



spring 4강-게시판

양 명 속

[now4ever7@gmail.com]



목차

- 스프링, mybatis 이용 게시판

root-context.xml => 공통설정
servlet-context.xml => 개별설정

모든 작업은 c:사용자:사용자 메인 파일:repository에 저장된다

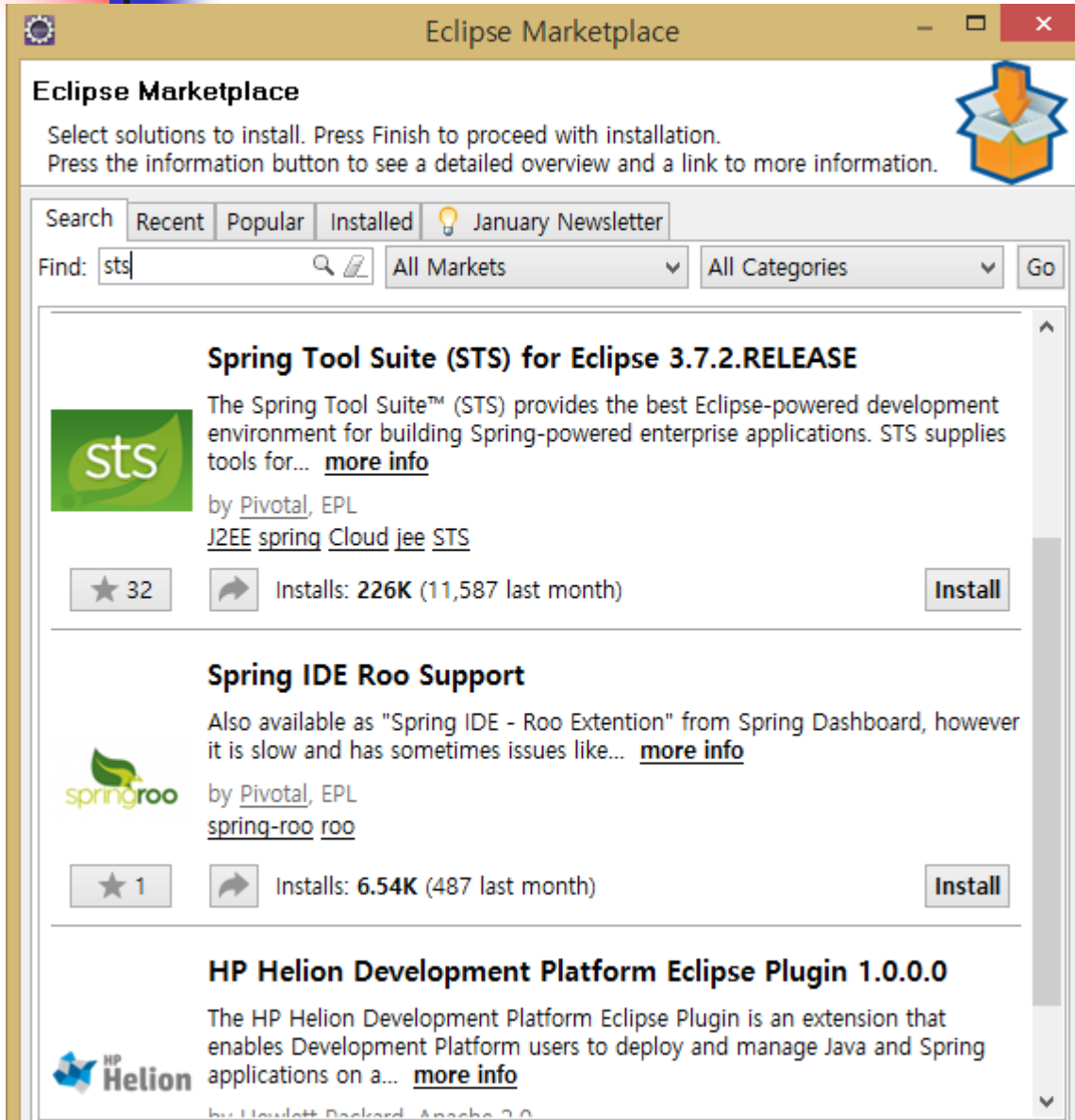
pom.xml에 설정해두면 자동으로 다운받아짐~>dependency 부분에 설정



STS(Spring Tool Suits)

- STS는 기본적으로 이클립스와 동일함
- 거기에 Spring 기반의 프로젝트를 생성할 때 Spring MVC 처럼 템플릿화 해서 불러올 수 있으며 별도의 확장 플러그인 없이도 기본적인 Maven build 타입의 Project를 생성할 수 있음
- 기존의 Dynamic Web Project 등과 같은 이전의 이클립스 프로젝트들도 지원됨
- <http://www.springsource.org/download>
- <http://spring.io/tools/sts/all>

Eclipse Marketplace 이용



The screenshot shows the Eclipse Marketplace window. At the top, it says "Eclipse Marketplace" and "Select solutions to install. Press Finish to proceed with installation. Press the information button to see a detailed overview and a link to more information." Below this is a search bar with tabs for "Search", "Recent", "Popular", "Installed", and "January Newsletter". The search bar contains the text "sts". To the right of the search bar are dropdown menus for "All Markets" and "All Categories", and a "Go" button. Below the search bar, there are three search results. The first result is "Spring Tool Suite (STS) for Eclipse 3.7.2.RELEASE" by Pivotal, EPL. It has a green icon with "sts" and a description: "The Spring Tool Suite™ (STS) provides the best Eclipse-powered development environment for building Spring-powered enterprise applications. STS supplies tools for... [more info](#)". It has 32 stars and 226K installs (11,587 last month). There is an "Install" button. The second result is "Spring IDE Roo Support" by Pivotal, EPL. It has a green icon with "springroo" and a description: "Also available as 'Spring IDE - Roo Extention' from Spring Dashboard, however it is slow and has sometimes issues like... [more info](#)". It has 1 star and 6.54K installs (487 last month). There is an "Install" button. The third result is "HP Helion Development Platform Eclipse Plugin 1.0.0.0" by Hewlett-Packard, Apache 2.0. It has a blue icon with "HP Helion" and a description: "The HP Helion Development Platform Eclipse Plugin is an extension that enables Development Platform users to deploy and manage Java and Spring applications on a... [more info](#)".

Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Search Recent Popular Installed January Newsletter

Find: sts All Markets All Categories Go

Spring Tool Suite (STS) for Eclipse 3.7.2.RELEASE

The Spring Tool Suite™ (STS) provides the best Eclipse-powered development environment for building Spring-powered enterprise applications. STS supplies tools for... [more info](#)

by Pivotal, EPL
[J2EE](#) [spring](#) [Cloud](#) [jee](#) [STS](#)

★ 32 Installs: 226K (11,587 last month) **Install**

Spring IDE Roo Support

Also available as "Spring IDE - Roo Extention" from Spring Dashboard, however it is slow and has sometimes issues like... [more info](#)

by Pivotal, EPL
[spring-roo](#) [roo](#)

★ 1 Installs: 6.54K (487 last month) **Install**

HP Helion Development Platform Eclipse Plugin 1.0.0.0

The HP Helion Development Platform Eclipse Plugin is an extension that enables Development Platform users to deploy and manage Java and Spring applications on a... [more info](#)

by Hewlett-Packard, Apache 2.0



Maven



Maven

- JAVA 를 위한 빌드 툴이자 프로젝트 전반의 라이프사이클 관리를 목적으로 하는 툴
- 프로젝트 관리 도구. 빌드 생명주기(Build Lifecycle)에 따라 프로젝트 표준을 제공하고 의존 관계를 관리하고 플러그인이 제공하는 부가 기능을 사용할 수 있게 하는 도구
- 메이븐에서 라이브러리 혹은 프로젝트 의존 관계(Dependency)를 설정하는 것은 매우 간단함. 설정 파일에 그 의존 관계를 선언하기만 하면 자동으로 처리해줌
- pom.xml, phase, plugin, goal
 - 메이븐을 이해하고 사용하는데 가장 중요한 키워드
- pom.xml
 - 메이븐을 이용하는 프로젝트의 root에 존재하는 xml 파일
 - 파일명 pom은 프로젝트 객체 모델(Project Object Model)을 뜻함
 - pom.xml 에는 개발자가 해당 프로젝트에 대한 정보 및 종속성(dependency) 등을 기술하여 관리함. 메이븐에서는 phase 실행 시 pom.xml 에 기술된 정보를 읽어들이어 사용함



Maven

■ Phase

- 단계를 뜻하는데, 메이븐에서는 프로젝트 라이프사이클의 각 단계를 미리 정해놓았음.
- 기본적인 phase들을 순서대로 나열해보면 아래와 같다.

Clean Lifecycle

clean : 이전 빌드에서 생성된 파일들을 삭제하는 단계. 보통 빌드 단계를 실행하기 전 사용.

Build Lifecycle

process-resources : 컴파일전 필요한 리소스들을 target 디렉토리에 복사하고 필요한 처리 작업을 진행하는 단계

compile : 프로젝트의 소스 코드를 컴파일하는 단계

process-test-resources : 테스트 코드 컴파일에 필요한 리소스들을 target 디렉토리에 복사하고 필요한 처리 작업을 진행하는 단계

test-compile : 프로젝트의 테스트 코드를 컴파일하는 단계

test : 테스트를 실행하는 단계

package : 컴파일된 소스 코드와 리소스들을 배포를 위한 패키지(ex. JAR, WAR)로 만드는 단계

install : 패키지를 로컬 저장소에 설치하는 단계

deploy : 만들어진 패키지를 외부 저장소에 release 하는 단계

Site Lifecycle

site : 프로젝트 문서를 생성하는 단계

Maven

• 각 phase는 실행할 goal을 가지고 있고, goal은 plugin에서 제공해주는 것이며, 어떤 plugin을 어떻게 사용할지, 어떤 phase에 어떤 goal을 사용할지는 pom.xml에서 설정

- 각 Lifecycle에서 phase를 실행시 이전 phase부터 순차적으로 실행이 됨
- 예) test phase를 실행시에는 process-resources => compile => process-test-resources => test 순으로 실행됨
- 각 phase가 실행된다는 것은 곧, 해당 phase에서 실행되도록 설정된 goal들을 실행한다는 뜻
- 보통 goal 들은 compiler:compile 형식으로 나타냄.
 - 콜론 앞의 compiler는 plugin을 나타내고, 콜론 뒤의 compile은 goal 명을 나타냄
 - 즉, **플러그인명:골명** 으로 나타냄
 - goal을 이렇게 나타내는 이유는 메이븐의 구현 아키텍처와도 관련이 있는데, **메이븐 자체는 아주 기본적인 기능만을 가지고 있고 대부분의 기능들은 plugin을 통해서 제공하도록 되어있기 때문**
 - plugin 들은 몇 가지 goal 기능들을 제공해주고, 프로젝트 마다의 필요에 따라서 유연하게 설정해 사용하도록 되어있음.
 - 각 빌드 phase에서 실행하는 기본적인 goal들을 매치해보면 아래와 같다.

