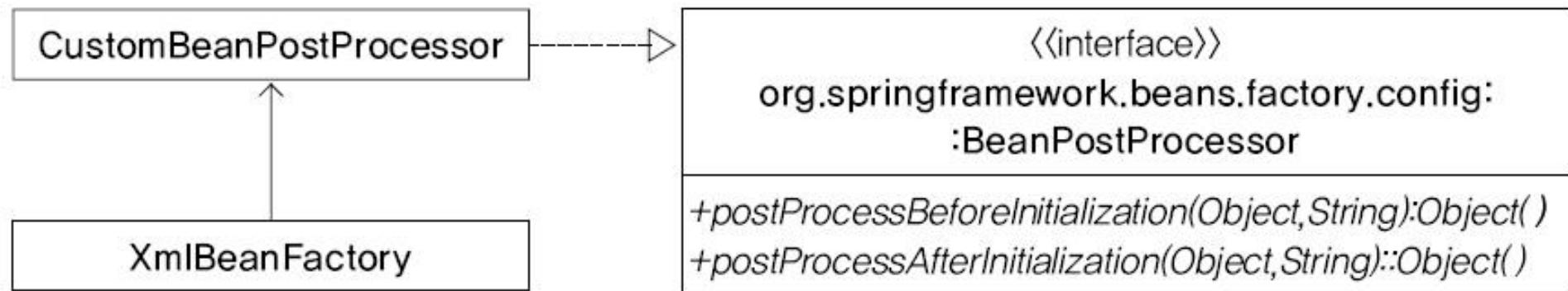
* lazy-init="true"

- 이 속성은 빈의 scope가 singleton일 경우에 사용할 수 있다.
- 기본적으로 빈은 ApplicationContext 생성시 singleton 객체들을 로딩하게 되지만 lazy-init 을 true로 바꾸면 명시적을 객체를 참조할 때 생성하게 된다.

3.7.1 구현 가능한 빈 라이프 사이클 메서드



3.7.1 구현 가능한 빈 라이프 사이클 메서드



3.7.2 빈 라이프 사이클 - BeanFactory

1. 빈의 인스턴스화(생성자 호출)
2. 필드값 설정
3. setBeanName() 메서드 호출
 - ▣ BeanNameAware 인터페이스를 구현하고 있을 경우
4. setBeanFactory() 메서드 호출
 - ▣ BeanFactoryAware 인터페이스를 구현하고 있을 경우
5. BeanPostProcessor의 postProcessBeforeInitialization() 메서드 호출
 - ▣ BeanFactory에 BeanPostProcessor 클래스가 관련되어 있을 경우
 - ▣ BeanPostProcessor 인터페이스를 구현한 클래스를 작성한 다음
`factory.addBeanPostProcessor(new CustomBeanPostProcessor());`
6. afterPropertiesSet() 메서드 호출
 - ▣ InitializingBean 인터페이스를 구현하고 있을 경우
7. Custom 초기화 메서드 호출
 - ▣ Custom 초기화 메서드가 정의되어 있을 경우
 - ▣ bean 설정시 init-method 속성으로 초기화 메서드 정의(`init-method="init"`)
8. BeanPostProcessor의 postProcessAfterInitialization() 메서드 호출
 - ▣ BeanFactory에 BeanPostProcessor 클래스가 관련되어 있을 경우
9. 빈 사용

3.7.2 빈 라이프 사이클 - BeanFactory

- 그리고 컨테이너가 종료할 때에는 다음 순서로 메서드가 호출된다.

1. **destroy()** 메서드 호출

- DisposableBean 인터페이스를 구현하고 있을 경우

2. **Custom Destroy** 메서드 호출

- bean 설정시 destroy-method 속성 이용해서 설정

3.7.3 빈 라이프 사이클 - ApplicationContext

- BeanFactory일 때의 setBeanFactory() 단계 이후에 다음과 같은 4단계의 라이프 사이클을 추가로 수행한다.
 1. setResourceLoader() 메서드 호출
 - ▣ ResourceLoaderAware 인터페이스를 구현하고 있을 경우
 2. setApplicationEventPublisher() 메서드 호출
 - ▣ ApplicationEventPublisherAware 인터페이스를 구현하고 있을 경우
 3. setMessageSource() 메서드 호출
 - ▣ MessageSourceAware 인터페이스를 구현하고 있을 경우
 4. setApplicationContext() 메서드 호출
 - ▣ ApplicationContextAware 인터페이스를 구현하고 있을 경우

3.7.4 커스텀 초기화 및 소멸 메서드

```
public class Foo {  
    public void start() {  
        //커스텀 초기화 메서드 구현부  
    }  
    public void stop() {  
        //커스텀 소멸 메서드 구현부  
    }  
}
```

```
<bean id="foo" class="Foo" init-method="start" destroy-method="stop">  
...  
</bean>
```

3.7.5 InitializingBean/DisposableBean 구현

- 빈 초기화 콜백 메서드는 InitializingBean 인터페이스를 implements
 - ▣ org.springframework.beans.factory.InitializingBean
 - ▣ void afterPropertiesSet() throws Exception;

- 빈 소멸 콜백 메서드는 DisposableBean 인터페이스를 implements
 - ▣ org.springframework.beans.factory.DisposableBean
 - ▣ void destroy() throws Exception;

3.7.6 Combining lifecycle mechanisms

- 스프링 2.5부터 빈 라이프사이클을 제어하기 위해 3가지를 사용할 수 있음
 - ▣ InitializingBean and DisposableBean callback interfaces;
 - ▣ custom init() and destroy() methods
 - ▣ @PostConstruct and @PreDestroy annotations.
- 모두 같이 사용되었을 때 순서
 - ▣ @PostConstruct 어노테이션을 갖는 메서드
 - ▣ InitializingBean인터페이스의 afterPropertiesSet() 메서드
 - ▣ 커스텀 init() 메서드
 - ▣ ...
 - ▣ @PreDestroy 어노테이션을 갖는 메서드
 - ▣ DisposableBean인터페이스의 destroy() 메서드
 - ▣ 커스텀 destroy() 메서드

3.8 Annotation-based container configuration

- 자바 5 부터 추가된 아노테이션은 코드에 설정 정보를 입력함으로 설정 파일을 사용하지 않거나 설정 파일의 크기를 줄이고 싶을 때 유용하게 사용됨

- 설정파일에

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">
```

```
    <context:annotation-config/>
```

...

또는

```
<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor"/>
<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>
<bean class="org.springframework.beans.factory.annotation.CommonAnnotationBeanPostProcessor"/>
<bean class="org.springframework.beans.factory.annotation.ConfigurationClassPostProcessor"/>
```

3.8.1 Annotation

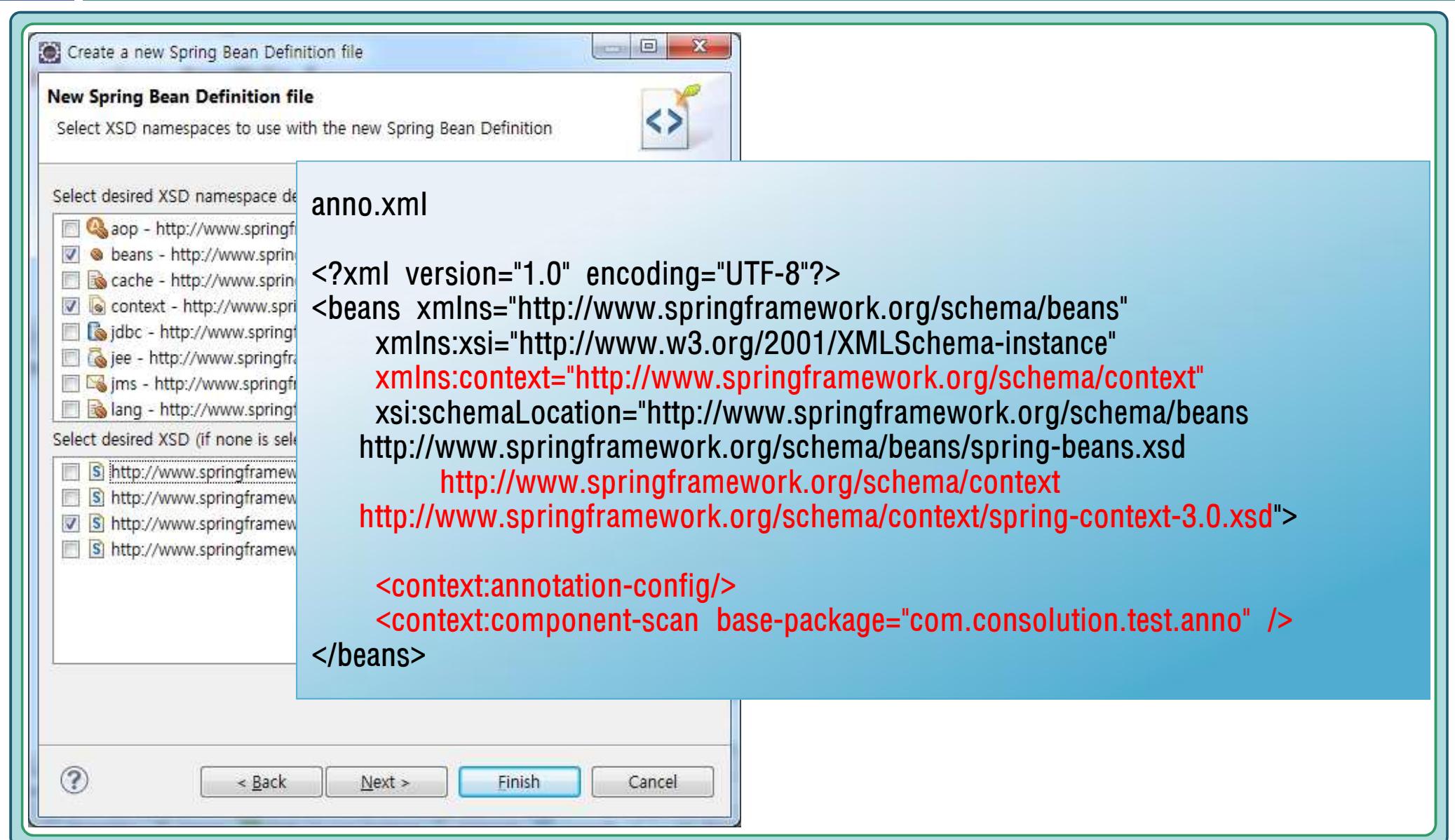
- **@Required**
 - ▣ org.springframework.beans.factory.annotation.Required
 - ▣ setter메서드에
 - ▣ 필수 속성이 되게 함
 - ▣ 설정파일에 <property/> 또는 <constructor-arg/> 엘리먼트를 통해 명시적으로 값이 설정되거나, @Autowiring에 의해 값이 설정되어야 함.
- **@Autowired**
 - ▣ org.springframework.beans.factory.annotation.Autowired
 - ▣ 속성, 생성자, setter메서드, 멤버변수를 설정하는 일반 메서드
 - ▣ 타입에 의존하는 객체를 삽입해 줌
- **@Qualifier**
 - ▣ org.springframework.beans.factory.annotation.Qualifier
 - ▣ @Autowired 와 같이 사용
 - ▣ 같은 타입의 빈 객체들이 있을 경우 특정 빈을 사용하도록 함
 - ▣ 설정파일의 <qualifier>태그의 value 속성의 값을 아노테이션 값으로 사용
 - <bean id="foo" class="x.y.Foo">
 - <qualifier value="action"/>
 - ▣ @Qualifier("action")
 - ▣ public void setFoo(@Qualifier("action") Foo foo) {}
- **@Resource**
 - ▣ javax.annotation.Resource
 - ▣ Java SE 6 과 Java EE5에 추가
 - ▣ 필요한 자원을 자동으로 연결시켜 줌
 - ▣ name 속성을 이용하면 속성과 빈의 이름이 틀릴 때 사용
 - @Resource(name="myFoo")
- **@PostConstruct, @PreDestroy**
 - ▣ javax.annotation.PostConstruct, javax.annotation.PreDestroy;
 - ▣ lifecycle 아노테이션

3.8.2 빈 객체 스캔

- 설정파일에 <context:component-scan>태그의 base-package속성으로 지정된 패키지 내의 클래스를 검색하여 자동으로 빈으로 등록하는 기능을 제공
 - ▣ xml 설정파일에 여러 빈 정보를 추가하지 않고 특정한 클래스를 빈으로 등록 가능
 - ▣ <context:component-scan base-package="com.abc.prj"/>
- **스프링 2.0 이후**
 - ▣ @Repository
- **스프링 2.5 이후**
 - ▣ @Component
 - org.springframework.stereotype.Component
 - 자동으로 빈으로 등록되게 함
 - 빈의 이름은 첫 문자만 소문자이고 나머지는 클래스 이름과 동일
 - 빈의 이름을 지정해 주려면 @Component("myFoo");
 - @Scope("prototype")으로 빈의 범위를 지정할 수 있음
 - org.springframework.context.annotation.Scope
 - ▣ @Service
 - ▣ @Controller

예제

91



Foo & Bar

92

Foo.java

```
package com.consoluton.test.anno;  
import javax.annotation.Resource;  
import org.springframework.beans.factory.annotation.Required;  
import org.springframework.stereotype.Component;  
  
@Component  
public class Foo {  
  
    private Bar bar;  
  
    @Required  
    @Resource  
    public void setBar(Bar bar) {  
        this.bar = bar;  
    }  
  
    public void doFoo() {  
        bar.doBar();  
    }  
}
```

Bar.java

```
package com.consoluton.test.anno;  
import javax.annotation.*; //PostConstruct, PreDestroy  
import org.springframework.stereotype.Component;  
  
@Component  
public class Bar {  
    private String name;  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public void doBar() {  
        System.out.println("Bar.doBar");  
    }  
  
    @PostConstruct  
    public void start() {  
        System.out.println("Bar.start");  
    }  
  
    @PreDestroy  
    public void stop() {  
        System.out.println("Bar.stop");  
    }  
}
```

AnnoMain.java

93

```
package com.consoltion.test.anno;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AnnoMain {

    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(new String[] { "anno.xml" });
        Foo foo = (Foo)context.getBean("foo");
        foo.doFoo();
    }

}
```

3.8.3 스캔 필터

- <context:component-scan base-package="x.y" scoped-proxy="no">
- <context:include-filter type="regex" expression=".HibernateRepository"/>
- <context:exclude-filter type="aspectj" expression="..*iBatisRepository"/>
- </context:component-scan>

| Type속성 | 설명 |
|------------|--|
| annotation | 클래스에 지정한 아노테이션이 적용됐는지의 여부 expression 속성 예 : org.example.SomeAnnotation |
| assignable | 클래스가 지정한 타입으로 할당가능한지의 여부. expression 속성 예 : org.example.SomeClass |
| regex | 클래스 이름이 정규 표현식에 매칭되는지의 여부 expression 속성 예 : org\example\Default.* |
| aspectj | 클래스 이름이 AspectJ 표현식에 매칭되는지의 여부 expression 속성 예 : org.example..*Service+ |
| custom | TypeFilter 인터페이스를 구현한 커스텀 인터페이스 인지의 여부 expression 속성 예 : org.example.MyTypeFilter |

3.8.4 컴포넌트 안에 빈 메타데이터 정의

- org.springframework.context.annotation 패키지의 @Configuration 아노테이션과 @Bean 아노테이션을 이용해 스프링 컨테이너에 새로운 빈 객체를 제공할 수 있다.
- @Configuration
 - ▣ 클래스에 선언
- @ImportResource
 - ▣ 클래스에 선언
 - ▣ xml 설정 정보를 사용할 때
 - ▣ @ImportResource("classpath:/article-repository.xml");
- @Import
 - ▣ 클래스에 선언
 - ▣ @Configuration 클래스에서 다수의 @Configuration 클래스를 묶을 수 있음
 - ▣ @import({ ArticleServiceConfig.class, ArticleRepositoryConfig.class })
- @Bean
 - ▣ 새로운 빈 객체를 제공할 때 사용
 - ▣ 메서드 리턴 유형의 빈이 자동으로 만들어짐
 - ▣ 적용된 메서드의 이름을 빈의 식별 값으로 사용
 - ▣ name 속성으로 빈의 이름을 수정할 수 있음
 - ▣ autowire 속성을 사용해 자동 연관 처리가 가능
 - autowire=Autowire.BY_NAME, Autowire.BY_TYPE, Autowire.NO
 - ▣ initMethod 속성으로 라이프사이클 처리 가능
 - @Bean(initMethod="init")
 - ▣ @Bean 아래 @Scope(value="prototype") 으로 빈의 범위 지정 가능

AnnoMain.java

```
Foo bar2 = (Foo)context.getBean("scottFoo");
System.out.println(bar2);
```

Baz.java

```
@Configuration
public class Baz {

    @Bean(name="scottBar")
    public Bar activeBar() {
        Bar bar = new Bar();
        bar.setName("Scott");
        return bar;
    }
}
```

- 설정파일을 통해
- bean 태그, id속성, class 속성
- constructor-arg 태그, property 태그
- ref, value, list, set, map, props
- 생명주기
 - ▣ 아노테이션, 인터페이스 구현, 커스텀 설정(init-method, destroy-method)
- 아노테이션 설정
 - ▣ context 네임스페이스 지정
 - ▣ 컴포넌트 스캔 패키지 지정, 스캔필터 추가 설정 가능
- 아노테이션
 - ▣ @Component, @Service, @Controller, @Repository
 - ▣ @Configuration, @Bean, ...
 - ▣ @Autowired, @Resource

빈 자동등록을 사용할 경우 기본타입 또는 String 타입 멤버변수의 setter 메서드가 인터페이스에 정의되어 있어야 합니다.

그래야 인터페이스를 통해 빈을 받아 setter 메서드로 필드의 값을 설정할 수 있습니다.

4. Aspect Oriented Programming

1. AOP 개요
2. AOP 주요 용어
3. 스프링에서의 AOP
4. 스프링 API를 이용한 AOP구현
5. POJO클래스를 이용한 AOP구현

4.1 AOP 개요

- AOP는 문제를 바라보는 관점을 기준으로 프로그래밍하는 기법을 말한다.
- 문제를 해결하기 위한 핵심 관심 사항과 전체에 적용되는 공통관심 사항을 기준으로 프로그래밍함으로써 공통모듈을 여러 코드에 쉽게 적용할 수 있도록 도와 준다.
- AOP에서 중요한 개념은 「횡단 관점의 분리(Separation of Cross-Cutting Concern)」이다.
- OOP를 더욱 OOP답게 만들어 준다.

공통관심사항(cross-cutting concern)

- 공통기능으로 어플리케이션 전반에 걸쳐 필요한 기능
예) 로깅, 트랜잭션, 보안 등

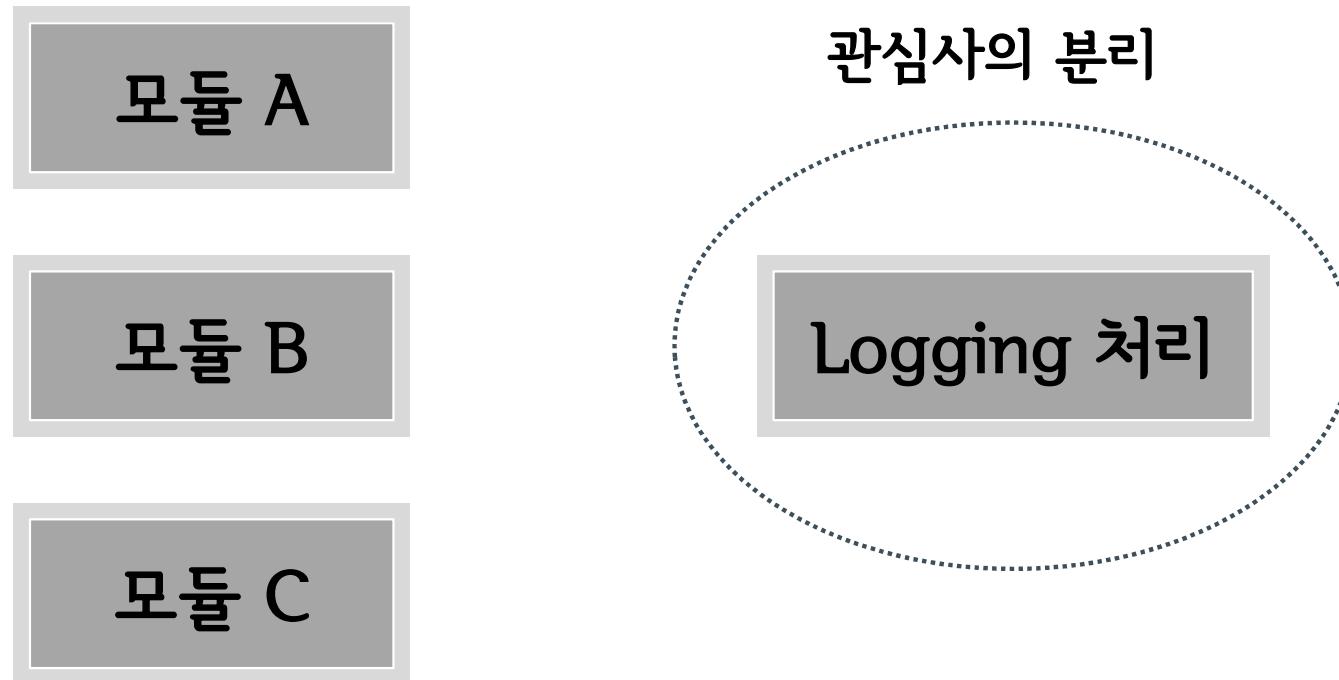
핵심관심사항(core concern)

- 핵심로직, 핵심 비즈니스 로직
예) 계좌이체, 이자계산, 대출처리 등

4.1.1 횡단관점의 분리 - OOP

99

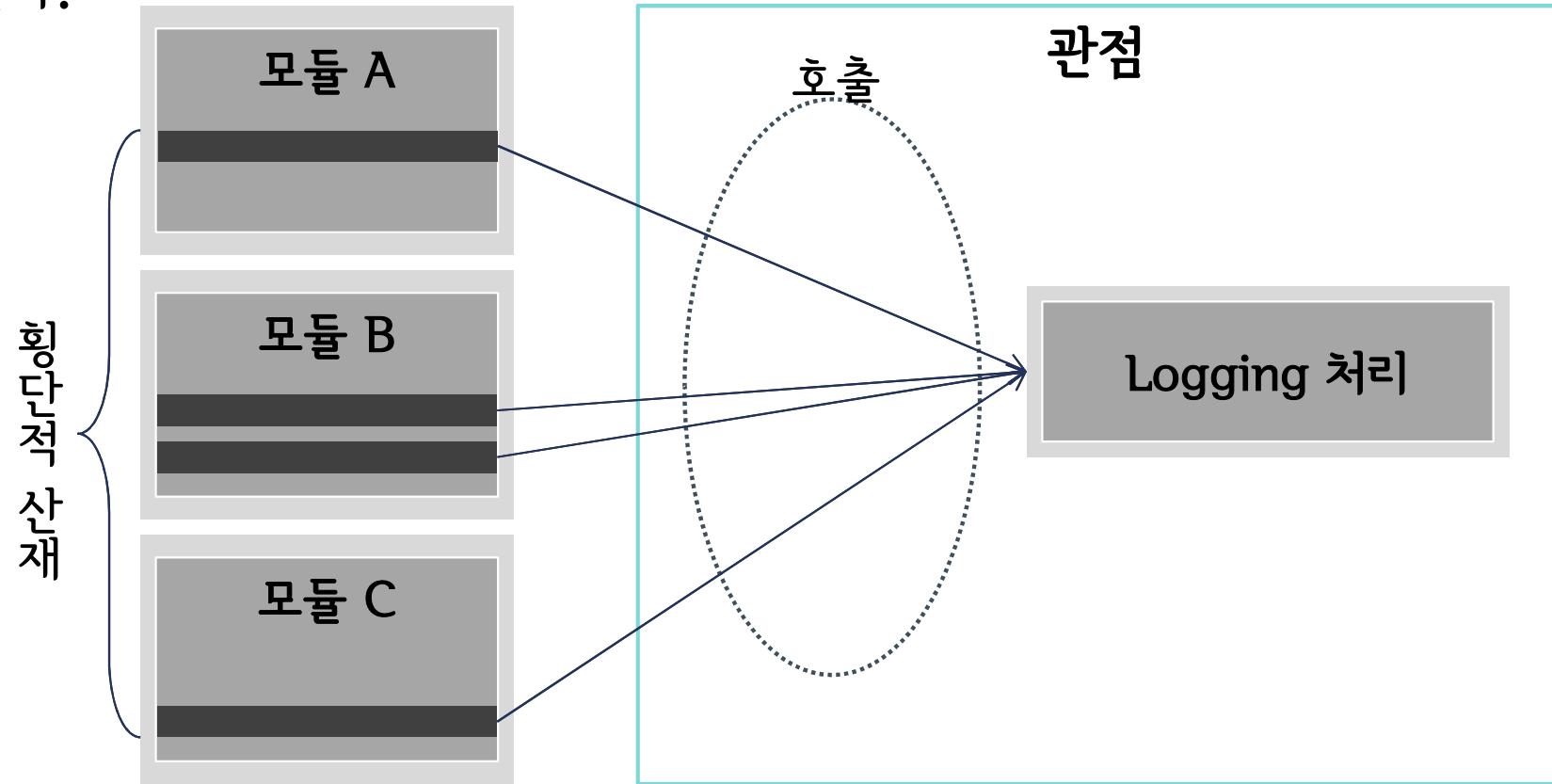
- OOP에서는 횡단관점을 분리를 위해 공통 기능들을 하나의 클래스라는 단위로 모으고 그것들을 모듈로부터 분리함으로써 재사용성과 보수성을 높이고 있다.



4.1.1 횡단관점의 분리 - OOP

100

- 각 모듈로부터 공통기능으로 분리하는 것으로 성공했지만 그 기능을 사용하기 위해 공통 기능을 호출하는 코드까지는 각 모듈로부터 분리할 수 없다. 그렇기 때문에 분리한 공통 기능을 이용하기 위한 코드가 각 모듈에 횡단으로 산재하게 된다.

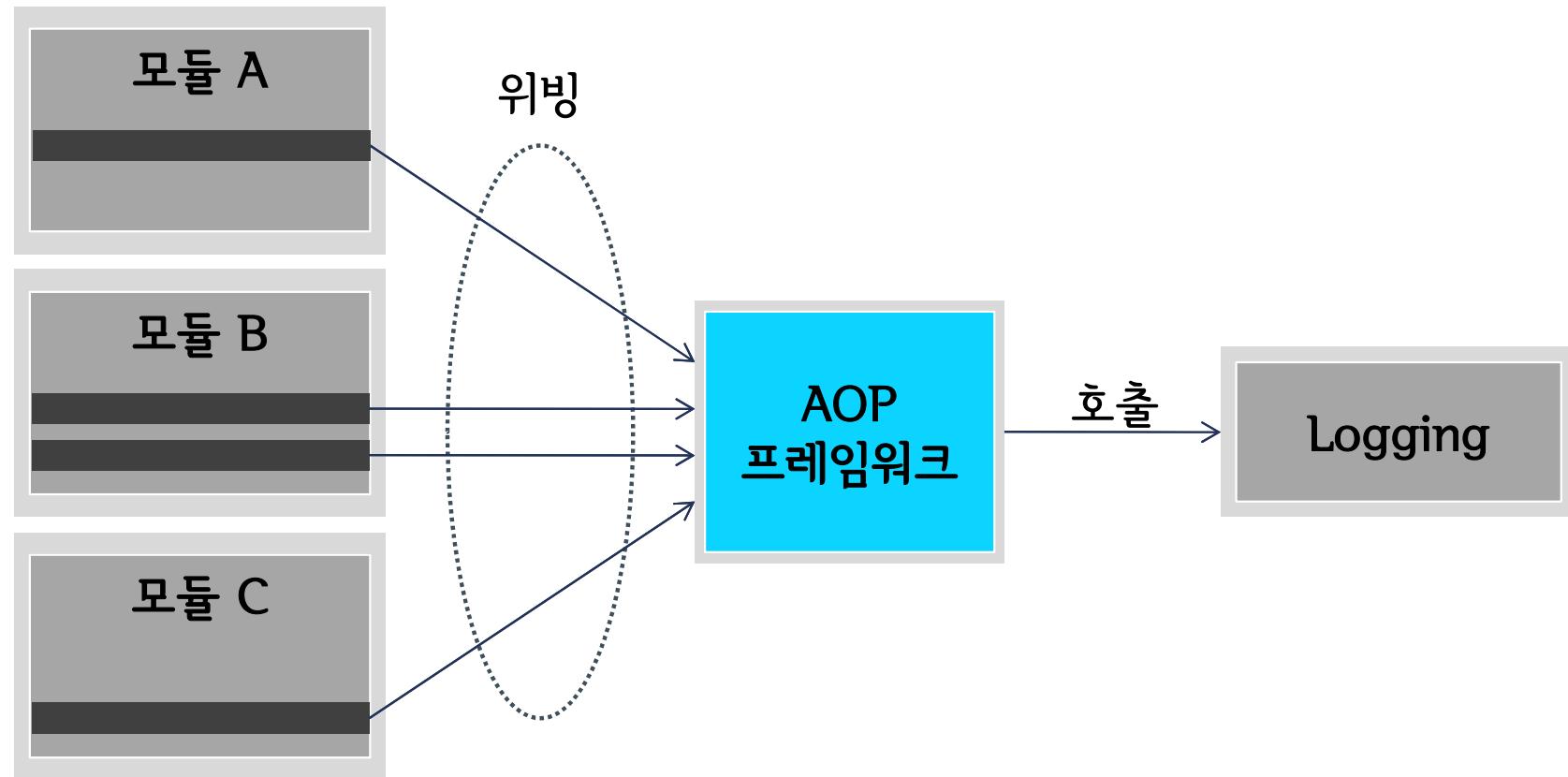


4.1.1 횡단관점의 분리 - AOP

- AOP에서는 핵심 로직을 구현한 코드에서 공통 기능을 직접적으로 호출하지 않는다.
- AOP에서는 분리한 공통 기능의 호출까지도 관점으로 다룬다. 그리고 이러한 각 모듈로 산재한 관점을 횡단 관점이라 부르고 있다.
- AOP에서는 이러한 횡단 관점까지 분리함으로써 각 모듈로부터 관점에 관한 코드를 완전히 제거하는 것을 목표로 한다.

4.1.1 횡단관점의 분리 - AOP

- AOP에서는 핵심 로직을 구현한 코드를 컴파일 하거나, 컴파일된 클래스를 로딩하거나 또는 로딩한 클래스의 객체를 생성할 때 핵심 로직 구현 코드 안에 공통 기능이 삽입된다.

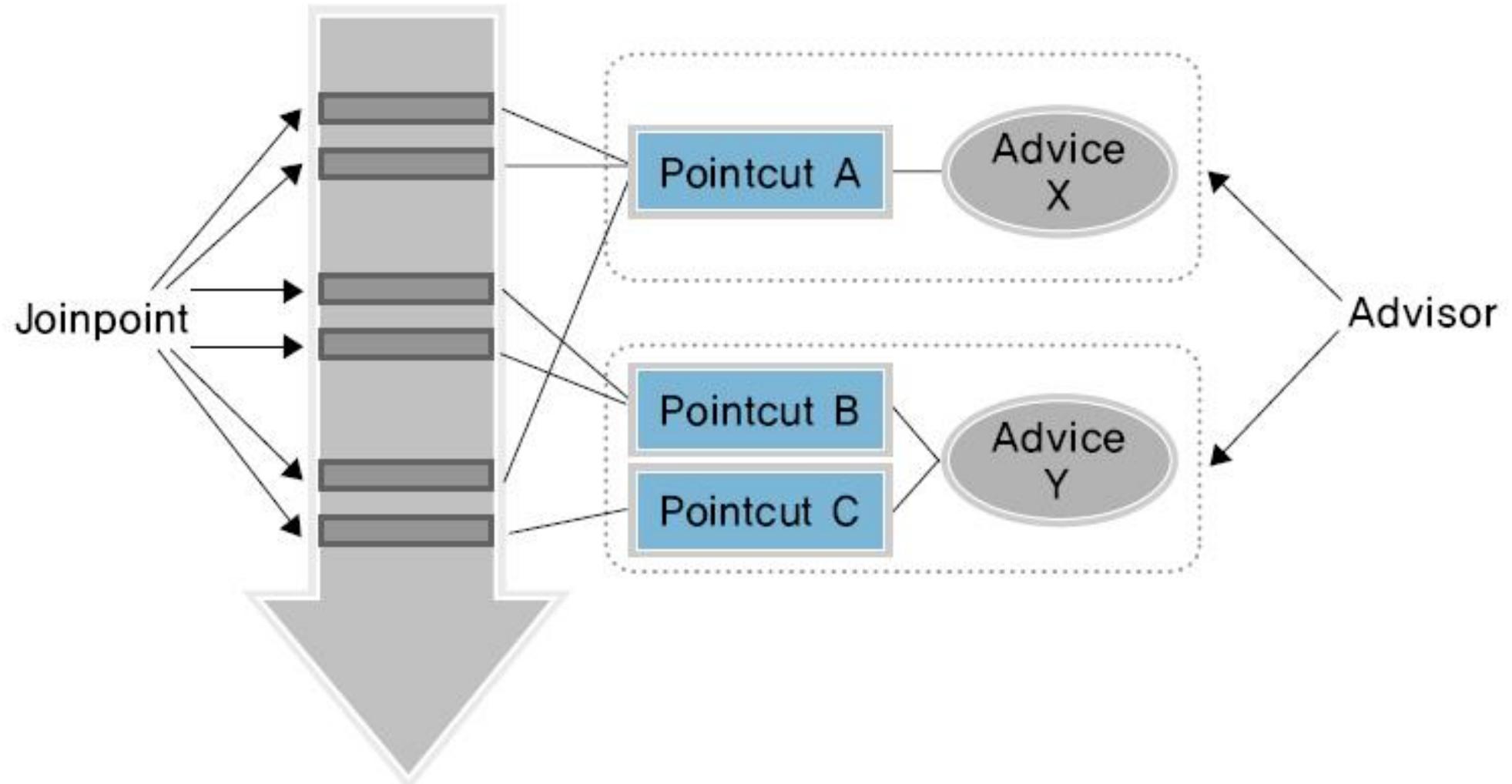


4.1.2 AOP 주요 용어

- Join point
 - ▣ 「클래스의 인스턴스 생성 시점」, 「메서드 호출 시점」 및 「예외 발생 시점」과 같이 애플리케이션을 실행할 때 특정 작업이 시작되는 시점
 - ▣ Advice를 적용 가능한 지점.
 - ▣ 스프링 AOP에서는 메서드 실행으로 표현됨
- Advice
 - ▣ 특정한 Join point에서의 행위(삽입되어져 동작할 수 있는 코드).
 - ▣ Before advice, After returning advice, After throwing advice, After (finally) advice, Around advice
- Pointcut
 - ▣ Join point의 부분집합
 - ▣ 실제로 Advice가 적용되는 Join point
 - ▣ 스프링에서는 정규 표현식이나 AspectJ의 문법을 이용하여 Pointcut을 정의할 수 있음
- Weaving
 - ▣ Advice(공통코드)를 핵심 로직 코드에 삽입하는 것.
- Target object
 - ▣ 하나 또는 그 이상의 Aspect에 의해 Advice되는 객체
 - ▣ 핵심 로직을 구현하는 클래스.
 - ▣ 스프링에서는 런타임 프록시를 사용해 구현됨
- Aspect
 - ▣ 여러 객체에 공통으로 적용되는 공통 관점 사항.
 - ▣ 트랜잭션이나 보안등은 Aspect의 좋은 예

4.1.2 AOP 주요 용어

프로그램 실행 순서



4.1.3 Weaving 방식

- Advice를 위빙하는 방식에는 다음과 같이 3가지 방식이 존재한다.
 - 1. 컴파일 시에 위빙하기
 - 별도 컴파일러를 통해 핵심 관심사 모듈의 사이 사이에 관점(Aspect) 형태로 만들어진 횡단 관심사 코드들이 삽입되어 관점(Aspect)이 적용된 최종 바이너리가 만들어지는 방식(ex. AspectJ)
 - 2. 클래스 로딩시에 위빙하기
 - 별도의 Agent를 이용하여 JVM이 클래스를 로딩할 때 해당 클래스의 바이너리 정보를 변경한다. 즉, Agent가 횡단 관심사 코드가 삽입된 바이너리 코드를 제공함으로써 AOP를 지원하게 된다.(ex. AspectWerkz)
 - 3. 런타임 시에 위빙하기
 - 소스 코드나 바이너리 파일의 변경 없이 프록시를 이용하여 AOP를 지원하는 방식이다. 프록시를 통해 핵심 관심사를 구현한 객체에 접근하게 되는데, 프록시는 핵심 관심사 실행 전후에 횡단 관심사를 실행한다. 따라서 프록시 기반의 런타임 엣기의 경우 메소드 호출시에만 AOP를 적용할 수 있다는 제한점이 있다.(ex. Spring AOP)

위빙은 공통코드를 핵심코드에 삽입하는 것

4.2 스프링에서의 AOP

- 스프링에서는 자체적으로 런타임시에 위빙하는 "프록시 기반의 AOP"를 지원하고 있다.
- 프록시 기반의 AOP는 메서드 호출 조인포인트만 지원한다.
- 스프링에서 어떤 대상 객체에 대해 AOP를 적용할지의 여부는 설정 파일을 통해서 지정한다.
 - ▣ 스프링은 설정 정보를 이용하여 런타임에 대상 객체에 대한 프록시 객체를 생성하게 된다.
 - ▣ 따라서, 대상 객체를 직접 접근하는 것이 아니라 프록시를 통한 간접 접근을 하게 된다.
- 스프링은 완전한 AOP 기능을 제공하는 것이 목적이 아니라 엔터프라이즈 어플리케이션을 구현하는데 필요한 기능을 제공하는 것을 목적으로 하고 있다.
- 필드 값 변경 등 다양한 조인포인트를 이용하려면 AspectJ와 같은 다른 AOP솔루션을 이용해야 한다.

4.2.1 스프링에서의 AOP 구현 방법

- 스프링에서는 다음 3가지 방식으로 AOP구현을 지원한다.
 1. 스프링 API를 이용한 AOP구현
 2. XML 기반의 POJO 클래스를 이용한 AOP구현
 3. AspectJ 5/6에서 정의한 @Aspect 아노테이션 기반의 AOP구현
 - ❖ 개발자가 직접 스프링 AOP API를 사용해서 AOP를 구현하는 경우는 많지 않음
 - ❖ 전자정부 표준프레임워크는 XML Schema 기반 POJO클래스를 이용한 AOP 구현을 사용함

4.3 스프링 API를 이용한 AOP

- 스프링 API를 이용한 AOP를 구현하는 과정은 다음과 같다.
 1. Advice 클래스를 작성한다.
 2. 설정 파일에 Pointcut을 설정한다.
 3. 설정 파일에 Advice와 Pointcut을 묶어 놓는 Advisor를 설정한다.
 4. 설정 파일에 ProxyFactoryBean 클래스를 이용하여 대상 객체에 Advisor를 적용한다.
 5. getBean() 메서드로 빈 객체(프록시 객체)를 가져와 사용한다.

aopalliance.jar 파일이 필요

- AOP 를 사용하기 위해서 필요한 라이브러리
- Advice 용 Class 작성시 사용됨.
- org.springframework.aop.MethodBeforeAdvice and AfterReturningAdvice and ThrowsAdvice 세가지 Advice 를 하나로 묶음.
- <http://www.jarfinder.com/>
- <http://sourceforge.net/projects/aopalliance/>

4.3.1 Advice 작성

- 스프링 AOP는 메서드 호출 관련 Advice만 제공하며 이들 Advice는 인터페이스 형태로 제공된다.

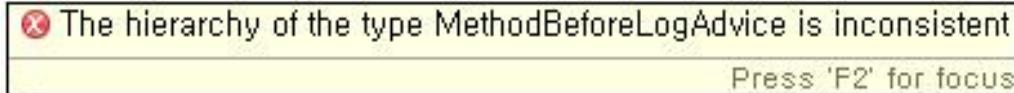
| 인터페이스 | 설명 |
|----------------------|---|
| MethodBeforeAdvice | 대상 객체의 메서드를 실행하기 전에 공통기능을 실행할 때 사용되는 Advice. |
| AfterReturningAdvice | 대상 객체의 메서드 실행 이후에 공통기능을 실행할 때 사용되는 Advice. |
| ThrowsAdvice | 대상 객체의 메서드가 실행하는 도중 예외가 발생할 경우 공통기능을 실행할 때 사용되는 Advice. |
| MethodInterceptor | 위 세 가지를 하나로 묶은 Advice로 메서드 실행 전, 후, 예외 발생시 공통 기능을 실행할 수 있다. |

4.3.2 Advice 작성 - MethodBeforeAdvice

MethodBeforeLogAdvice.java