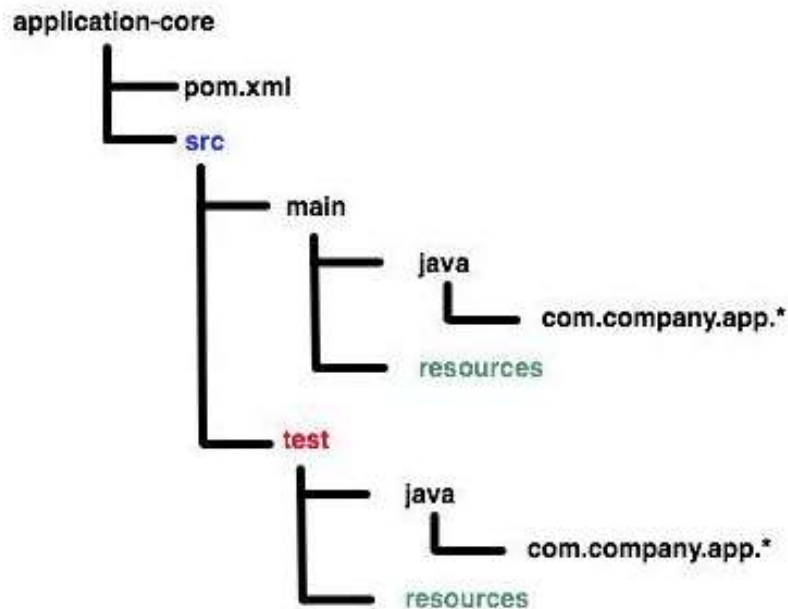
```
process-resources => resources:resource
compile => compiler:compile
process-test-resources => resources:testResources
test-compile => compiler:testCompile
test => surefire:test
package => jar:jar
install => install:install
deploy => deploy:deploy
```




Maven

- **groupId** – 프로젝트 속하는 그룹 식별 값. 회사, 본부, 또는 단체를 의미하는 값이 오며, 패키지 형식으로 계층을 표현함.
- **artifactId** – 프로젝트 결과물의 식별 값. 프로젝트나 모듈을 의미하는 값이 옴
- **version** – 결과물의 버전을 입력
- **package** – 기본적으로 생성할 패키지를 입력
 - 별도로 입력하지 않을 경우 groupId와 동일한 구조의 패키지를 생성

Maven 프로젝트의 기본 디렉토리 정책



- 메이븐의 프로젝트 관리 디렉토리는 최상위에 프로젝트를 기준으로 pom.xml 이라는 메이븐 프로젝트 설정 파일이 존재하며 그것을 바탕으로 프로젝트와 관련된 정보를(라이브러리 및 빌드 정보등) 기술
- src 밑으로는 main 과 test 라는 디렉토리가 존재하며 각 하위에 java 와 resources 가 위치
- 기본적인 메이븐의 주요 디렉토리
 - src/main/java - 자바 소스 파일이 위치함
 - src/main/resources - 프로퍼티나 XML 등 리소스 파일이 위치함. 클래스패스에 포함됨
 - src/main/webapp - 웹 어플리케이션 관련 파일이 위치한다. (WEB-INF 디렉터리, JSP 파일 등)
 - src/test/java - 테스트 자바 소스 파일이 위치함
 - JUnit등의 테스트 파일
 - src/test/resources - 테스트 과정에서 사용되는 리소스 파일이 위치함. 테스트 시에 사용되는 클래스패스에 포함됨



Maven

- 컴파일 해보기/테스트 실행 해보기/패키지 해보기
- [1] 소스 코드를 컴파일 하려면 다음 명령어를 실행
 - mvn **compile**
 - 컴파일 된 결과는 target/classes 디렉터리에 생성
- [2] 테스트 클래스를 실행해보고 싶다면 다음과 같은 명령어를 사용
 - mvn **test**
 - 테스트 코드를 컴파일한 뒤 테스트 코드를 실행함. 그리고 테스트 성공 실패 여부를 화면에 출력.
 - 컴파일 된 테스트 클래스들은 target/test-classes 디렉터리에 생성되고, 테스트 결과 리포트는 target/surefire-reports 디렉터리에 저장
- [3] 배포 가능한 jar 파일 만들기
 - mvn **package**
 - 프로젝트를 패키징해서 결과물을 생성함
 - mvn package가 성공적으로 실행되면, target 디렉터리에 프로젝트 이름과 버전에 따라 알맞은 이름을 갖는 jar 파일이 생성됨.
 - 예) simple-app-1.0-SNAPSHOT.jar 파일이 생성



Maven

■ POM 파일 기본

- Maven 프로젝트를 생성하면 pom.xml 파일이 프로젝트 루트 디렉터리에 생성됨.
- 이 pom.xml 파일은 Project Object Model 정보를 담고 있는 파일

■ POM 파일에서 다루는 주요 설정 정보

- 프로젝트 정보 - 프로젝트의 이름, 개발자 목록, 라이선스 등의 정보를 기술
- 빌드 설정 - 소스, 리소스, 라이프 사이클 별 실행할 플러그인 등 빌드와 관련된 설정을 기술
- 빌드 환경 - 사용자 환경 별로 달라질 수 있는 프로파일 정보를 기술
- POM 연관 정보 - 의존 프로젝트(모듈), 상위 프로젝트, 포함하고 있는 하위 모듈 등을 기술



Maven-[기본으로 생성되는 pom.xml 파일]

```
<?xml version="1.0" encoding="UTF-8"?>
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.spmaven</groupId>
  <artifactId>myapp</artifactId>
  <name>spmaven3</name>
  <packaging>war</packaging>
  <version>1.0.0-BUILD-SNAPSHOT</version>
  <properties>
    <java-version>1.6</java-version>
    <org.springframework-version>3.1.1.RELEASE</org.springframework-version>
    <org.aspectj-version>1.6.10</org.aspectj-version>
    <org.slf4j-version>1.6.6</org.slf4j-version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${org.springframework-version}</version>
    </dependency>
```



Maven

- POM 파일에서 프로젝트 정보를 기술하는 태그는 다음과 같다.
 - <name> - 프로젝트 이름
 - <url> - 프로젝트 사이트 URL
- POM 연관 정보는 프로젝트간 연관 정보를 기술하는데, 관련 태그는 다음과 같다.
 - <groupId> - 프로젝트의 그룹 ID 설정
 - <artifactId> - 프로젝트의 Artifact ID 설정
 - <version> - 버전 설정
 - <packaging> - 패키징 타입 설정. 프로젝트의 결과 Artifact가 jar 파일로 생성됨을 의미한다. jar 뿐만 아니라 웹 어플리케이션을 위한 war나 JEE를 위한 ear 등의 패키징 타입이 존재한다.
 - <dependencies> - 이 프로젝트에서 의존하는 다른 프로젝트 정보를 기술한다.
 - <dependency> - 의존하는 프로젝트 POM 정보를 기술
 - <groupId> - 의존하는 프로젝트의 그룹 ID
 - <artifactId> - 의존하는 프로젝트의 artifact ID
 - <version> - 의존하는 프로젝트의 버전
 - <scope> - 의존하는 범위를 설정



Maven

- 의존 설정 - <dependency> 부분의 설정
 - Maven을 사용하지 않을 경우 개발자들은 코드에서 필요로 하는 라이브러리를 각각 다운로드 받아야 함
 - 코드에서 필요로 하는 라이브러리 뿐만 아니라 그 라이브러리가 필요로 하는 또 다른 라이브러리도 직접 찾아서 설치해 주어야 함
 - Maven을 사용할 경우에는 **코드에서 직접적으로 사용하는 모듈에 대한 의존만 추가해주면 됨**
 - 예를 들어, commons-dbcp 모듈을 사용하고 싶은 경우 다음과 같은 <dependency> 코드만 추가해주면 됨

```
<dependency>
  <groupId>commons-dbcp</groupId>
  <artifactId>commons-dbcp</artifactId>
  <version>1.4</version>
</dependency>
```
 - 그러면, Maven은 commons-dbcp 뿐만 아니라 commons-dbcp가 **의존하는 라이브러리도 자동으로 처리해줌.**



Maven

- Maven은 commons-logging 모듈을 다운로드 받을 때 관련 POM 파일도 함께 다운로드 받음. (다운로드 받은 파일은 로컬 리포지토리에 저장됨)
- 그리고 POM 파일에 명시한 의존 모듈을 함께 다운로드 받음
- 이런 식으로 반복해서 다운로드 받은 모듈이 필요로 하는 모듈을 다운로드 받고 이들 모듈을 현재 프로젝트에서 사용할 클래스패스에 추가해줌
- 개발자는 일일이 필요한 모듈을 다운로드 받을 필요가 없으며, 현재 코드에서 직접적으로 필요로 하는 모듈에 대해서만 <dependency>로 추가해주면 됨
- 나머지 의존은 모두 Maven이 알맞게 처리해줌
- mvnrepository.com 사이트에서 POM 정보 찾기
 - <http://www.mvnrepository.com/> 는 Maven 중앙 리포지토리에 등록된 POM 정보를 검색해주는 기능을 제공해줌
 - 이 사이트를 통해서 추가할 라이브러리의 <dependency> 설정 정보를 구할 수 있음



Maven

- 원격 리포지토리 and 로컬 리포지토리
 - Maven은 컴파일이나 패키징 등 작업을 실행할 때 필요한 플러그인이나 pom.xml 파일의 <dependency> 등에 설정한 모듈을 **Maven 중앙 리포지토리에서 다운로드 받음**
 - **원격 리포지토리에서 다운로드 받은 모듈은 로컬 리포지토리에 저장됨**
 - 로컬 리포지토리는 [USER_HOME]/.m2/repository 디렉터리에 생성되며, 로컬 리포지토리에는 다음과 같은 형식의 디렉터리를 생성한 뒤 다운로드 받은 모듈을 저장함
 - [groupId]/[artifactId]/[version]
- 예) commons-dbcp 1.4 버전의 경우, 모듈 및 관련 POM 파일이 저장되는 디렉터리
 - [USER_HOME]/.m2/repository/commons-dbcp/commons-dbcp/1.4
 - 위 디렉터리에 저장되는 파일은 패키징 된 모듈 파일, pom 파일, 그리고 소스 코드 다운로드 옵션을 실행한 경우에는 소스 코드를 포함한 jar 파일이 포함
- 일단 원격 리포지토리로부터 파일을 다운로드해서 로컬 리포지토리에 저장하면, 그 뒤로는 로컬 리포지토리에 저장된 파일을 사용하게 됨



사설 메이븐 저장소

```
<repositories>
  <repository>
    <id>springsource-repo</id>
    <name>SpringSource Repository</name>
    <url>http://repo.springsource.org/release</url>
  </repository>

  <repository>
    <id>oracle</id>
    <name>ORACLE JDBC Repository</name>
    <url>http://maven.jahia.org/maven2</url>
  </repository>
</repositories>
```