```
public class MethodBeforeLogAdvice implements MethodBeforeAdvice {
    public void before(Method method, Object[] args, Object target) {
        System.out.println("[Log]METHOD Before : "
            + method.getName() + " on " + target + " calling");
    }
}
```

아래와 같이 발생하는 에러는

 The hierarchy of the type MethodBeforeLogAdvice is inconsistent
Press 'F2' for focus

→ aopalliance-1.0.jar 파일이 필요함

<http://sourceforge.net/projects/aopalliance/>

MethodBeforeAdvice 인터페이스는
대상 객체의 메서드를 실행하기 전에 공통기능을 실행할 때 사용되는 Advice.

4.3.3 Advice 작성 - AfterReturningAdvice

MethodAfterLogAdvice.java

```
public class MethodAfterLogAdvice implements AfterReturningAdvice {  
  
    public void afterReturning(Object returnValue, Method method, Object[] args,  
        Object target) throws Throwable {  
        System.out.println("[Log]METHOD After : "+method.getName() +  
            " on "+target+" called");  
    }  
  
}
```

AfterReturningAdvice는
대상 객체의 메서드 실행 이후에 공통기능을 실행할 때 사용되는 Advice

4.3.4 Advice 작성 - ThrowsAdvice

MethodThrowsLogAdvice.java

```
public class MethodThrowsLogAdvice implements ThrowsAdvice {  
  
    public void afterThrowing(Exception e){  
        System.out.println("예외 발생 ");  
    }  
  
}
```

ThrowsAdvice는

대상 객체의 메서드가 실행하는 도중 예외가 발생할 경우 공통기능
을 실행할 때 사용되는 Advice.

4.3.5 Advice 작성 - MethodInterceptor

PerformanceCheckAdvice.java

```

import org.aopalliance.intercept.*;
public class PerformanceCheckAdvice implements MethodInterceptor {
    public Object invoke(MethodInvocation invocation) throws Throwable {
        String methodName = invocation.getMethod().getName();
        long startTime = System.nanoTime();
        System.out.println("[Log]METHOD Before --> " + methodName+ " time check start");
        Object obj = null;
        try{
            obj= invocation.proceed();
        }catch(Exception e){
            System.out.println("[Log]METHOD error --> "+ methodName);
        }

        long endTime = System.nanoTime();
        System.out.println("[Log]METHOD After --> " + methodName+ " time check end");
        System.out.println("[Log] " + methodName + " Processing time is "+(endTime - startTime)+"ns");
        return obj;
    }
}

```

MethodInterceptor는

MethodBeforeAdvice, AfterReturningAdvice, ThrowsAdvice 세 가지를 하나로 묶은 Advice로 메서드 실행 전, 후, 예외 발생시 공통 기능을 실행할 수 있다.

4.3.6 Advice 빈 설정

- Advice 자체는 스프링 빈이므로 다른 스프링 빈과 마찬가지로 설정 파일에 <bean>태그를 이용하여 설정한다.

applicationContext.xml

```
...
<bean id="greetingTarget" class="myspring.aop.GreetingServiceImpl">
    <property name="greeting">
        <value>Hello</value>
    </property>
</bean>
<bean id="beforeLogAdvice" class="myspring.aop.MethodBeforeLogAdvice"/>
<bean id="afterLogAdvice" class="myspring.aop.MethodAfterLogAdvice"/>
<bean id="throwsLogAdvice" class="myspring.aop.MethodThrowsLogAdvice"/>
<bean id="performanceCheckAdvice" class="myspring.aop.PerformanceCheckAdvice"/>
...
```

4.3.7 Pointcut 설정

- Advice를 어떤 조인포인트에 적용할지를 설정한다.
- 스프링은 정규표현식이나 AspectJ의 문법을 이용하여 포인트 컷을 정의할 수 있는 클래스를 제공한다.

applicationContext.xml

```
...
<bean id="helloPointcut"
      class="org.springframework.aop.support.JdkRegexpMethodPointcut">
    <property name="pattern">
      <value>.*sayHello.*</value>
    </property>
</bean>
...
```

모든 메서드를 포인트컷으로 하려면 .:.*

4.3.8 Advisor 설정

- 어떤 Advice를 어떤 포인트컷에 적용할지를 설정한다.

applicationContext.xml

```
...
<bean id="helloAdvisor"
      class="org.springframework.aop.support.DefaultPointcutAdvisor">
    <property name="advice">
        <ref bean="beforeLogAdvice" />
    </property>
    <property name="pointcut">
        <ref bean="helloPointcut" />
    </property>
</bean>
...
```

4.3.9 ProxyFactoryBean을 이용한 Advice 적용

- 프록시를 만들어서 실제 대상 빈 객체의 포인트 컷에 Advice를 적용하도록 한다.

프록시 객체

applicationContext.xml

```
...
<bean id="greeting" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="target">
        <ref bean="greetingTarget"/>
    </property>
    <property name="interceptorNames">
        <list>
            <value>helloAdvisor</value>
        </list>
    </property>
</bean>
...
```

4.3.10 프록시 객체 사용

- 실제 대상 빈이 아니라 대상 빈 객체에 대한 프록시 빈을 가져와서 사용한다.

GreetingTest.java

```
public class GreetingTest {
    public static void main(String[] args) {
        ApplicationContext ctx =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        GreetingService bean = (GreetingService)ctx.getBean("greeting");
        bean.sayHello("홍길동"); //AOP 적용되어 실행됨
        bean.sayGoodbye("홍길동");//AOP 적용 안됨
    }
}
```

↑

greetingTarget 이 아니고 greeting임
프록시 객체임

4.3.11 프록시 객체를 위한 클래스

□ 서비스 구현 클래스

GreetingServiceImpl.java

```
public class GreetingServiceImpl implements GreetingService {  
  
    private String greeting;  
  
    public void setGreeting(String greeting) {  
        this.greeting = greeting;  
    }  
  
    @Override  
    public void sayHello(String name) {  
        System.out.println("sayHello : " + greeting + ":" + name);  
    }  
  
    @Override  
    public void sayGoodbye(String name) {  
        System.out.println("sayGoodbye : " + greeting + ":" + name);  
    }  
}
```

4.4 POJO를 이용한 AOP

- 스프링2 버전부터 스프링 API를 사용하지 않은 POJO클래스를 이용하여 Advice를 개발하고 적용할 수 있는 방법이 추가되었다.
- 스프링2 버전의 xml 스키마 확장 기법을 통해 설정 파일도 보다 쉽게 설정이 가능하다.
- AspectJ 모듈 다운로드
 - ▣ <http://www.eclipse.org/aspectj/downloads.php>에서 다운로드
 - ▣ aspectj-1.x.x.j ar파일 안에 포함되어 있는 **aspectjweaver.jar** 파일을 빌드패스에 추가
 - 2012.9월에는 1.7.1 버전이 안정화된 버전임
 - 1.6 버전 사용 가능

4.4.1 Aspect 작성

LogAspect.java

```
public class LogAspect {  
    public void beforeLogging(){  
        System.out.println("** 메서드 호출 전**");  
    }  
    public void afterLogging(Object returnValue){  
        System.out.println("** 메서드 호출 후**");  
    }  
    public void throwingLogging(Exception ex){  
        System.out.println("** 예외 발생 : "+ex.getMessage()+"**");  
    }  
    public void alwaysLogging(){  
        System.out.println("** 항상 실행 **");  
    }  
}
```

4.4.2 Aspect 작성

PerformanceAspect.java

```
import org.aspectj.lang.*;  
public class PerformanceAspect {  
    public Object timeCheck(ProceedingJoinPoint joinPoint) throws Throwable{  
        Signature s= joinPoint.getSignature();  
        String methodName = s.getName();  
        long startTime = System.nanoTime();  
        System.out.println("[Log]METHOD Before : " + methodName+" time check start");  
  
        Object obj = null;  
        try{  
            obj = joinPoint.proceed();  
        }catch(Exception e){  
            System.out.println("[Log]METHOD error : "+ methodName);  
        }  
  
        long endTime = System.nanoTime();  
        System.out.println("[Log]METHOD After : " + methodName+" time check end");  
        System.out.println("[Log] "+methodName + " Processing time is "+(endTime - startTime)+"ns");  
        return obj;  
    }  
}
```

4.4.3 Aspect 설정

- 설정 파일에 aop 네임스페이스 및 네임스페이스와 관련된 스키마를 추가한다.
- <aop:config>태그를 이용하여 AOP관련정보를 설정한다.

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
    ...
</beans>
```

4.4.4 Aspect 설정 – Advice 정의 관련 태그

| 태그 | 설명 |
|-----------------------|--|
| <aop:before> | 메서드를 실행하기 전에 적용되는 Advice를 정의한다. |
| <aop:after-returning> | 메서드가 정상적으로 실행된 이후에 적용되는 Advice를 정의한다. |
| <aop:after-throwing> | 메서드가 실행하는 도중 예외가 발생할 경우 적용되는 Advice를 정의한다. |
| <aop:after> | 메서드가 정상적으로 실행되던지, 예외를 발생시키던지 여부에 상관없이 적용되는 Advice를 정의한다. |
| <aop:around> | 메서드 실행 전, 후, 예외 발생시 적용 가능한 Advice를 정의한다. |

4.4.5 Aspect 설정

applicationContext.xml

```
<bean id="greetingTarget" class="com.consoluton.test.aop.GreetingServiceImpl">
    <property name="greeting">
        <value>Hello</value>
    </property>
</bean>

<bean id="logAspect" class="com.consoluton.test.aop.pojo.LogAspect" />
<bean id="performanceAspect" class="com.consoluton.test.aop.pojo.PerformanceAspect" />

<aop:config>
    <aop:pointcut id="publicMethod" expression="execution(public * *(..))" />
    <aop:aspect ref="logAspect">
        <aop:before method="beforeLogging" pointcut-ref="publicMethod" />
        <aop:after-returning method="afterLogging" pointcut-ref="publicMethod" returning="returnValue" />
        <aop:after-throwing method="throwingLogging" pointcut-ref="publicMethod" throwing="ex" />
        <aop:after method="alwaysLogging" pointcut-ref="publicMethod" />
    </aop:aspect>
    <aop:aspect ref="performanceAspect">
        <aop:around method="timeCheck" pointcut-ref="publicMethod"/>
    </aop:aspect>
</aop:config>
```

4.4.6 빈 객체 사용

- 스프링 API로 구현한 경우처럼 프록시 객체를 설정하지 않으므로 빈 객체를 얻어와 사용하면 된다.
- 포인트 컷에 expression에 설정한 정보로 자동으로 Advice가 해당 포인트컷에 적용된다.

GreetingTest.java

```
public class GreetingTest {  
    public static void main(String[] args) {  
        ApplicationContext ctx =  
            new ClassPathXmlApplicationContext("applicationContext.xml");  
        GreetingService bean = (GreetingService)ctx.getBean("greetingTarget");  
        bean.sayHello("홍길동");  
        bean.sayGoodbye("홍길동");  
    }  
}
```

4.4.7 적용 순서

127

- 정상 실행인 경우
 - ▣ before
 - ▣ around(대상 메서드 수행 전)
 - ▣ 대상 메서드
 - ▣ around(finally)
 - ▣ around(대상 메서드 수행 후)
 - ▣ afterReturning
 - ▣ after(finally)
- 예외 발생할 경우
 - ▣ before
 - ▣ around(대상 메서드 수행 전)
 - ▣ 대상 메서드(만약 예외 발생한다면...)
 - ▣ around(throws)
 - ▣ around(대성 메서드 후행 후)
 - ▣ afterThrowing
 - ▣ after(finally)

4.4.8 Pointcut 표현식

- execution
 - ▣ 빈의 조건에 맞는 메서드나 생성자의 실행을 포인트컷으로 함
- within
 - ▣ 빈이 조건에 설정한 타입이라면, 메서드의 실행을 포인트컷으로 함
- this
 - ▣ 빈이 조건에 설정한 타입에 대입할 수 있는 인스턴스라면, 메서드 실행을 포인트컷으로 함
- target
 - ▣ 대상이 되는 객체가 조건에 설정된 타입에 대입할 수 있는 인스턴스라면, 메서드의 실행을 포인트컷으로 함
- args
 - ▣ 메서드의 인수가 조건에 설정한 타입에 대입할 수 있는 인스턴스라면, 메서드의 실행을 포인트컷으로 함

4.4.9 Pointcut 표현식

- execution(수식어패턴? 리턴타입패턴 클래스이름패턴?이름패턴(파라미터패턴))
 - ▣ 수식어패턴 : 생략 가능, public, protected 등이 옴
 - ▣ 리턴타입패턴 : 리턴 타입 명시
 - ▣ 클래스이름패턴 : 생략 가능, 클래스 이름 명시
 - ▣ 이름패턴 : 메서드 이름 명시
 - ▣ 파라미터패턴 : 매칭될 파라미터에 대해서 명시
- 각 패턴은 * 을 이용해 모든 값을 표현 가능
- .. 을 이용하면 0개 이상이라는 의미를 표현
 - ▣ 클래스이름패턴에서 패키지 이름 뒤에 ..을 쓰면 서브패키지도 찾는다.
 - ▣ 파라미터패턴에 .. 을 쓰면 파라미터가 0개 이상
 - (*) 으로 하면 파라미터를 1개 포함해야 함
 - (*, *) 으로 하면 파라미터가 2개 이어야 함
 - (Integer, ..)으로 하면 첫 번째 파라미터는 Integer형이며, 1개 이상의 파라미터를 가짐
- 논리연산자 &&, ||, ! 사용 가능
 - ▣ execution(public * set*(..)) || execution(* sample1..*.*(..))
 - ▣ public 메서드이면서 set으로 시작되는 모든 메서드 또는 sample1 패키지에 있는 모든 클래스의 모든 메서드를 대상으로 함
- 예)
 - ▣ execution(* com.myapp.aop..*.select*(..))
 - com.myapp.aop패키지 및 하위 패키지에 있는 파라미터가 0개 이상인 메서드 이름이 select 로 시작하는 메서드 호출(리턴타입과도 무관함)

4.5 @Aspect 아노테이션을 이용한 AOP

- AspectJ 5 버전에 추가
- 스프링 2부터 Aspect 아노테이션을 지원
- 구현과정
 - ▣ XML 스키마 기반의 AOP를 구현하는 과정과 유사
 - ▣ 차이점은
 - @Aspect 아노테이션을 이용해서 Aspect 클래스를 구현한다. 이때 Aspect 클래스는 Advice를 구현한 메서드와 Pointcut을 포함한다.
 - XML 설정에서 <aop:aspectj-autoproxy/>를 설정한다.
- 아노테이션
 - ▣ @Aspect
 - ▣ @Pointcut
 - ▣ @Around : 메서드 수행 전 후
 - ▣ @Before : 메서드 수행 전
 - ▣ @AfterReturning : 메서드 정상 실행 후(예외 발생시 실행 안됨)
 - ▣ @AfterThrowing : 예외 발생 시
 - ▣ @After : 메서드 정상 종료 및 예외 발생시 무조건 실행

4.5.1 Aspect클래스 작성(예 1)

PerformanceAspect.java

```

import org.aspectj.lang.*;
import org.aspectj.lang.annotation.*;

@Aspect
public class PerformanceAspect {

    @Pointcut("execution(public * com.consoluton.test.aop..*sayHello(..))")
    private void profileTarget() {}

    @Around("profileTarget()")
    public Object trace(ProceedingJoinPoint joinPoint) throws Throwable{
        Signature s= joinPoint.getSignature();
        String methodName = s.getName();
        System.out.println("[Log]METHOD Before --> " + methodName+" time check start");

        long startTime = System.nanoTime();

        Object o = null;
        try{
            o= joinPoint.proceed();
        }catch(Exception e){
            System.out.println("[Log]METHOD error --> "+ methodName);
        }

        long endTime = System.nanoTime();
        System.out.println("[Log]METHOD After --> " + methodName+" time check end");
        System.out.println("[Log] " + methodName + " Processing time is "+(endTime - startTime)+"ns");
        return o;
    }
}

```

4.5.1 Aspect클래스 작성(예 2)

RandomAspect.java

```

import java.io.IOException;

import org.aspectj.lang.*;
import org.aspectj.lang.annotation.*;

@Aspect
public class RandomAspect {

    @Pointcut("execution (public int com.aop.RandomBiz.calc(..))")
    private void calc() {}

    @Before("calc()")
    public void beforeLog(JoinPoint joinPoint) {
        System.out.println("@before - " + joinPoint.getSignature().getName());
    }

    @AfterReturning(pointcut="calc()", returning="returnValue")
    public void afterLog(JoinPoint joinPoint, int returnValue) {
        System.out.println("@after returning - " + returnValue);
    }

    @AfterThrowing(pointcut="calc()", throwing="exception")
    public void afterThrowLog(Exception exception) {
        System.out.println("@after throwing - " + exception.getMessage());
    }
}

```

```

@Around("calc()")
public int aroundLog(ProceedingJoinPoint joinPoint) throws
    Throwable {
    System.out.println("@around - before");
    Object obj = null;
    long startTime = System.nanoTime();
    try {
        obj = joinPoint.proceed();
    } catch(IOException e) {
        System.out.println("@around - throws");
        throw new RuntimeException(e.getMessage());
    } catch(RuntimeException e) {
        System.out.println("@around - runtimeException");
    } finally {
        System.out.println("@around - finally");
    }
    long endTime = System.nanoTime();
    System.out.println("@around - after");
    System.out.println("@around - result : " + (int)obj);
    System.out.println(endTime - startTime + "ns");
    return (int)obj;
}

```

4.5.2 설정파일

applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">

    <aop:aspectj-autoproxy />
    <bean id="greetingTarget" class="com.consoltion.test.aop.GreetingServiceImpl">
        <property name="greeting">
            <value>Hello_annot</value>
        </property>
    </bean>
    <bean id="randomBiz" class="com.aop.RandomBiz"/>

    <bean id="performanceAspect" class="com.consoltion.test.aop.annot.PerformanceAspect" />
    <bean id="randomAspect" class="com.aop.anno.RandomAspect"/>
</beans>

```

component-scan
사용 가능

New Spring Bean Definition file
Select XSD namespaces to use with the new Spring Bean Definition

Select desired XSD namespace declarations:

- aop - http://www.springframework.org/schema/aop
- beans - http://www.springframework.org/schema/beans
- c - http://www.springframework.org/schema/c
- cache - http://www.springframework.org/schema/cache
- context - http://www.springframework.org/schema/context
- jdbc - http://www.springframework.org/schema/jdbc
- lio - http://www.springframework.org/schema/lio

4.5.3 빈 객체 사용

GreetingTest.java

```
public class GreetingTest {  
    public static void main(String[] args) {  
        ApplicationContext ctx =  
            new ClassPathXmlApplicationContext("applicationContext.xml");  
        GreetingService bean = (GreetingService)ctx.getBean("greetingTarget");  
        bean.sayHello("홍길동");  
        bean.sayGoodbye("홍길동");  
    }  
}
```

4.5.4 적용 순서

135

- 정상 실행인 경우
 - ▣ around(대상 메서드 수행 전)
 - ▣ before
 - ▣ 대상 메서드
 - ▣ around(finally)
 - ▣ around(대상 메서드 수행 후)
 - ▣ after(finally)
 - ▣ afterReturning
- 예외 발생할 경우
 - ▣ before
 - ▣ around(대상 메서드 수행 전)
 - ▣ 대상 메서드(만약 예외 발생한다면...)
 - ▣ after(finally)
 - ▣ afterThrowing

4.5.4 Pointcut 표현식

- execution
 - ▣ 빈의 조건에 맞는 메서드나 생성자의 실행을 포인트컷으로 함
- within
 - ▣ 빈이 조건에 설정한 타입이라면, 메서드의 실행을 포인트컷으로 함
- this
 - ▣ 빈이 조건에 설정한 타입에 대입할 수 있는 인스턴스라면, 메서드 실행을 포인트컷으로 함
- target
 - ▣ 대상이 되는 객체가 조건에 설정된 타입에 대입할 수 있는 인스턴스라면, 메서드의 실행을 포인트컷으로 함
- args
 - ▣ 메서드의 인수가 조건에 설정한 타입에 대입할 수 있는 인스턴스라면, 메서드의 실행을 포인트컷으로 함

4.5.4 Pointcut 표현식

- execution(수식어패턴? 리턴타입패턴 클래스이름패턴?이름패턴(파라미터패턴))
 - ▣ 수식어패턴 : 생략 가능, public, protected 등이 옴
 - ▣ 리턴타입패턴 : 리턴 타입 명시
 - ▣ 클래스이름패턴 : 생략 가능, 클래스 이름 명시
 - ▣ 이름패턴 : 메서드 이름 명시
 - ▣ 파라미터패턴 : 매칭될 파라미터에 대해서 명시
- 각 패턴은 * 을 이용해 모든 값을 표현 가능
- .. 을 이용하면 0개 이상이라는 의미를 표현
 - ▣ 클래스이름패턴에서 패키지 이름 뒤에 ..을 쓰면 서브패키지도 찾는다.
 - ▣ 파라미터패턴에 .. 을 쓰면 파라미터가 0개 이상
 - (*) 으로 하면 파라미터를 1개 포함해야 함
 - (*, *) 으로 하면 파라미터가 2개 이어야 함
 - (Integer, ..)으로 하면 첫 번째 파라미터는 Integer형이며, 1개 이상의 파라미터를 가짐
- 논리연산자 &&, ||, ! 사용 가능
 - ▣ execution(public * set*(..)) || execution(* sample1..*.*(..))
 - ▣ public 메서드이면서 set으로 시작되는 모든 메서드 또는 sample1 패키지에 있는 모든 클래스의 모든 메서드를 대상으로 함
- 예)
 - ▣ execution(* com.myapp.aop..*.select*(..))
 - com.myapp.aop패키지 및 하위 패키지에 있는 파라미터가 0개 이상인 메서드 이름이 select 로 시작하는 메서드 호출(리턴타입과도 무관함)

4.5.5 정리

138

- AspectJ 아노테이션 이용할 경우
 - ▣ XxxAspect 클래스를 작성한다
 - Pointcut 메서드 정의
 - Pointcut을 설정하기 위한 메서드이다.
 - Advice 메서드 정의
 - @Around
 - 첫 번째 인자로 ProceedingJoinPoint가 선언되어야 한다.
 - @Before
 - @AfterReturning
 - @AfterThrowing
 - ▣ 설정파일을 작성한다.
 - <aop:aspectj-autoproxy />
 - 타겟 빈과 Aspect 빈 등록
 - <context:annotation-config/>
 - <context:component-scan base-package="com.myapp.aop"/>
 - ▣ Main에서 테스트

5. Database 연동

1. 스프링의 데이터베이스 연동 지원
2. JDBC Template 클래스 이용
3. JDBC DaoSupport 클래스 이용
4. 스프링의 iBatis Template 클래스 이용
5. 스프링의 iBatis Dao Support 클래스 이용
6. 스프링의 MyBatis Template 클래스 이용
7. 스프링의 MyBatis Dao Support 클래스 이용
8. 스프링과 MyBatis Mapper Framework
9. Spring과 MyBatis로 Emp 테이블 다루기
10. Hibernate와 MyBatis

5.1 스프링의 데이터베이스 연동 지원

140

- 스프링은 JDBC를 비롯하여 ORM 프레임워크를 직접적으로 지원하고 있기 때문에 간단하게 JDBC뿐만 아니라 ORM 프레임워크를 스프링과 연동할 수 있다.
- 스프링은 JDBC, ORM 프레임워크 등의 다양한 기술을 이용해서 손쉽게 DAO클래스를 구현할 수 있도록 지원한다.
- 여러분은 실제 프로젝트 진행 전에 DAO를 어떤 방법으로 구현해야 할 것인가를 결정해야 합니다.
 - ▣ Java SE의 JDBC를 이용
 - ▣ 스프링의 JdbcTemplate 또는 JdbcDaoSupport 클래스를 이용(설명됨)
 - ▣ MyBatis의 Template 또는 DaoSupport 클래스를 이용(설명됨)
 - ▣ Hibernate의 Template 또는 DaoSupport 클래스를 이용
 - ▣ MyBatis SQL Mapper 이용(설명됨)
 - ▣ Hibernate ORM 이용

5.1.1 Template 클래스와 DaoSupport 클래스 지원

141

- 템플릿 클래스 지원
 - ▣ 개발자가 중복된 코드를 입력해야 하는 성가신 작업을 줄일 수 있도록 한다.
 - ▣ JDBC : JdbcTemplate
 - ▣ iBatis : SqlMapClientTemplate
 - ▣ MyBatis : SqlSessionTemplate
 - ▣ Hibernate : HibernateTemplate
- DaoSupport 클래스 지원
 - ▣ DAO에서 기본적으로 필요로 하는 기능을 제공한다.
 - ▣ DaoSupport 클래스를 상속받아 DAO클래스를 구현한 뒤, DaoSupport 클래스가 제공하는 기능을 사용하여 보다 편리하게 코드를 작성할 수 있게 된다.
 - ▣ JDBC : JdbcDaoSupport
 - ▣ iBatis : SqlMapClientDaoSupport
 - ▣ MyBatis : SqlSessionDaoSupport
 - ▣ Hibernate : HibernateDaoSupport

5.1.2 예외 클래스 지원

142

- 의미 있는 예외 클래스 제공한다.
- 스프링은 데이터베이스 처리 과정에서 발생한 예외가 왜 발생했는지를 좀 더 구체적으로 확인 할 수 있도록 하기 위해, **데이터베이스 처리와 관련된 예외 클래스를 제공하고 있다.**
- 데이터베이스 처리 과정에서 SQLException이 발생하면 **스프링이 제공하는 예외 클래스 중 알맞은 예외 클래스로 변환해서 예외를 발생 시킨다.**
- **스프링의 모든 예외 클래스들은 DataAccessException을 상속 받는다.**
- **DataAccessException은 RuntimeException의 하위클래스이다.**
 - 필요한 경우에만 try~catch
- 주요 예외 클래스
 - DuplicateKeyException
 - DataRetrievalFailureException
 - PermissionDeniedDataAccessException
 - BadSqlGrammarException
 - TypeMismatchDataAccessException

5.1.3 DataSource 설정

143

- 스프링은 DataSource를 통해서 Connection을 제공한다.
- 따라서, DataSource 정보를 설정해야 한다.
- 스프링은 다음과 같은 3가지 설정 방식을 제공한다.
 1. 커넥션 풀을 이용한 DataSource 설정
 2. JNDI를 이용한 DataSource 설정
 3. DriverManager를 이용한 DataSource 설정

`org.apache.commons.dbcp.BasicDataSource`

`org.springframework.jdbc.datasource.DriverManagerDataSource`

5.1.4 DataSource설정 – 커넥션 풀 이용

applicationContext.xml

```
<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="dataSource"
          class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:orcl" />
        <property name="username" value="scott"/>
        <property name="password" value="tiger"/>
    </bean>
    ...
</beans>
```

Commons DBCP 커넥션풀 데이터소스 사용하려면...
org.apache.commons.dbcp.BasicDataSource

oracle.jdbc.driver.OracleDriver
jdbc:oracle:thin:@127.0.0.1:1521:orcl

5.2 Template 클래스

145

- 데이터 베이스 연동을 위한 Connection에 관련된 코드 및 예외 처리 등 반복적인 코드를 제거 할 수 있도록 해준다.
- 스프링은 3개의 Template 클래스를 제공한다.
 1. JdbcTemplate
 2. NamedParameterJdbcTemplate
 3. SimpleJdbcTemplate

5.2.1 JdbcTemplate API - select : multi row

- List query(String sql, Object[] args, RowMapper rowMapper)
 - ▣ PreparedStatement를 이용해서 select 수행할 경우
- List query(String sql, RowMapper rowMapper)
 - ▣ 정적 SQL을 이용해서 select 수행할 경우

RowMapper 인터페이스

- ResultSet에서 값을 가져와 원하는 타입으로 매팅할 때 사용된다.