- 로컬 저장소에서 저 위치에 해당하는 파일이 없다면, 메이븐은 원격 저장소에서 로컬 저장소로 파일을 받아옴



Log4j

모든 파일을 세팅할 때 `src/main/resources`와 `src/test/resources` 파일을 잘 구분해서 넣기... 비슷해서 헷갈림



Log4j

- Log4j
 - 자바기반 로깅 유틸리티
 - 디버그용 도구로 주로 사용됨.
 - 높은 등급에서 낮은 등급으로의 6개 로그 레벨을 가지고 있음
 - 설정 파일에 대상별로 레벨 지정이 가능하고 그 등급 이상의 로그만 저장하는 방식임
 - TRACE, DEBUG, INFO, WARN, ERROR, FATAL
 - 예) INFO레벨로 지정해두면 logger.debug로 찍는 로그는 출력되지 않고, INFO레벨 이상의 것만 출력됨
 - FATAL : 가장 크리티컬한 에러가 일어 났을 때 사용
 - ERROR : 일반 에러가 일어 났을 때 사용
 - WARN : 에러는 아니지만 주의할 필요가 있을 때 사용
 - INFO : 일반 정보를 나타낼 때 사용
 - DEBUG : 일반 정보를 상세히 나타낼 때 사용



Log4j.xml

```
<!-- Root Logger -->
```

```
<root>
```

```
  <priority value="warn" />
```

```
  <appender-ref ref="console" />
```

```
</root>
```

```
<!-- Appenders -->
```

```
<appender name="console" class="org.apache.log4j.ConsoleAppender">
```

```
  <param name="Target" value="System.out" />
```

```
  <layout class="org.apache.log4j.PatternLayout">
```

```
    <param name="ConversionPattern" value="%p: %c - %m%n" />  <
```

```
  </layout>
```

```
</appender>
```

```
<param name="ConversionPattern"
  value="[%d{HH:mm:ss}] %p: %c:%M:%L - %m%n" />
```

```
<!-- Application Loggers -->
```

```
<logger name="com.herb.app">
```

```
  <level value="info" />
```

```
</logger>
```

Log4j

- root Logger는 최상위 로거
 - 모든 warn 레벨 이상의 로그는 다 console로 찍겠다는 것
- console – ConsoleAppender라는 클래스
 - 콘솔에 로그를 찍어준다는 것
- layout에는 PatternLayout을 지정 – 레벨, 클래스, 메시지 등
- 파일에다 출력 할 수 있는데, DailyRollingFileAppender클래스를 이용하면 파일에 출력
 - 매일매일 다른 로그를 사용하게 만듦
 - 로그이름이 data.log라면, 해당로그가 어제날짜인데 로그를 찍으려고 하면 기존에 있던 파일은 data.log.2014-06-17 로 바뀌어줌
- logger name에 패키지명이나 클래스명을 지정해놓고, 로그레벨과 출력할 로그를 지정할 수 있는데, 해당 클래스나 패키지의 로그는 지정한 것으로 찍겠다는 것

```
<appender name="file" class="org.apache.log4j.DailyRollingFileAppender">
  <param name="File" value="d:\□□logs□□debug.log" />    <<디버그 파일을 매일 새로 만들어줌 해당 위치에..
  <param name="Append" value="true" />
  <param name="DatePattern" value="'.yyyy-MM-dd' />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="[%d{yyyy-MM-dd HH:mm}] %p
                                                                [%C{1}(%M:%L)] :%m%n%n" />
  </layout>
</appender>
```



Log4j

```
<param name="ConversionPattern"
value="[%d{HH:mm:ss}] %p: %c:%M:%L - %m%n" />
```

- **%p** – debug, info, warn, error, fatal 등의 priority 가 출력된다.
- **%m** – 로그내용이 출력됩니다
- **%d** – 로깅 이벤트가 발생한 시간을 기록합니다.
포맷은 %d{HH:mm:ss, SSS}, %d{yyyy MMM dd HH:mm:ss, SSS}같은 형태로 사용하며 SimpleDateFormat에 따른 포맷팅을 하면 된다
- **%t** – 로그이벤트가 발생한 스레드의 이름을 출력합니다.
- **%%** – % 표시를 출력하기 위해 사용한다.
- **%n** – 플랫폼 종속적인 개행문자가 출력된다. (WrWn 또는 Wn)
- **%c** – 카테고리를 표시합니다
예) 카테고리가 a.b.c 처럼 되어있다면 %c{2}는 b.c가 출력
- **%C** – 클래스명을 표시함
예)클래스구조가 org.apache.xyz.SomeClass 처럼 되어있다면 %C{2}는 xyz.SomeClass 가 출력
- **%F** – 로깅이 발생한 프로그램 파일명을 나타냅니다.
- **%l** – 로깅이 발생한 caller의 정보를 나타냅니다
- **%L** – 로깅이 발생한 caller의 라인수를 나타냅니다
- **%M** – 로깅이 발생한 method 이름을 나타냅니다.
- **%r** – 어플리케이션 시작 이후 부터 로깅이 발생한 시점의 시간(milliseconds)
- **%x** – 로깅이 발생한 thread와 관련된 NDC(nested diagnostic context)를 출력합니다.
- **%X** – 로깅이 발생한 thread와 관련된 MDC(mapped diagnostic context)를 출력



Log4j

- 소스코드에서 로그 출력하기

```
// Logger 클래스의 인스턴스를 받아온다.  
static Logger logger = Logger.getLogger(SimpleTest.class);  
  
public static void main(String[] args) {  
    logger.debug("[DEBUG] Hello log4j.");  
    logger.info ("[INFO] Hello log4j.");  
    logger.warn ("[WARN] Hello log4j.");  
    logger.error("[ERROR] Hello log4j.");  
    logger.fatal("[FATAL] Hello log4j.");  
    //logger.log( Level.DEBUG , "debug") 와 동일  
}
```




slf4j

- slf4j – jdk, common_log, log4j 등등의 로깅 framework를 통합해서 사용할 수 있는 로깅 framework
- slf4j와 log4j를 연동해서 사용 가능
- slf4j와 log4j를 위한 필수 jar들
 - slf4j-api-1.6.6.jar
 - log4j-1.2.15.jar
 - slf4j-log4j12-1.6.6.jar



LOG4J와 SLF4J 연동하기

```
import org.slf4j.LoggerFactory;
```

```
import org.slf4j.Logger;
```

```
public class HelloWorld {
```

```
    public static void main(String[] args) {
```

```
        Logger logger = LoggerFactory.getLogger(HelloWorld.class);
```

```
        String str="Hello World";
```

```
        logger.info("변수값={}", str); // {}안에 "hello world"가 입력이 되서
```

출력

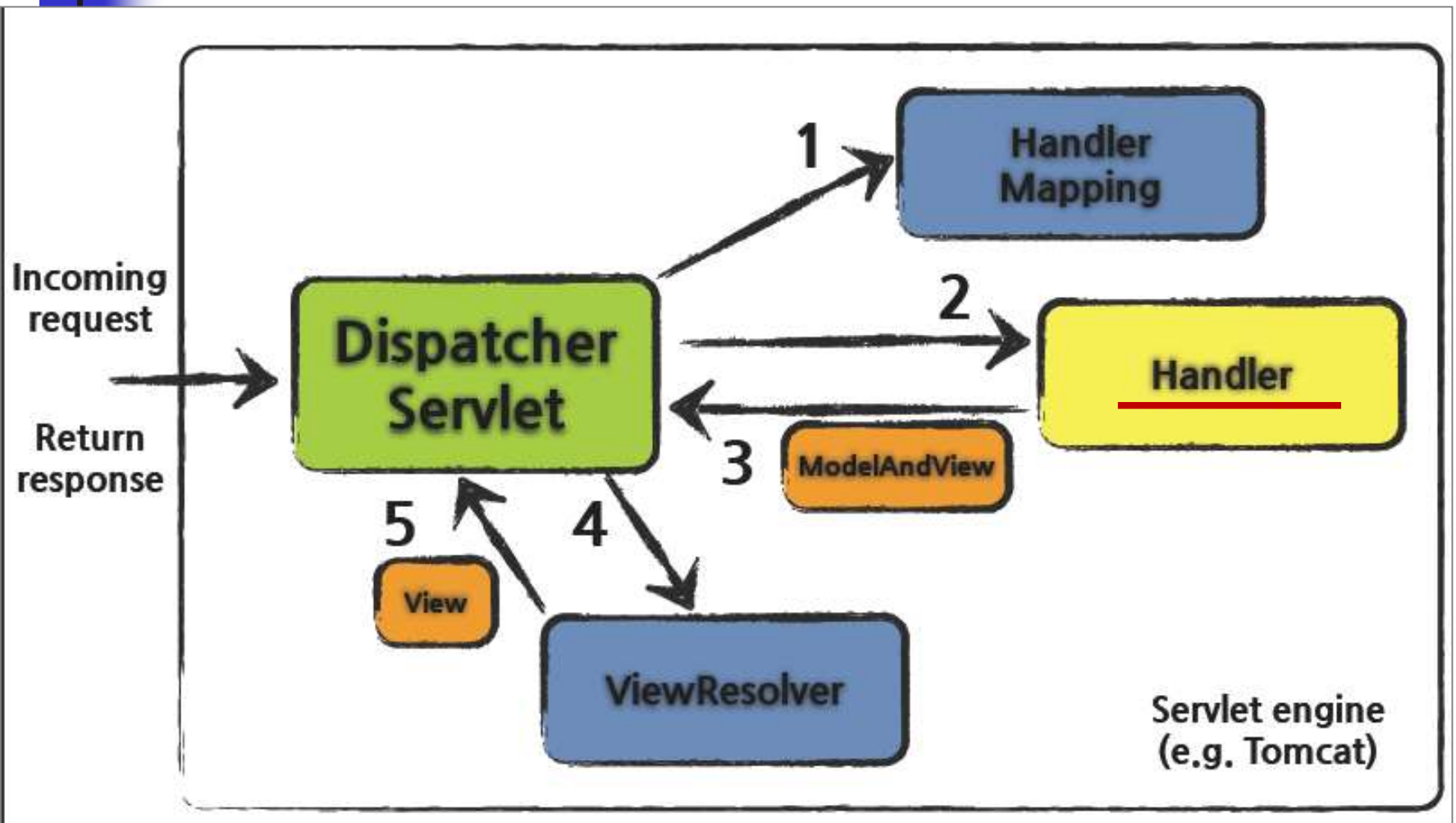
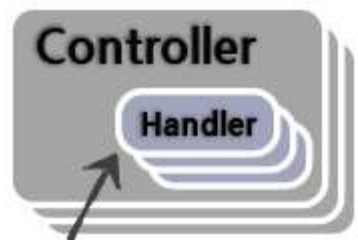
```
    }
```

```
}
```



스프링 @MVC

스프링 @MVC





Spring MVC 이용 - 게시판 만들기

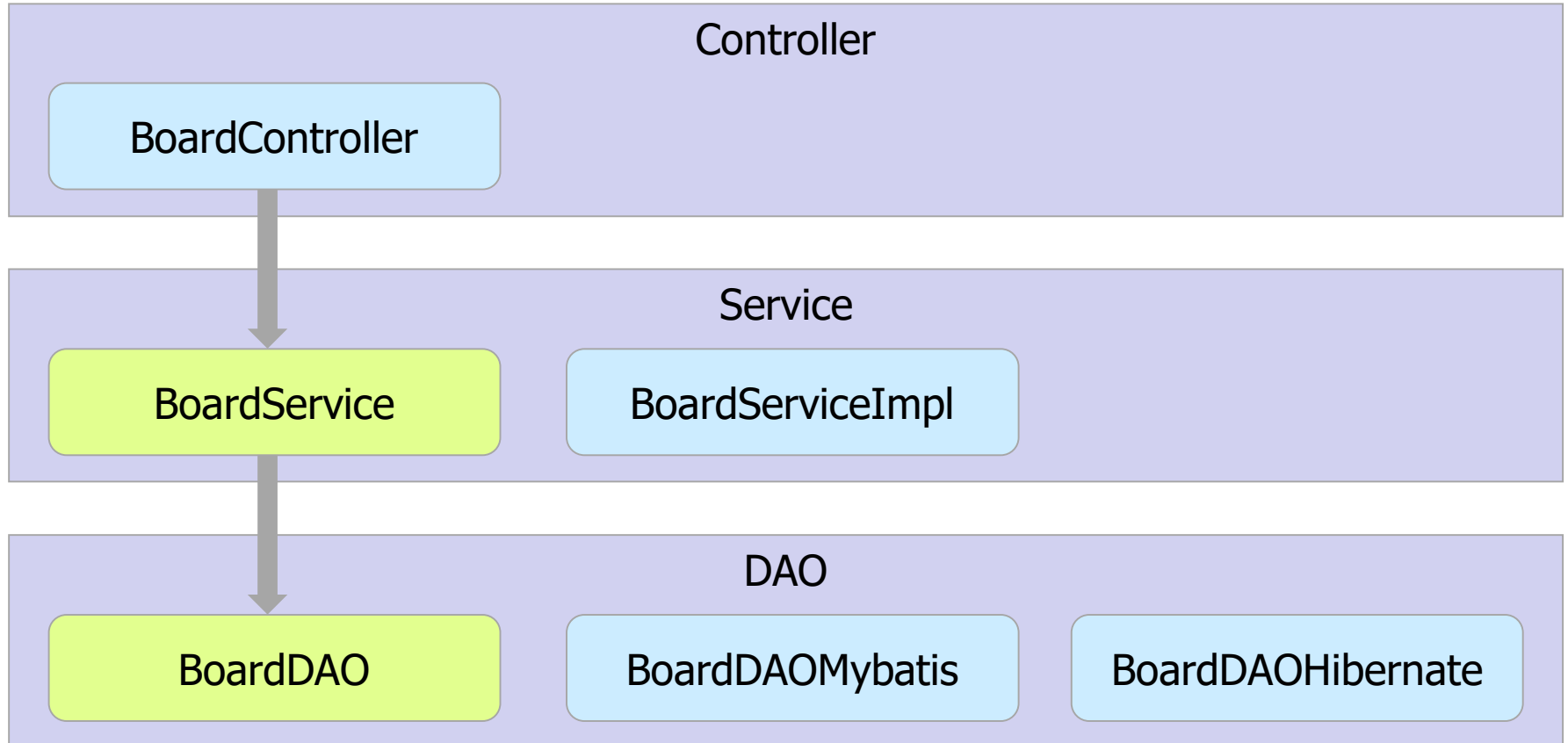
springherb

- springherb3
 - src/main/java
 - com.herb.app
 - com.herb.app.board.controller
 - BoardCountUpdateControll
 - BoardDeleteController.java
 - BoardEditController.java
 - BoardListController_BAK.jav
 - BoardListController.java
 - BoardViewController.java
 - BoardWriteController.java
 - com.herb.app.board.model
 - BoardBean.java
 - BoardDAO.java
 - BoardDAOIbatis.java
 - BoardService.java
 - BoardServiceImpl.java
 - com.herb.app.common
 - PaginationInfo.java
 - PagingBean.java
 - SearchBean.java
 - Utility.java

- src/main/resources
 - config.mybatis.mapper.oracle
 - Board.xml
 - config.mybatis.oracle
 - mybatis-config.xml
 - config.props
 - database.properties
 - fileUpload.properties
 - paging.properties
 - config.spring
 - context-common.xml
 - context-database.xml
 - META-INF
 - log4j.xml
 - src/test/java
 - src/test/resources
 - JRE System Library [JavaSE-1.6]

- src
 - main
 - webapp
 - common
 - css
 - images
 - js
 - pdimages
 - pds_upload
 - resources
 - WEB-INF
 - classes
 - spring
 - appServlet
 - servlet-context.xml
 - views
 - board
 - inc
 - home.jsp
 - web.xml
 - test
 - target
 - pom.xml

계층형 아키텍처





web.xml

```
<context-param>
```

```
    <param-name>contextConfigLocation</param-name>
```

```
    <param-value>classpath*:config/spring/context-*.xml</param-value>
```

```
</context-param>
```

```
<listener>
```

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

```
</listener>
```

```
<servlet>
```

```
    <servlet-name>appServlet</servlet-name>
```

```
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
```

```
    <init-param>
```

```
        <param-name>contextConfigLocation</param-name>
```

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

```
    </init-param>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
    <servlet-name>appServlet</servlet-name>
```

```
    <url-pattern>*.do</url-pattern>
```

```
</servlet-mapping>
```




web.xml

```
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>utf-8</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>*.do</url-pattern>
</filter-mapping>

<error-page>
  <error-code>404</error-code>
  <location>/code404.jsp</location>
</error-page>
<error-page>
  <error-code>500</error-code>
  <location>/code500.jsp</location>
</error-page>
```

```
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

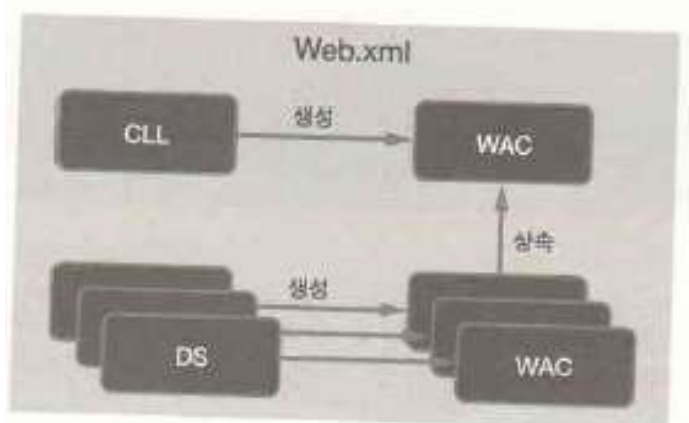


설정

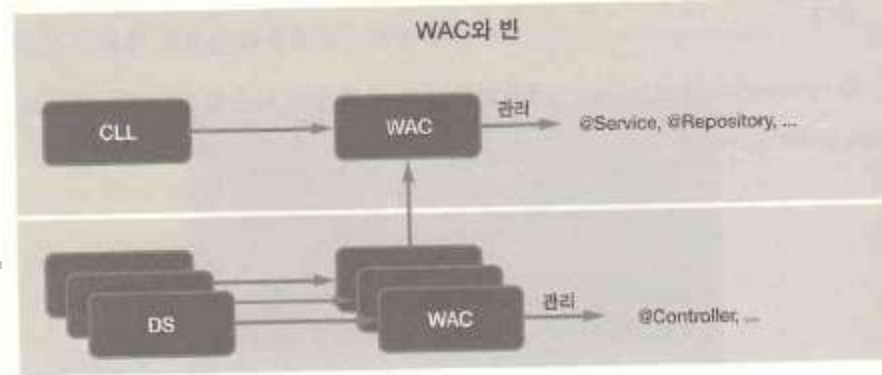
- 전체 설정 구조(web.xml)
 - 스프링 설정 파일은 스프링의 핵심인 ApplicationContext 타입의 객체를 만들 때 사용됨
 - 이 객체는 웹 어플리케이션에서 계층 구조로 구성될 수 있음
- ContextLoaderListener
 - 서블릿에서 제공하는 ServletContextListener 를 확장하여 만든 것으로, 웹 애플리케이션이 서블릿 컨테이너에 로딩될 때 실행되는 리스너
 - contextConfigLocation 매개변수에 ContextLoaderListener에서 사용할 스프링 설정파일을 지정
 - ContextLoaderListener가 하는 일 – 웹 어플리케이션이 로딩될 때 WebApplicationContext 를 만드는 것
 - **WebApplicationContext** – contextConfigLocation 에 설정한 빈 설정 파일을 사용해서 웹 어플리케이션에서 사용할 객체를 관리해주는 역할을 함

설정

- DispatcherServlet도 ContextLoaderListener처럼 WebApplicationContext 를 생성하지만,
- ContextLoaderListener에서 만든 WebApplicationContext 가 있다면, 그것을 상속받는 WebApplicationContext 를 만든다
- ContextLoaderListener 는 애플리케이션당 WebApplicationContext 한 개, DispatcherServlet은 서블릿 설정당 WebApplicationContext 를 한 개씩 만든다
- ContextLoaderListener 는 애플리케이션 전체에서 사용할 WebApplicationContext 를 만들고, DispatcherServlet은 해당 서블릿에서만 사용할 WebApplicationContext 를 만든다



상속 구조의 특징



- 자식 WebApplicationContext (WAC) 에서 부모 WAC의 빈을 참조할 수 있다
- 부모 WAC에서는 자식 WAC의 빈을 참조할 수 없다
- 자식 WAC에서 어떤 빈을 참조할 때, 먼저 자기 자신 내부에 있는 빈을 참조한다. 만약 자기 자신 내부에 없다면 부모 쪽에서 찾아보고, 부모 쪽에도 원하는 빈이 없다면 예외를 발생시킴
- 보통 빈을 다음과 같이 나눠서 관리함
- [1] WAC(ContextLoaderListener) : 웹에 종속적이지 않은 빈, 예) 서비스, DAO
- [2] WAC(DispatcherServlet) : 웹에 종속적인 빈, 예) 컨트롤러, 스프링 MVC관련 빈

context-common.xml (Root)

```
<context:component-scan base-package="com.herb.app">
    <context:exclude-filter expression="org.springframework.stereotype.Controller"
        type="annotation" />
</context:component-scan>
```

- 웹 관련 빈을 제외하고 빈을 등록
- **@Controller** 애노테이션을 사용한 클래스는 빈으로 등록하지 않도록 추가적인 설정

```
<!-- MULTIPART RESOLVERS -->
<!-- regular spring resolver -->
<bean id="spring.RegularCommonsMultipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="maxUploadSize" value="100000000" />
    <property name="maxInMemorySize" value="100000000" />
</bean>
```