```
public interface RowMapper{  
    Object mapRow(ResultSet rs, int rowNum) throws SQLException;  
}
```

5.2.2 JdbcTemplate API - select : single row

147

- Object queryForObject(String sql, Object[] args, RowMapper rowMapper)
- Object queryForObject (String sql, RowMapper rowMapper)
- int queryForInt(String sql, Object[] args)
- int queryForInt(String sql)
- ...

```
jdbcTemplate.queryForObject(  
    sql,  
    new Object[] {emp.getEname(), emp.getSal()},  
    empMapper);
```

5.2.3 JdbcTemplate API - insert, update, delete

148

- `public int update(String sql, Object[] args)`
- `public int update(String sql)`

5.2.4 JdbcTemplate 클래스 이용

- JdbcTemplate 클래스는 DataSource를 필요로 한다.
- JdbcTemplate 클래스를 빈으로 설정하고 미리 설정한 DataSource 빈을 dataSource 프로퍼티로 전달한다.

applicationContext.xml

```
<bean id="dataSource"  
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>  
    <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:orcl" />  
    <property name="username" value="scott"/>  
    <property name="password" value="tiger"/>  
</bean>  
  
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">  
  <property name="dataSource" ref="dataSource"/>  
</bean>  
...
```

5.2.5 JdbcTemplate을 사용하는 DAO 클래스 작성

150

- DAO작성시 JdbcTemplate 클래스를 주입 받을 수 있도록 프로퍼티와 setter 메서드 또는 생성자를 작성한다.
- JdbcTemplate클래스가 제공하는 메서드를 이용해서 DAO를 작성한다.
- select 쿼리 결과 집합을 매팅할 커스텀 RowMapper를 작성한다.
- DAO 클래스를 빈으로 설정하고 미리 설정한 JdbcTemplate 빈을 전달 받도록 설정한다

5.2.5 JdbcTemplate을 사용하는 DAO 클래스 작성

151

UserDAOImpl1.java

```
public class UserDAOImpl1 implements UserDAO {  
  
    private JdbcTemplate jdbcTemplate;  
  
    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {  
        this.jdbcTemplate = jdbcTemplate;  
    }  
    public List<User> findUserList() {  
        String sql = "SELECT USERID,PASSWORD,NAME,EMAIL FROM USERINFO";  
        UserRowMapper rowMapper = new UserRowMapper();  
        List<User> userList = jdbcTemplate.query(sql, rowMapper);  
        return userList;  
    }  
}
```

5.2.6 RowMapper

UserRowMapper.java

```
public class UserRowMapper implements RowMapper<User> {  
  
    public User mapRow(ResultSet rs, int rowNum) throws SQLException {  
        User user = new User();  
        user.setPassword(rs.getString("PASSWORD"));  
        user.setName(rs.getString("NAME"));  
        user.setEmail(rs.getString("EMAIL"));  
        user.setUserId(rs.getString("USERID"));  
        return user;  
    }  
}
```

5.2.7 설정파일에 JdbcTemplate 클래스 등록

153

applicationContext.xml

```
...
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:orcl" />
    <property name="username" value="scott"/>
    <property name="password" value="tiger"/>
</bean>

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
</bean>
...

<bean id="userDao" class="myspring.jdbc.UserDAOImpl1"
      p:jdbcTemplate-ref="jdbcTemplate" />
...
```

5.3 DaoSupport 클래스 이용

- DaoSupport 클래스는 각각의 템플릿 클래스 별로 존재한다.
- 각 DaoSupport 클래스는 템플릿 객체를 구할 수 있는 메서드를 제공한다.
 1. JdbcDaoSupport
 - JdbcTemplate 지원
 2. NamedParameterJdbcDaoSupport
 - NamedParameterJdbcTemplate 지원
 3. SimpleJdbcDaoSupport
 - SimpleJdbcTemplate 지원

5.3.1 JdbcDaoSupport 클래스 이용

- DAO작성시 JdbcDaoSupport 클래스를 상속받는다.
 - JdbcDaoSupport 클래스가 제공하는 getJdbcTemplate() 메서드를 이용해서 JdbcTemplate 객체를 얻어내어 DAO를 작성한다.
 - 따라서, JdbcTemplate객체를 주입 받을 생성자나 setter메서드가 필요 없다.
 - select 쿼리 결과 집합을 맵핑 할 커스텀 RowMapper를 작성한다.
- DAO 클래스를 빈으로 설정하고 미리 설정한 DataSource 빈을 전달 받도록 설정한다.

5.3.1 JdbcDaoSupport 클래스를 상속받은 클래스

156

UserDAOImpl2.java

```
public class UserDAOImpl2 extends JdbcDaoSupport implements UserDAO {  
    public List<User> findUserList() {  
        String sql = "SELECT userid, password, name, email FROM userinfo";  
        UserRowMapper rowMapper = new UserRowMapper();  
        List<User> userList = getJdbcTemplate().query(sql, rowMapper);  
        return userList;  
    }  
}
```

5.3.2 설정파일에 빈 객체 등록

157

applicationContext.xml

```
...
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource"
      p:driverClassName="com.mysql.jdbc.Driver"
      p:url="jdbc:mysql://127.0.0.1:3306/struts"
      p:username="scott"
      p:password="tiger" />

<bean id="userDao2" class="myspring.jdbc.UserDAOImpl2"
      p:dataSource-ref="dataSource"/>
...
```

5.3.3 Demo

158

- EmpSpringJdbcPrj 프로젝트 파일을 import 해 보세요.
- 데이터베이스를 설치해야 합니다.
- Oracle Express Edition을 설치하면 빠릅니다.
 - ▣ 오라클 설치 시 system 계정 비밀번호를 입력해야 합니다.
- 설치가 완료된 다음 시작 -> 프로그램 -> Oracle Database 11g Express Edition -> Run SQL Command Line 을 선택해서 SQL 콘솔을 엽니다.
- conn /as sysdba 로 접속한 다음 hr 계정 락을 풀어주고 비번을 설정합니다.
 - ▣ conn /as sysdba
 - ▣ alter user hr account unlock;
 - ▣ alter user hr identified by hr;
- hr 계정으로 접속해서 준비해둔 sql 파일의 내용을 붙여 넣습니다.
 - ▣ conn hr/hr
- 프로젝트 빌드패스에 jdbc driver를 추가합니다.
- applicationContext.xml 파일의 url, username, password의 값을 수정합니다.
- 실행은 a.bc.emp.EmpWin을 실행합니다.

5.4 스프링의 iBatis지원

159

- iBatis는 데이터베이스 테이블과 자바 객체 사이의 단순한 매핑을 비교적(?) 간단한 설정을 통해 처리할 수 있기 때문에 널리 사용되는 프레임워크 중 하나이다.
- 스프링은 SqlMapClient를 사용할 때 발생하는 코드 중복을 없애고 SQLException을 스프링이 제고아는 예외로 변환해 주는 SqlMapClientTemplate 클래스를 제공하고 있다.
- MyBatis(iBatis)는 SQL 매퍼 프레임워크, ORM이 아닙니다.

5.4.1 SqlMapClientFactoryBean을 이용한 SqlMapClient 설정

applicationContext.xml

```
...
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource"
      p:driverClassName="com.mysql.jdbc.Driver"
      p:url="jdbc:mysql://127.0.0.1:3306/struts"
      p:username="scott"
      p:password="tiger" />

<bean id="sqlMapClient"
      class="org.springframework.orm.ibatis.SqlMapClientFactoryBean"
      p:dataSource-ref="dataSource"
      p:configLocation="SqlMapConfig.xml" />
...
```

5.4.2 SqlMapConfig.xml를 이용한 쿼리문 작성

161

SqlMapConfig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMapConfig
PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>

    <sqlMap resource="movie/Director.xml"/>

```

```
</sqlMapConfig>
```

SQL문이 매핑되어 있는 파일

iBatis 설정 파일 및 매핑 파일은 <http://ibatis.apache.org> 사이트를 참고

5.4.3 SqlMapClientTemplate 클래스를 이용한 iBatis 연동

162

- SqlMapClient를 사용할 때에 관련된 반복적인 코드를 제거 할 수 있도록 해준다.
- SqlMapClientTemplate 클래스를 제공한다.
- SqlMapClientTemplate 클래스는 내부적으로 iBatis의 SqlMapClient를 사용한다.
- SqlMapClientTemplate 클래스를 빈으로 설정하고 이미 설정한 SqlMapClient 빈을 설정한다.

5.4.4 SqlMapClientTemplate 등록

163

applicationContext.xml

```
...
<bean id="sqlMapClient"
      class="org.springframework.orm.ibatis.SqlMapClientFactoryBean"
      p:dataSource-ref="dataSource"
      p:configLocation="SqlMapConfig.xml" />

<bean id="sqlMapClientTemplate"
      class="org.springframework.orm.ibatis.SqlMapClientTemplate"
      p:sqlMapClient-ref="sqlMapClient" />
...

```

5.4.5 Dao 클래스에서 SqlMapClientTemplate 클래스 사용

164

- DAO 작성
 - ▣ SqlMapClientTemplate 클래스를 주입 받을 수 있도록 프로퍼티와 setter메서드를 작성한다.
- DAO 메서드 작성
 - ▣ SqlMapClientTemplate 클래스가 제공하는 메서드를 이용해서 DAO 메서드를 작성한다.
- DAO 클래스를 빈으로 설정
 - ▣ 미리 설정한 SqlMapClientTemplate 빈을 전달 받도록 설정한다.

5.4.5.1 DAO 클래스 작성

165

DirectorDAOImpl.java

```
public class DirectorDAOImpl implements DirectorDAO {  
  
    private SqlMapClientTemplate sqlMapClientTemplate;  
  
    public void setSqlMapClientTemplate(SqlMapClientTemplate  
        sqlMapClientTemplate){  
        this.sqlMapClientTemplate = sqlMapClientTemplate;  
    }  
  
    public List<Director> selectAllDirector() {  
        return sqlMapClientTemplate.queryForList("selectAllDirector");  
    }  
}
```

5.4.5.2 설정파일에 DAO 등록

166

applicationContext.xml

```
...
<bean id="sqlMapClientTemplate"
      class="org.springframework.orm.ibatis.SqlMapClientTemplate"
      p:sqlMapClient-ref="sqlMapClient" />

<bean id="directorDao"
      class="myspring.ibatis.DirectorDAOImpl"
      p:sqlMapClientTemplate-ref="sqlMapClientTemplate" />
...

```

5.4.5.3 JDBC Template와 iBatis Template 비교

| | JDBC | iBatis | 비고 |
|--------------|-------------------------------|--------------------------------------|-----------------------|
| 클래스 이름 | JdbcTemplate | SqlMapClientTemplate | |
| DAO | DAO에서 jdbcTemplate을 이용해 쿼리 전송 | 쿼리에 대한 정의는 xml 문서에 정의 | |
| 매퍼 | 클래스로 작성 | Xml 문서에 정의 | |
| dataSource | 드라이버, URL, ID, PW 정의 | 드라이버, URL, ID, PW 정의 | |
| SqlMapClient | - | dataSource 설정 SqlMapConfig.xml 설정 | Config 파일은 매퍼 xml을 정의 |
| Template | dataSource 설정 | SqlMapClient 설정 | |
| DAO | jdbcTemplate 설정 | sqlMapClientTemplate 설정 | |

5.5.1 SqlMapClientDaoSupport 클래스 이용

168

- SqlMapClientTemplate 클래스를 DAO클래스에서 좀 더 쉽게 사용할 수 있도록 SqlMapClientDaoSupport 클래스를 제공한다.
 - ▣ SqlMapClientDaoSupport 클래스가 제공하는 getSqlMapClientTemplate() 메서드를 이용해서 SqlMapClientTemplate 객체를 얻어내어 DAO를 작성한다.
 - ▣ 따라서, SqlMapClientTemplate객체를 주입받을 생성자나 setter메서드가 필요 없다.
- DAO 클래스를 빙으로 설정하고 미리 설정한 SqlMapClient 빙을 전달 받도록 설정한다.

5.5.2 SqlMapClientDaoSupport 클래스를 상속받아 DAO 작성

DirectorDAOImpl.java

```
public class DirectorDAOImpl extends SqlMapClientDaoSupport  
    implements DirectorDAO {  
  
    public List<Director> selectAllDirector() {  
        return getSqlMapClientTemplate().queryForList("selectAllDirector");  
    }  
  
}
```

SqlMapClientTemplate 을 위한 프로퍼티나 setter 메서드가 없어도 된다.

5.5.3 설정파일에 DAO 클래스 등록

170

applicationContext.xml

```
...
<bean id="sqlMapClient"
      class="org.springframework.orm.ibatis.SqlMapClientFactoryBean"
      p:dataSource-ref="dataSource"
      p:configLocation="SqlMapConfig.xml" />

<bean id="directorDao2"
      class="myspring.ibatis.DirectorDAOImpl"
      p:sqlMapClient-ref="sqlMapClient" />
...
```

5.6.1 SqlSessionTemplate

- MyBatis SQL 메서드를 호출하는데 사용
- thread safe하며, 여러 DAO들에서 사용 가능하다.

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

- DAO 클래스에 SqlSession 타입의 프로퍼티와 setter 메서드를 정의해야 한다.

Mapper XML 문서가 아닌 Daolmpl
클래스를 만들 경우에 사용합니다.

5.7.1 SqlSessionDaoSupport

172

```
public class EmpDaolmpl extends SqlSessionDaoSupport implements EmpDao {  
  
    @Override  
    public EmpVo selectEmp(Integer empno) {  
        // EmpVo emp = getSqlSession().selectOne("com.myapp.dao.EmpDao", empno);  
  
        EmpDao dao = this.getSqlSession().getMapper(EmpDao.class);  
        EmpVo emp = dao.selectEmp(empno);  
        return emp;  
    }  
}
```

Mapper XML 문서가 아닌 Daolmpl 클래스를 만들 경우에 사용합니다.

```
<bean id="empDao" class="com.myapp.dao.EmpDaolmpl">  
    <property name="sqlSessionFactory" ref="sqlSessionFactory"></property>  
</bean>
```

5.7.2 MapperFactoryBean

- Mybatis-Spring은 MapperFactoryBean 프락시 펙토리를 제공한다.

```
<bean id="empDao" class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface" value="com.myapp.dao.EmpDao" />
    <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

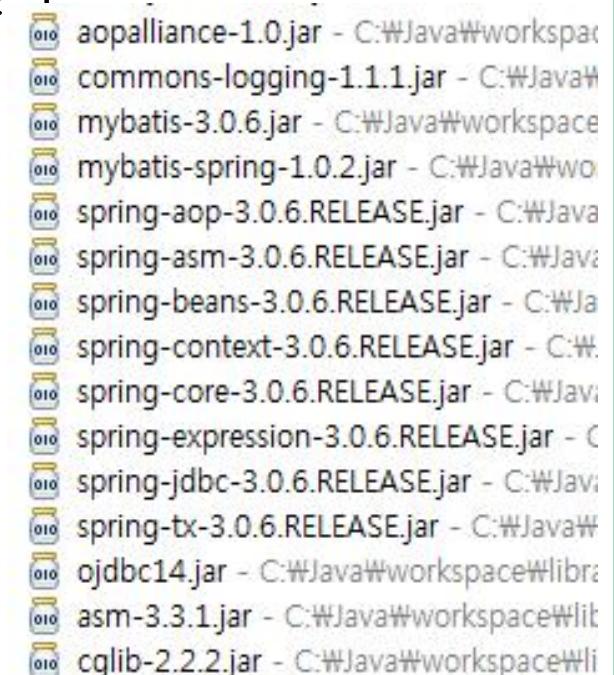
Impl 클래스를 빈으로 등록하지 않고 프락시 펙토리를 통해 빈 객체를 등록할 수 있다.

5.8 MyBatis framework 설치

174

- iBatis 팀이 구글의 MyBatis로 이름을 바꿔 옮겨갔다.
- 스프링에서 MyBatis를 사용하기 위해서 <http://www.mybatis.org>에서
 - ▣ MyBatis Core Framework 과 MyBatis Spring integration 모듈을 다운로드 받아야 한다.
 - ▣ <http://code.google.com/p/mybatis/downloads/list?can=3&q=Product%3DSpring>
 - ▣ 다운로드 받은 프로그램 중에서 다음 라이브러리를 추가한다
 - mybatis-3.2.1.jar
 - mybatis-spring-1.2.1.jar
 - cglib.jar와 asm.jar도 추가한다.

<http://code.google.com/p/mybatis/>



5.8.1 설정파일(데이터베이스 등 기본 설정)

- 설정파일들을 별도의 폴더에 관리
 - ▣ com/myapp/config/applicationContext.xml
 - org.springframework.beans.factory.config.PropertyPlaceholderConfigurer
 - dataSource 정보를 별도의 파일로 관리
 - org.springframework.jdbc.datasource.DriverManagerDataSource
 - dataSource 설정, \${} 이용
 - 컴포넌트 자동 스캔
 - <context:component-scan base-package="com.myapp.service" />
 - 아노테이션 사용
 - <context:annotation-config />
 - 아노테이션 기반 트랜잭션 사용
 - <tx:annotation-driven />
 - ▣ com/myapp/config/jdbc.properties
 - 데이터베이스 정보(driver, url, username, password)
- UI 클래스에서 설정파일 사용
 - ▣ 서비스 클래스를 멤버변수로 선언해 놓고
 - ▣ 생성자 또는 초기화 메서드에서
 - ApplicationContext ctx = new ClassPathXmlApplicationContext("com/myapp/config/applicationContext.xml");
 - service = (EmpService)ctx.getBean("empService");

5.8.1 설정파일(com/myapp/config/applicationContext.xml)

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

    <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="location">
            <value>classpath:com/myapp/config/jdbc.properties</value>
        </property>
    </bean>
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="${driver}" />
        <property name="url" value="${url}" />
        <property name="username" value="${username}" />
        <property name="password" value="${password}" />
    </bean>

    <!-- enable component scanning (beware that this does not enable mapper scanning!) -->
    <context:component-scan base-package="com.myapp.service" />                                com/myapp/config/jdbc.properties

    <!-- enable autowire -->
    <context:annotation-config />

    <!-- enable transaction demarcation with annotations -->
    <tx:annotation-driven />
</bean>
```

driver=oracle.jdbc.driver.OracleDriver
 url=jdbc:oracle:thin:@127.0.0.1:1521:ora11g
 username=scott
 password=tiger

5.8.2 Mapper xml & Interface

177

- com/myapp/dao/EmpMapper.xml

```
<?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.myapp.dao.EmpDao">
```

```
<select id="selectEmp" parameterType="int" resultType="com.myapp.domain.EmpVo">
    select * from emp where empno = #{empno}
</select>
```

```
</mapper>
```

- com/myapp/dao/EmpDao.java

```
public interface EmpDao {
    EmpVo selectEmp(@Param("empno") int empno);
}
```

iBatis와 MyBatis는
DOCTYPE 선언부가
다릅니다.

#{}안의 문자는 대/소
문자 구분합니다.

5.8.2.1 mapper xml과 interface의 리턴타입

| java | xml |
|--------------------------------|--------------------|
| 기본 타입(int, double) | 기본 타입(int, double) |
| String | string |
| Wrapper Class(Integer, Double) | 기본 타입(int, double) |
| String[] | string |
| List<DeptVo> | DeptVo |
| List<Integer> | int |

Supported JDBC Types

| | | | | | |
|----------|---------|-------------|---------------|---------|-----------|
| BIT | FLOAT | CHAR | TIMESTAMP | OTHER | UNDEFINED |
| TINYINT | REAL | VARCHAR | BINARY | BLOG | NVARCHAR |
| SMALLINT | DOUBLE | LONGVARCHAR | VARBINARY | CLOB | NCHAR |
| INTEGER | NUMERIC | DATE | LONGVARBINARY | BOOLEAN | NCLOB |
| BIGINT | DECIMAL | TIME | NULL | CURSOR | ARRAY |

5.8.2.2 파라미터 타입이 여러 개 일 경우

- Dao 인터페이스에...
 - ▣ Emp selectXxx(@Param("mgr") int mgr, @Param("job") String job);
- Mapper 클래스에
 - ▣ <select id="selectXxx" parameterType="**map**" resultType="Emp">
 - ▣ select * from emp where mgr=#{mgr} and job=\${job}
 - ▣ </select>

5.8.2.3 에러...

180

- "Mapped Statements collection does not contain value for" 에러
 - ▣ Mapper namespace + id = Dao + method 이어야 한다.
 - ▣ 설정파일로 namespace 와 dao 를 연결한 후 id 와 dao의 method 가 일치해야 한다. 즉 MyBatis 의 경우에는 Dao 선언한 메서드와 Mapper 의 id 와 일치해야 하는데 일치하지 않는다는 것이다

5.8.3 SqlSessionFactoryBean

- MyBatis에서 SqlSessionFactory 는 SqlSessionFactoryBuilder를 이용해 생성하지만 MyBatis-Spring에서는 SqlSessionFactoryBean 이 사용된다.
- SqlSessionFactoryBean는 스프링의 FactoryBean 인터페이스를 구현하고 있다.
- SqlSessionFactoryBean 클래스의 getObject() 메서드를 통해 SqlSessionFactory 인스턴스를 리턴받을 수 있다.
- SqlSessionFactory는 mapperLocations 프로퍼티를 통해 Sql Mapper 파일을 지정할 수 있다.

5.8.4 설정파일에 SqlSessionFactoryBean 정의

```
<!-- define the SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
<property name="dataSource" ref="dataSource" />
<property name="mapperLocations" value="classpath:com/myapp/dao/**/*.xml" />
<property name="typeAliasesPackage" value="com.myapp.domain" />
</bean>

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

- mapperLocation
 - ▣ Mapper 파일의 위치를 지정함.
 - ▣ 예에서는 dao폴더의 하위 폴더까지 검색함
- typeAliasesPackage
 - ▣ Mapper에서 타입을 지정할 때 도메인 클래스의 패키지 명을 생략할 수 있도록 함
 - ▣ Mapper에서 resultType="com.myapp.domain.Emp"를 resultType="EmpVo"로 사용할 수 있도록 함

5.8.5 DataSourceTransactionManager

183

- Standard Configuration
 - ▣ <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
 - ▣ <property name="dataSource" ref="dataSource" />
 - ▣ </bean>
- Container Managed Transactions
 - ▣ <tx:jta-transaction-manager />
- Programmatic Transaction Management
 - ▣ DefaultTransactionDefinition def = new DefaultTransactionDefinition();
 - ▣ def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);
 - ▣ TransactionStatus status = txManager.getTransaction(def);
 - ▣ try {
 - ▣ userMapper.insertUser(user);
 - ▣ }
 - ▣ catch (MyException ex) {
 - ▣ txManager.rollback(status);
 - ▣ throw ex;
 - ▣ }
 - ▣ txManager.commit(status);

자바코드 내에서
@Transactional 으로 트랜잭션 처리 가능

5.8.6 설정파일에 transactionManager 정의

184

```
<!-- standard transaction configuration -->
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

- 아래와 같이 서비스 클래스에서 **@Transactional** 어노테이션을 사용할 수 있다.

5.8.8 환경설정 파일에서 Mapper 자동 등록

185

```
<!-- scan for mappers and let them be autowired -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.myapp.dao" />
</bean>
```

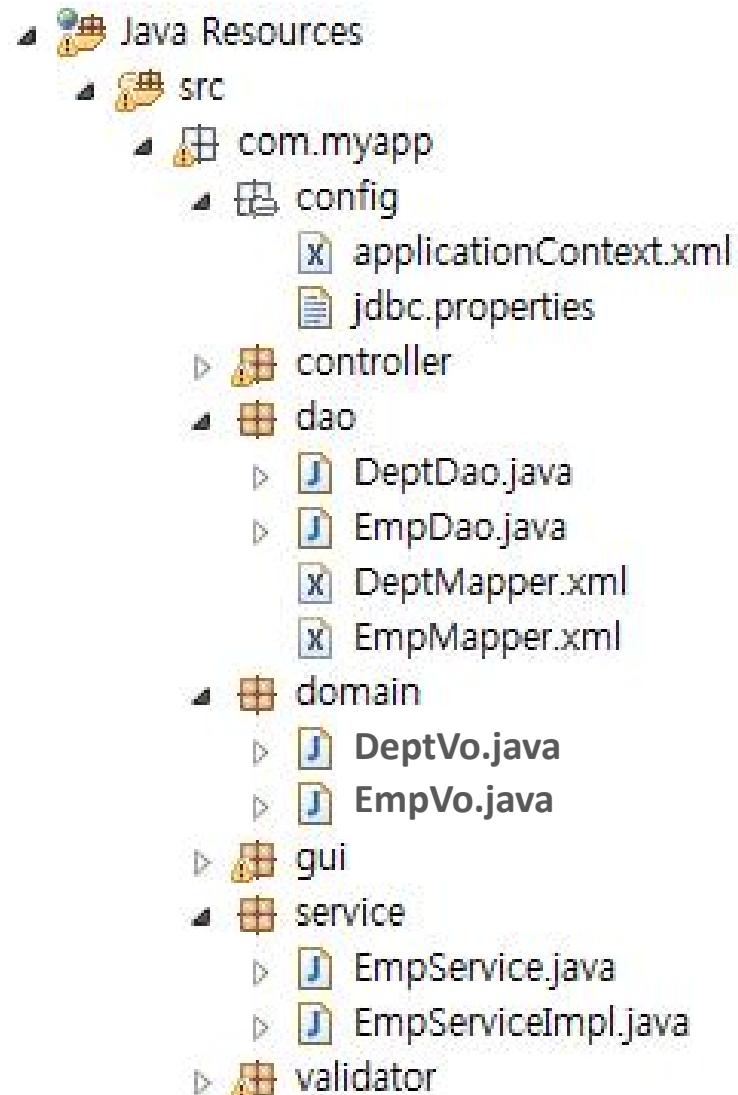
↑
Mapper 가 있는 패키지

5.8.7 테스트

186

```
ApplicationContext ctx =  
    new ClassPathXmlApplicationContext("com/myapp/config/applicationContext.xml");  
EmpService service = (EmpService)ctx.getBean("empService");  
System.out.println(service);  
System.out.println(service.selectEmp(7902));
```

- 제공된 GUI와 일부 구현된 클래스를 프로젝트를 생성한 다음 import 하세요
- 프로젝트
 - ▣ 이름 : EmpSpringMyBatisPrj
 - ▣ 프로젝트 종류 : Spring Project
- 라이브러리
 - ▣ MyBatis 라이브러리
 - ▣ MyBatis-Spring 라이브러리
 - ▣ JDBC Driver(ojdbc5.jar)
 - ▣ cglib-nodep-2.2.3.jar
 - ▣ asm-4.0.jar
 - ▣ aopalliance-1.0.jar
 - ▣ commons-logging-1.1.1.jar



```

<?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.myapp.dao.EmpDao">

<select id="selectEmp" parameterType="int" resultType="com.myapp.domain.EmpVo">
    select * from emp where empno = #{empno}
</select>

<select id="getAllEmpno" resultType="int">
    select empno from emp
</select>

<insert id="insertEmp" parameterType="EmpVo">
    insert into emp (empno, ename, job, mgr, hiredate, sal, comm, deptno)
    values (#{empno}, #{ename}, #{job}, #{mgr}, #{hiredate}, #{sal}, #{comm}, #{deptno})
</insert>

<select id="getAllEmps" resultType="EmpVo">
    select * from emp
</select>

<update id="updateEmp" parameterType="EmpVo">
    update emp set ename=#{ename}, job=#{job} where empno = #{empno}
</update>

<delete id="deleteEmp" parameterType="int">
    delete from emp where empno=#{empno}
</delete>
</mapper>

```

오라클이라면
SYSDATE 사용 가능

5.9.2 Emp 테이블 다루기(DeptMapper.xml)

189

```
<?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.myapp.dao.DeptDao">
  <select id="getAllDepts" resultType="DeptVo">
    select deptno, dname, loc from dept
  </select>
  <select id="getDept" parameterType="int" resultType="DeptVo">
    select deptno, dname, loc from dept where deptno = #{deptno}
  </select>
</mapper>
```

```

@Service("empService")
public class EmpServiceimpl implements
    EmpService {

    @Autowired
    private EmpDao empDao;

    @Autowired
    private DeptDao deptDao;

    public List<Dept> getAllDepts() {
        return deptDao.getAllDepts();
    }

    @Transactional
    public EmpVo selectEmp(int empno) {
        EmpVo emp = empDao.selectEmp(empno);
        return emp;
    }
}

```

```

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

```

```

public List<Integer> getAllEmpno() {
    return empDao.getAllEmpno();
}

public void insertEmp(EmpVo emp) {
    empDao.insertEmp(emp);
}

public List<Emp> getAllEmps() {
    return empDao.getAllEmps();
}

public void updateEmp(EmpVo emp) {
    empDao.updateEmp(emp);
}

public void deleteEmp(int empno) {
    empDao.deleteEmp(empno);
}

```

5.9.4 Emp 테이블 다루기(인터페이스)

191

EmpDao.java

```
package com.myapp.dao;

import java.util.List;

import org.apache.ibatis.annotations.Param;
import com.myapp.domain.Emp;

public interface EmpMapper {
    EmpVo selectEmp(@Param("empno") int empno);
    List<Integer> getAllEmpno();
    void insertEmp(Emp emp);
    List<Emp> getAllEmps();
    String[] getColumnNames();
    void updateEmp(Emp emp);
    void deleteEmp(int empno);
}
```

DeptDao.java

```
package com.myapp.dao;

import java.util.List;

import com.myapp.domain.Dept;

public interface DeptMapper {
    List<Dept> getAllDepts();
    DeptVo getDept(int deptno);
}
```

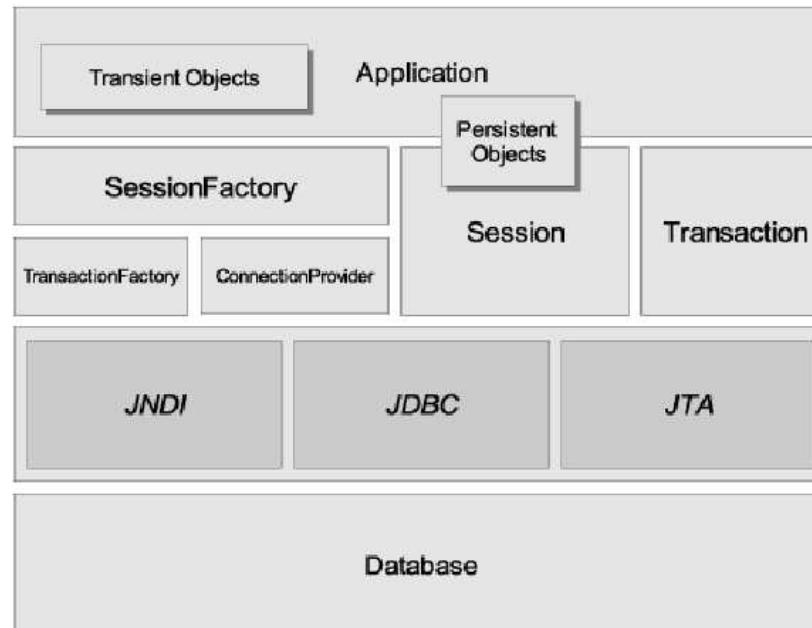
5.10 Hibernate와 MyBatis

- Hibernate는 자바 객체와 관계형 데이터 모델간의 매팅을 위한 도구이며 쿼리 서비스를 지원하는 강력한 고성능의 퍼시스턴스 프레임워크임
 - 관계형 데이터모델에 대한 객체지향 관점을 제공하는 객체/관계 매팅(Object Relational Mapping) 프레임워크
 - Gavin King (JBoss, 현재 Red Hat)을 중심으로한 소프트웨어 개발팀에 의해 개발됨.
- MyBatis는 단순성이라는 사상을 강조한 퍼시스턴스 프레임워크로, SQL 맵을 이용하여 반복적이고 복잡한 DB 작업코드를 최소화함
 - 단순성이라는 사상을 강조하여, XML을 이용하여 Stored Procedure 혹은 SQL문과 자바 객체간의 매팅을 지원
 - 2001년 Clinton Begin (Apache 소프트웨어재단)에 의해 개발된 퍼시스턴스 프레임워크