```
<alias name="spring.RegularCommonsMultipartResolver" alias="multipartResolver" />
```

context:component-scan

- 스프링에 빈으로 등록할 클래스를 찾아서 빈으로 등록해주는 설정
- **com.herb.app** 패키지과 그 밑에 있는 모든 패키지에 들어있는 클래스에서 **@Component**, **@Controller**, **@Service**, **@Repository**를 사용한 클래스를 빈으로 등록하고 해당 빈에서 필요로 하는 의존성을 주입해줌
- 애노테이션 정보를 확인하고 특정 클래스를 빈으로 등록해주는 기능뿐 아니라 **@Autowired**, **@Inject**, **@Resource** 와 같은 애노테이션 정보를 확인하고 필요한 객체를 찾아서 주입해 주는 기능도 추가함

servlet-context.xml (Sub, web)

- 기존 dispatcher-servlet.xml
<annotation-driven />

DefaultAnnotationHandlerMapping, AnnotationMethodHandlerAdapter, ConfigurableWebBindingInitializer, 메시지컨버터, validator, <spring:eval> 을 위한 컨버전 서비스 노출용 인터셉터 등을 자동 등록해줌

```
<resources mapping="/resources/**" location="/resources/" />
<!--<resources mapping="/images/**" location="/images/" />
<resources mapping="/css/**" location="/css/" /> -->
<default-servlet-handler/>
```

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

```
<context:component-scan base-package="com.herb.app" use-default-filters="false">
    <context:include-filter expression="org.springframework.stereotype.Controller"
        type="annotation"/>
</context:component-scan>
```

<mvc:annotation-driven>

- 애노테이션 방식의 컨트롤러를 사용할 때 필요한 DispatcherServlet 전략빈을 자동으로 등록해 준다. 또, 최신 @MVC 지원 기능을 제공하는 빈도 함께 등록하고 전략 빈의 프로퍼티로 설정해준다. 라이브러리의 존재 여부를 파악해서 자동으로 관련 빈을 추가해주는 기능도 제공된다.

context-database.xml (Root)

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

빈 설정에서 사용하고 있는 일부 문자열을 프로퍼티스 파일을 사용해서 설정 파일 밖으로 빼낼수 있는 기능을 제공

```
<!-- DataSource -->
```

```
<alias name="dataSource-`${Globals.DbType}`" alias="dataSource"/>
```

```
<!-- MySQL -->
```

```
<bean id="dataSource-mysql" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
```

```
    <property name="driverClassName" value="${Globals.DriverClassName}"/>
```

```
    <property name="url" value="${Globals.Url}" />
```

```
    <property name="username" value="${Globals.UserName}"/>
```

```
    <property name="password" value="${Globals.Password}"/>
```

```
</bean>
```

database.properties 파일

Globals.DbType = oracle

Globals.DriverClassName=oracle.jdbc.driver.OracleDriver

Globals.Url=jdbc:oracle:thin:@303-0:1521:xe

```
<!-- Oracle -->
```

```
<bean id="dataSource-oracle" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
```

```
    <property name="driverClassName" value="${Globals.DriverClassName}"/>
```

```
    <property name="url" value="${Globals.Url}" />
```

```
    <property name="username" value="${Globals.UserName}"/>
```

```
    <property name="password" value="${Globals.Password}"/>
```

```
</bean>
```

context-database.xml (Root)

<!-- 트랜잭션 관련 -->

```
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
      p:dataSource-ref="dataSource" />
```

<tx:annotation-driven />

<tx:annotation-driven />

- **@Transactional** 이라는 애노테이션을 사용해서 트랜잭션 경계를 선언하고, 트랜잭션 특성을 선언하겠다는 것 (선언적인 트랜잭션 관리)

<!-- mybatis 설정 -->

```
<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="configLocation"
    value="classpath:/config/mybatis/${Globals.DbType}/mybatis-config.xml" />
  <property name="typeAliasesPackage" value="com.herb.app" />
  <property name="mapperLocations">
    <array>
      <value>classpath*/config/mybatis/mapper/${Globals.DbType}/**/*.xml</value>
    </array>
  </property>
</bean>
```

classpath: 해당 리소스를 클래스 패스(class, src)를 기준으로 찾겠다는 것



context-database.xml (Root)

<tx:annotation-driven />

- **@Transactional** 이라는 애노테이션을 사용해서 트랜잭션 경계를 선언하고, 트랜잭션 특성을 선언하겠다는 것 (선언적인 트랜잭션 관리)

- 클래스에 **@Transactional** 을 붙이면

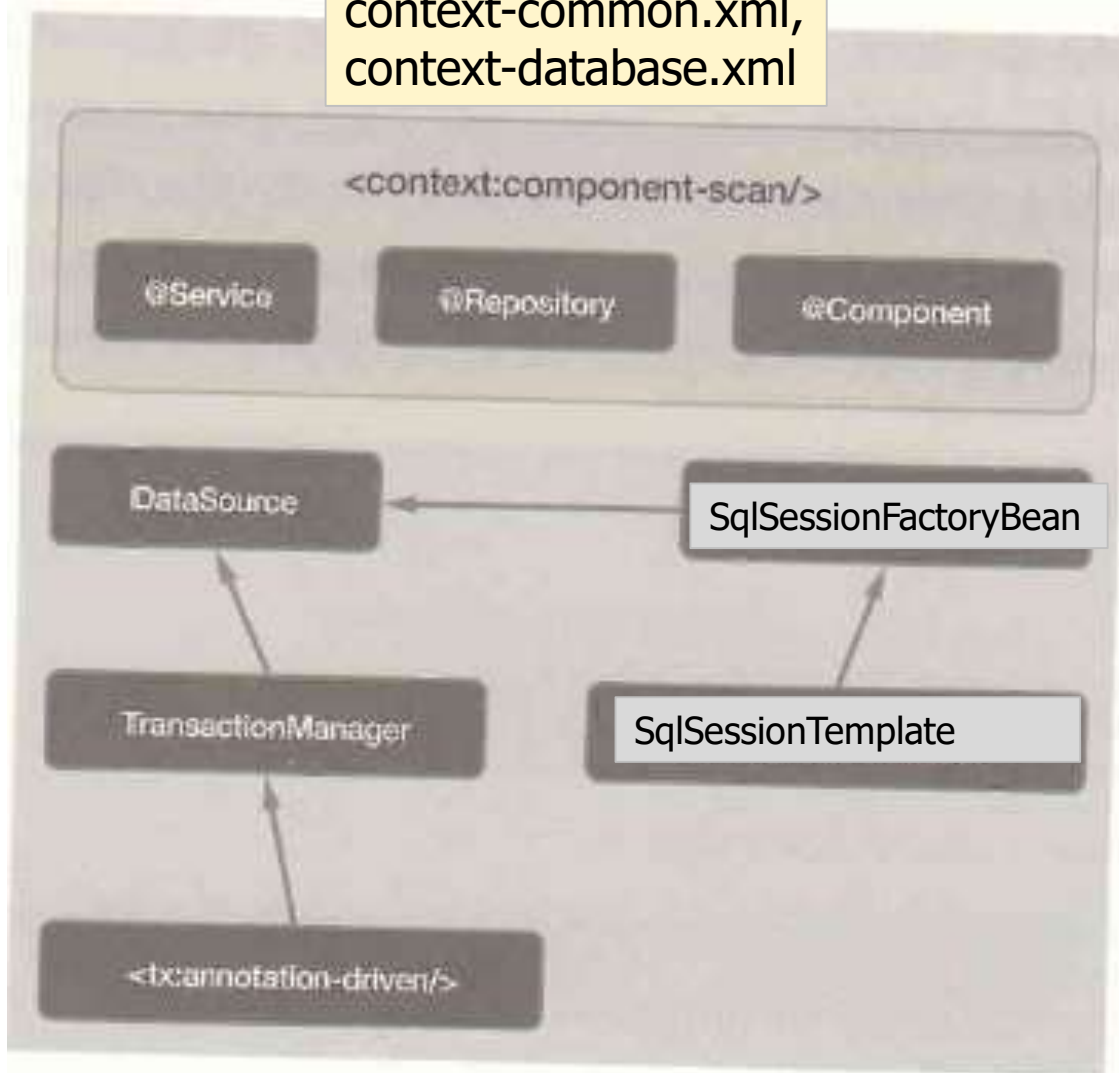
[1] 해당 클래스의 모든 메서드가 각각의 트랜잭션 경계가 됨

[2] 트랜잭션 내부에서 **RuntimeException**이 발생하면 해당 트랜잭션을 **rollback**하고, 그렇지 않으면 **commit**함

transactionManager 빈 – 스프링에서 **JDBC API**를 사용하여 트랜잭션을 처리할 때 사용하는 빈

context-common.xml

context-common.xml,
context-database.xml



servlet-context.xml

servlet-context.xml

```
<context:component-scan/>
```

@Controller

```
<mvc:annotation-driven />
```

ViewResolver



database.properties

```
# database.properties
```

```
# key = value
```

```
#Globals.DbType = mysql
```

```
#Globals.DriverClassName=com.mysql.jdbc.Driver
```

```
#Globals.Url=jdbc:mysql://localhost:1623/herbmall
```

```
#Globals.UserName=herb
```

```
#Globals.Password=herb123
```

```
Globals.DbType = oracle
```

```
Globals.DriverClassName=oracle.jdbc.driver.OracleDriver
```

```
Globals.Url=jdbc:oracle:thin:@userpc:1521:orcl
```

```
Globals.UserName=herb
```

```
Globals.Password=herb123
```



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>
<!-- 마이바티스의 작동 규칙정의 -->
<settings>
    <setting name="cacheEnabled" value="true"/>
    <setting name="useGeneratedKeys" value="false"/>
    <setting name="mapUnderscoreToCamelCase" value="true"/>
</settings>
</configuration>
```



Board.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="config.mybatis.mapper.oracle.board">
    <sql id="searchWhere">
        <where>
            <if test="searchKeyword !=null and searchKeyword !='' ">
                ${searchCondition} like '%'||#{searchKeyword}||'%'
            </if>
            $는 이전 ?로는 쓸 수 없었던 것에만 붙인다
        </where>
    </sql>

    <!--Insert -->
    <insert id="insertBoard" parameterType="boardVO">
        <selectKey resultType="int" keyProperty="no" order="BEFORE">
            select board_seq.nextval as no from dual
        </selectKey>
        insert into board (no, name, pwd, title, email, content)
        values(#{no},#{name},#{pwd},#{title},#{email}, #{content})
    </insert>
```

```
<update id="updateReadcount" parameterType="Integer">
    update board set readcount=readcount+1
    where no=#{no}
</update>
```

```
<select id="getBoard" parameterType="Integer" resultType="boardVO">
    select no, name, pwd, title, email, content, regdate, readcount
    from board
    where no=#{no}
</select>
```

firstRecordIndex : 각 페이지에서의 시작 인덱스, 0, 5, 10, 15
recordCountPerPage : 한 페이지에 보여줄 레코드 개수, 5

<!-- 해당 페이지의 레코드만 조회하도록 변경 -->

```
<select id="getBoardList" parameterType="searchVO" resultType="boardVO">
    SELECT *
    FROM (
        SELECT ROWNUM RNUM, ALL_LIST.*
        FROM (
            SELECT no, name, pwd, title, email, content, regdate, readcount,
                   (sysdate-regdate)*24 as newImgTerm
            FROM board
            <include refid="searchWhere"/>
            ORDER BY no DESC
        ) ALL_LIST
    )
    <![CDATA[
        WHERE RNUM > #{firstRecordIndex}
        AND RNUM <= #{firstRecordIndex} + #{recordCountPerPage} ]]>
</select>
```

```
<select id="getTotalRecord" parameterType="searchVO"    resultType="Integer">
    select count(*) from board
    <include refid="searchWhere"/>
</select>
```

```
<select id="getPwd" parameterType="Integer"    resultType="String">
    select pwd from board where no=#{no}
</select>
```

```
<!-- Update -->
<update id="updateBoard" parameterType="boardVO">
    update board set name=#{name}, title=#{title},
        content=#{content}, email=#{email}
    where no=#{no}
</update>
```

```
<!-- delete -->
<delete id="deleteBoard" parameterType="Integer">
    delete board where no=#{no}
</delete>
```



```

<!-- main notice -->
<select id="getMainNotice" resultType="boardVO">
<![CDATA[
        select no, title
        from
                (select no, title from board order by no asc)
        where rownum<=6
]]>
</select>
</mapper>

```

* 페이지 처리에 필요한 변수

한 페이지에 보여줄 레코드 개수 - **recordCountPerPage** 5 (기존의 **pageSize**)
블록 사이즈 - **blockSize** 10 (1~10)
총 레코드 개수 - **totalRecord** 총 17건
총 페이지 개수 - **totalPage** 총 4 페이지

- 현재 페이지를 이용해서 계산하는 변수

현재페이지 - **currentPage** 17
블록의 시작 페이지 - **firstPage** 1, 11, 21
블록의 마지막 페이지 - **lastPage** 10, 20, 30
선택한 페이지의 시작 인덱스 - **firstRecordIndex** (기존의 **curPos**과 동일) 0, 5, 10, ..



/inc/message.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"    pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:if test="${!empty msg }">
    <script type="text/javascript">
        alert("${msg}");
        location.href="<c:url value='${url}'/>";
    </script>
</c:if>
<c:if test="${empty msg }">
    <script type="text/javascript">
        location.href="<c:url value='${url}'/>";
    </script>
</c:if>
```



list.jsp

```
<%@page import="java.util.ArrayList"%>
<%@ page language="java" contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

<!DOCTYPE HTML>
<html lang="ko">
<head>
<title>글 목록</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<link rel="stylesheet" type="text/css" href="<c:url value='/css/mainstyle.css'/>" />
<link rel="stylesheet" type="text/css" href="<c:url value='/css/clear.css'/>" />
<link rel="stylesheet" type="text/css" href="<c:url value='/css/layout.css'/>" />
<link rel="stylesheet" type="text/css" href="<c:url value='/css/mystyle.css'/>" />
<style type="text/css">
    .divList, .divPage, .divSearch, .divList table{
        width:700px;
    }
    .divList{ margin:10px 0; }
    caption{
        visibility:hidden;
        font-size:0.1em; }
```

```

.divPage{
    text-align:center;
    padding:5px 0;
}
.divSearch{
    text-align:center;
    padding:5px 0 2px 0;
}
.divBtn{
    text-align:right;
    width:700px;
}
.divBtn a{
    font-size:1.0em;
}

```

```

</style>
<script type="text/javascript">
    function boardList(currentPage) {
        frmPage.currentPage.value=currentPage;
        frmPage.submit();
    }
</script>
</head>
<body>
<h2>공지사항</h2>
<c:if test="${!empty searchVO.searchKeyword}">
    검색어 : ${searchVO.searchKeyword}, ${pagingInfo.totalRecord }건이 검색되었습니다.
</c:if>
<form name="frmPage" method="post" action='<c:url value="/board/boardList.do"></c:url>'>
    <input type="hidden" name="searchCondition" value="${param.searchCondition }">
    <input type="hidden" name="searchKeyword" value="${param.searchKeyword }">

```

```

<input type="hidden" name="currentPage" >
</form>
<div class="divList">
<table cellpadding="0" cellspacing="0" class="box2"
    summary="기본 게시판에 관한 표로써, 번호, 제목, 작성자, 작성일, 조회수에 대한 정보를 제공합니다.">
<caption>기본 게시판</caption>
<colgroup>
    <col style="width:10%; height:20px" /><col style="width:50%; height:20px" />
    <col style="width:15%; height:20px" /><col style="width:15%; height:20px" />
    <col style="width:10%; height:20px" />
</colgroup>
<thead>
    <tr>
        <th scope="col">번호</th> <th scope="col">제목</th> <th scope="col">작성자</th>
        <th scope="col">작성일</th> <th scope="col">조회수</th>
    </tr>
</thead>
<tbody> <!--게시판 내용 반복문 시작 -->
    <c:forEach var="bean" items="${boardList}">
        <tr onMouseOver="this.style.backgroundColor='lightblue';this.style.cursor='hand'"
            onMouseOut="this.style.backgroundColor=''" align="center">
            <td>${bean.no }</td>
            <td align="left"> <!--제목이 30자 이상인 경우 처리 -->
                <a href='<c:url value="/board/boardCountUpdate.do?no=${bean.no}"></c:url>'>
                    <c:if test="${fn:length(bean.title)>30}">
                        ${fn:substring(bean.title, 0, 30) }....
                    </c:if>
                    <c:if test="${fn:length(bean.title)<=30}">
                        ${bean.title}
                    </c:if>
                </a>
            </td>
        </tr>
    </c:forEach>
</tbody>
</table>
</div>

```

```

        <!-- new 이미지 처리 -->
        <c:if test="${bean.newImgTerm < 24}">
            <img src='<c:url value="/images/new.gif"></c:url>' border="0">
        </c:if>
        </td>
    <td><c:if test="${ !empty bean.email}">
        <a href=mailto:${bean.email}> ${bean.name}</a> </c:if>
        <c:if test="${empty bean.email}"> ${bean.name} </c:if>  </td>
    <td><fmt:formatDate value="${bean.regdate }" type="date" pattern="yyyy-MM-dd"/></td>
    <td>${bean.readcount}</td>
</tr>
</c:forEach>  <!--반복처리 끝 -->
</tbody>
</table>
</div>
<div class="divPage">
    <!-- 페이지 번호 추가 -->
    <c:if test="${pagingInfo.firstPage>1 }">
        <a href="#" onclick="boardList(${pagingInfo.firstPage-1})">
            <img src='<c:url value="/images/first.JPG" />' border="0">  </a>
    </c:if>
    <!-- [1][2][3][4][5][6][7][8][9][10] -->
    <c:forEach var="i" begin="${pagingInfo.firstPage }" end="${pagingInfo.lastPage }">
        <c:if test="${i<=pagingInfo.totalPage}">
            <c:if test="${i==pagingInfo.currentPage }">
                <span style="color:blue;font-weight:bold">${i }</span>
            </c:if>
            <c:if test="${i!=pagingInfo.currentPage }">
                <a href="#" onclick="boardList(${i})">[ ${i }]</a>
            </c:if>
        </c:if>
    </c:forEach>

```

```
<c:if test="${pagingInfo.lastPage<pagingInfo.totalPage}">
    <a href="#" onclick="boardList(${pagingInfo.lastPage+1})">
        " border="0">
    </a>
</c:if> <!-- 페이지 번호 끝 -->
```

```
</div>
```

```
<div class="divSearch">
```

```
<form name="frmSearch" method="post" action='<c:url value="/board/boardList.do"></c:url>'>
```

```
<select name="searchCondition">
```

```
<option value="title" <c:if test="${param.searchCondition=='title' }">
```

```
    selected</c:if>
    제목</option>
```

```
<option value="content" <c:if test="${param.searchCondition=='content' }">
```

```
    selected</c:if>
    내용</option>
```

```
<option value="name" <c:if test="${param.searchCondition=='name' }">
```

```
    selected</c:if>
    작성자</option>
```

```
</select>
```

```
<input type="text" name="searchKeyword" size="20" value="${param.searchKeyword}" title="검색어 입력">
```

```
<input type="submit" value="검색">
```

```
</form>
```

```
</div>
```

```
<div class="divBtn">
```

```
<a href='<c:url value="/board/boardWrite.do"></c:url>'>글쓰기</a>
```

```
</div>
```

```
</body>
```

```
</html>
```



BoardVO

```
package com.herb.app.board.model;
```

```
import java.sql.Timestamp;
```

```
public class BoardVO {
```

```
    private int no;
```

```
    private String name;
```

```
    private String pwd;
```

```
    private String title;
```

```
    private String email;
```

```
    private Timestamp regdate;
```

```
    private int readcount;
```

```
    private String content;
```

```
    //24시간 이내의 글에 new 이미지 추가관련
```

```
    private long newImgTerm;
```

```
    ...
```




SearchVO

```
/** 검색 정보를 담고 있는 Bean, 페이징 처리 관련 변수 포함 */
public class SearchVO {
    /** 검색조건 */
    private String searchCondition = "";
    /** 검색키워드 */
    private String searchKeyword = "";

    /** 현재 페이지 */
    private int currentPage = 1;
    /**페이지 별 레코드 개수, 한 페이지에 보여줄 레코드 수 (pageSize) */
    private int recordCountPerPage;
    /**블럭당 보여질 페이지 수 */
    private int blockSize;

    /** 시작 인덱스 */
    private int firstRecordIndex = 1;  //(기존 curPos)
    /** 끝 인덱스 */
    private int lastRecordIndex = 1;
```

PaginationInfo

```
public class PaginationInfo {
    /** Required Fields
        currentPage : 현재 페이지
        recordCountPerPage : 페이지당 보여질 레코드수
        blockSize : 블록당 보여질 페이지 수
        totalRecord : 총 레코드 수
    */
    private int currentPage; //현재 페이지
    private int recordCountPerPage; //pageSize 페이지당 보여질 레코드수
    private int blockSize; //블록당 보여질 페이지 수
    private int totalRecord; // 총 레코드 수
    .....

    /** Not Required Fields - currentPage를 이용하여 계산하는 변수 */
    private int totalPages; //총 페이지수
    private int firstPage; //블록당 시작 페이지, 1, 11, 21, 31, ...
    private int lastPage; //블록당 마지막 페이지 10, 20, 30, 40, ...
    private int firstRecordIndex; //페이지당 시작 인덱스 0, 5, 10, 15 ...
    private int lastRecordIndex; //페이지당 마지막 인덱스 5,10,15,20....
    public int getTotalPage() {
        totalPages=(int)Math.ceil((float)totalRecord/recordCountPerPage);
        //totalPage = ((getTotalRecord()-1)/getRecordCountPerPage()) + 1;
        return totalPages;
    }
}
```

```
public int getFirstPage() {  
    firstPage= currentPage-((currentPage-1)%blockSize);  
    //firstPage = ((getCurrentPage()-1)/getBlockSize())*getBlockSize() + 1;  
    return firstPage;  
}
```

```
public int getLastPage() {  
    lastPage = getFirstPage()+(blockSize-1);  
    if(lastPage > getTotalPage()){  
        lastPage = getTotalPage();  
    }  
    return lastPage;  
}
```

```
public int getFirstRecordIndex() {  
    //curPos=(currentPage-1)*pageSize;  
    firstRecordIndex = (getCurrentPage() - 1) * getRecordCountPerPage();  
    return firstRecordIndex;  
}
```

```
public int getLastRecordIndex() {  
    lastRecordIndex = getCurrentPage() * getRecordCountPerPage();  
    return lastRecordIndex;  
}
```