5.10.1 Hibernate

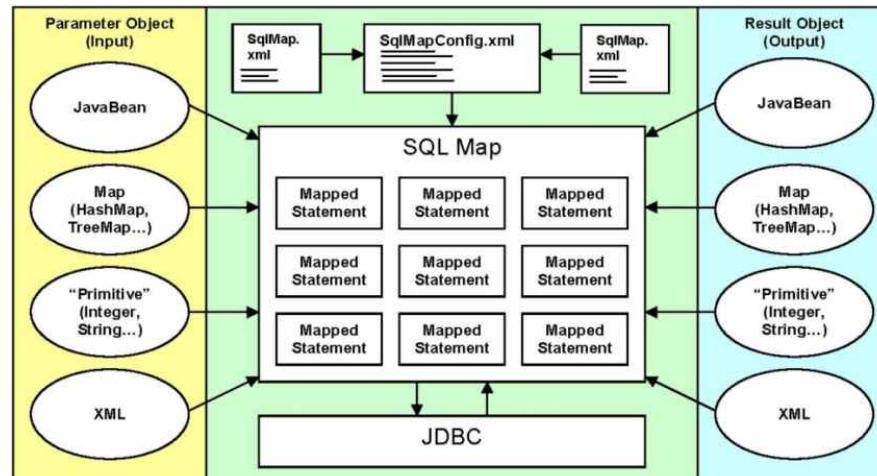
- Hibernate는 J2EE 표준인 JNDI, JDBC, JTA를 기반으로 객체관계형 매팅(ORM), 데이터베이스 연결 및 트랜잭션 관리 기능 등을 제공함



| 아키텍처 구성요소 | 설명 |
|---|--|
| SessionFactory | <ul style="list-style-type: none"> 단일 데이터베이스에 대한 캐시로서, Session에 대한 팩토리 기능을 제공한다. |
| Session | <ul style="list-style-type: none"> 어플리케이션과 영속 저장소 사이의 연결을 표현하는 객체로서, JDBC 커넥션을 Wrapping한다. Transaction에 대한 팩토리 기능을 제공한다. |
| Persistent Objects and Collections | <ul style="list-style-type: none"> Session과 연관되어 있는 영속(persistent) 상태의 객체로서, 일반적인 JavaBeans/POJO이다. Session이 닫히면, Session과 분리되어 Application 내에서 자유롭게 사용할 수 있게 된다. |
| Transient and Detached Objects and Collections | <ul style="list-style-type: none"> Session과 연관되어 있지 않은 영속 클래스들의 인스턴스로서. 어플리케이션에 의해 초기화된 후 영속화되지 않았거나, 닫혀진 Session에 의해 초기화되었을 수도 있다. |
| Transaction | <ul style="list-style-type: none"> 작업의 완전성을 보장하기 위한 어플리케이션에 의해 사용되는 객체이다. |
| ConnectionProvider | <ul style="list-style-type: none"> JDBC 커넥션들에 대한 팩토리 기능을 제공한다. |
| TransactionFactory | <ul style="list-style-type: none"> Transaction 인스턴스들에 대한 팩토리 기능을 제공한다. |

5.10.2 iBatis(MyBatis)

- MyBatis는 소스코드 외부에 정의된 SqlMap.xml 파일정보를 바탕으로 생성된 Mapped Statement를 이용하여 SQL과 객체간의 매핑기능을 제공함



* iBATIS SQL Maps 개발자 가이드 Version 2.0

| 아키텍처 구성요소 | 설명 |
|---------------------------------|--|
| Parameter Object (Input) | 파라미터 객체는 JavaBean, Map, Primitive 객체로서, update문 내에 입력 값을 설정하기 위해 사용되거나 쿼리문의 where절을 설정하기 위해서 사용된다. |
| SqlMapConfig.xml | Data Mapper에서 사용하는 설정을 담고 있는 파일로서, DataSource, Data Mapper 및 Thread Management 등과 같은 상세 설정 정보를 담고 있다. |
| SqlMap.xml | 하나의 SqlMap.xml 파일은 많은 Cache Models, Parameters Maps, Result Maps, Statements 정보를 담고 있다. |
| SQL Map | Data Mapper 프레임워크는 PreparedStatement 인스턴스를 생성하고 제공된 파라미터 객체를 사용해서 파라미터를 설정한다. 그리고 statement를 실행하고 ResultSet으로부터 결과 객체를 생성한다. |
| Mapped Statement | Mapped Statement는 Data Mapper 프레임워크의 핵심으로서, Parameter Maps과 Result Maps를 이용하여 SQL statement로 치환된다. |
| Result Object (Output) | 결과 객체는 JavaBean, Map, Primitive 객체로서, 쿼리문의 결과값을 담고 있다. |

5.10.3 Hibernate vs MyBatis

- **Hibernate: Object Relational Mapper**
 - ▣ Database 엔티티(일종의 테이블 row)와 자바 객체를 동기화 하는 역할을 담당
 - ▣ Hibernate는 이러한 역할을 하는 프레임워크
 - ▣ 모든 sql문은 프레임워크에서 생성되고 실행됨
 - ▣ sql작업이 필요할 경우 HSQL을 통하여 이루어짐(EJB-QL과 유사)
 - ▣ HSQL은 실제적인 sql의 앞단에서 처리되는 객체지향 쿼리 랭귀지
 - ▣ 종류: hibernate, TopLink, Cocobase, JDO 구현체
- **MyBatis: SQL mapper**
 - ▣ 자바객체를 실제 sql 문장에 맵핑.(자바 코드에서 sql 관련부분 제거)
 - ▣ Sql 문장은 자동 생성되는 것은 아니고 개발자가 기술해 줌
 - ▣ 맵핑 자체는 데이터베이스의 엔티티와 관계(relationship)에 독립적임.(mapping 자체가 sql문에 국한)
 - ▣ 실제로 모든 임베디드 sql 시스템은 모두 sql mapper로 간주가능
 - 예: MyBATIS SQL Maps, Oracle SQLJ, Forte 4GL Embedded SQL, Pro*C Embedded SQL
 - ▣ MyBatis sql map의 경우 xml에 임베디드된 sql (자바코드의 sql을 xml 파일로 분리)

5.10.4 Hibernate와 MyBatis의 비교 우위

- Hibernate와 MyBatis는 다른 특성을 갖는 프레임워크
- 일차원적인 비교는 불가능
- 상황에 따라 적용 프레임워크의 효율성이 달라짐

- Hibernate가 적절한 경우
 - ▣ 새로운 프로젝트가 시작된 상태
 - ▣ 객체 모델과 데이터베이스 디자인이 미완성인 상태

- MyBatis가 적절한 경우
 - ▣ 3rd party databases에 접근하는 경우
 - ▣ 레거시 데이터베이스와 연동이 필요한 경우
 - ▣ 디비 디자인이 부적절한 상태일 경우
 - ▣ O/R Mapper가 이러한 상황을 제어할 능력이 없을 수도 있음.
 - ▣ SQL Mapper를 사용할 경우 객체 모델과 데이터 모델사이의 매핑에는 아무런 제약 사항이 없음.
 - ▣ sql문을 인력을 사용하여 수작업으로 tuning이나 최적화를 해야 할 경우

5.10.5 Hibernate와 iBatis의 Performance 측면의 비교

197

- 과거 embeded sql mapper
 - ▣ 컴파일 랭귀지를 사용하여 제작됨
 - ▣ 매우 빠른속도를 제공하고 시스템 환경에 최적화 되어 있음
- O/R mapper
 - ▣ sql mapper에 비하여 다양한 일을 수행
 - ▣ 대부분 reflection 방식 (hibernate),
 - ▣ binary code enhancement 방식(JDO).
 - ▣ hibernate의 향후 버전에서는 binary code enhancement방식을 채용
 - ▣ reflection 방식을 사용한다는 측면은 iBatis와 공통점
- 프레임워크 성능비교는 무의미
 - ▣ 프레임워크 성능이란 프레임워크를 어떻게 사용하는 방식에 따라서 결정
 - ▣ 일반적으로 O/R mapper가 sql mapper에 비해서 훨씬더 효율적인 맵핑을 하고 수행전략을 수립.
 - ▣ O/R mapper는 객체 모델과 데이터베이스 모델에 대한 광범위한 정보를 포함있음
 - ▣ 간단한 CRUD 어플리케이션에 테이블-클래스 맵핑을 사용한다면 단순성과 성능이란 측면에서 O/R mapper 많은 장점을 갖고 있음
 - ▣ 복잡한 데이터 전송방식의 환경에서는 sql mapper가 효율적임
 - ▣ Sql mapper가 더 효율적인 sql의 장점들을 표출할 수 있음

5.10.5 그러면... Hibernate와 iBatis 중에서...

198

- 하나이상을 선택하여 테스트 해보라.
- 프로젝트에 대한 컨셉에 따라 세밀하게 테스트 해보라.
- 모든 프로젝트의 특정은 모두 다르며 상황에 따라 Hibernate, iBATIS SQL Maps, TopLink, raw JDBC를 유연하게 사용해야 함

- 이러한 이유에서 다양한 툴(프레임워크)을 빠르고 효과적으로 선택하고 테스트 하는 방법을 배우는 것이 더 중요하고 유용하다. 프레임워크 중 하나만을 사용할 줄 아는 것은 중요한 것이 아니다.
- 다양한 상황에서 연습을 해보고, 더 좋은 결정을 내려보시길 바랍니다. 성배를 찾는 것은 중요한 것이 아닙니다.

199

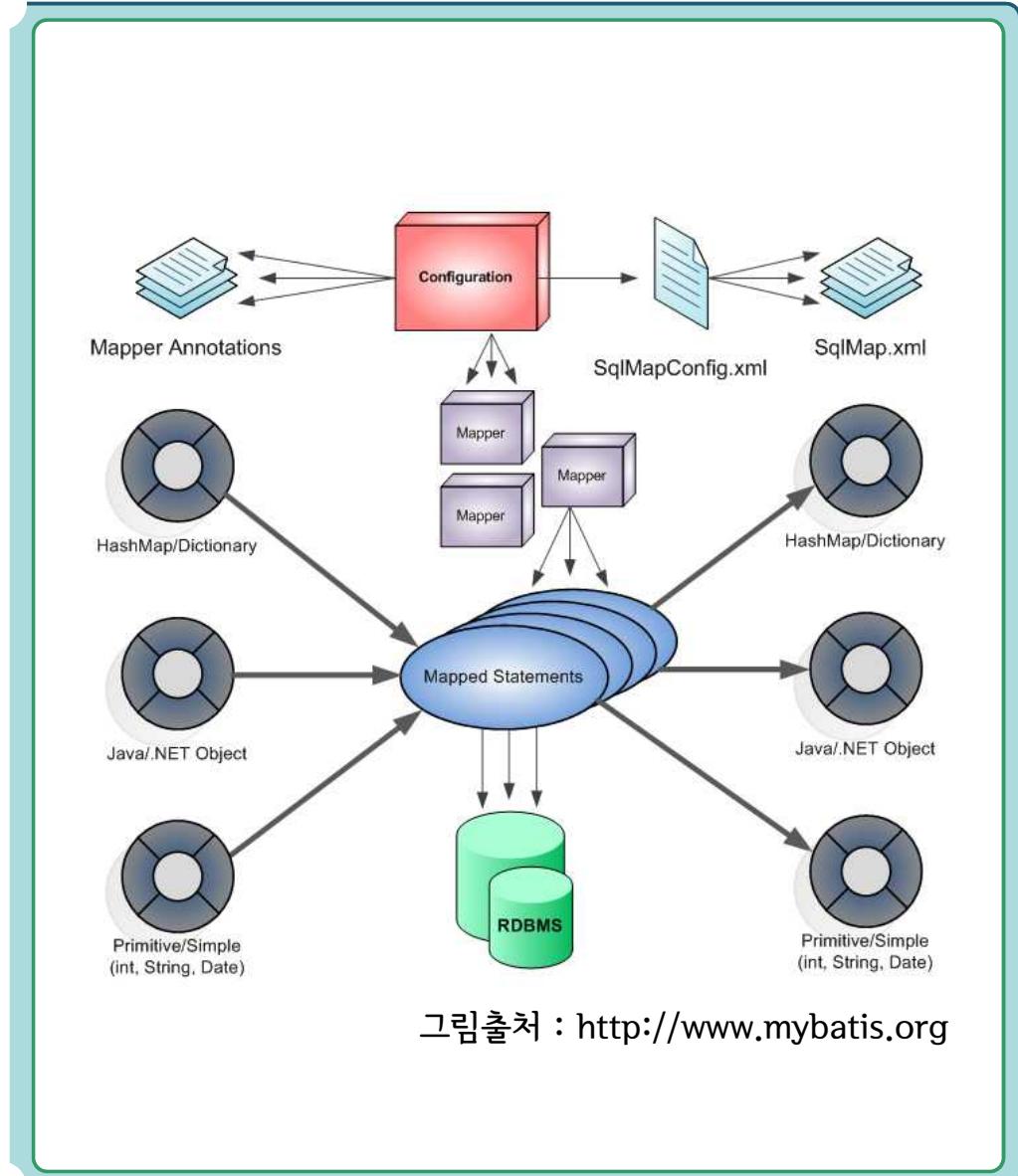
6. MyBatis 프레임워크

6.1 MyBatis

200

- MyBatis 는 개발자가 지정한 SQL, 저장 프로시저 그리고 몇 가지 고급 매핑을 지원하는 퍼시스턴스 프레임워크이다.
- MyBatis 는 JDBC 코드와 수동으로 셋팅 하는 파라미터와 결과 매핑을 제거한다.
- MyBatis 는 데이터베이스 레코드에 원시 타입과 Map 인터페이스 그리고 자바 POJO 를 설정하고 매핑하기 위해 XML 과 아노테이션을 사용할 수 있다.

- <http://code.google.com/p/mybatis/>에서 MyBatis Persistence Framework 다운로드
- mybatis-3.x.x.jar 파일을 빌드패스에 포함한다.
- 스프링에서 사용하려면 MyBatis-Spring integration module을 다운로드 받아야 한다.



6.1.1 SqlSessionFactory 빌드

201

- 모든 MyBatis 애플리케이션은 SqlSessionFactory 인스턴스를 사용한다. SqlSessionFactory 인스턴스는 SqlSessionFactoryBuilder 를 사용하여 만들 수 있다. SqlSessionFactoryBuilder 는 XML 설정파일에서 SqlSessionFactory인스턴스를 빌드할 수 있다.

```
String resource = "com/myapp/config/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

SqlSession session = sqlSessionFactory.openSession();
try {
    BlogMapper mapper = session.getMapper(BlogMapper.class);
    Blog blog = mapper.selectBlog(101);
} finally {
    session.close();
}
```

6.2 환경설정 파일(com/myapp/config/mybatis-config.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <properties resource="com/myapp/config/ibatis.properties">
    <typeAliases>
      <typeAlias type="com.myapp.model.Emp" alias="emp"/>
    </typeAliases>
    <environments default="development">
      <environment id="development">
        <dataSource type="POOLED">
          <property name="driver" value="${driver}"/>
          <property name="url" value="${url}"/>
          <property name="username" value="${username}"/>
          <property name="password" value="${password}"/>
        </dataSource>
      </environment>
    </environments>
    <mappers>
      <mapper resource="com/myapp/sqlmap/EmpMapper.xml"/>
    </mappers>
  </configuration>
```

com/myyapp/conf/ibatis.properties

```
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@127.0.0.1:1521:ora11g
username=scott
password=tiger
```

6.2.1 properties

203

- 이 설정은 외부에 옮길 수 있다. 자바 프로퍼티 파일 인스턴스에 설정할 수도 있고, properties 요소의 하위 요소에 둘 수도 있다.
- <properties resource="org/mybatis/example/config.properties">
 - <property name="username" value="scott"/>
 - <property name="password" value="tiger"/>
 - </properties>
- 속성들은 파일 도처에 둘 수도 있다.
- <dataSource type="POOLED">
 - <property name="driver" value="\${driver}"/>
 - <property name="url" value="\${url}"/>
 - <property name="username" value="\${username}"/>
 - <property name="password" value="\${password}"/>
- </dataSource>
- 가장 우선순위가 높은 속성은 메서드의 파라미터로 전달된 값이고 그 다음은 resource 및 url 속성이고 마지막은 properties 요소에 명시된 값이다

6.2.2 typeAliases

204

- 타입 별칭은 자바 타입에 대한 좀더 짧은 이름이다. 오직 XML 설정에서만 사용되며, 타이핑을 줄이기 위해 존재한다.
- <typeAliases>
- <typeAlias alias="Author" type="domain.blog.Author"/>
- <typeAlias alias="Blog" type="domain.blog.Blog"/>
- <typeAlias alias="Comment" type="domain.blog.Comment"/>
- <typeAlias alias="Post" type="domain.blog.Post"/>
- <typeAlias alias="Section" type="domain.blog.Section"/>
- <typeAlias alias="Tag" type="domain.blog.Tag"/>
- </typeAliases>

6.2.3 environments

205

- MyBatis 는 여러개의 환경으로 설정될 수 있다. 여러 가지 이유로 여러 개의 데이터베이스에 SQL Map 을 적용하는데 도움이 된다. 예를 들어, 개발, 테스트, 리얼 환경을 위해 별도의 설정을 가지거나, 같은 스키마를 여러 개의 DBMS 제품을 사용할 경우들이다. 그 외에도 많은 경우가 있을 수 있다.
- 중요하게 기억해야 할 것은, 다중 환경을 설정할 수는 있지만, SqlSessionFactory 인스턴스마다 한 개만 사용할 수 있다는 것이다.
 - 두 개의 데이터베이스에 연결하고 싶다면, SqlSessionFactory 인스턴스를 두 개 만들 필요가 있다. 세 개의 데이터베이스를 사용한다면, 역시 3 개의 인스턴스를 필요로 한다. 기억하기 쉽게
- <environments default="development">
- <environment id="development">
- <transactionManager type="JDBC">
- <property name="..." value="..."/>
- </transactionManager>
- <dataSource type="POOLED">
- <property name="driver" value="\${driver}"/>
- <property name="url" value="\${url}"/>
- <property name="username" value="\${username}"/>
- <property name="password" value="\${password}"/>
- </dataSource>
- </environment>
- </environments>

6.2.4 dataSource

- dataSource 요소는 표준 JDBC DataSource 인터페이스를 사용하여 JDBC Connection 객체의 소스를 설정한다.
- UNPOOLED - 이 구현체는 매번 요청에 대해 커넥션을 열고 닫는 간단한 DataSource이다. 조금 늦긴 하지만 성능을 크게 필요로 하지 않는 간단한 애플리케이션을 위해서는 괜찮은 선택이다. UNPOOLED DataSource는 5개의 프로퍼티만으로 설정된다.
 - ▣ driver - JDBC 드라이버의 패키지 경로를 포함한 드라이버 클래스명
 - ▣ url - 데이터베이스 인스턴스에 대한 JDBC URL.
 - ▣ username - 데이터베이스에 로그인 할 때 사용할 사용자명
 - ▣ password - 데이터베이스에 로그인 할 때 사용할 패스워드
 - ▣ defaultTransactionIsolationLevel - 커넥션에 대한 디폴트 트랜잭션 격리 레벨
- POOLED - DataSource에 풀링이 적용된 JDBC 커넥션을 위한 구현체이다. 이는 새로운 Connection 인스턴스를 생성하기 위해 매번 초기화하는 것을 피하게 해준다. 그래서 빠른 응답을 요구하는 웹 애플리케이션에서는 가장 흔히 사용되고 있다. UNPOOLED DataSource에 비해, 많은 프로퍼티를 설정할 수 있다.
 - ▣ poolMaximumActiveConnections - 주어진 시간에 존재할 수 있는 활성화된(사용중인) 커넥션의 수. 디폴트는 10이다.
 - ▣ poolMaximumIdleConnections - 주어진 시간에 존재할 수 있는 유휴 커넥션의 수
 - ▣ 강제로 리턴되기 전에 풀에서 “체크아웃” 될 수 있는 커넥션의 시간. 디폴트는 20000ms(20초)
 - ▣ poolTimeToWait - 풀이 로그 상태를 출력하고 비정상적으로 긴 경우 커넥션을 다시 얻을려고 시도하는 로우 레벨 셋팅. 디폴트는 20000ms(20초)
 - ▣ poolPingQuery - 커넥션이 작업하기 좋은 상태이고 요청을 받아서 처리할 준비가 되었는지 체크하기 위해 데이터베이스에 던지는 펑쿼리(Ping Query). 디폴트는 “펑 쿼리가 없음”이다. 이 설정은 대부분의 데이터베이스로 하여금 에러메시지를 보게 할 수도 있다.
 - ▣ poolPingEnabled - 펑쿼리를 사용할지 말지를 결정. 사용한다면, 오류가 없는(그리고 빠른) SQL을 사용하여 poolPingQuery 프로퍼티를 셋팅해야 한다. 디폴트는 false이다.
 - ▣ poolPingConnectionsNotUsedFor - poolPingQuery 가 얼마나 자주 사용될지 설정한다. 필요이상의 펑을 피하기 위해 데이터베이스의 타임아웃 값과 같을 수 있다. 디폴트는 0이다. 디폴트 값은 poolPingEnabled 가 true 일 경우에만, 모든 커넥션이 매번 펑을 던지는 값이다.
- JNDI - 이 DataSource 구현체는 컨테이너에 따라 설정이 변경되며, JNDI 컨텍스트를 참조한다. 이 DataSource는 오직 두개의 프로퍼티만을 요구한다.
 - ▣ initial_context - 이 프로퍼티는 InitialContext에서 컨텍스트를 찾기(예를 들어, initialContext.lookup(initial_context)) 위해 사용된다. 이 프로퍼티는 선택적인 값이다. 이 설정을 생략하면, data_source 프로퍼티가 InitialContext에서 직접 찾을 것이다.
 - ▣ data_source - DataSource 인스턴스의 참조를 찾을 수 있는 컨텍스트 경로이다. initial_context 륙업을 통해 리턴된 컨텍스트에서 찾을 것이다. initial_context 가 지원되지 않는다면, InitialContext에서 직접 찾을 것이다.

6.2.5 mappers

207

- 매핑된 SQL 구문을 정의하기 전에 설정을 어디에 둘지 결정해야 한다.
- 자바는 자동으로 리소스를 찾기 위한 좋은 방법을 제공하지 않는다. 그래서 가장 좋은 건 어디서 찾으라고 지정하는 것이다. 클래스패스에 상대적으로 리소스를 지정할 수도 있고, url 을 통해서 지정할 수도 있다.
- <!-- Using classpath relative resources -->
- <mappers>
- <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
- <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
- <mapper resource="org/mybatis/builder/PostMapper.xml"/>
- </mappers>
- <!-- Using url fully qualified paths -->
- <mappers>
- <mapper url="file:///var/mappers/AuthorMapper.xml"/>
- <mapper url="file:///var/mappers/BlogMapper.xml"/>
- <mapper url="file:///var/mappers/PostMapper.xml"/>
- </mappers>
- <!-- Using mapper interface classes -->
- <mappers>
- <mapper class="org.mybatis.builder.AuthorMapper"/>
- <mapper class="org.mybatis.builder.BlogMapper"/>
- <mapper class="org.mybatis.builder.PostMapper"/>
- </mappers>
- <!-- Register all interfaces in a package as mappers -->
- <mappers>
- <package name="org.mybatis.builder"/>
- </mappers>

6.3 Mapper XML

- MyBatis 의 가장 큰 장점은 매핑된 구문이다.
- MyBatis 는 SQL 을 작성하는데 집중하도록 만들어졌다.
- SQL Map XML 파일은 첫 번째(first class)요소만을 가진다.
 - ▣ cache - 해당 명명공간을 위한 캐시 설정
 - ▣ cache-ref - 다른 명명공간의 캐시 설정에 대한 참조
 - ▣ resultMap- 데이터베이스 결과데이터를 객체에 로드하는 방법을 정의하는 요소
 - ▣ parameterMap - 비권장됨! 예전에 파라미터를 매핑하기 위해 사용되었으나 현재는 사용하지 않음
 - ▣ sql - 다른 구문에서 재사용하기 위한 SQL 조각
 - ▣ insert- 매핑된 INSERT 구문.
 - ▣ update- 매핑된 UPDATE 구문.
 - ▣ delete- 매핑된 DELETE 구문.
 - ▣ select- 매핑된 SELECT 구문.
- sqlmap 에 대한 자세한 내용이 있는 곳은...
 - ▣ <http://www.mybatis.org/core/ko/sqlmap-xml.html>

6.3.1 com/myapp/sqlmap/EmpMapper

209

```
<?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.empapp.dao.EmpMapper">
  <select id="selectEmp" parameterType="int" resultType="com.empapp.model.Emp">
    select * from emp where empno = #{id}
  </select>
</mapper>
```

6.3.2 select

210

- select 구문은 MyBatis 에서 가장 흔히 사용할 요소이다. 데이터베이스에서 데이터를 가져온다. 아마도 대부분의 애플리케이션은 데이터를 수정하기보다는 조회하는 기능을 많이 가진다. 그래서 MyBatis 는 데이터를 조회하고 그 결과를 매핑하는데 집중하고 있다. Select 는 다음 예처럼 단순한 경우에는 단순하게 설정된다.
- ```
<select id="selectPerson" parameterType="int" resultType="hashmap">
 SELECT * FROM PERSON WHERE ID = #{id}
</select>
```
- #{id} 표기법은 MyBatis 에게 PreparedStatement 파라미터를 만들도록 지시 한다. JDBC 를 사용할 때 PreparedStatement 에는 “?” 형태로 파라미터가 전달된다. 즉 결과적으로 위 설정은 아래와 같이 작동하게 되는 셈이다.

## 6.3.2 select

211

- <select id="selectPerson" parameterType="int" parameterMap="deprecated" resultType="hashmap" resultMap="personResultMap" flushCache="false" useCache="true" timeout="10000" fetchSize="256" statementType="PREPARED" resultSetType="FORWARD\_ONLY">

## 6.3.3 Select Attributes

| 속성                   | 설명                                                                                                                                                          |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>id</b>            | 구문을 찾기 위해 사용될 수 있는 명명공간내 유일한 구분자                                                                                                                            |
| <b>parameterType</b> | 구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭                                                                                                                      |
| <b>parameterMap</b>  | 외부 parameterMap 을 찾기 위한 비권장된 접근방법. 인라인 파라미터 매핑과 parameterType 을 대신 사용하라.                                                                                    |
| <b>resultType</b>    | 이 구문에 의해 리턴되는 기대타입의 패키지 경로를 포함한 전체 클래스명이나 별칭. collection 이 경우, collection 타입 자체가 아닌 collection 이 포함된 타입이 될 수 있다. resultType 이나 resultMap 을 사용하라.            |
| <b>resultMap</b>     | 외부 resultMap 의 참조명. 결과맵은 MyBatis 의 가장 강력한 기능이다. resultType 이나 resultMap 을 사용하라.                                                                             |
| <b>flushCache</b>    | 이 값을 true 로 셋팅하면, 구문이 호출될때마다 캐시가 지원될것이다(flush). 디폴트는 false 이다.                                                                                              |
| <b>useCache</b>      | 이 값을 true 로 셋팅하면, 구문의 결과가 캐시될 것이다. 디폴트는 true 이다.                                                                                                            |
| <b>timeout</b>       | 예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간을 설정한다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.                                                                      |
| <b>fetchSize</b>     | 지정된 수만큼의 결과를 리턴하도록 하는 드라이버 힌트 형태의 값이다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.                                                                           |
| <b>statementType</b> | STATEMENT, PREPARED 또는 CALLABLE 중 하나를 선택할 수 있다. MyBatis 에게 Statement, PreparedStatement 또는 CallableStatement 를 사용하게 한다. 디폴트는 PREPARED 이다.                   |
| <b>resultSetType</b> | FORWARD_ONLY SCROLL_SENSITIVE SCROLL_INSENSITIVE 중 하나를 선택할 수 있다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.                                                 |
| <b>databaseld</b>    | 설정된 databaseldProvider 가 있는 경우, MyBatis 는 databaseld 속성이 없는 모든 구문을 로드하거나 일치하는 databaseld 와 함께 로드될 것이다. 같은 구문에서 databaseld 가 있거나 없는 경우 모두 있다면 뒤에 나온 것이 무시된다. |

## 6.3.4 insert, update and delete

213

```
<insert
 id="insertAuthor"
 parameterType="domain.blog.Author"
 flushCache="true"
 statementType="PREPARED"
 keyProperty=""
 keyColumn=""
 useGeneratedKeys=""
 timeout="20000">

<update
 id="insertAuthor"
 parameterType="domain.blog.Author"
 flushCache="true"
 statementType="PREPARED"
 timeout="20000">

<delete
 id="insertAuthor"
 parameterType="domain.blog.Author"
 flushCache="true"
 statementType="PREPARED"
 timeout="20000">
```

## 6.3.5 Insert, Update and Delete Attributes

| 속성                      | 설명                                                                                                                                                 |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>id</b>               | 구문을 찾기 위해 사용될 수 있는 명명공간내 유일한 구분자                                                                                                                   |
| <b>parameterType</b>    | 구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭                                                                                                             |
| <b>parameterMap</b>     | 외부 parameterMap 을 찾기 위한 비권장된 접근방법. 인라인 파라미터 매핑과 parameterType을 대신 사용하라.                                                                            |
| <b>flushCache</b>       | 이 값을 true 로 셋팅하면, 구문이 호출될때마다 캐시가 지원질것이다(flush). 디폴트는 false 이다.                                                                                     |
| <b>timeout</b>          | 예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간을 설정한다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.                                                             |
| <b>statementType</b>    | STATEMENT, PREPARED 또는 CALLABLE 중 하나를 선택할 수 있다. MyBatis 에게 Statement, PreparedStatement 또는 CallableStatement 를 사용하게 한다. 디폴트는 PREPARED 이다.          |
| <b>useGeneratedKeys</b> | (입력(insert)에만 적용) 데이터베이스에서 내부적으로 생성한 키(예를 들어, MySQL 또는 SQL Server 와 같은 RDBMS 의 자동 증가 필드)를 받는 JDBC getGeneratedKeys 메서드를 사용하도록 설정하다. 디폴트는 false 이다. |
| <b>keyProperty</b>      | (입력(insert)에만 적용) getGeneratedKeys 메서드나 insert 구문의 selectKey 하위 요소에 의해 리턴된 키를 셋팅할 프로퍼티를 지정. 디폴트는 셋팅하지 않는 것이다.                                      |
| <b>keyColumn</b>        | (입력(insert)에만 적용) 생성키를 가진 테이블의 칼럼명을 셋팅. 키 칼럼이 테이블이 첫번째 칼럼이 아닌 데이터베이스 (PostgreSQL 처럼)에서만 필요하다.                                                      |

## 6.3.6 Insert, update, delete 구문의 예

215

```
<insert id="insertAuthor" parameterType="domain.blog.Author">
 insert into Author (id,username,password,email,bio)
 values (#{id},#{username},#{password},#{email},#{bio})
</insert>

<update id="updateAuthor" parameterType="domain.blog.Author">
 update Author set
 username = #{username},
 password = #{password},
 email = #{email},
 bio = #{bio}
 where id = #{id}
</update>

<delete id="deleteAuthor" parameterType="int">
 delete from Author where id = #{id}
</delete>
```

## 6.3.7 insert를 위한 key 생성 기능

- 사용하는 데이터베이스가 자동생성키(예를 들면, MySQL 과 SQL 서버)를 지원한다면, `useGeneratedKeys="true"` 로 설정하고 대상 프로퍼티에 `keyProperty` 를 셋팅 할 수 있다. 예를 들어, Author 테이블이 id 칼럼에 자동 생성키를 적용했다고 하면, 구문은 아래와 같은 형태일 것이다.

```
<insert id="insertAuthor" parameterType="domain.blog.Author"
 useGeneratedKeys="true" keyProperty="id">
 insert into Author (username,password,email,bio)
 values (#{username},#{password},#{email},#{bio})
</insert>
```

## 6.3.8 selectKey

217

- MyBatis 는 자동생성키 칼럼을 지원하지 않는 다른 데이터베이스를 위해 다른 방법 또한 제공한다.