BoardDAO

```
public interface BoardDAO {  
    public void insertBoard(BoardVO board) ;  
    public int updateReadCount(int no) ;  
    public ArrayList<BoardVO> getBoardList(BoardVO boardVO) ;  
    public BoardVO getBoard(int no) ;  
    public String getPwd(int no) ;  
    public int updateBoard(BoardVO board) ;  
    public int deleteBoard(int no) ;  
    public List<BoardVO> getMainNotice() ;  
    public int selectTotalRecord(SearchVO searchVO) ;  
}
```



BoardService

```
public interface BoardService {  
    public void insertBoard(BoardVO board) ;  
    public int updateReadCount(int no) ;  
    public ArrayList<BoardVO> getBoardList(BoardVO boardVO) ;  
    public BoardVO getBoard(int no) ;  
    public String getPwd(int no) ;  
    public int updateBoard(BoardVO board) ;  
    public int deleteBoard(int no) ;  
    public List<BoardVO> getMainNotice() ;  
    public boolean checkPwd(int no, String pwd) ;  
    public int selectTotalRecord(SearchVO searchVO) ;  
}
```

@Repository

```
public class BoardDAOMybatis extends SqlSessionDaoSupport implements BoardDAO {
    private static final Logger logger = LoggerFactory.getLogger(BoardDAOMybatis.class);
    private String namespace="config.mybatis.mapper.oracle.board";

    public BoardDAOMybatis() {
        logger.info("BoardDAOMybatis 생성자");
    }

    public void insertBoard(BoardVO board) {
        getSession().insert(namespace + ".insertBoard", board);
    }

    public int updateReadCount(int no) {
        return getSession().update(namespace + ".updateReadcount", no);
    }

    public List<BoardVO> getBoardList(SearchVO searchVO) {
        List<BoardVO> list=
            getSession().selectList(namespace + ".getBoardList", searchVO);
        return list;
    }

    public BoardVO getBoard(int no) {
        BoardVO board=(BoardVO)getSession().selectOne(namespace + ".getBoard",no);
        return board;
    }
}
```

BoardDAOMybatis

@Repository 이용하여 스프링에 빈으로 등록

@Autowired
private SqlSessionTemplate sqlSession;

SqlSession org.mybatis.spring.support.SqlSessionDaoSupport.getSession()

```
public String getPwd(int no) {
    String pwd
        =(String)getSession().selectOne(namespace + ".getPwd", no);
    return pwd;
}

public int updateBoard(BoardVO board) {
    return getSession().update(namespace + ".updateBoard", board);
}

public int deleteBoard(int no) {
    return getSession().delete(namespace + ".deleteBoard", no);
}

public List<BoardVO> getMainNotice() {
    List<BoardVO> list = getSession().selectList(namespace + ".getMainNotice");
    return list;
}

@Override
public int selectTotalRecord(SearchVO searchVO) {
    Integer totCount
        =(Integer)getSession().selectOne(namespace + ".getTotalRecord", searchVO);
    return totCount;
}

} //class
```

@Service
@Transactional

애플리케이션의 주요 로직에 트랜잭션 경계를 설정하고,
BoardService 인터페이스를 구현한 구현체

BoardServiceImpl

```
public class BoardServiceImpl implements BoardService{
```

```
    private @Autowired BoardDAO boardDAO;
```

```
    private static final Logger logger = LoggerFactory.getLogger(BoardServiceImpl.class);
```

```
    public void insertBoard(BoardVO board) {  
        boardDAO.insertBoard(board);  
    }
```

```
    public List<BoardVO> getBoardList(BoardVO boardVO) {  
        return boardDAO.getBoardList(boardVO);  
    }
```

```
    public BoardVO getBoard(int no) {  
        return boardDAO.getBoard(no);  
    }
```

```
    public boolean checkPwd(int no, String pwd) {  
        String dbPwd = boardDAO.getPwd(no);  
        boolean result=false;  
        if(dbPwd.equals(pwd)){  
            result=true;  
        }  
        return result;  
    }
```

```
/*public void setBoardDAO(BoardDAO boardDAO) {  
    this.boardDAO = boardDAO;  
}*/
```

@Transactional – 클래스와 메서드에서 사용 가능
클래스에 사용하면 해당 클래스의 모든 public 메서드에 트랜잭션을 적용함

@Transactional – 단순히 트랜잭션 경계와 그 트랜잭션의 속성을 나타내는 설정일 뿐이고,
실제로 이 애노테이션을 읽어서 트랜잭션 처리를 하는 클래스는 transactionManager와 <tx:annotation-driven />으로 등록되는 빈 이다


```
public int updateBoard(BoardVO board) {  
    return boardDAO.updateBoard(board);  
}  
  
public int deleteBoard(int no) {  
    return boardDAO.deleteBoard(no);  
}  
  
public int updateReadCount(int no) {  
    return boardDAO.updateReadCount(no);  
}  
  
public List<BoardVO> getMainNotice() {  
    return boardDAO.getMainNotice();  
}  
  
public int selectTotalRecord(SearchVO searchVO) {  
    return boardDAO.selectTotalRecord(searchVO);  
}  
}  
} //class
```

```
/*public void setBoardService(BoardService boardService) {  
    this.boardService = boardService;  
}*/
```

BoardWriteController

```
@Controller
```

```
@RequestMapping("/board/boardWrite.do")
```

```
public class BoardWriteController {
```

```
    @Autowired BoardService boardService;
```

```
    private static final Logger logger=LoggerFactory.getLogger(BoardWriteController.class);
```

```
    @RequestMapping(method=RequestMethod.GET)
```

```
    public String insertGet(){
```

특정 요청을 처리할 메서드를 스프링 MVC에서는 handler 라고 부름

```
        logger.info("get : insertGet()");  
        String viewName = "/board/write";  
        return viewName;
```

```
    }
```

@Autowired 애노테이션을 사용해서 **BoardService** 타입의 객체를 주입 받는다

클래스와 메서드에 모두 **@RequestMapping** 이 설정되어 있을 경우
- 클래스에 붙인 **@RequestMapping** 정보와 메서드에 붙인 **@RequestMapping** 정보를 조합해서 매핑함

```
    @RequestMapping(method=RequestMethod.POST)
```

```
    public String insertPost(@ModelAttribute BoardVO bBean){
```

```
        logger.info("POST : insertPost(), bBean :{}", bBean);  
        int n = boardService.insertBoard(bBean);  
        logger.info("입력 결과 n={}", n);  
        return "redirect:/board/boardList.do";
```

```
    }
```

```
}//class
```

```
@Controller
public class BoardListController {
    @Autowired BoardService boardService;
    private static final Logger logger = LoggerFactory.getLogger(BoardListController.class);

    @RequestMapping("/board/boardList.do")
    public String getBoardList(SearchVO searchVO, Model model) {
        logger.info("getBoardList() :searchVO={}", searchVO);
        searchVO.setRecordCountPerPage(Utility.RECORD_COUNT_PER_PAGE);

        /** paging setting */
        PaginationInfo paginationInfo = new PaginationInfo();
        paginationInfo.setCurrentPage(searchVO.getCurrentPage());
        paginationInfo.setRecordCountPerPage(searchVO.getRecordCountPerPage());
        paginationInfo.setBlockSize(searchVO.getBlockSize());

        searchVO.setFirstRecordIndex(paginationInfo.getFirstRecordIndex());

        List<BoardVO> boardList= boardService.getBoardList(searchVO);
        logger.info("getBoardList() 결과, list.size()={}, boardList.size()");

        int totCnt = boardService.selectTotalRecord(searchVO);
        paginationInfo.setTotalRecord(totCnt);

        model.addAttribute("boardList", boardList);
        model.addAttribute("pagingInfo", paginationInfo);
        return "/board/list";
    }
}
```

@Controller

public class BoardViewController{

BoardViewController

```
    @Autowired BoardService boardService;
    private static final Logger logger = LoggerFactory.getLogger(BoardViewController.class);
    @RequestMapping("/board/boardView.do")
    public String getBoard(@RequestParam(defaultValue="0") int no, ModelMap modelMap){
        logger.info("getBoard() :{}", no);
        if(no==0){
            modelMap.addAttribute("msg", "잘못된 url 정보입니다!!, no가 없습니다");
            modelMap.addAttribute("url", "boardList.do");
            return "inc/message";
        }

        BoardVO bean=boardService.getBoard(no);

        String content = bean.getContent();
        //content 줄바꿈 처리
        if(content !=null && content!=""){
            content = content.replace("WrWn", "<br>"); //엔터를 <br> 태그로 치환하여 줄바꿈처리
        }
        bean.setContent(content);
        logger.info("getBoard() 결과, bean={}", bean);

        //결과 저장
        modelMap.addAttribute("bean", bean);
        return "/board/view";
    }
}
```



BoardCountUpdateController

@Controller

```
public class BoardCountUpdateController{  
    @Autowired BoardService boardService;  
    private static final Logger logger =  
        LoggerFactory.getLogger(BoardCountUpdateController.class);  
  
    @RequestMapping("/board/boardCountUpdate.do")  
    public String updateCount(int no){  
        logger.info("updateCount() :{}", no);  
  
        int n=boardService.updateReadCount(no);  
        logger.info("updateReadCount() 결과, n={}", n);  
  
        return "redirect:/board/boardView.do?no="+no;  
    }  
}
```



BoardEditController

```
@Controller
@RequestMapping("/board/boardEdit.do")
public class BoardEditController {
    @Autowired BoardService boardService;
    private static final Logger logger = LoggerFactory.getLogger(BoardEditController.class);

    @RequestMapping(method=RequestMethod.GET)
    public String editGet(int no, Model model){
        logger.info("get : editGet(), no={}", no);

        BoardVO bean= boardService.getBoard(no);
        logger.info("editGet:getBoard() 결과, bean={}", bean);

        model.addAttribute("bean", bean);
        return "/board/edit";
    }
}
```

Model 타입의 매개변수 - 이 핸들러에서 뷰로 넘길 객체들을 담을 수 있는 객체 일종의 **Map** 형식으로 키/값 쌍으로 뷰에서 참조할 객체를 담을 수 있음

핸들러에서 리턴타입이 **void** 인 경우 - 기본 뷰 이름을 사용



BoardEditController

```
@RequestMapping(method=RequestMethod.POST)
public String editPost(@ModelAttribute BoardVO bBean, Model model){
    logger.info("POST : editPost(), bBean :{}",bBean);

    String viewName="";
    if(boardService.checkPwd(bBean.getNo(), bBean.getPwd())){
        int n = boardService.updateBoard(bBean);
        logger.info("editPost:updateBoard() 결과 n={}", n);
        viewName="redirect:/board/boardView.do?no="+bBean.getNo();
    }else{
        model.addAttribute("msg", "비밀번호 불일치!!");
        model.addAttribute("url",
            "/board/boardEdit.do?no="+bBean.getNo());
        viewName="/inc/message";
    }

    return viewName;
}

}

} //class
```



BoardDeleteController

```
@Controller
@RequestMapping("/board/boardDelete.do")
public class BoardDeleteController {
    @Autowired BoardService boardService;
    private static final Logger logger = LoggerFactory.getLogger(BoardDeleteController.class);
    @RequestMapping(method=RequestMethod.GET)
    public String deleteGet(){
        logger.info("get : deleteGet()");
        return "/board/delete";
    }
    @RequestMapping(method=RequestMethod.POST)
    public String deletePost(int no, String pwd, Model model){
        logger.info("POST : deletePost(), no ={}, pwd={}", no, pwd);
        String viewName="";
        if(boardService.checkPwd(no, pwd)){
            int n= boardService.deleteBoard(no);
            logger.info("deletePost:deleteBoard() 결과, n={}", n);
            viewName="redirect:/board/boardList.do";
        }else{
            model.addAttribute("msg", "비밀번호 불일치!!");
            model.addAttribute("url", "/board/boardDelete.do?no="+no);
            viewName="/inc/message";
        }
        return viewName;
    }
}
```




참고



DispatcherServlet 설정과 ApplicationContext 의 관계

- DispatcherServlet – 클라이언트의 요청을 중앙에서 처리하는 스프링 MVC의 핵심 구성 요소
 - web.xml 파일에 한 개 이상의 DispatcherServlet 을 설정할 수 있으며, 각 DispatcherServlet 은 한 개의 WebApplicationContext 를 갖게 됨
 - 또한 각 DispatcherServlet 이 공유할 수 있는 빈을 설정할 수도 있음

DispatcherServlet 설정

- DispatcherServlet 은 기본적으로 웹 어플리케이션의 /WEB-INF/디렉토리에 위치한 [서블릿이름]-servlet.xml 파일로부터 스프링 설정 정보를 읽어옴
- 한 개 이상의 설정 파일을 사용해야 하는 경우나 기본 설정 파일 이름이 아닌 다른 이름의 설정 파일을 사용하고 싶은 경우
 - DispatcherServlet 을 설정할 때 contextConfigLocation 초기화 파라미터에 설정 파일 목록을 지정하면 됨

```
<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/main.xml
            /WEB-INF/bbs.xml
        </param-value>
    </init-param>
</servlet>
```

- contextConfigLocation 초기화 파라미터는 설정 파일 목록을 값으로 갖는데,
- 각 설정파일은 콤마(,), 공백 문자(" "), 탭(□t), 줄 바꿈(□n), 세미콜론(";") 을 이용하여 구분
- 각 설정 파일의 경로는 웹 어플리케이션 루트 디렉토리를 기준으로 함



웹 어플리케이션을 위한 ApplicationContext 설정

- DispatcherServlet 은 그 자체가 서블릿이기 때문에 한 개 이상의 DispatcherServlet 을 설정하는 것이 가능
- 예) 웹 페이지를 위한 DispatcherServlet 과 REST 기반의 웹 서비스 연동을 위한 DispatcherServlet 을 나누어 설정했다면.

```
<servlet>
    <servlet-name>front</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/front.xml</param-value>
    </init-param>
</servlet>

<servlet>
    <servlet-name>rest</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/rest.xml</param-value>
    </init-param>
</servlet>
```

웹 어플리케이션을 위한 ApplicationConext 설정

- 이 경우 두 DispatcherServlet은 각각 별도의 WebApplicationContext를 생성하게 됨
- front DispatcherServlet 은 front.xml 설정 파일을 사용하고, rest DispatcherServlet 은 rest.xml 설정 파일을 사용=> front.xml 에서는 rest.xml 에 설정된 빈 객체를 사용할 수 없게 됨
- 웹 어플리케이션에서 컨트롤러는 클라이언트의 요청을 비즈니스 로직을 구현한 서비스 레이어를 이용하여 처리하는 것이 일반적
- 서비스 레이어는 영속성 레이어(db연동)를 사용해서 데이터 접근을 처리
- front 관련 컨트롤러와 rest 관련 컨트롤러는 동일한 서비스 레이어에 대한 의존 관계를 가질 것임

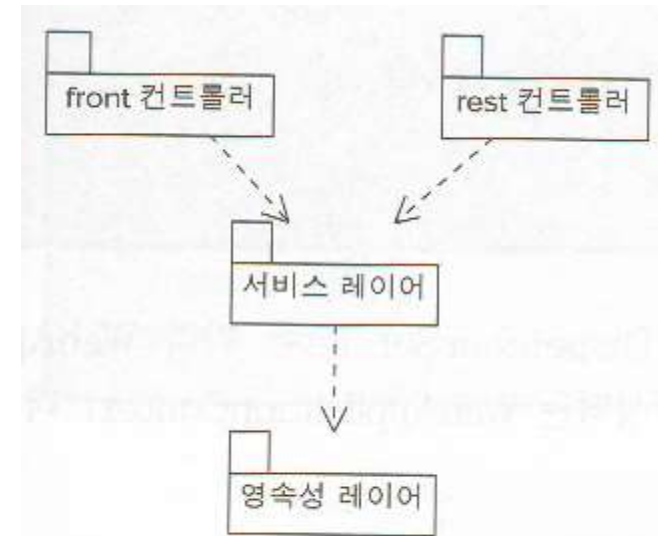


그림 6.5 웹 어플리케이션의 전형적인 레이어 구성



웹 어플리케이션을 위한 ApplicationConext 설정

- 서로 다른 DispatcherServlet 이 공통 빈을 필요로 하는 경우
 - ContextLoaderListener 를 사용하여 공통으로 사용될 빈을 설정할 수 있게 됨
 - ContextLoaderListener 를 ServletListener 로 등록하고, contextConfigLocation 컨텍스트 파라미터를 이용하여 공통으로 사용될 빈 정보를 담고 있는 설정 파일 목록을 지정하면 됨

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

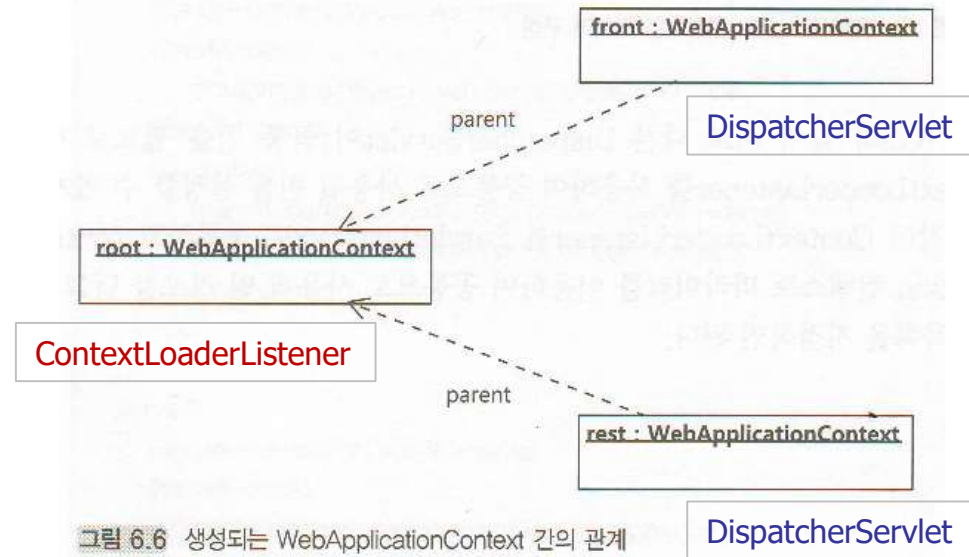
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<servlet>
    <servlet-name>front</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>

<servlet>
    <servlet-name>rest</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
```

웹 어플리케이션을 위한 ApplicationConext 설정

- ContextLoaderListener 와 DispatcherServlet 은 각각 WebApplicationContext 객체를 생성하는데, 이때 생성되는 WebApplicationContext 객체 간의 관계는 아래 그림과 같다



- ContextLoaderListener 가 생성하는 WebApplicationContext 는 웹 어플리케이션에서 루트 컨텍스트가 되며, DispatcherServlet 이 생성하는 WebApplicationContext 는 루트 컨텍스트를 부모로 사용하는 자식 컨텍스트가 됨
 - 이때 자식은 root 가 제공하는 빈을 사용할 수 있음
 - ContextLoaderListener 는 contextConfigLocation 컨텍스트 파라미터를 명시하지 않으면 /WEB-INF/applicationContext.xml 을 설정 파일로 사용함
 - 클래스패스에 위치한 파일로부터 설정 정보를 읽어오고 싶으면 classpath: 접두어 사용



웹 어플리케이션을 위한 ApplicationContext 설정

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath:config/service.xml
        classpath:common.xml
        /WEB-INF/config/message_conf.xml
    </param-value>
</context-param>
```


IoC 컨테이너

- 스프링 컨테이너 - IoC 컨테이너
 - 스프링 애플리케이션에서는 **오브젝트의 생성과 관계설정, 사용, 제거 등의 작업을 애플리케이션 코드 대신 독립된 컨테이너가 담당함**
 - 이를 컨테이너가 코드 대신 오브젝트에 대한 제어권을 갖고 있다고 해서 **IoC**라고 부름
 - 스프링에선 IoC를 담당하는 컨테이너를 빈 팩토리 또는 애플리케이션 컨텍스트라고 부르기도 함
 - 오브젝트의 생성과 오브젝트 사이의 런타임 관계를 설정하는 DI 관점으로 볼때는 컨테이너를 빈 팩토리하고 함
 - DI를 위한 빈 팩토리에 엔터프라이즈 애플리케이션을 개발하는데 필요한 여러 가지 컨테이너 기능을 추가한 것을 **애플리케이션 컨텍스트**라고 부름
 - 애플리케이션 컨텍스트 - 그 자체로 IoC와 DI를 위한 빈 팩토리이면서 그 이상의 기능을 가졌다
 - 스프링의 IoC 컨테이너는 일반적으로 애플리케이션 컨텍스트를 말함
- 스프링 애플리케이션은 최소한 하나 이상의 IoC 컨테이너, 즉 애플리케이션 컨텍스트 오브젝트를 갖고 있다.



WebApplicationContext

- 스프링 애플리케이션에서 가장 많이 사용되는 애플리케이션 컨텍스트
- ApplicationContext를 확장한 인터페이스
- 웹 환경에서 사용할 때 필요한 기능이 추가된 애플리케이션 컨텍스트