```
<insert id="insertAuthor" parameterType="domain.blog.Author">
 <selectKey keyProperty="id" resultType="int" order="BEFORE">
 select CAST(RANDOM()*1000000 as INTEGER) a from
 SYSIBM.SYSDUMMY1
 </selectKey>
 insert into Author (id, username, password, email, bio, favourite_section)
 values (#{id}, #{username}, #{password}, #{email}, #{bio},
 #{favouriteSection,jdbcType=VARCHAR})
</insert>
```

- 위 코드는 selectKey 구문이 먼저 실행되고, Author id 프로퍼티에 셋팅된다. 그리고 나서 insert 구문이 실행된다. 이건 복잡한 자바코드 없이도 데이터베이스에 자동생성키의 행위와 비슷한 효과를 가지도록 해준다.

## 6.3.9 selectKey Attributes

| 속성            | 설명                                                                                                                                                                                                       |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| keyProperty   | selectKey 구문의 결과가 셋팅될 대상 프로퍼티                                                                                                                                                                            |
| resultType    | 결과의 타입. MyBatis 는 이 기능을 제거할 수 있지만 추가하는게 문제가 되지는 않을것이다. MyBatis 는 String 을 포함하여 키로 사용될 수 있는 간단한 타입을 허용한다.                                                                                                 |
| order         | BEFORE 또는 AFTER 를 셋팅할 수 있다. BEFORE 로 셋팅하면, 키를 먼저 조회하고 그 값을 keyProperty 에 셋팅한 뒤 insert 구문을 실행한다. AFTER 로 셋팅하면, insert 구문을 실행한 뒤 selectKey 구문을 실행한다. Oracle 과 같은 데이터베이스에서는 insert 구문 내부에서 일관된 호출 형태로 처리한다. |
| statementType | 위 내용과 같다. MyBatis 는 Statement, PreparedStatement 그리고 CallableStatement 을 매핑하기 위해 STATEMENT, PREPARED 그리고 CALLABLE 구문 타입을 지원한다.                                                                           |

## 6.4 전자정부 표준프레임워크와 iBatis

- 전자정부 표준프레임워크에서는 DataAccess 프레임워크로 MyBatis가 아닌 iBatis를 채택하고 있음
- DAO클래스는 SqlMapClientDaoSupport 클래스를 상속받고, SqlMapClient 빈 객체를 setter injection 방식을 이용하여 사용하고 있음
- 쿼리의 구현은 매퍼 클래스를 이용함
- 테이블과 매핑되는 Value Object 클래스 작성
- Spring 설정파일(예: dispatcher-servlet.xml)에 sql-map-config.xml 설정파일의 위치를 지정해야 함
  - ▣ 결과 셋을 Value Object 클래스를 직접 지정할 수 있지만, resultMpa에 타입을 명확하게 지시해주면 java 의 reflection 기술을 사용하여 대상 클래스의 개별 속성에 대한 type 을 구하는 것보다 성능상 이점이 있을 수 있다.
- sql-map-config.xml 설정파일에 sql-map.xml 파일의 위치를 지정해야 함
- sql-map 파일에 CRUD 쿼리를 작성함
- 쿼리문에 <, >등 파싱 하면 안 되는 문자는 <!CDATA[<>]>로 묶어준다.

## 6.4.1 com/emp/service/impl/EmpDAO

220

```
□ package com.emp.service.impl;

□ import java.util.List;
□ import javax.annotation.Resource;
□ import org.springframework.orm.ibatis.support.SqlMapClientDaoSupport;
□ import org.springframework.stereotype.Repository;
□ import com.ibatis.sqlmap.client.SqlMapClient;
□ import com.emp.vo.EmpVo;

□ @Repository("empDao")
□ public class EmpDao extends SqlMapClientDaoSupport {
 □ @Resource(name="sqlMapClient")
 □ public void setSuperSqlMapClient(SqlMapClient sqlMapClient) {
 □ super.setSqlMapClient(sqlMapClient);
 }
 □ @SuppressWarnings("unchecked")
 □ public List<EmpVo> selectAllEmp() {
 □ return getSqlMapClientTemplate().queryForList("empDao.selectAllEmp", null);
 }
}
```

SQL mapper의  
namespace 명

SQL mapper의 id  
와 일치해야 함

## 6.4.2 com/sqlmap/emp/EmpSQL\_oracle.xml

221

```


- <?xml version="1.0" encoding="UTF-8"?>
- <!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
 "http://ibatis.apache.org/dtd/sql-map-2.dtd" >
- <sqlMap namespace="empDao">
- <typeAlias alias="empVo" type="daou.emp.vo.EmpVo"/>
- <resultMap id="empResult" class="empVo">
 <result property="empno" column="EMPNO" javaType="int" jdbcType="NUMERIC"/>
 <result property="ename" column="ENAME" javaType="string" jdbcType="VARCHAR" nullValue="" />
 <result property="job" column="JOB" javaType="string" jdbcType="VARCHAR" nullValue="" />
 <result property="mgr" column="MGR" javaType="int" jdbcType="NUMERIC" nullValue="0" />
 <result property="hiredate" column="HIREDATE" javaType="date" jdbcType="DATE" />
 <result property="sal" column="SAL" javaType="double" jdbcType="DOUBLE" />
 <result property="comm" column="COMM" javaType="double" jdbcType="DOUBLE" nullValue="0" />
 <result property="deptno" column="DEPTNO" javaType="int" jdbcType="NUMERIC" />
 </resultMap>
- <select id="selectAllEmp" resultMap="empResult">
 SELECT * FROM emp
 </select>
- </sqlMap>

```

insert, update 등 파라미터에 Value Object를 사용하기 위해서 parameterMap을 정의함

Select 쿼리 결과를 Value Object 클래스와 매핑시킨 resultMap

날짜의 경우 DATE, DATETIME, TIMESTAMP 를 구분해야 함

널인 필드의 경우 널일 경우 대체되어야 할 값을 지정

결과 값들이 매핑될 resultMap의 id

namespace를 statementName으로 사용하려면 sqlmap-config 파일에 <settings useStatementNamespaces="true" /> 해야 함

DAO클래스의 statementName이름은 empDao.selectAllEmp가 됨

## 6.4.3 com/sqlmap/config/sqlmap-config-oracle-emp.xml

6. MyBatis Framework

222

```
□ <?xml version="1.0" encoding="UTF-8"?>
□ <!DOCTYPE sqlMapConfig
PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-config-2.dtd" >
□ <sqlMapConfig>
□ <settings useStatementNamespaces= "true" />
□ <sqlMap resource="com/sqlmap/emp/EmpSQL_oracle.xml" />
□ </sqlMapConfig>
```

SQL mapper 파일에서 namespace를  
지정하고 namespace명.id이름을  
statementName으로 사용함

SQL mapper 파일의  
경로를 지정

## 6.4.4egovframework/spring/com/contest-sqlMap.xml

223

```
□ <!-- lob Handler -->
□ <bean id="lobHandler" class="org.springframework.jdbc.support.lob.DefaultLobHandler" lazy-
init="true" />

□ <!-- SqlMap setup for iBATIS Database Layer -->
□ <bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
□ <property name="configLocations">
□ <list>
□ <value>classpath:/egovframework/sqlmap/config/${Globals.DbType}/*.xml</value>
□ <value>classpath:/com/sqlmap/config/*.xml</value>
□ </list>
□ </property>
□ <property name="dataSource" ref="dataSource-${Globals.DbType}" />
□ <property name="lobHandler" ref="lobHandler" />
□ </bean>
```

SQL config 파일의  
경로를 지정

## 6.4.5 WEB-INF/config/egovframework/springmvc/egov-com-servlet.xml

b. MyBatis Framework

224

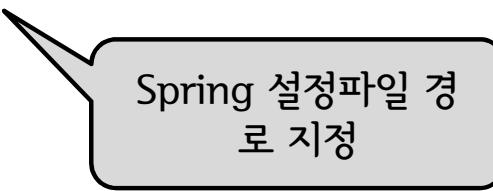
- <!-- 패키지 내 Controller, Service, Repository 클래스의 auto detect를 위한 mvc 설정 -->
- <context:component-scan base-package="com.emp.service.impl" />

Dao 클래스를 빈으로  
자동 등록시키기 위해  
컴포넌트 스캔 지정

## 6.4.6 WEB-INF/web.xml

225

- <context-param>
- <param-name>contextConfigLocation</param-name>
- <param-value>
- classpath\*:egovframework/spring/com/context-\*.xml
- </param-value>
- </context-param>



Spring 설정파일 경  
로 지정

226

## 7. Spring Web MVC

# 7.1 Spring MVC - 개요(1/2)

227

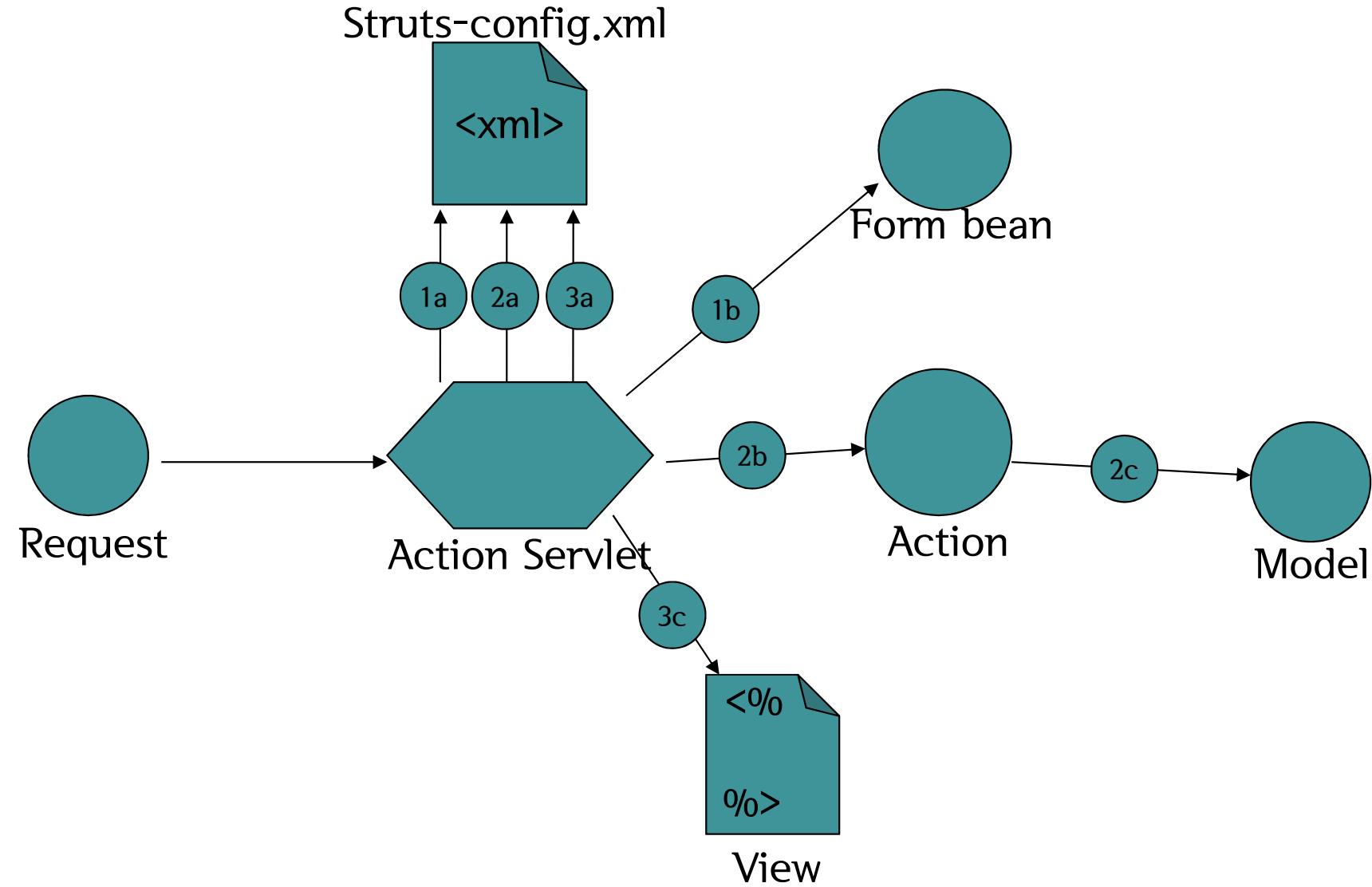
7. Spring Web MVC

- MVC(Model-View-Controller) 패턴은 코드를 기능에 따라 Model, View, Controller 3가지 요소로 분리한다.
  - ▣ Model : 어플리케이션의 데이터와 비즈니스 로직을 담는 객체이다.
  - ▣ View : Model의 정보를 사용자에게 표시한다. 하나의 Model을 다양한 View에서 사용할 수 있다.
  - ▣ Controller : Model과 View의 중계역할을 한다. 사용자의 요청을 받아 Model에 변경된 상태를 반영하고, 응답을 위한 View를 선택한다.
- MVC 패턴은 UI 코드와 비즈니스 코드를 분리함으로써 종속성을 줄이고, 재사용성을 높이고, 보다 쉬운 변경이 가능하도록 한다.
- 오픈소스 Web MVC Framework
  - ▣ Spring MVC, Struts, Webwork 등이 있다.

# 7.1 Spring MVC - 개요(2/2)

- Spring Framework
  - ▣ Framework내의 특정클래스를 상속하거나, 참조, 구현해야 하는 등의 제약사항이 비교적 적다.
  - ▣ IOC Container가 Spring 이라면 연계를 위한 추가 설정없이 Spring MVC를 사용할 수 있다.
- Spring MVC
  - ▣ DispatcherServlet, HandlerMapping, Controller, Interceptor, ViewResolver, View등 각 컴포넌트들의 역할이 명확하게 분리된다.
  - ▣ HandlerMapping, Controller, View등 컴포넌트들에 다양한 인터페이스 및 구현클래스를 제공한다.
  - ▣ Controller(@MVC)나 폼클래스(커맨드클래스) 작성시에 특정클래스를 상속받거나 참조할 필요 없이 POJO 나 POJO-style의 클래스를 작성함으로써 비즈니스 로직에 집중한 코드를 작성할 수 있다.
  - ▣ 웹 요청 파라미터와 커맨드 클래스간에 데이터매핑 기능을 제공한다.
  - ▣ 데이터검증을 할 수 있는, Validator와 Error 처리 기능을 제공한다.
  - ▣ JSP Form을 쉽게 구성하도록 Tag를 제공한다

## 7.1.1 스트ր츠는?



## 7.1.2 MVC패턴의 서블릿(컨트롤러)이 하는일

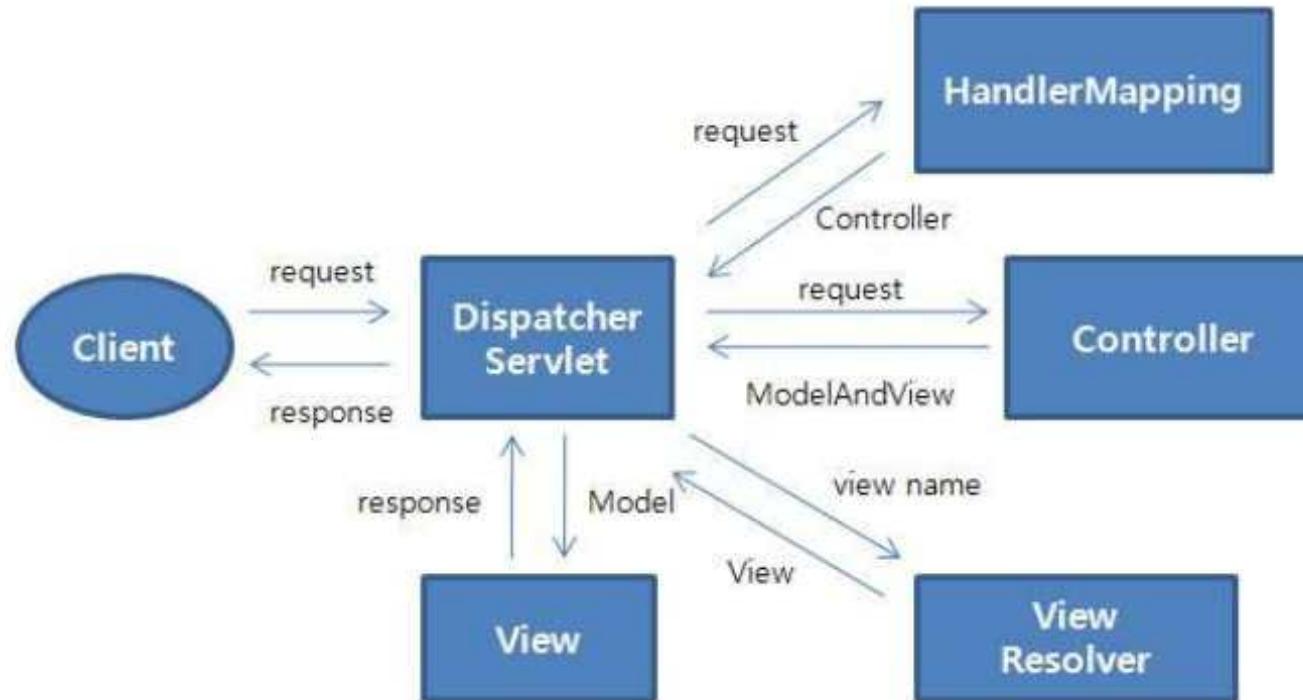
230

1. 요청받기
  2. 요청 처리하기 위해 값 추출하기
  3. 비지니스 메서드 호출하기
  4. 뷰단에 공유해야 할 일이 있을 때 적당한 스코프에 객체 속성으로 추가
  5. 뷰에 forward, redirect
- 
- 컨트롤러가 너무 복잡하다.
  - 모든 컨트롤러들이 비슷한 일을 하기 때문에 중복된다.
  
  - 그래서 '스프링'이 컨트롤러단을 미리 만들어서 제공해 줍니다.
  - 그럼 스프링의 장점은?
    - ▣ 스프링이 제공하는 트랜잭션처리, DI, AOP 적용 등을 쉽게 사용할 수 있습니다.
    - ▣ 스트럿츠와 같은 프레임워크와 연동을 위한 추가적인 설정이 필요 없습니다.

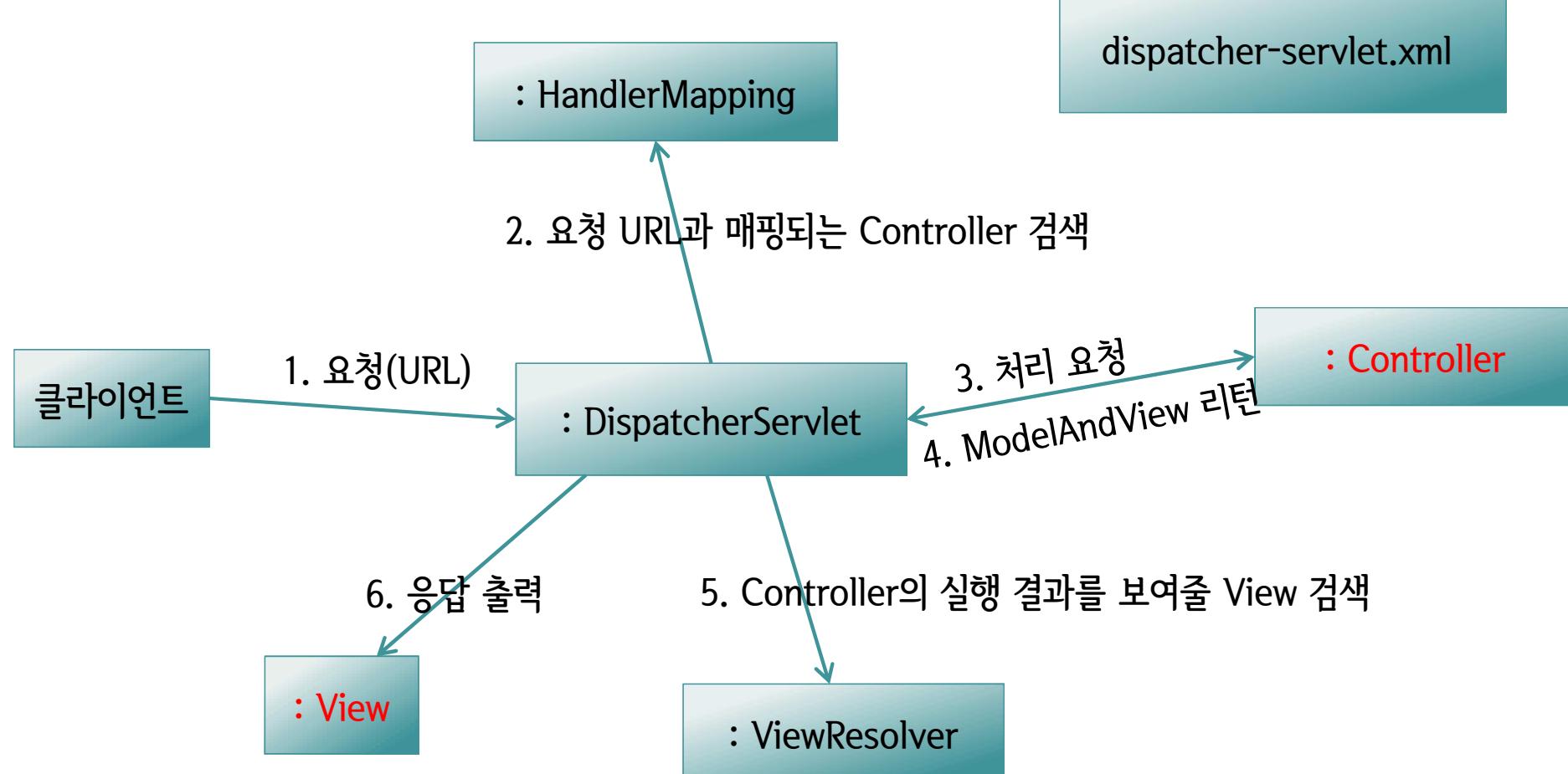
## 7.1.3 Spring Web MVC의 핵심 Component

- **DispatcherServlet**
  - ▣ 클라이언트의 요청을 전달받아,
  - ▣ 컨트롤러에게 클라이언트의 요청을 전달하고
  - ▣ 컨트롤러가 리턴 한 결과 값을 View에게 전달하여 알맞은 응답을 생성하도록 함
- **HandlerMapping**
  - ▣ 클라이언트의 요청 URL을 어떤 컨트롤러가 처리할지를 결정함
- **컨트롤러(Controller)**
  - ▣ 클라이언트의 요청을 처리한 뒤, 그 결과를 DispatcherServlet에 알려줌
  - ▣ 스트럿츠의 Action과 동일한 역할을 수행
- **ModelAndView**
  - ▣ 컨트롤러가 처리한 결과 정보 및 뷰 선택에 필요한 정보를 담는다.
- **ViewResolver**
  - ▣ 컨트롤러의 처리 결과를 생성할 뷰를 결정
- **뷰(View)**
  - ▣ 컨트롤러의 처리 결과 화면을 생성

- Client의 요청이 들어오면 DispatcherServlet이 가장 먼저 요청을 받는다.
- HandlerMapping이 요청에 해당하는 Controller를 return한다.
- Controller는 비지니스로직을 수행(호출)하고 결과데이터를 ModelAndView에 반영하여 return한다.
- ViewResolver는 view name을 받아 해당하는 View 객체를 return한다.
- View는 Model 객체를 받아 rendering한다.

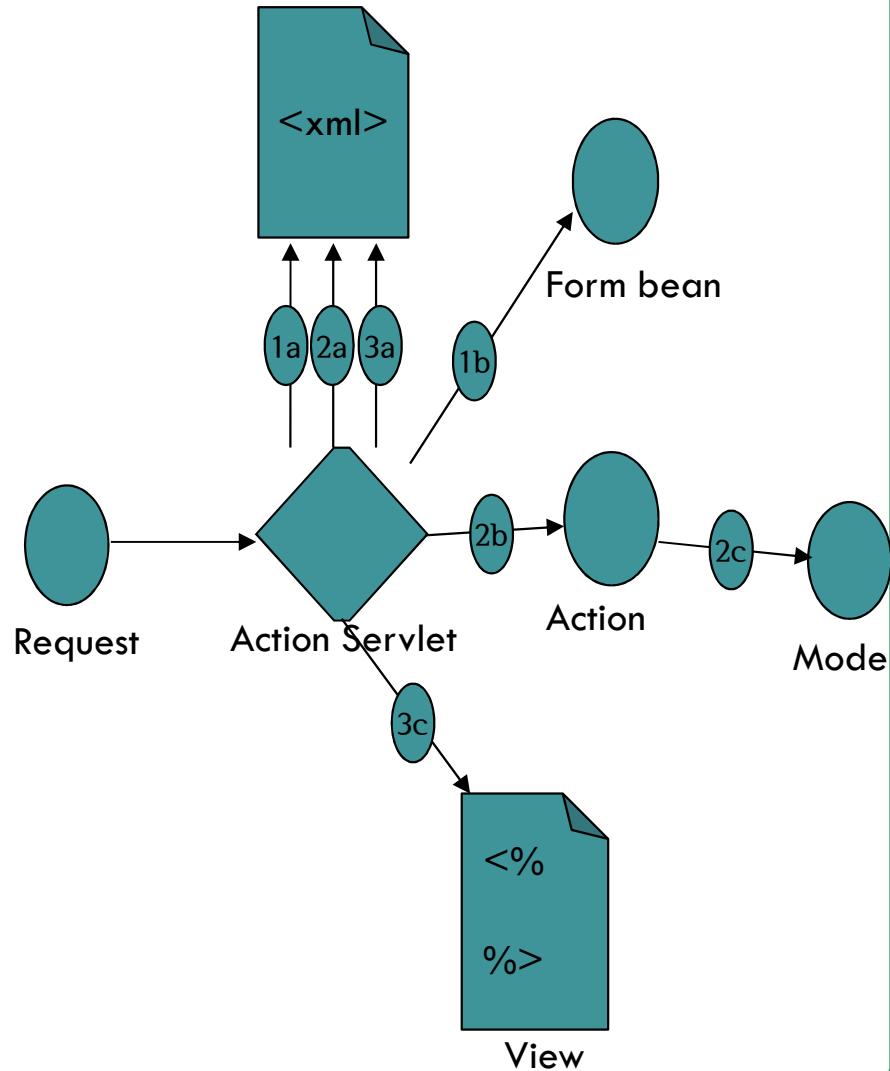


## 7.1.5 스프링 MVC의 클라이언트 요청 처리 과정



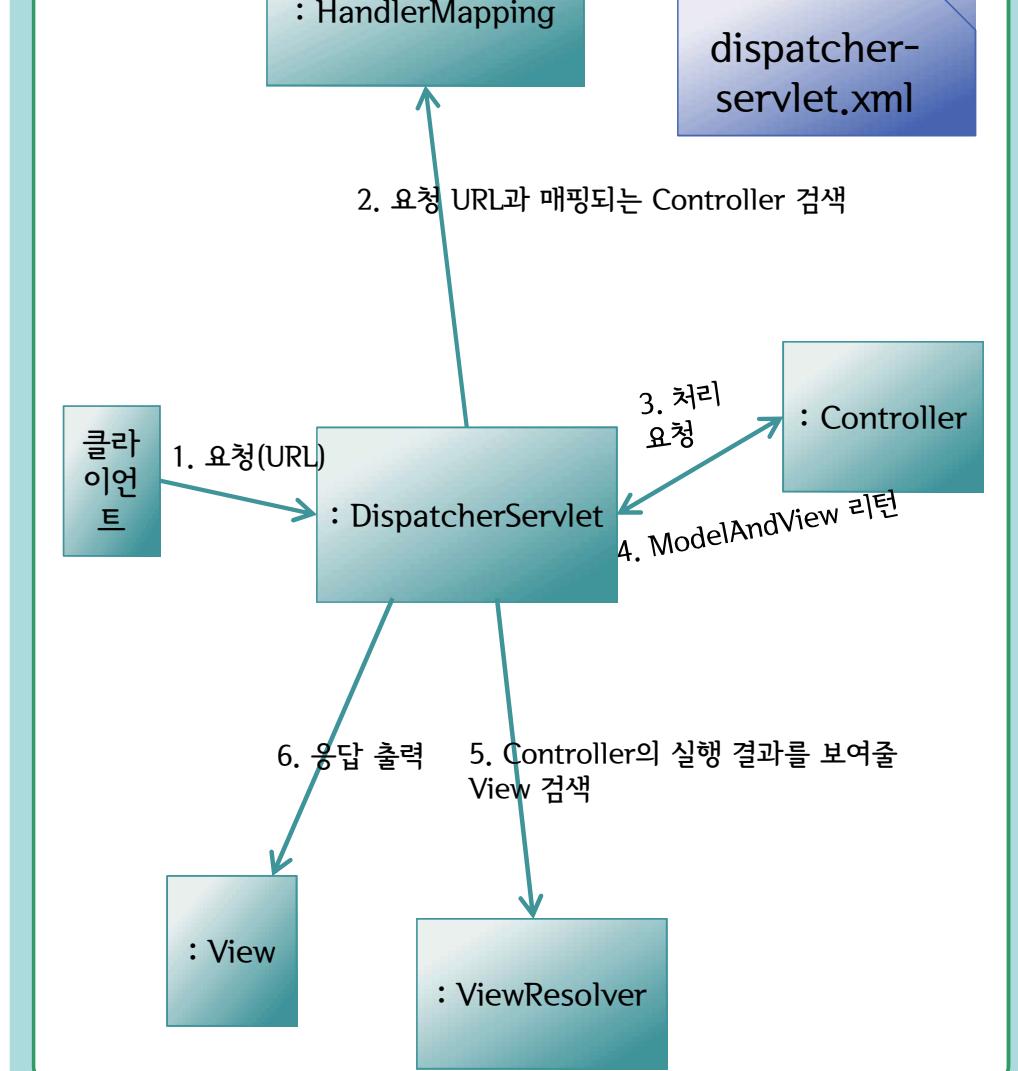
## 7.1.6 스트럿츠와 Spring MVC 비교

Struts-config.xml



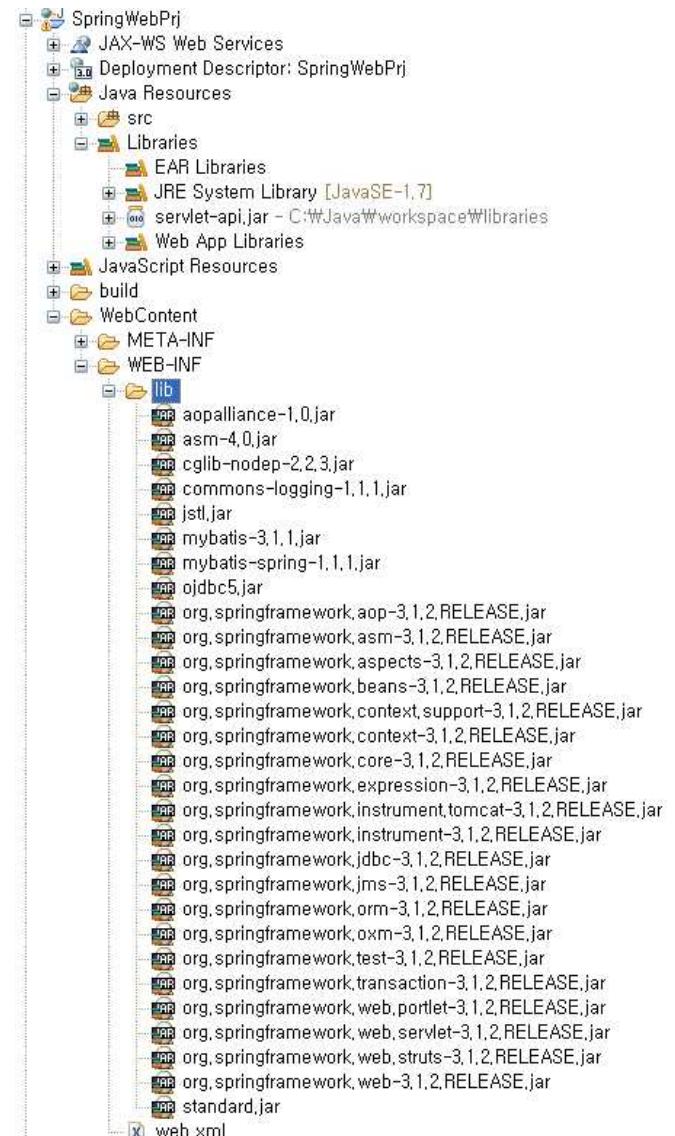
: HandlerMapping

dispatcher-servlet.xml



- DispatcherServlet을 web.xml 파일에 설정
  - ▣ 모든 요청이 디스패쳐 서블릿으로 들어오도록
- 요청 URL과 컨트롤러의 매팅 방식을 설정
  - ▣ HandlerMapping을 이용
- 컨트롤러 작성
  - ▣ 클라이언트의 요청을 처리
- ViewResolver 설정
  - ▣ 어떤 뷰를 이용하여 컨트롤러의 처리 결과 응답 화면을 생성할지 결정
- 뷰 코드 작성
  - ▣ JSP 이용

- Dynamic Web Project 만들기
  - ▣ File > New > Other > Web > Dynamic Web Project
  
- Jar 파일 삽입하기(Import)
  - ▣ WEB-INF/lib 폴더에
    - Spring jar 파일들
    - commons-logging-1.1.1.jar
    - cglib & asm 모듈
    - aop 모듈
    - mybatis.jar & mybatis-spring.jar
    - standard.jar
    - jstl.jar
    - jdbc driver
  - ▣ 빌드패스에 추가
    - servlet-api.jar
      - tomcat/lib 에 있음
      - 서블릿 컴파일하기 위해서



## 7.2.1 DispatcherServlet 설정

- 클라이언트의 요청을 전달받을 DispatcherServlet 설정
- 공통으로 사용할 어플리케이션 컨텍스트 설정
- /WEB-INF/ 디렉토리에 [서블릿이름]-servlet.xml 파일로부터 스프링 설정 정보를 읽어온다.
- 위와 같이 했다면 설정파일은 /WEB-INF/dispatcher-servlet.xml 파일이 됨

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
...
<servlet>
 <servlet-name>dispatcher</servlet-name>
 <servlet-class>
 org.springframework.web.servlet.DispatcherServlet
 </servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>dispatcher</servlet-name>
 <url-pattern>*.do</url-pattern>
</servlet-mapping>
...
</web-app>

```

설정파일을 다르게 할 경우...>뒤에서 자세하게 설명됩니다.

```

</servlet-class>
<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>
 /WEB-INF/common.xml
 /WEB-INF/emp.xml
 /WEB-INF/login.xml
</param-value>
</init-param>
</servlet>

```

## 7.2.2 컨트롤러 구현

238

```
@Controller
```

```
public class HelloController {
```

```
 @RequestMapping("/hello.do")
```

```
 public ModelAndView hello() {
```

```
 ModelAndView mav = new ModelAndView();
```

```
 mav.setViewName("hello");
```

```
 mav.addObject("greeting", getGreeting());
```

```
 return mav;
```

```
}
```

```
private String getGreeting() {
```

```
 int hour = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);
```

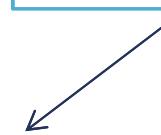
```
 //생략
```

```
 return hour;
```

```
}
```

```
}
```

portlet 패키지에 있는 것이 아니고  
servlet 패키지에 있는 ModelAndView 입니다.



## 7.2.3 dispatcher-servlet.xml

- File -> New -> Other -> Spring -> Spring Bean Configuration File
- 설정파일 만들 때 beans, context, tx 선택하고 파일 생성
- 위치는 /WEB-INF/ 폴더에 생성
- 컨트롤러 빈 설정
- 뷰리졸버 설정(위치, 파일 형식)

빈 자동 등록 설정해도 됨

<context:annotation-config/>

<context:component-scan base-package="com.kdb.controller"/>

```
<bean id="helloController" class="com.myweb.controller.HelloController" />
```

```
<bean id="viewResolver"
 class="org.springframework.web.servlet.view.InternalResourceViewResolver">
 <property name="prefix" value="/WEB-INF/view/" />
 <property name="suffix" value=".jsp" />
</bean>
```

## 7.2.4 뷰 코드(/WEB-INF/view/hello.jsp)

240

- <%@ page contentType="text/html;charset=euc-kr" %>
- 인사말 : \${greeting}
  
- 브라우저에서 <http://localhost:8080/hello.do> 로 실행함
- 컨텍스트 이름이 있을 경우 <http://localhost:8080/contextName/hello.do> 로 실행함

오라클 http 데몬이 실행되고 있을 경우 오라클의 http 데몬 포트번호를 변경해야 톰캣에서 8080 포트를 사용할 수 있습니다.

```
SQL> conn /as sysdba
Connected.
SQL> exec dbms_xdb.sethttpport(9090)
```

PL/SQL procedure successfully completed.

## 7.3 DispatcherServlet

241

```
□ <servlet>
 □ <servlet-name>dispatcher</servlet-name>
 □ <servlet-class>
 □ org.springframework.web.servlet.DispatcherServlet
 □ </servlet-class>
 □ <init-param>
 □ <param-name>contextConfigLocation</param-name>
 □ <param-value>
 □ /WEB-INF/common.xml
 □ /WEB-INF/emp.xml
 □ /WEB-INF/login.xml
 □ </param-value>
 □ </init-param>
 □ </servlet>
 □ <servlet-mapping>
 □ <servlet-name>dispatcher</servlet-name>
 □ <url-pattern>*.do</url-pattern>
 □ </servlet-mapping>
```

## 7.3.1 DispatcherServlet

242

- 디스패처 서블릿을 두 개 이상 두고 다른 이름의 설정파일을 지정할 수 있다.
- <servlet>
- < servlet-name>front</servlet-name>
- < servlet-class>org. … .DispatcherServlet</servlet-class>
- < init-param>
- < param-name>contextConfigLocation</param-value>
- < param-value>/WEB-INF/front.xml</param-value>
- </init-param>
- </servlet>
- <servlet>
- < servlet-name>rest</servlet-name>
- < servlet-class>org. … .DispatcherServlet</servlet-class>
- < init-param>
- < param-name>contextConfigLocation</param-value>
- < param-value>/WEB-INF/rest.xml</param-value>
- </init-param>
- </servlet>

## 7.3.2 공통 빈

243

```
<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>/WEB-INF/servlet.xml, /WEB-INF/persistence.xml</param-value>
</context-param>

<listener>
 <listener-class>
 org.springframework.web.context.ContextLoaderListener
 </listener-class>
</listener>
```

### 7.3.3 인코딩 필터

244

- 인코딩 필터를 이용하면 `request.setCharacterEncoding("euc-kr");`를 넣지 않아도 됨

```
<filter>
 <filter-name>encodingFilter</filter-name>
 <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
 <init-param>
 <param-name>encoding</param-name>
 <param-value>EUC-KR</param-value>
 </init-param>
</filter>
<filter-mapping>
 <filter-name>encodingFilter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
```

- 스프링 MVC의 SimpleFormController를 비롯한 스프링의 MVC의 몇몇 컨트롤러는 기존의 MVC 프레임워크에 비해 그 처리 흐름이 복잡한 편이었고, 많은 종류의 부모클래스를 제공하고 있었다.
- Annotation을 이용하여 복잡함을 줄임과 동시에 기능을 구현할 수 있으므로 Annotation 기반 스프링 MVC 구현을 선호하고 있다.
  - ▣ @Controller
  - ▣ @RequestMapping
    - value 속성은 요청 URL
    - method 속성은 요청 방식 지정(지정하지 않으면 모든 HTTP 전송방식 처리)
    - method=RequestMethod.GET
    - method=RequestMethod.POST

## 7.4.1 용도에 따른 Controller 클래스 분류

8. Spring을 이용한 웹 어플리케이션

246

| 용도             | 클래스(인터페이스)                                                 | 설명                                                            |
|----------------|------------------------------------------------------------|---------------------------------------------------------------|
| 단순처리           | Controller<br>AbstractController                           | 별도 기능을 제공하지 않는 컨트롤러.<br>요청 파라미터 처리 등의 작업을 직접 구현해 주어야 한다.      |
| 파라미터 매팅        | AbstractCommandController                                  | 요청 파라미터를 객체에 저장해 주며, 파라미터 값 검증 기능을 제공한다.                      |
| 입력 폼 처리        | SimpleFormController                                       | 폼을 출력하고 폼에 입력한 데이터를 처리할 때 사용된다.                               |
| 다중 페이지 입력 폼 처리 | AbstractWizardFormController                               | 여러 페이지에 걸쳐서 데이터를 입력하는 경우, 입력 폼의 흐름을 제어하고 입력한 데이터를 처리할 때 사용한다. |
| 정적 뷰 매팅        | ParameterizableViewController<br>UrlFilenameViewController | 컨트롤러에서 어떤 기능도 수행하지 않고, 단순히 클라이언트의 요청을 뷰로 전달할 때 사용된다.          |
| 다중 액션          | MultiActionController                                      | 연관된거나 비슷한 로직을 수행하는 다수의 기능을 하나의 컨트롤러에서 구현할 때 사용된다.             |

## 7.4.1 Annotation

247

- **@Controller**
  - ▣ 컨트롤러 클래스 선언부 위에 지정
- **@RequestMapping**
  - ▣ 컨트롤러 메서드 선언부 위에 지정
  - ▣ value 속성은 요청 URL
  - ▣ method 속성은 요청 방식 지정(지정하지 않으면 모든 HTTP 전송방식을 처리)
  - ▣ method=RequestMethod.GET
  - ▣ method=RequestMethod.POST

## 7.4.2 폼에 입력한 데이터 전달

248

- HTML 폼에 입력한 데이터를 자바빈 객체로 전달받는 방법
- @RequestMapping 어노테이션이 적용된 메서드의 파라미터에 자바빈 타입을 추가해 주기만 하면 됨
- HTML 폼 name 속성의 값과 자바빈 클래스의 프로퍼티 이름이 일치할 경우 폼에 입력한 값을 해당 자바빈 클래스의 프로퍼티로 설정해 주는 기능이 있다.
- **타입변환도 알아서 해준다.**
- 폼으로부터 하나의 name 속성에 값이 여러 개 넘어올 경우 List<E> 타입으로 선언해 놓으면 자동으로 바인딩 된다.
  - List<OrderItem> orderItems;로 선언되어 있을 경우
  - HTML에서
  - <form method="post">
  - 상품1 <input type="text" name="orderItems[0].itemId" />
  - 개수 <input type="text" name="orderItems[0].number" />
  - 주의 <input type="text" name="orderItems[0].remark" /><br>
  - 상품2 <input type="text" name="orderItems[1].itemId" />
  - 개수 <input type="text" name="orderItems[1].number" />
  - 주의 <input type="text" name="orderItems[1].remark" />
  - <input type="submit" />
  - </form>

## 7.4.3 컨트롤러 메서드의 파라미터 타입

249

- HttpServletRequest, HttpServletResponse
- HttpSession
- org.springframework.web.context.request.WebRequest or  
org.springframework.web.context.request.NativeWebRequest
- java.util.Locale : 현재 요청에 대한 로케일
- java.io.InputStream / java.io.Reader : 요청 컨텐츠에 직접 접근
- java.io.OutputStream / java.io.Writer : 응답 컨텐츠를 생성
- java.security.Principal : 인증된 사용자
- @PathVariable : URI 템플릿 변수에 접근
- @RequestParam
  - ▣ 파라미터 값 받을 때, 선언된 타입으로 형변환 안되면 400(Bad Request)에러
  - ▣ public ModelAndView search(@RequestParam("num") int pageNumber) {
  - ▣ 필수 파라미터가 아닌 경우 required=false
  - ▣ 기본 값 지정은 defaultValue로 지정(defaultValue="1")
- @RequestHeader : 요청 헤더 매핑
- @CookieValue : 쿠키 매핑
- @RequestBody : 요청 몸체 내용에 접근
- HttpEntity<?> : Servlet request HTTP 헤더와 컨텐츠
- java.util.Map, org.springframework.ui.Model, org.springframework.ui.ModelMap : 뷰에 전달한 모델 데이터 설정
- org.springframework.web.servlet.mvc.support.RedirectAttributes : redirect에 사용되는 값, 뷰이름에 redirect:  
가 들어갈 때 사용
- 커맨드 객체 : 요청 파라미터 저장 객체, 클래스 이름을 모델명으로 사용 @ModelAttribute로 모델명 설정 가능
- org.springframework.validation.Errors, org.springframework.validation.BindingResult : 커멘트 또는 폼 객체의 유효성 검증 결과
- org.springframework.web.bind.support.SessionStatus
- org.springframework.web.util.UriComponentsBuilder

# @PathVariable과 URI 템플릿 변수

- RESTful 서비스의 URI 형태를 지원하기 위한 것임
- xxxx/yyyy/info?user=id 형태의 url이 있다면 xxxx/yyyy/info/user/id 처럼 파라미터가 uri에 포함되도록 하는 형태.

## 1. 설정방법

- @RequestMapping 어노테이션값으로 { 템플릿변수 } 를 사용한다.
- @PathVariable 어노테이션을 이용해서 { 템플릿 변수 } 와 동일한 이름을 갖는 파라미터를 추가한다.

### 1. @Controller

```

2. public class CharacterInfoController {
3. @RequestMapping("/game/users/{userId}/characters/{characterId}")
4. public String characterInfo(@PathVariable String userId, @PathVariable int characterId,ModelMap model) {
5. model.addAttribute("userId",userId);
6. model.addAttribute("characterId",characterId);
7. ...
8. }

```

- RequestMapping 어노테이션에 변수를 포함하고 있다. 이들변수는 @PathVariable 어노테이션이 적용된 동일한 이름을 갖는 파라미터에 매칭된다.
- 즉 /game/usres/ezsnote/characters/sorcerer 하면 userId / characterId에 ezsnotes , sorcerer 가 들어간다.
- 만약 @PathVariable 어노테이션에 값을 주면 변수이름을 동일하게 하지 않고 지정할 수 있게 된다.

# @PathVariable과 URI 템플릿 변수

251

## 2. 추가 설정방법

- @Controller@RequestMapping("/game/users/{userId}")
- public class CharacterInfoController {
- @RequestMapping("/characters/{characterId}")
- public String characterInfo(@PathVariable String userId,
- @PathVariable int characterId, ModelMap model) {
- model.addAttribute("userId", userId);
- model.addAttribute("characterId", characterId);
- return "game/character/info";
- }
- }
- 위의 코드처럼 클래스와 메소드에 각각 RequestMapping을 걸어놓게 되면?
- 메소드에 적용한 @RequestMapping 어노테이션의 값은 클래스에 적용한 @RequestMapping 어노테이션의 값을 기본경로로 쓰게 된다.
- 메소드에 적용된 어노테이션의 값은 /characters/{characterId} 인데 실제 매칭되는 값은 클래스에 적용된 어노테이션의 값을 포함한 /game/users/{userId}/characters/{characterId} 가 된다.

## 7.4.4 컨트롤러 메서드의 리턴 타입

252

- ModelAndView
- Model
- Map
- String
- View 객체
- void
- @ResponseBody
- ResponseEntity<?> or ReponseEntity<?>

## 7.4.5 컨트롤러 자동스캔

- @Controller 어노테이션이 적용된 컨트롤러 클래스를 자동으로 로딩할 수 있다.
- context 네임스페이스 지정되어야 함
- <context:component-scan base-package="com.myweb.controller" />

- 컨트롤러 메서드는 처리 결과를 보여줄 뷰 이름이나 View 객체를 리턴하고 DispatcherServlet은 뷰 이름이나 View 객체를 이용해서 뷰를 생성하게 된다.

## 7.5.1 뷰 이름 지정

- mav.setViewName("list");
  - ▣ 리턴타입이 ModelAndView
- return "list"
  - ▣ 리턴타입이 String
- 뷰 이름 자동 지정
  - ▣ 리턴 타입이 Model이나 Map인 경우
  - ▣ 리턴 타입이 void 이면서 ServletResponse 또는 HttpServletResponse 타입의 파라미터가 없는 경우
  - ▣ 요청 URI에서 맨 앞 /와 확장자를 제외한 나머지 부분이 사용
  - ▣ /search/game2.do -> search/game2
- 리다이렉트
  - ▣ 뷰 이름에 redirect: 접두어를 붙이면 리다이렉트 됨

```
@RequestMapping("/empinsert.do")
public String empInsert(ModelMap map, EmpVo vo) {
 try {
 dao.insertEmp(vo);
 return "redirect:/emplist.do";
 } catch(Exception e) {
 map.addAttribute("msg", e.getMessage());
 return "empform";
 }
}
```

## 7.6 모델 생성

- @RequestMapping 어노테이션이 적용된 메서드의 파라미터나 리턴 타입으로 ModelAndView, Model, ModelMap, Map, 커맨드 객체 등을 이용해서 모델을 뷰에 전달하게 된다.

## 7.6.1 뷰에 전달되는 모델 데이터

257

- @RequestMapping 메서드가 ModelAndView, Model, Map을 리턴하는 경우 모델 데이터가 뷰에 전달된다.
- 추가로 다음 항목도 뷰에 함께 전달된다.
  - ▣ 커맨드 객체
  - ▣ @ModelAttribute 가 적용된 메서드가 리턴한 객체
  - ▣ 메서드의 Map, Model, ModelMap 타입 파라미터를 통해 설정된 모델

## 7.6.2 Map, Model, ModelMap을 통해 모델을 설정

258

- Map, Model, ModelMap을 통해 모델을 설정하기 위해서는 파라미터로 들어온 객체에 메서드를 통해 값을 지정하거나, 인터페이스를 구현한 클래스의 객체를 생성해서 리턴한다.
- Model, ModelMap 인터페이스의 주요 메서드
  - ▣ Model addAttribute(String name, Object value)
  - ▣ Model addAttribute(Object value)
  - ▣ Model addAllAttributes(Collection<?> values)
  - ▣ Model addAllAttributes(Map<String, ?> attributes)
  - ▣ Model mergeAttributes(Map<String, ?> attributes)
  - ▣ boolean containsAttributes(String name)
- ModelAndView는 객체를 생성한 다음 setViewName() 메서드로 뷰를 지정하며, addObject() 메서드로 뷰에 전달할 값을 추가한다.

## 7.6.3 @ModelAttribute

- `@RequestMapping` 이 적용되지 않은 별도 메서드로 모델에 추가될 객체를 생성
- 커맨드 객체의 초기화 작업을 수행

- org.springframework.validation.Validator 인터페이스
  - ▣ 객체 검증에 사용됨
- org.springframework.validation.Errors 인터페이스
  - ▣ 유효성 검증 결과를 저장할 때 사용
- org.springframework.validation.BindResult 인터페이스
  - ▣ Errors 인터페이스의 하위 인터페이스
  - ▣ 폼 값을 커맨드 객체에 바인딩 한 결과를 저장하고 여러 코드로부터 여러 메시지를 가져옴
- Validator 인터페이스를 이용해 폼 값을 저장한 커맨드 객체의 유효성 여부를 검사하고, 입력한 값이 유효하지 않은 경우 Errors에 여러 메시지를 저장해서 뷰에 전달하게 된다.

## 7.7.1 Validator 인터페이스

- boolean supports(Class<?> clazz)
  - ▣ Validator가 해당 클래스에 대한 값 검증을 지원하는지의 여부를 리턴
- void validate(Object target, Errors errors)
  - ▣ target 객체에 대한 검증을 실행한다. 검증 결과 문제가 있을 경우 errors 객체에 어떤 문제인지에 대한 정보를 저장한다.
- 컨트롤러 클래스에는...
  - ▣ @RequestMapping 어노테이션 메서드에서 커맨드 객체 다음 파라미터로 BindResult 타입이나 Errors 타입의 파라미터를 추가(사이에 다른 타입 들어가면 안 됨)
    - @RequestMapping(method=RequestMethod.POST)
    - public String submit(@Valid EmpVo emp, BindingResult result) {
  - ▣ Errors.hasErrors() 메서드를 이용해 에러가 존재하는지 확인하고, 에러가 존재할 경우 알맞은 처리를 수행할 수 있다.
    - if(result.hasErrors()) {
    - //에러처리
    - }

## 7.7.2 Errors 인터페이스

- 유효성 검증 결과를 저장할 때 사용
- Errors 인터페이스의 메서드
  - ▣ void reject(String errorCode)
    - 전체 객체에 대한 글로벌 에러코드를 추가한다.
  - ▣ void reject(String errorCode, String defaultMessage)
    - 전체 객체에 대한 글로벌 에러코드를 추가한다.
    - 에러 코드에 대한 메시지가 존재하지 않을 경우 defaultMessage를 사용한다.
  - ▣ void reject(String errorCode, Object[] errorArgs, String defaultMessage)
    - 전체 객체에 대한 글로벌 에러코드를 추가한다.
    - 메시지 인자로 errorArgs를 전달한다.
    - 에러 코드에 대한 메시지가 존재하지 않을 경우 defaultMessage를 사용한다.
  - ▣ void rejectValue(String field, String errorCode)
    - 필드에 대한 에러코드를 추가한다.
  - ▣ void rejectValue(String field, String errorCode , String defaultMessage)
    - 필드에 대한 에러코드를 추가한다.
    - 에러코드에 대한 메시지가 존재하지 않을 경우 defaultMessage를 사용한다.
  - ▣ void rejectValue(String field, String errorCode , Object[] errorArgs, String defaultMessage)
    - 필드에 대한 에러코드를 추가한다.
    - 메시지 인자로 errorArgs를 전달한다.
    - 에러코드에 대한 메시지가 존재하지 않을 경우 defaultMessage를 사용한다.

## 7.7.3 DefaultMessageCodesResolver

- MessageSource로부터 에러 코드에 해당하는 메시지를 로딩한다.
- DefaultMessageCodesResolver 를 이용하여 에러 메시지를 생성하려면 MessageSource를 빈 객체로 등록해 주어야 한다.
- ```
<bean id="messageSource"
      class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
      <list>
        <value>message.validation</value>
      </list>
    </property>
  </bean>
```

7.7.4 뷰가 JSP일 경우 에러 메시지 설정

- <spring:hasBindErrors> 태그 이용
 - ▣ <%@ taglib uri="http://www.springframework.org/tags" prefix="spring" %>
 - ▣ <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
 - ▣ <**spring:hasBindErrors name="command"** />
 - ▣ <form:errors path="command" />
 - ▣ <form:errors path="command.userid" />

- <form:form> 태그 이용
 - ▣ <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
 - ▣ <**form:form commandName="command"** />
 - ▣ <form:errors element="div" />
 - ▣ <form:errors path="userid" />

7.7.5 @Valid @InitBinder

- 폼의 입력 값 검증을 위해 커멘드 객체에 `@Valid` 아노테이션을 지정할 수 있다.
 - ▣ `@RequestMapping(method=RequestMethod.POST)`
 - ▣ `public String submit(@Valid EmpVo emp, BindingResult result) {`
 - ▣ `...`
 - ▣ `}`
- 컨트롤러에서 `@InitBinder` 아노테이션이 지정된 메서드에서 데이터 타입 변환 처리 및 Validator를 지정할 수 있다.
 - ▣ `@InitBinder`
 - ▣ `protected void initBinder(WebDataBinder binder) {`
 - ▣ `binder.setValidator(new EmpValidator());`
 - ▣ `}`
- 커멘드 클래스를 `BaseObjectBean` 클래스를 상속받도록 하고 필드에 `@Size`, `@NotNull` 아노테이션을 이용해 유효성 설정도 가능하다.
 - ▣ `validation-api.jar` 파일이 필요함
 - ▣ 설정파일에
 - `<mvc:annotation-driven validator="validator"/>`
 - `<beans:bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean"/>`
 - ▣ 커맨트클래스에
 - `@Size(min=5, max=20, message="아이디는 5자이상 20자 미만이어야 합니다.")`
 - `@NotNull`
 - `private String userid;`

7.7.6 @ExceptionHandler를 이용한 예외처리

- AnnotationMethodHandlerExceptionResolver 클래스는 @Controller 어노테이션이 적용된 클래스에서 @ExceptionHandler 어노테이션이 적용된 메서드를 이용해서 예외 처리를 한다.
- **@ExceptionHandler(NullPointerException.class)**
- ```
public String handleNullPointerException(NullPointerException ex) {
```
- ```
    return "error/nullException";
```
- ```
}
```
- ExceptionHandler 메서드가 가질 수 있는 파라미터 타입
  - HttpServletRequest, HttpServletResponse, HttpSession
  - Locale
  - InputStream/Reader, OutputStream/Writer
  - 예외 타입
- ExceptionHandler 메서드가 가질 수 있는 리턴 타입
  - ModelAndView, Model, Map, View
  - String
  - void

## 7.8 파일 업로드 처리

- 스프링은 업로드 한 파일을 커맨드 객체에 저장할 수 있는 기능을 제공하고 있다.
- 파일을 업로드 할 때에는 `<form>` 태그의 `enctype` 속성의 값을 "multipart/form-data"로 지정해야 함
  - ▣ `<form method="post" enctype="multipart/form-data">`
  - ▣ 제목 : `<input type="text" name="title" /><br>`
  - ▣ 파일 : `<input type="file" name="file" />`
  - ▣ `</form>`

## 7.8.1 MultipartResolver 설정

- Multipart 지원 기능을 사용하기 위해 MultipartResolver를 스프링 설정 파일에 등록해야 한다.
- MultipartResolver는 CommonsMultipartResolver이다.
- CommonsMultipartResolver는 Commons FileUpload API를 이용해서 Multipart를 처리해 준다.
- CommonsMultipartResolver를 MultipartResolver를 사용하려면 빈 이름으로 multipartResolver를 사용해서 등록하면 된다.
  - ▣ <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
  - ▣ </bean>

## 7.8.2 CommonsMultipartResolver 클래스의 프로퍼티

- <bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
- <property name="defaultEncoding" value="utf-8"/>
- <property name="maxUploadSize" value="10000000"/>
- <property name="uploadTempDir" value="uploadtemp"/>
- </bean>

WebContent에 uploadtemp 폴더를 생성해 놓으세요. 그리고... 아래 폴더도...

C:\Java\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\프로젝트이름\uploadtemp

| 프로퍼티            | 설명                                                                                     | 기본값              |
|-----------------|----------------------------------------------------------------------------------------|------------------|
| defalutEncoding | 기본 인코딩을 설정한다. 만약 필터를 통해서 request.setCharacterEncoding()을 통해 인코딩을 설정한 경우, 해당 인코딩을 사용한다. | ISO-8859-1       |
| maxUploadSize   | 허용되는 최대 크기를 바이트 단위로 지정한다. -1인 경우 제한을 두지 않는다.                                           | -1               |
| maxInMemorySize | 업로드 한 내용을 메모리에 저장할 수 있는 최대 크기를 바이트 단위로 지정한다. 지정한 크기를 넘어서면 임시 디렉터리에 파일로 저장한다.           | 10240            |
| uploadTempDir   | 업로드 한 파일이 저장될 임시 디렉터리 경로                                                               | 서블릿 컨테이너 임시 디렉터리 |

## 7.8.3 @RequestParam 어노테이션을 이용한 업로드 파일 접근

270

- MultipartFile 인터페이스를 이용해 업로드 한 파일의 이름, 실제 데이터 크기, 파일 크기 등을 구할 수 있다.
  - ▣ @RequestMapping(value="/upload.do", method=RequestMethod.POST)
  - ▣ public String writeArticle(@RequestParam("file") MultipartFile report) {
  - ▣ ...
  - ▣ }
- 폼은 enctype이 설정되어야 하며, input 태입의 type 속성을 file로 한다.
  - ▣ <form action="/upload.do" method="post" enctype="multipart/form-data">
  - ▣ 제목 : <input type="text" name="title" /><br>
  - ▣ 파일 : <input type="file" name="file" /><br>
  - ▣ <input type="submit" value="Upload" />
  - ▣ </form>

## 7.8.4 MultipartFile 인터페이스의 주요 기능

| 메서드                                                                        | 설명                                                                                                                                          |
|----------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| String getName()                                                           | 입력 폼의 파라미터 이름을 리턴한다.                                                                                                                        |
| String getOriginalFilename()                                               | 업로드 한 파일의 이름을 리턴한다. 업로드 할 파일을 선택하지 않은 경우 빈 문자열 ("")을 리턴한다.                                                                                  |
| String getContentType()                                                    | 업로드 한 파일의 컨텐트 타입을 리턴한다. 타입이 정의되지 않았거나 업로드 할 파일을 선택하지 않은 경우 null을 리턴 한다.                                                                     |
| boolean isEmpty()                                                          | 업로드 한 파일이 내용을 갖고 있지 않거나 또는 업로드 한 파일을 선택하지 않은 경우 true를 리턴한다.(if(multipartFile.isEmpty())                                                     |
| long getSize()                                                             | 파일의 크기를 구한다.                                                                                                                                |
| byte[] getBytes()<br>throws IOException                                    | 파일의 내용을 byte 배열로 구한다. byte 배열을 파일/DB/네트워크 등으로 전송할 때 사용한다.(byte[] fileData = multipartFile.getBytes());                                      |
| InputStream getInputStream()<br>throws IOException                         | 파일을 내용을 읽어올 수 있는 InputStream을 구한다.                                                                                                          |
| void transferTo(File dest)<br>throws IOException,<br>IllegalStateException | 업로드 한 파일의 내용을 대상 파일(dest)에 저장한다. 만약, 대상 파일이 이미 존재하면 삭제한 뒤 저장한다.<br><pre>File file=new File(fileName); multipartFile.transferTo(file);</pre> |

# 컨트롤러 예

272

```

@RequestMapping(value="/board/upload.do", method=RequestMethod.POST)
public String fileUpload(@RequestParam("file") CommonsMultipartFile[] file, HttpServletRequest request) {
 String title = request.getParameter("title");
 System.out.println(title);
 String rootdir = request.getSession().getServletContext().getRealPath("/");
 System.out.println(rootdir);
 System.out.println(file[0].getOriginalFilename());

 File dir = new File(rootdir);
 //File dir = new File(rootdir + "upload");

 File uploadfile = new File(dir, file[0].getOriginalFilename());
 //File uploadfile = isExist(dir, file[0].getOriginalFilename());
 try {
 file[0].transferTo(uploadfile);
 } catch (IllegalStateException e) {
 e.printStackTrace();
 return "/error/error";
 } catch (IOException e) {
 e.printStackTrace();
 return "/error/error";
 }
 return "redirect:/index.jsp";
}

```

```

private File isExist(File dir, String filename) {
 File file = new File(dir, filename);
 if(file.exists()) {
 int i = filename.lastIndexOf(".");
 String firstname = filename.substring(0, i) + "0";
 String lastname = filename.substring(i);
 file = isExist(dir, firstname+lastname);
 }
 return file;
}//파일이 이미 존재할 때...

```

## 7.8.5 MultipartHttpServletRequest를 이용한 업로드 파일 접근

Spring Web MVC

273

- @RequestMapping 어노테이션이 지정된 메서드의 인자로 MultipartHttpServletRequest 타입을 정의한다.
- MultipartHttpServletRequest 인터페이스는 HttpServletRequest와 MultipartRequest 인터페이스의 하위 인터페이스이다.

| 메서드                                       | 설명                                             |
|-------------------------------------------|------------------------------------------------|
| Iterator<String> getFileNames()           | 업로드 된 파일의 이름 목록을 제공한다.                         |
| MultipartFile getFile(String name)        | 파라미터 이름이 name인 업로드 파일 정보를 구한다.                 |
| List<MultipartFile> getFiles(String name) | 파라미터 이름이 name인 업로드 파일 정보 목록을 구한다.              |
| Map<String, MultipartFile> getFileMap()   | 파라미터 이름을 키로, 파라미터에 해당하는 파일정보를 값으로 하는 Map을 구한다. |

## 7.8.6 커맨드 객체를 통한 업로드 파일 접근

- 파라미터와 동일한 이름의 MultipartFile 타입 프로퍼티를 추가해 주기만 하면 된다.
- files 타입을 MultipartFile[] 타입으로 선언할 수 있으나 그렇게 하면 스프링 의존적이 된다.
  - ▣ public class BoardCommand {
  - ▣     private String writerid;
  - ▣     private String subject;
  - ▣     private String content;
  - ▣     private byte[] files;
  - ▣
  - ▣     public byte[] getFiles() {
  - ▣         return files;
  - ▣     }
  - ▣     public void setFiles(byte[] files) {
  - ▣         this.files = files;
  - ▣     }
  - ▣ }
- byte[]을 사용하기 위해서는 initBinder 메서드에서 byte[] 타입을 위한 Editor로 ByteArrayMultipartFileEditor로 등록해야 한다.
  - ▣ protected void initBinder(HttpServletRequest request, ServletRequestDataBinder binder) throws ServletException {
  - ▣     binder.registerCustomEditor(byte[].class, new ByteArrayMultipartFileEditor());
  - ▣ }

## 7.9 Tiles 2를 이용한 View 레이아웃 템플릿 처리

275

- Tiles 2 이용 레이아웃 템플릿 처리
- Tiles는 sitemesh와 함께 가장 유명한 템플릿 프레임워크이다.
- Tiles는 Composite View 패턴을 사용하고 있다. 그래서 sitemesh에서 사용하는 Decorator 패턴 보다 성능이 더 좋다. 그리고 실행시 설정을 변경할 수도 있는 장점이 있다.
- Tiles와 같은 템플릿 라이브러리를 사용해서 레이아웃을 처리하면 뷰 관련 코드에서 레이아웃을 처리하기 위한 코드의 중복을 제거할 수 있는 장점이 있다.

- 스프링은 널리 사용되고 있는 템플릿 라이브러리인 Tiles 2 버전을 지원하고 있다.
- Tiles 2 연동을 위한 모듈을 클래스 패스에 추가해야 한다.
  - ▣ <http://tiles.apache.org/>에서 다운로드
  - ▣ tiles-2.2.2-bin.zip 다운로드
    - tiles-api-2.2.2.jar
    - tiles-core-2.2.2.jar
    - tiles-jsp-2.2.2.jar
    - tiles-servlet-2.2.2.jar
    - tiles-template-2.2.2.jar
  - ▣ <http://commons.apache.org>에서 다운로드
    - commons-beanutils-1.8.3.jar
    - commons-digester-2.1.jar(digester3 다운받으면 안됨)
    - commons-logging-1.1.1.jar
  - ▣ <http://www.slf4j.org/>에서 다운로드
    - 별도의 파라미터 포매팅 개념을 도입한 "Parameterized logging method."
    - slf4j-api-1.6.4.jar
    - slf4j-simple-1.6.4.jar
- Tiles 2의 뷰 정의 노가다(?)를 없애기 위해 Dynamic Tiles Spring MVC Module 을 다운로드 한다.
  - ▣ <http://www.springbyexample.org>에서 다운로드
  - ▣ org.springframework.dynamic.tiles2-1.2.jar

## 7.9.2 스프링 설정 파일

- TilesConfigurer 를 이용하여 Tiles2 레이아웃 설정 파일 명시

```
<bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
 <property name="definitions">
 <list>
 <value>/WEB-INF/tiles2-defs/templates.xml</value>
 </list>
 </property>
</bean>
```

- UrlBasedViewResolver의 viewClass프로퍼티를 TilesView로 지정

```
<bean id="tilesUrlBasedViewResolver"
 class="org.springframework.web.servlet.view.tiles2.TilesUrlBasedViewResolver">
 <property name="viewClass"
 value="org.springframework.web.servlet.view.tiles2.DynamicTilesView" />
 <property name="prefix" value="/WEB-INF/view/" />
 <property name="suffix" value=".jsp" />
 <property name="tilesDefinitionName" value="default" />
 <property name="tilesBodyAttributeName" value="body" />
 <property name="tilesDefinitionDelimiter" value="." />
 <property name="order" value="1" />
</bean>
```

기존의 viewResolver 에 아래 추가하여 우선순위를 낮춘다.  
 <property name="order" value="2" />

## 7.9.3 Tiles 2 설정 파일(/WEB-INF/tiles2-defs/templates.xml)

Spring Web MVC

278

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration 2.1//EN"
"http://tiles.apache.org/dtds/tiles-config_2_1.dtd">
```

} 이거도 입력해야 합니다. ㅠ.ㅠ

```
<tiles-definitions>
 <definition name=".default" template="/WEB-INF/tiles2-defs/templates/layout.jsp">
 <put-attribute name="title" value="Welcome" type="string" />
 <put-attribute name="header" value="/WEB-INF/tiles2-defs/templates/header.jsp" />
 <put-attribute name="footer" value="/WEB-INF/tiles2-defs/templates/footer.jsp" />
 <put-attribute name="menu" value="/WEB-INF/tiles2-defs/templates/menu.jsp" />
 <put-attribute name="body" value="/WEB-INF/view/temp/index.jsp" />
 </definition>

 <definition name=".template2" template="/WEB-INF/tiles2-defs/templates2/layout.jsp">
 <put-attribute name="title" value="template2" type="string" />
 <put-attribute name="header" value="/WEB-INF/tiles2-defs/templates2/header.jsp" />
 <put-attribute name="footer" value="/WEB-INF/tiles2-defs/templates2/footer.jsp" />
 <put-attribute name="menu" value="/WEB-INF/tiles2-defs/templates2/menu.jsp" />
 <put-attribute name="body" value="/WEB-INF/view/temp/index.jsp" />
 </definition>
</tiles-definitions>
```

## 7.9.4 layout.jsp

279

```
<%@ page contentType="text/html; charset=EUC-KR" pageEncoding="EUC-KR"%>
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title><tiles:getAsString name="title"/></title>
</head>
<body>
<tiles:insertAttribute name="header"/>
<hr/>
<tiles:insertAttribute name="body"/>
<hr/>
<tiles:insertAttribute name="footer"/>
</body>
</html>
```

## 7.9.5 컨트롤러 파일

```
□ @Controller
□ public class IndexController {
□
□ @RequestMapping(value="/index.do", method=RequestMethod.GET)
□ public String index() {
□ return "index";
□ }
□ }
```

index.jsp 파일을 /WEB-INF/view/ 폴더에 작성.

컨트롤러에서 /index.do 의 리턴 값이 index 이므로 ViewResolver 설정대로 파일을 작성해야 한다.

## 7.9.6 Welcome 페이지 처리

281

- /WEB-INF/web.xml
  - ▣ <welcome-file-list>
  - ▣    <welcome-file>**index.do**</welcome-file>
  - ▣ </welcome-file-list>
- 컨트롤러
  - ▣ @RequestMapping(**value="/index.do"**, method=RequestMethod.GET)
  - ▣ public String index() {
  - ▣    return "index";
  - ▣ }
- 파일
  - ▣ 컨텍스트 루트에 **index.do** 빙 페이지 하나 생성
- 모든 페이지는 컨트롤러를 통과해야 함

## 7.9.7 컨트롤러 파일에서 템플릿 변경하려면...

282

- @Controller 파일에서 필드 선언
  - ▣ @Autowired
  - ▣ private TilesUrlBasedViewResolver tilesUrlBasedViewResolver;
- 템플릿을 변경하고 싶으면 컨트롤러 메서드(@RequestMapping 메서드)에서
  - ▣ tilesUrlBasedViewResolver.clearCache();
    - clearCache() 안해주면 500에러
  - ▣ tilesUrlBasedViewResolver.setTilesDefinitionName("default2");
  - ▣ return "xxx"; //ModelAndView로 넘겨도 됨

## 8. Spring을 이용한 웹 어플리케이션

1. 개발 환경 설정
2. 설정파일
3. 코드작성
4. 조회
5. 폼을 이용한 데이터 저장

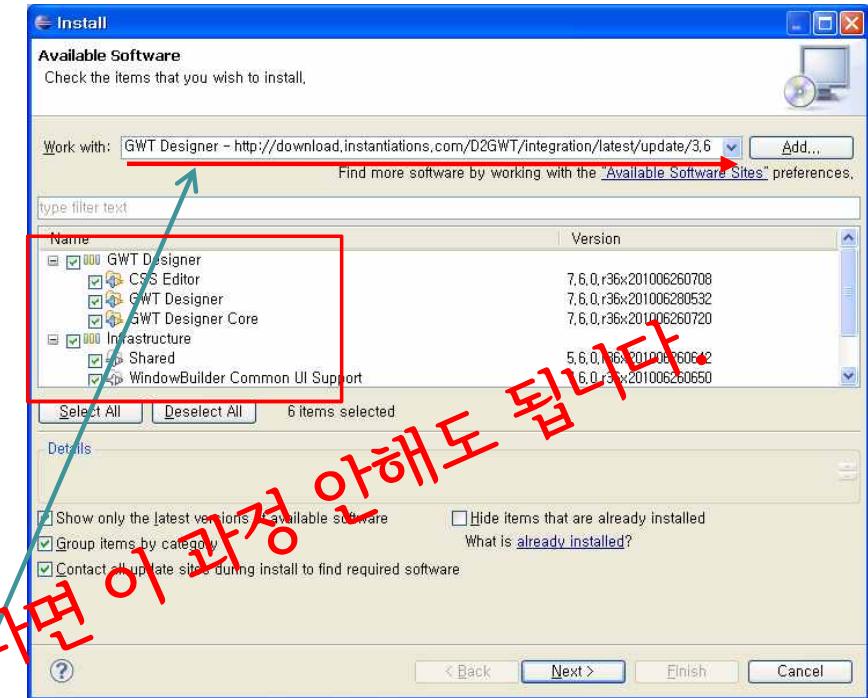
# 8.1 개발환경설정

- 서버 설치
- 서블릿 한글 인코딩 설정
- 라이브러리 추가
- 스프링 MVC 이용 웹 App 개발 과정 안내
- Sample Table
- Sample Data

## 8.1.1 서버 설치

285

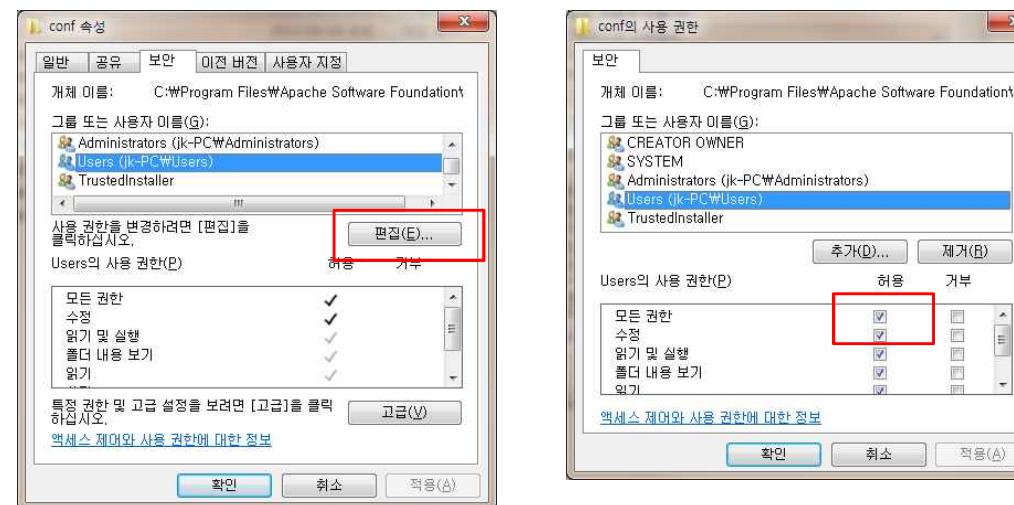
- 서버 설치
  - ▣ 다운로드
    - <http://tomcat.apache.org/>
    - 6.0/7.0 다운로드
  - ▣ 설치
    - Zip 파일은 압축 풀기만 하면 됨
  - ▣ 환경변수 설정
    - JAVA\_HOME
    - C:/Program Files/Java/jdk1.6.0\_20
- 웹 프로젝트 작성을 위한 IDE 설치
  - ▣ Window Builder
    - GWT Builder 업데이트사이트 이용
    - <http://download.instantiations.com/D2GWT/integration/latest/update/3.6/>
    - 모두 체크하고 업데이트



## 8.1.2 서블릿 한글 인코딩

286

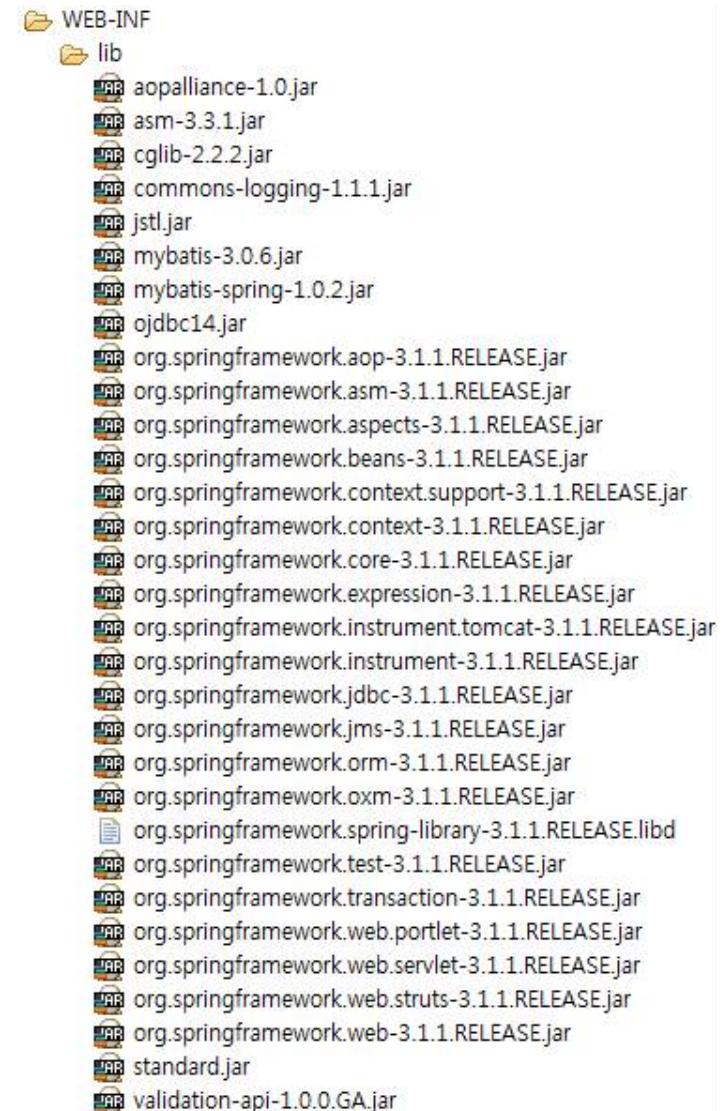
- GET방식에서 한글 깨짐 현상
  - ▣ tomcat/conf/server.xml 파일에서
  - ▣ <Connector port="8080" ... **URIEncoding="euc-kr" />**
  - ▣ 빨간 부분 추가
  
- Windows 7에서 “액세스가 거부되었습니다.” 라고 뜨면...
  - ▣ conf 폴더 속성에서 사용자 쓰기 권한 설정해 줘야 함.
    - 폴더 속성 -> 보안 탭 -> 편집 -> Users(xx-pc\Users) 에서
    - Users의 사용 권한에 모든권한 주세요.



## 8.1.3 프로젝트 생성 및 라이브러리 추가

287

- Dynamic Web Project 만들기
  - ▣ File > New > Other > Web > Dynamic Web Project
  
- Jar 파일 삽입하기(Import)
  - ▣ WEB-INF/lib 폴더에
    - Spring jar 파일들
    - commons-logging-1.1.1.jar
    - cglib & asm 모듈
    - aop 모듈
    - mybatis.jar & mybatis-spring.jar
    - standard.jar
    - validation-api.jar
    - jstl.jar
    - jdbc driver
  - ▣ 빌드패스에 추가
    - servlet-api.jar
      - tomcat6/lib 에 있음
      - 서블릿 컴파일하기 위해서



## 8.1.4 스프링 MVC 이용 웹 App 개발 과정

288

- DispatcherServlet을 web.xml 파일에 설정
  - ▣ 모든 요청이 디스패쳐 서블릿으로 들어오도록
- 요청 URL과 컨트롤러의 맵핑 방식을 설정
  - ▣ HandlerMapping을 이용
- 컨트롤러 작성
  - ▣ 클라이언트의 요청을 처리
- ViewResolver 설정
  - ▣ 어떤 뷰를 이용하여 컨트롤러의 처리 결과 응답 화면을 생성할지 결정
- 뷰 코드 작성
  - ▣ JSP 이용

## 8.1.5 Sample table(emp & dept)

289

```
create table dept(
 deptno int(2) primary key,
 dname varchar(14),
 loc varchar(13)
);
```

```
create table emp (
 empno int(4) primary key,
 ename varchar(10),
 job varchar(9),
 mgr int(4),
 hiredate date,
 sal float(6,1),
 comm float(6,1),
 deptno int(2)
);
```

```
ALTER TABLE emp ADD FOREIGN KEY (deptno) REFERENCES dept(deptno);
```

\* MySql에서 사용자 생성  
create database mydb;  
use mysql  
create user 'user'@'%' identified by 'user123';  
grant all privileges on \*.\* to 'user'@'%';  
flush privileges;  
\* User로 다시 접속해서  
use mydb;

## 8.1.6 Sample data(emp & dept)

```
insert into dept values(10, 'ACCOUNTING', 'NEW YORK');
insert into dept values(20, 'RESEARCH', 'DALLAS');
insert into dept values(30, 'SALES', 'CHICAGO');
insert into dept values(40, 'OPERATIONS', 'BOSTON');

insert into emp values(7369, 'SMITH', 'CLERK', 7902, '1980-12-17', 800, NULL, 20);
insert into emp values(7499, 'ALLEN', 'SALESMAN', 7698, '1981-02-20', 1600, 300, 30);
insert into emp values(7521, 'WARD', 'SALESMAN', 7698, '1981-02-22', 1250, 500, 30);
insert into emp values(7566, 'JONES', 'MANAGER', 7839, '1981-04-02', 2975, NULL, 20);
insert into emp values(7654, 'MARTIN', 'SALESMAN', 7698, '1981-09-28', 1250, 1400, 30);
insert into emp values(7698, 'BLAKE', 'MANAGER', 7839, '1981-05-01', 2850, NULL, 30);
insert into emp values(7782, 'CLARK', 'MANAGER', 7839, '1981-06-09', 2450, NULL, 10);
insert into emp values(7788, 'SCOTT', 'ANALYST', 7566, '1987-04-19', 3000, NULL, 20);
insert into emp values(7839, 'KING', 'PRESIDENT', NULL, '1981-11-17', 5000, NULL, 10);
insert into emp values(7844, 'TURNER', 'SALESMAN', 7698, '1981-09-08', 1500, 0, 30);
insert into emp values(7876, 'ADAMS', 'CLERK', 7788, '1987-05-23', 1100, NULL, 20);
insert into emp values(7900, 'JAMES', 'CLERK', 7698, '1981-12-03', 950, NULL, 30);
insert into emp values(7902, 'FORD', 'ANALYST', 7566, '1981-12-03', 3000, NULL, 20);
insert into emp values(7934, 'MILLER', 'CLERK', 7782, '1982-01-23', 1300, NULL, 10);
```

## 8.2 설정파일 작업

291

- /WEB-INF/web.xml
- /WEB-INF/dispatcher-servlet.xml
- /WEB-INF/jdbc.properties

- <?xml version="1.0" encoding="UTF-8"?>
  - <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd" id="WebApp\_ID" version="2.5">
  - <display-name>EmpSpringMybatisWebPrj</display-name>
  - <welcome-file-list>
  - <welcome-file>index.jsp</welcome-file>
  - </welcome-file-list>
  - <servlet>
  - <servlet-name>dispatcher</servlet-name>
  - <servlet-class>
  - org.springframework.web.servlet.DispatcherServlet
  - </servlet-class>
  - </servlet>
  - <servlet-mapping>
  - <servlet-name>dispatcher</servlet-name>
  - <url-pattern>\*.do</url-pattern>
  - </servlet-mapping>
  - </web-app>
  
  - /WEB-INF/ 디렉토리에 [서블릿이름]-servlet.xml 파일로부터 스프링 설정 정보를 읽어온다.
  - 위와 같이 했다면 설정파일은 /WEB-INF/dispatcher-servlet.xml 파일이 됨
- 설정파일을 다르게 할 경우 dispatcher 서블릿 설정부분에 init-param 추가
- 설정파일을 다르게 할 경우 dispatcher 서블릿 설정부분에 init-param 추가

## 8.2.2 인코딩 필터 설정(web.xml)

293

- 한글 처리가 되지 않아 수정 후 한글이 깨진다?
- web.xml에 필터를 추가. 모든 요청에 대해 적용(<url-pattern>/\*</url-pattern>)

```
<filter>
 <filter-name>encodingFilter</filter-name>
 <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
 <init-param>
 <param-name>encoding</param-name>
 <param-value>EUC-KR</param-value>
 </init-param>
</filter>
<filter-mapping>
 <filter-name>encodingFilter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
```

- File -> New -> Other -> Spring -> Spring Bean Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:aop="http://www.springframework.org/schema/aop"
 xmlns:context="http://www.springframework.org/schema/context"
 xmlns:jdbc="http://www.springframework.org/schema/jdbc"
 xmlns:p="http://www.springframework.org/schema/p"
 xmlns:tx="http://www.springframework.org/schema/tx"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd
 http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd
 http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-3.0.xsd
 http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

</beans>
```



- 스프링 MVC의 구성요소인 HandlerMapping, Controller, ViewResolver, View등의 빈을 설정.
- 설정파일에 DB관련 DataSource 빈으로 등록
- 사용할 서비스 객체 빈으로 등록
- ViewResolver등록 (<bean>의 id속성에 등록)

## 8.2.4 DataSource 설정(/WEB-INF/dispatcher-servlet.xml)

Spring을 이용한 웹 어플리케이션

295

```
□ <?xml version="1.0" encoding="UTF-8"?>
□ <beans ...>

□ <context:property-placeholder location="/WEB-INF/jdbc.properties"/>

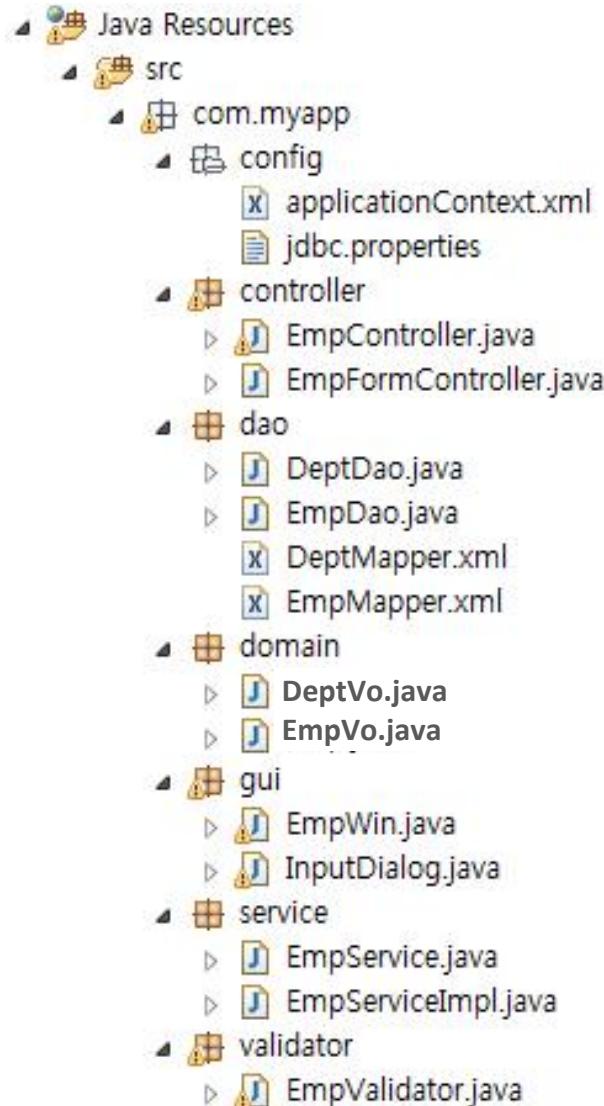
□ <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
 <property name="driverClassName" value="${driver}" />
 <property name="url" value="${url}" />
 <property name="username" value="${username}" />
 <property name="password" value="${password}" />
 </bean>

□ </beans>
```

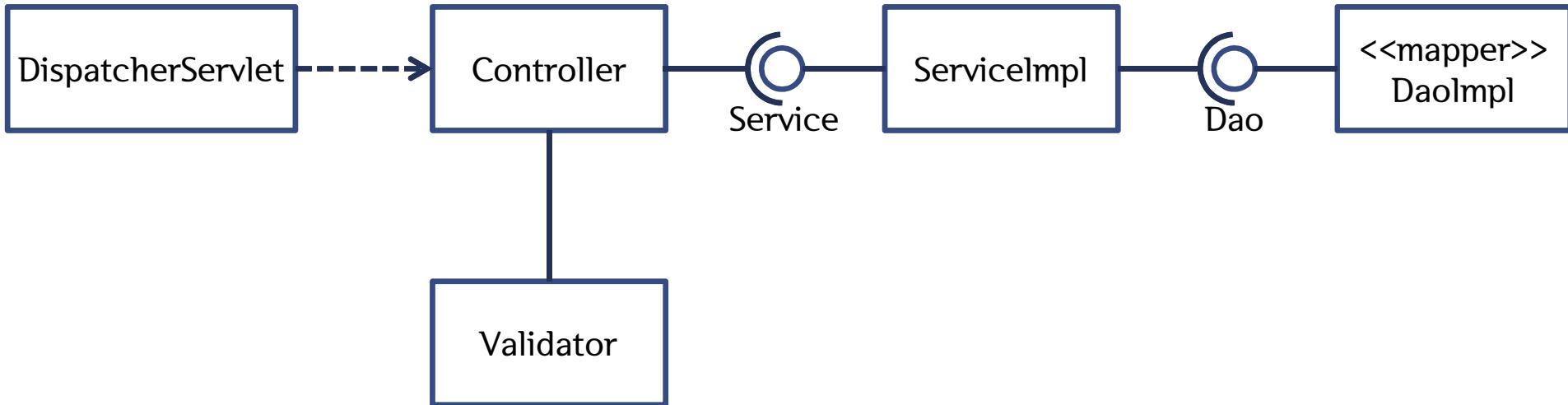
/WEB-INF/jdbc.properties

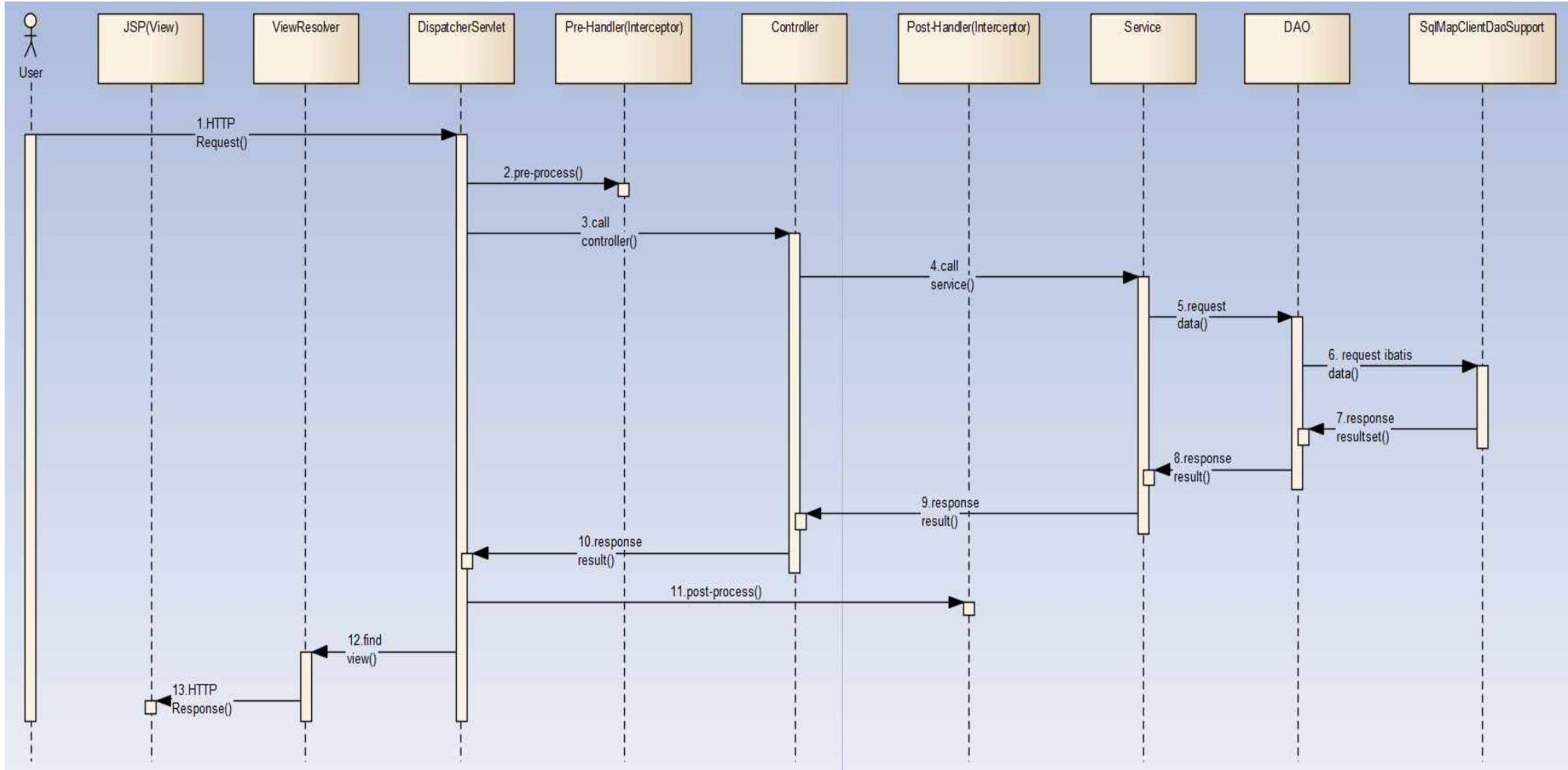
```
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@127.0.0.1:1521:ora11g
username=scott
password=tiger
```

- controller
  - ▣ 컨트롤러 클래스
- dao
  - ▣ Dao 인터페이스
  - ▣ Dao 메퍼
- domain
  - ▣ Value Object
- service
  - ▣ 서비스 인터페이스/클래스
- validator
  - ▣ 폼 검증기
  
- config와 gui의 파일들은 윈도우용 애플리케이션에서 사용되는 파일임



## 8.3.1 시스템 구성도





```
package com.myapp.domain;

import java.io.Serializable;

public class DeptVo implements Serializable {
 private static final long serialVersionUID = 212765432L;

 private int deptno;
 private String dname;
 private String loc;

 //기본생성자

 //setter/getter
}
```

```
package com.myapp.domain;

import java.io.Serializable;
import java.sql.Date;
import java.util.List;

public class EmpVo implements Serializable {
 private static final long serialVersionUID = 237654222L;

 private int empno;
 private String ename;
 private String job;
 private int mgr;
 private Date hiredate;
 private String hiredateStr; //날짜 형식 변환을 위해 두었음
 private double sal;
 private double comm;
 private int deptno;
 private List<DeptVo> deptList; //formBacking때문에...
 private List<Integer> empnoList;//formBacking때문에...

 //기본 생성자

 //setter/getter 메서드
}
```

```
package com.myapp.dao;

import java.util.List;

import com.myapp.domain.DeptVo;

public interface DeptDao {
 List<Dept> getAllDepts();
 DeptVo getDept(int deptno);
}
```

```
package com.myapp.dao;

import java.util.List;

import org.apache.ibatis.annotations.Param;
import com.myapp.domain.EmpVo;

public interface EmpDao {
 Integer getAllEmpCount();
 EmpVo selectEmp(@Param("empno") int empno);
 List<Integer> getAllEmpno();
 void insertEmp(Emp emp);
 List<Emp> getAllEmps();
 String[] getColumnNames();
 void updateEmp(Emp emp);
 void deleteEmp(int empno);
}
```

```
<?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.myapp.dao.DeptDao">
 <select id="getAllDepts" resultType="DeptVo">
 select deptno, dname, loc from dept
 </select>
 <select id="getDept" parameterType="int" resultType="DeptVo">
 select deptno, dname, loc from dept where deptno = # ${deptno}
 </select>
</mapper>
```

## 8.3.4 Daolmpl[1/3] (EmpMapper.xml - resultMap 설정)

Spring을 이용한 웹 어플리케이션

302

```
<?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.myapp.dao.EmpDao">

 <resultMap type="com.myapp.domain.EmpVo" id="emp">
 <result property="empno" column="empno" javaType="int"/>
 <result property="ename" column="ename" javaType="string"/>
 <result property="job" column="job" javaType="string"/>
 <result property="mgr" column="mgr" javaType="int"/>
 <result property="hiredate" column="hiredate" javaType="java.sql.Date" jdbcType="DATE" />
 <result property="sal" column="sal" javaType="double"/>
 <result property="comm" column="comm" javaType="double"/>
 <result property="deptno" column="deptno" javaType="int"/>
 </resultMap>

 <!-- 다음 페이지에 있는 쿼리문 이곳에 넣어야 함 -->
</mapper>
```

```
<select id="getAllEmpCount" resultType="int">
 select count(*) from emp
</select>
```

모두 mapper 엘리먼트 안에 있어야 함

```
<select id="selectEmp" parameterType="int" resultType="com.myapp.domain.EmpVo">
 select empno, ename, job, mgr, hiredate,
 sal, comm, deptno from emp where empno = #{empno}
</select>
```

```
<select id="getAllEmpno" resultType="int">
 select empno from emp
</select>
```

SqlSessionFactory 빈 설정시  
*typeAliasesPackage*에 패키지명 설정하면 이곳에서 패키지 이름 생략 가능

```
<insert id="insertEmp" parameterType="EmpVo">
 insert into emp
 (empno, ename, job, mgr, hiredate, sal, comm, deptno)
 values
 (#{empno}, #{ename}, #{job}, #{mgr}, sysdate, #{sal}, #{comm}, #{deptno})
</insert>
```

```
<select id="getAllEmps" resultType="EmpVo">
 select * from emp
</select>

<update id="updateEmp" parameterType="EmpVo">
 update emp set ename=#{ename}, job=#{job}, mgr=#{mgr},
 hiredate=#{hiredate,jdbcType=DATE} ,
 sal=#{sal}, comm=#{comm}, deptno=#{deptno} where empno=#{empno}
</update>

<delete id="deleteEmp" parameterType="int">
 delete from emp where empno=#{empno}
</delete>

</mapper> <!-- 매퍼 루트 엘리먼트 닫아줌 -->
```

```
package com.myapp.service;

import java.util.List;

import com.myapp.domain.Dept;
import com.myapp.domain.Emp;

public interface EmpService {
 int getAllEmpCount() ;
 List<Dept> getAllDepts();
 EmpVo selectEmp(int empno);
 List<Integer> getAllEmpno();
 void insertEmp(Emp emp);
 List<Emp> getAllEmps();
 void updateEmp(Emp emp);
 void deleteEmp(int empno);
}
```

```
@Service("empService")
public class EmpServiceImpl {
 @Autowired
 private EmpDao empDao;
 @Autowired
 private DeptDao deptDao;

 public int getAllEmpCount() {
 return empDao.getAllEmpCount();
 }
 public List<Dept> getAllDepts() {
 return deptDao.getAllDepts();
 }

 public EmpVo selectEmp(int empno) {
 EmpVo emp = empDao.selectEmp(empno);
 return emp;
 }
}
```

```
public List<Integer> getAllEmpno() {
 return empDao.getAllEmpno();
}

@Transactional
public void insertEmp(Emp emp) {
 empDao.insertEmp(emp);
}

public List<Emp> getAllEmps() {
 return empDao.getAllEmps();
}

public void updateEmp(Emp emp) {
 empDao.updateEmp(emp);
}

public void deleteEmp(int empno) {
 empDao.deleteEmp(empno);
}
}//end EmpServiceImpl
```

```

package com.myapp.controller;

import javax.servlet.http.HttpServletRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.myapp.service.EmpService;

@Controller
public class EmpController {
 @Autowired
 private EmpService empService;

 @RequestMapping(value="/emp/count.do", method=RequestMethod.GET)
 public String getEmpCount(Model model, HttpServletRequest request) {
 System.out.println(request.getParameter("data"));
 model.addAttribute("count", empService.getAllEmpCount());
 return "emp/count";
 }
}

```

<context:component-scan base-package="com.myapp.service" />

**<bean id="empController" class="com.abc.controller.EmpController">**  
**<property name="empService" ref="empService" />**  
**</bean>**

**@Autowired는 타입으로 의존성 삽입함**  
**@Resource(name="empService")을 이용하면 빈의 이름으로 의존성 삽입함**

**<bean id="handler" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">**  
**<property name="mappings" >**  
**<props>**  
**<prop key="/emp/count.do">empController</prop>**  
**</props>**  
**</property>**  
**</bean>**

Controller URL Mapping 설정과 동일

- InternalResourceViewResolver 이용
- dispatcher-servlet.xml 에서 다음과 같이 설정 하고...

```
<bean id="viewRelolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
 <property name="prefix" value="/WEB-INF/view/" />
 <property name="suffix" value=".jsp" />
</bean>
```

→ 이렇게 하면 뷰를 직접 호출할 수 없겠죠?

- 컨트롤러에서 다음과 같이 뷰 이름을 지정했을 경우
  - ModelAndView mav = new ModelAndView("hello");
  - Return mav;
- InternalResourceViewResolver 가 사용하는 자원의 경로는
  - Context\_Path/WEB-INF/view/hello.jsp

## 8.3.9 빈 자동등록 및 Auto wiring(dispatcher-servlet.xml)

8. Spring을 이용한 웹 어플리케이션

309

```
<!-- enable component scanning(beware that this does not enable mapper scanning!)-->
<context:component-scan base-package="com.myapp.service" />
<context:component-scan base-package="com.myapp.controller" />

<!-- enable autowire -->
<context:annotation-config />

<!-- scan for mappers and let them be autowired -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
 <property name="basePackage" value="com.myapp.dao" />
</bean>
```

## 8.3.10 메퍼위치 설정 및 타입 별칭 패키지 정의(dispatcher-servlet.xml)

8. Spring을 이용한 웹 어플리케이션

310

```
<!-- define the SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
 <property name="dataSource" ref="dataSource" />
 <property name="mapperLocations" value="classpath:com/myapp/dao/*.xml" />
 <property name="typeAliasesPackage" value="com.myapp.domain" />
</bean>

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

```
<!--bean id="empMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">
 <property name="mapperInterface" value="com.myapp.dao.EmpMapper" />
 <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean-->

<!-- enable transaction demarcation with annotations -->
<tx:annotation-driven />

<!-- standard transaction configuration -->
<bean id="transactionManager"
 class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
 <property name="dataSource" ref="dataSource" />
</bean>
```

## 8.3.12 View 코드 구현 - WebRoot/view/emp/count.jsp

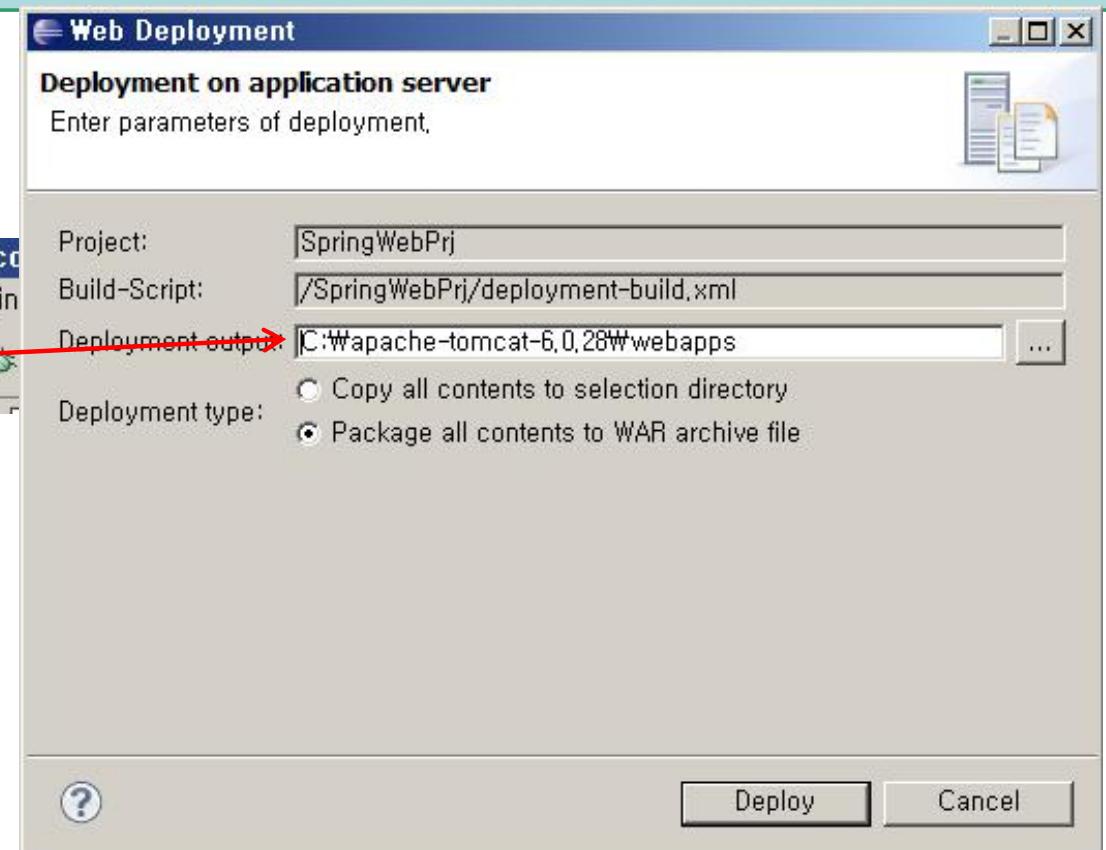
312

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
 pageEncoding="EUC-KR" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>고객의 수</title>
</head> 컨트롤러에서 ModelAndView에 addObject 할 때의 키
<body>
고객의 수 : ${count}.

<%= new java.util.Date()%>
<%= request.getAttribute("count")%>
<%= session.getAttribute("count")%>
<%= application.getAttribute("count")%>
</body>
</html>
```

- 브라우저 주소에 이 파일을 직접 실행시키는 것이 아닙니다.
- 이 파일은 결과를 나타내는 용도로 사용됩니다.
- 실행은 브라우저에서 <http://localhost:8080/contextName/emp/count.do>

- 톰켓 실행시키고...
- 디플로이 하고...



- 브라우저에서...
  - ▣ <http://localhost:8080/컨텍스트이름/emp/count.do>

## 8.3.14 디플로이 후 서버 콘솔에 다음과 같이 뜰 때...

8. Spring을 이용한 웹 어플리케이션

314

- could not be completely deleted.

```
심각: [C:\Apache-tomcat-6.0.28\webapps\SpringWebPrj] could not be completely deleted. The presence of the remaining files may cause problems
2010. 7. 20 오후 11:24:16 org.apache.catalina.startup.HostConfig deployWAR
정보: Deploying web application archive SpringWebPrj.war
```

톰켓 서버를 종료 후 다시 실행 시키세요.

- cafe24의 웹호스팅에서 테스트



## 8.4.1 조회 - 컨트롤러

- Emp 테이블 전체 조회
- <http://localhost:8080/emp/emplist.do>
  
- `@RequestMapping("/emp/emplist.do")`
- `public String empList(ModelMap map, HttpServletRequest request) {`
- `Collection<EmpVo> emps = empService.getAllEmps();`
- `map.addAttribute("emps", emps);`
- `return "emplist";`
- }

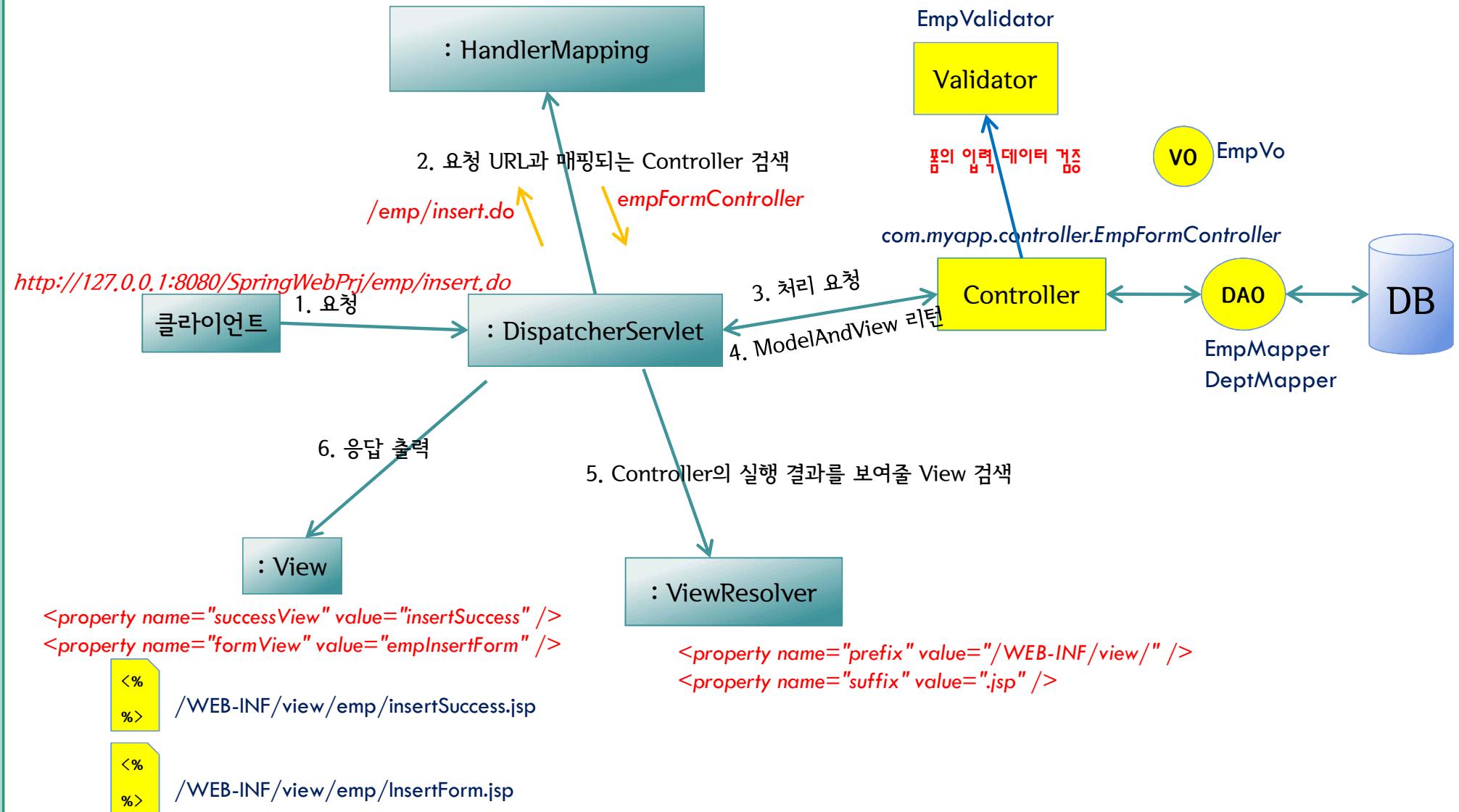
## 8.4.2 조회 - JSP 파일

```
1. <%@ page language="java" contentType="text/html; charset=UTF-8%> pageEncoding="UTF-8"%>
2. <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3. <!DOCTYPE html>
4. <html>
5. <head>
6. <meta charset="UTF-8">
7. <title>Insert title here</title>
8. </head>
9. <body>
10. <h1>사원정보입니다.</h1>
11. <table>
12. <tr>
13. <th>사원번호</th><th>이름</th><th>매니저</th><th>집</th><th>입사일</th><th>급여</th><th>커미션</th><th>부서번호</th>
14. </tr>
15. <!-- standard tag library -->
16. <c:if test="${!empty emps}">
17. <c:forEach var="emp" items="${emps}">
18. <tr>
19. <td>${emp.empno}</td>
20. <td>${emp.ename}</td>
21. <td>${emp.mgr}</td>
22. <td>${emp.job}</td>
23. <td>${emp.hiredate}</td>
24. <td>${emp.sal}</td>
25. <td>${emp.comm}</td>
26. <td>${emp.deptno}</td>
27. </tr>
28. </c:forEach>
29. </c:if>
30. </table>
31. </body>
32. </html>
```

## 8.4.3 상세 조회(컨트롤러와 JSP)

- EmpController.java에 메서드 추가
- ```
@RequestMapping(value="/emp/select.do")  
public String getEmp(@RequestParam int empno, Model model) {  
    model.addAttribute("emp", empService.selectEmp(empno));  
    return "emp/select";  
}
```
- /WEB-INF/web/emp/select.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" pageEncoding="EUC-KR"%>  
<!DOCTYPE html>  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">  
<title>사원정보</title>  
</head>  
<body>  
    사원번호 : ${emp.empno}<br>  
    이름 : ${emp.ename}<br>  
    직업 : ${emp.job}<br>  
    관리자 : ${emp.mgr}<br>  
    입사일 : ${emp.hiredate}<br>  
    급여 : ${emp.sal}<br>  
    보너스 : ${emp.comm}<br>  
    부서번호 : ${emp.deptno}<br>  
    <a href="../insert.do?empno=${emp.empno}">수정</a>  
</body>  
</html>
```



```

@Controller
@RequestMapping("/emp/insert.do")
public class EmpFormController {

    @Autowired
    private EmpService empService;

    @RequestMapping(method=RequestMethod.GET)
    public String form(@ModelAttribute("emp") EmpVo emp) {
        return "emp/empform";
    }

    @ModelAttribute("emp")
    public EmpVo formBacking(HttpServletRequest request) {
        EmpVo emp = new EmpVo();
        String empnoStr =request.getParameter("empno");
        if(empnoStr != null) {
            int empno = Integer.parseInt(empnoStr);
            emp = empService.selectEmp(empno);
        }
        if(request.getMethod().equalsIgnoreCase("GET")) {
            emp.setEmpnoList(empService.getAllEmpno());
            emp.setDeptList(empService.getAllDepts());
        }
        return emp;
    }
}

```

```

@RequestMapping(method=RequestMethod.POST)
public String submit(@Valid EmpVo emp, BindingResult
result) {
    if(result.hasErrors()) {
        System.out.println(result.toString());
        emp.setDeptList(empService.getAllDepts());
        emp.setEmpnoList(empService.getAllEmpno());
        return "emp/empform";
    }
    empService.updateEmp(emp);
    return "redirect:/emp/insert.do?empno=" +
emp.getEmpno();
}

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

```

8.4.2 EmpValidator.java (1/2)

```
public class EmpValidator implements Validator {  
  
    private EmpVo emp;  
  
    public boolean supports(Class<?> clazz) {  
        return Emp.class.isAssignableFrom(clazz);  
    }  
  
    public void validate(Object target, Errors errors) {  
  
        emp = (EmpVo) target;  
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "empno", "required", "Field is required.");  
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "ename", "required", "Field is required.");  
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "job", "required", "Field is required.");  
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "hiredate", "required", "Field is required.");  
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "sal", "required", "Field is required.");  
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "deptno", "required", "Field is required.");  
  
        if (isValidSal(emp.getSal())) {  
            errors.rejectValue("sal", "required", "급여는 0보다 커야 합니다.");  
        }  
        if (!isValidHiredate(emp.getHiredateStr())) {  
            errors.rejectValue("hiredateStr", "required", "날짜 형식(YYYY-MM-DD)에 맞지 않습니다.");  
        }  
    }  
}//end validate()
```

8.4.2 EmpValidator.java (2/2)

321

```
private boolean isValidSal(double value) {  
    if (value < 0) {  
        return true;  
    }  
    return false;  
}  
  
private boolean isValidHiredate(String date) {  
    try {  
        Date hiredate = Date.valueOf(date);  
        emp.setHiredate(hiredate);  
        return true;  
    } catch(InvalidArgumentException e){  
        return false;  
    }  
}  
}//end class
```

이게 말입니다...

EmpVo 클래스에 String hiredateStr 변수 추가했죠?

입력된 날짜를 받아서 Date 타입으로 만들어 주면서 날짜의 유효성까지 체크하는 메서드입니다.

8.4.3 empform.jsp

322

```

<%@ page contentType="text/html; charset=EUC-KR" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="spring"
   uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="form"
   uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
   charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>
<spring:hasBindErrors name="emp" />
<form:errors path="emp" />
<form action=".//insert.do" method="post">
EMPNO :
<input type="text" name="empno" value="${emp.empno}"><br>
ENAME :
<input type="text" name="ename" value="${emp.ename}">
<form:errors path="emp.ename"/><br>
JOB :
<input type="text" name="job" value="${emp.job}">
<form:errors path="emp.job"/><br>
MGR :
<select name="mgr">
<c:forEach var="mgrno" items="${emp.empnoList}">
<option value="${mgrno}" <c:if test="${emp.mgr ==
   dept.mgrno}">selected</c:if>>
${mgrno}
</option>
</c:forEach>
</select>
<form:errors path="emp.mgr"/> <br>

```

HIREDATE :

```

<input type="text" name="hiredateStr" value="${emp.hiredate}"
   placeholder="2012-01-01">
<form:errors path="emp.hiredateStr"/><br>
SAL :
<input type="text" name="sal" value="${emp.sal}">
<form:errors path="emp.sal"/><br>
COMM :
<input type="text" name="comm" value="${emp.comm}">
<form:errors path="emp.comm"/><br>
DEPTNO :
<select name="deptno">
<c:forEach var="dept" items="${emp.deptList}">
<option value="${dept.deptno}" <c:if test="${emp.deptno ==
   dept.deptno}">selected</c:if>>
${dept.dname}
</option>
</c:forEach>
</select>
<form:errors path="emp.deptno"/><br>
<input type="submit" value="저장">
<input type="reset" value="취소">
</form>
</body>
</html>

```

8.5.1 라이브러리 다운로드

- Excel 파일 포맷을 다룰 수 있는 자바 라이브러리를 제공하여 Java 어플리케이션에서도 Excel 파일을 다룰 수 있는 서비스
- Apache POI 오픈소스를 사용하여 Excel 파일 접근 및 Excel 업/다운로드 기능을 서비스할 수 있음
- 주요 기능
 - ▣ 엑셀 파일을 읽어 특정 셀의 값 조회
 - ▣ 엑셀 파일 내 특정 셀의 내용을 변경
 - ▣ 엑셀 파일 내 특정 셀의 속성(폰트, 사이즈 등) 변경
 - ▣ 엑셀 파일 생성
 - ▣ 엑셀 파일 다운로드
 - ▣ 엑셀 파일의 속성(셀의 크기, Border의 속성, 셀의 색상, 정렬 등) 변경
 - ▣ 엑셀 파일 문서의 속성(Header, Footer)을 변경
 - ▣ 공통 템플릿 사용을 통한 일관성 제공
- 라이브러리
 - ▣ The Apache POI Project
 - <http://poi.apache.org>
 - ▣ jXLS
 - <http://jxls.sourceforge.net>
- 라이브러리 파일을 WEB-INF/lib 폴더에 추가

8.5.2 컨트롤러

```
□ @RequestMapping("/emp/exceldata.do")
□ public ModelAndView getEmpByExcelData() throws Exception {
    □ Collection<EmpVo> emps = dao.selectAllEmp();
    □ Map<String, Object> map = new HashMap<String, Object>();
    □ map.put("emps", emps);
    □
    □
    □ return new ModelAndView("empExcelView", "empMap", map);
□ }
```



뷰 클래스로 작성됨

8.5.3 뷰 클래스

```

1. import java.util.*;
2. import javax.servlet.http.*;
3. import org.apache.poi.hssf.usermodel.*;
4. import org.springframework.web.servlet.view.document.AbstractExcelView;
5. import com.emp.vo.EmpVo;
6. public class EmpExcelView extends AbstractExcelView {
7.     @Override
8.     protected void buildExcelDocument(Map<String, Object> model, HSSFWorkbook wb, HttpServletRequest req, HttpServletResponse res) throws Exception {
9.         //시트 생성
10.        HSSFSheet sheet = wb.createSheet("Emp List");
11.        sheet.setDefaultColumnWidth((short)12); //deprecated
12.
13.        HSSFCell cell = getCell(sheet, 0, 0); //시트객체, 행번호, 열번호
14.        setText(cell, "Emp Lists");
15.
16.        setText(getCell(sheet, 2, 0), "empno");
17.        setText(getCell(sheet, 2, 1), "ename");
18.        setText(getCell(sheet, 2, 2), "job");
19.        setText(getCell(sheet, 2, 3), "mgr");
20.        // 생략
21.
22.        //model에서 데이터 조회, 모델에 저장할 때의 키가 empMap임
23.        Map<String, Object> map = (Map<String, Object>) model.get("empMap");
24.        ArrayList<EmpVo> emps = (ArrayList<EmpVo>)map.get("emps");
25.
26.        for(int i=0; i<emps.size(); i++) {
27.            EmpVo vo = (EmpVo)emps.get(i);
28.            setText(getCell(sheet, i+3, 0), Integer.toString(vo.getEmpno()));
29.            setText(getCell(sheet, i+3, 1), vo.getEname());
30.            setText(getCell(sheet, i+3, 2), vo.getJob());
31.            setText(getCell(sheet, i+3, 3), vo.getMgr()+"");
32.            setText(getCell(sheet, i+3, 4), vo.getHiredate().toString());
33.            //생략
34.        }
35.    }
36. }
```

8.5.4 서블릿 설정파일

326

- <!-- 엑셀뷰 클래스를 빈으로 등록 -->
- <bean id="empExcelView" class="com.emp.view.EmpExcelView" />
- <!-- 뷰 리졸버 설정 -->
- <bean class="org.springframework.web.servlet.view.BeanNameViewResolver">
- <property name="order" value="0"></property>
- </bean>

8.5.5 기타

- index.jsp 파일
 - ▣ <%@ page language="java" contentType="text/html; charset=utf-8"
pageEncoding="utf-8"%>
 - ▣ <jsp:forward page="/emp/emplist.do"/>
- emplis.jsp의 링크
 - ▣ 엑셀파일로 다운로드
- Mavel을 사용할 경우 의존성 설정
 - ▣ <dependency>
 - ▣ <groupId>egovframework.rte</groupId>
 - ▣ <artifactId>egovframework.rte.fdl.excel</artifactId>
 - ▣ <version>2.5.0</version>
 - ▣ </dependency>

8.6.1 라이브러리 다운로드

328

- <http://commons.apache.org>
 - ▣ commons-fileupload
 - ▣ commons-io

8.6.2 uploadform.jsp

329

```
1. <%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
2. <!DOCTYPE html>
3. <html>
4.   <head>
5.     <meta charset="UTF-8">
6.     <title>Insert title here</title>
7.   </head>
8.   <body>
9.     <h1>파일 업로드 폼</h1>
10.    <form action="upload.do" method="post" enctype="multipart/form-data">
11.      Title <input type="text" name="title"><br>
12.      File 1 <input type="file" name="file"><br>
13.      File 2 <input type="file" name="file">
14.      <br>
15.      <input type="submit" value=" Upload ">
16.      <input type="reset" value=" Cancel ">
17.    </form>
18.  </body>
19. </html>
```

```

1. import java.io.*;
2. import javax.servlet.http.HttpServlet;
3. import org.springframework.stereotype.Controller;
4. import org.springframework.web.bind.annotation.*;
5. import org.springframework.web.multipart.commons.CommonsMultipartFile;
6. @Controller
7. public class UploadController {
8.     @RequestMapping(value="/upload.do", method=RequestMethod.POST)
9.     public String fileUpload(@RequestParam("file") CommonsMultipartFile[] files, HttpServletRequest request) {
10.         String title = request.getParameter("title");
11.         System.out.println("입력한 제목은 : " + title);
12.
13.         //컨텍스트 루트의 절대경로를 얻어온다.
14.         String rootDir = request.getSession().getServletContext().getRealPath("/upload");
15.
16.         File dir = new File(rootDir);
17.         if(files != null) {
18.             //업로드 처리
19.             for(int i=0; i<files.length; i++) {
20.                 File uploadFile = isExist(dir, files[i].getOriginalFilename());
21.                 //파일 저장
22.                 try {
23.                     files[i].transferTo(uploadFile);
24.                 } catch (IllegalStateException e) {
25.                     e.printStackTrace();
26.                 } catch (IOException e) {
27.                     e.printStackTrace();
28.                 }
29.             }
30.         }
31.         return "index";
32.     }
33.     private File isExist(File dir, String filename) {
34.         File file = new File(dir, filename);
35.         if(file.exists()) {
36.             int i= filename.lastIndexOf(".");
37.             String fname = filename.substring(0, i) + "0"; //파일명
38.             String lname = filename.substring(i); //확장자
39.             //제거호출해서 다시 파일 존재하는지 확인
40.             file = isExist(dir, fname+lname);
41.         }
42.         return file;
43.     }
44.     @RequestMapping(value="/upload.do",
45.                     method=RequestMethod.GET)
46.     public String uploadform() {
47.         return "uploadform";
48.     }
49. }

```

```
□ <bean id="multipartResolver">  
□   class="org.springframework.web.multipart.commons.CommonsMultipart  
Resolver">  
□     <property name="defaultEncoding" value="utf-8" />  
□     <property name="maxUploadSize" value="10000000"/><!-- 10M -->  
□     <property name="uploadTempDir" value="/uploadtemp" />  
□ </bean>
```

* 프로젝트의 WebContent 폴더에 uploadtemp 폴더와 upload 폴더 생성해 줘야 함

```
□ <bean id="messageSource"
      class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
  □ <property name="basenames">
    □ <list>
      □ <value>classpath:/com/props/message/message-common</value>
    □ </list>
  □ </property>
  □ <property name="cacheSeconds">
    □ <value>60</value>
  □ </property>
  □ </bean>
```

- com/props/message/message-common_en_US.properties 파일에 아래 내용 입력
 - ▣ title.empno=Employee Number
 - ▣ title.ename=Employee Name
- com/props/message/message-common_en_KR.properties 파일에 아래 내용 입력
 - ▣ title.empno=사원 번호
 - ▣ title.ename=사원 이름
- 이클립스에서 Property Editor에 의해 properties 파일은 자동으로 유니코드로 변환되어 저장됨(아래 내용은 위의 한글 내용이 저장된 것을 메모장으로 불렀을 때 보여지는 것임)
 - ▣ title.empno=\uc0ac\uc6d0 \ubc88\ud638
 - ▣ title.ename=\uc0ac\uc6d0 \uc774\ub984

8.7.3 컨트롤러

```
□ @Resource(name="messageSource")
□ private MessageSource msgSource;           //디폴트 메시지 소스를 이용해 국제화

□ @RequestMapping("/emp/emplist.do")
□ public String empList(ModelMap map, HttpServletRequest request) {
□     Collection<EmpVo> emps = dao.selectAllEmp();
□     map.addAttribute("emps", emps);
□
□     Locale locale = request.getLocale(); //사용자 브라우저의 언어
□     Map<String, String> title = new HashMap<>();
□     title.put("empno", messageSource.getMessage("title.empno",null, locale));
□     title.put("ename", messageSource.getMessage("title.ename",null ,locale));
□
□     map.addAttribute("title", title);
□     return "emplist";
□ }
□ //getMessage(String key, Object[] arguments, Locale locale)
```

8.7.4 View(jsp 파일)

- EL을 이용해 보여지게 할 수 있음
- 일부만 설정한 것입니다. 나머지는 직접 해보세요.

- <h1>사원정보입니다.</h1>
- 엑셀파일로 다운로드
- <table>
- <tr>
- <th>\${title.empno}</th>
- <th>\${title.ename}</th>
- <th>매니저</th>
- <th>잡</th>
- <th>입사일</th>
- <th>급여</th>
- <th>커미션</th>
- <th>부서번호</th>
- </tr>

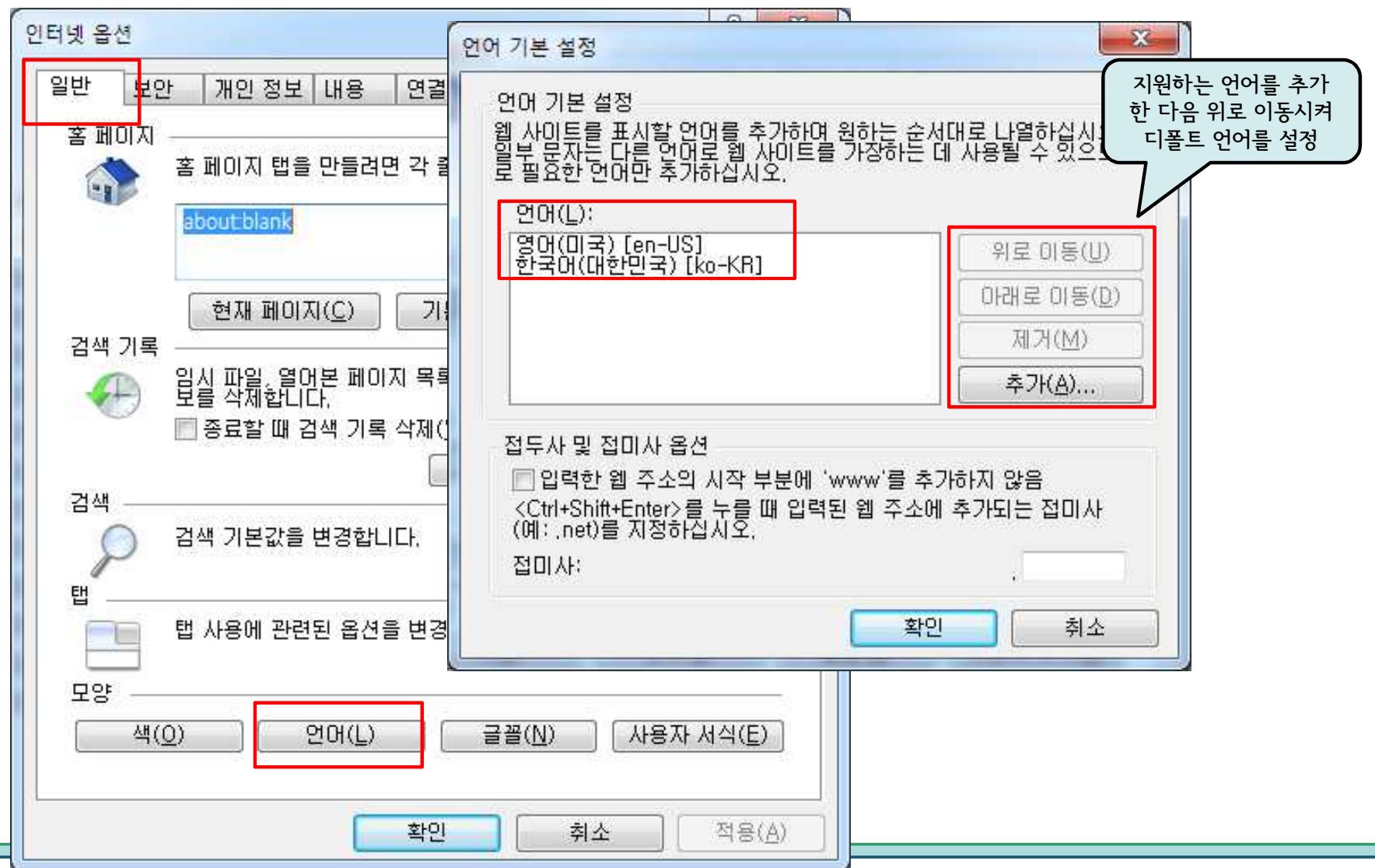
- 단순히 브라우저 화면에서만 국제화를 적용할 것이면 fmt JSTL을 이용하는 것이 더 편할 수 있음

8.7.5 브라우저에서 언어 변경

8. Spring을 이용한 웹 어플리케이션

336

- 브라우저 인터넷 옵션 -> 일반 탭 -> 언어(L)



8.7.6 실행 결과

8. Spring을 이용한 웹 어플리케이션

337

The screenshot displays two separate web browser windows side-by-side, both showing the same internationalized employee information page. The left window is in Korean, and the right window is in English.

Left Window (Korean):

- Header: "사원정보입니다."
- Link: "엑셀파일로 다운로드"
- Table Headers: "Employee Number", "Employee Name", "매니저"
- Data Rows (approximate):

| | | | |
|------|--------|------|-------|
| 7369 | SMITH | 7902 | CLERK |
| 7499 | ALLEN | 7698 | SALE |
| 7521 | WARD | 7698 | SALE |
| 7566 | JONES | 7839 | MAN |
| 7654 | MARTIN | 7698 | SALE |
| 7698 | BLAKE | 7839 | MAN |
| 7782 | CLARK | 7839 | MAN |
| 7788 | SCOTT | 7566 | ANA |
| 7839 | KING | 0 | PRES |
| 7844 | TURNER | 7698 | SALE |
| 7876 | ADAMS | 7788 | CLE |
| 7900 | JAMES | 7698 | CLE |
| 7902 | FORD | 7566 | ANA |
| 7934 | MILLER | 7782 | CLER |

Right Window (English):

- Header: "사원정보입니다."
- Link: "엑셀파일로 다운로드"
- Table Headers: "사원 번호", "사원 이름", "매니저", "잡", "입사일", "급여", "커미션", "부서번호"
- Data Rows (approximate):

| | | | | | | | |
|------|--------|------|-----------|------------|--------|--------|----|
| 7369 | SMITH | 7902 | CLERK | 1980-12-17 | 800.0 | 0.0 | 20 |
| 7499 | ALLEN | 7698 | SALESMAN | 1981-02-20 | 1600.0 | 300.0 | 30 |
| 7521 | WARD | 7698 | SALESMAN | 1981-02-22 | 1250.0 | 500.0 | 30 |
| 7566 | JONES | 7839 | MANAGER | 1981-04-02 | 2975.0 | 0.0 | 20 |
| 7654 | MARTIN | 7698 | SALESMAN | 1981-09-28 | 1250.0 | 1400.0 | 30 |
| 7698 | BLAKE | 7839 | MANAGER | 1981-05-01 | 2850.0 | 0.0 | 30 |
| 7782 | CLARK | 7839 | MANAGER | 1981-06-09 | 2450.0 | 0.0 | 10 |
| 7788 | SCOTT | 7566 | ANALYST | 1987-04-19 | 3000.0 | 0.0 | 20 |
| 7839 | KING | 0 | PRESIDENT | 1981-11-17 | 5000.0 | 0.0 | 10 |
| 7844 | TURNER | 7698 | SALESMAN | 1981-09-08 | 1500.0 | 0.0 | 30 |
| 7876 | ADAMS | 7788 | CLERK | 1987-05-23 | 1100.0 | 0.0 | 20 |
| 7900 | JAMES | 7698 | CLERK | 1981-12-03 | 950.0 | 0.0 | 30 |
| 7902 | FORD | 7566 | ANALYST | 1981-12-03 | 3000.0 | 0.0 | 20 |
| 7934 | MILLER | 7782 | CLERK | 1982-01-23 | 1300.0 | 0.0 | 10 |

감사합니다.

