00장 스프링 개발 환경 구축

0.1 개발을 위한 준비

■ B.1.1 내용을 참고한다.

0.2 스프링 프로젝트 생성과 실행

0.2.1 'ex00Lab' 프로젝트 생성

■ File -> New -> Spring Legacy Project를 선택하고 가장 하단에 있는 Spring MVC 프로젝트를 선택한다. 프로젝트 설정은 다음과 같다.
- Project name: ex00Lab
- Templates: Spring MVC Project
- Top-level package: org.zerock.web

[꿀팁] 스프링 프로젝트를 시작하는 두 가지 방법

	Spring Starter Project (Spring Boot)	Spring Legacy Project
장점	별도의 설정이 필요 없다. WAS없이 실행이 가능하다. 로딩 시간이 짧아서 테스트 하기에 편하다.	현재까지 실무에서 많이 사용되고 있다. 다양한 자료가 존재한다. 기존 프로젝트를 이해하는데 도움이 된다. 모든 버전의 스프링에서 사용할 수 있다.
단점	기존의 설정과 다른 방식으로 사용한다. JSP 설정 등은 별도로 해야 한다.	초반 테스트 환경 구성 등이 어렵다. WAS와 연동하는 경우 결과 확인에 많은 리소스를 소모한다.

0.2.2 스프링 버젼 변경

```
// pom.xml의 수정 내용
        <properties>
                <java-version>11</java-version>
                <org.springframework-version>4.2.4.RELEASE</org.springframework-version>
                <org.aspectj-version>1.6.10</org.aspectj-version>
                <org.slf4j-version>1.6.6</org.slf4j-version>
        </properties>
```

0.2.3 Java version 변경

■ pom.xml 파일의 자바 라이브러리 버전 변경
- 생성된 프로젝트의 JRE System Library를 보면 'JavaSE-1.6' 버전으로 생성되었으므로, 스프링 5.x 버전을 이용하고 싶다면 JDK 11을 사용한다.

```
// pom.xml의 수정 내용
          <plugin>
              <groupId>org.apache.maven.plugins</groupId>
              <artifactId>maven-compiler-plugin</artifactId>
              <version>3.5.1</version>
              <configuration>
```

```
                <source>11</source>
                <target>11</target>
                <compilerArgument>-Xlint:all</compilerArgument>
                <showWarnings>true</showWarnings>
                <showDeprecation>true</showDeprecation>
            </configuration>
        </plugin>
```

■ Project Facets의 자바 버전 변경
- 해당 프로젝트를 선택한 상태에서 마우스 오른쪽 버튼을 클릭하여 맨 아래에 'Properties > Project Facets'를 선택하여 Java 버전을 우리가 설치한 11 버전으로 수정한다. 그리고 오른쪽에 'Runtimes' 탭을 선택하여 'Apache Tomcat v9.0'을 체크한다.
- 'Java Buil Path > Libaries' 탭을 클릭하여 아래 내용을 포함하는지 확인한다.
  - Apache Tomcat v9.0 [Apache Tomcat v89.0]
  - JRE System Library [jre]
  - ~~JUnit 5~~
  - Maven Dependencies

0.2.4 Tomcat을 이용한 프로젝트 실행 확인

■ 프로젝트의 'Run As > Run on Server'를 이용해서 처리한다.

0.2.5 라이브러리의 초기화

■ STS는 기본적으로 maven을 내장하고 있기 때문에 프로젝트 생성 시 바로 maven을 이용해서 스프링 관련 라이브러리가 다운로드 된다.
■ Maven이 다운로드를 받은 라이브러리를 사용하는 경로는 ' C:\Users\사용자계정\.m2'라는 이름의 폴더이다.

[꿀팁] 스프링 프로젝트 코드 실행의 문제점 및 해결책

■ 스프링 프로젝트는 필요한 라이브러리를 메이븐을 통해 다운로드 받아 실행하므로 간혹 네트워크 문제로 제대로 받아오지 못해 실행되지 않는 경우가 발생한다. 이때는 기존에 잘 수행되었던 ".m2" 디렉토리를 덮어쓰기하여 사용하도록 한다.

0.3 MySQL 설치 및 설정

0.3.1 MySQL의 설치

■ 다로운드: http://dev.mysql.com/downloads/

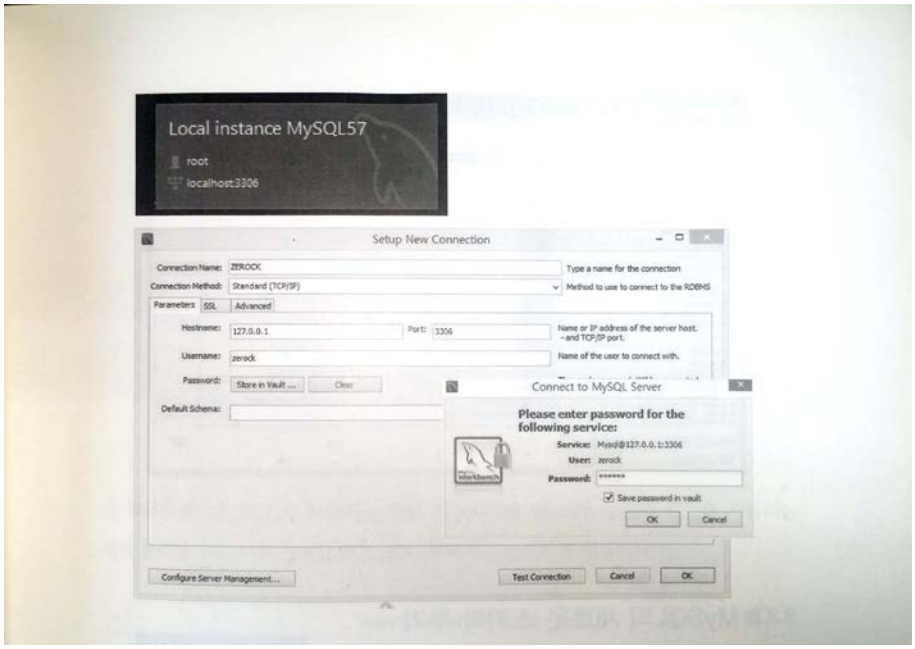
0.3.2 MySQL Workbench의 활용

■ MySQL의 새로운 스키마 추가

```
create database book_ex default character set utf8;
```

```
create user 'zerock'@'localhost' identified by 'zerock';
grant all privileges on book_ex.* to 'zerock'@'localhost';

create user 'zerock'@'%' identified by 'zerock';
grant all privileges on book_ex.* to 'zerock'@'%';
```

■ 계정 연결 설정
- Connection Name: zerock@book_ex
- Hostname: 127.0.0.1
- Port: 3306
- Username: zerock
- Password: zerock
- Default Schema: book_ex



■ DB 테스트: MySQL의 설정과 Workbench의 설정이 완료되면 간단히 테이블을 작성하고 SQL 문장에 대한 테스트를 진행한다. 이때 가장 중요한 작업은 문자열이 제대로 UTF-8로 지정됐는지 확인하는 것이다.

```
C:\Program Files\MySQL\MySQL Server 5.7\bin>mysql --user=zerock --password book_ex
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5095
Server version: 5.7.18-log MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> status
--------------
```

```
mysql  Ver 14.14 Distrib 5.7.18, for Win64 (x86_64)

Connection id:          5095
Current database:       book_ex
Current user:           zerock@localhost
SSL:                    Not in use
Using delimiter:        ;
Server version:         5.7.18-log MySQL Community Server (GPL)
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    utf8
Db     characterset:    utf8
Client characterset:    euckr
Conn.  characterset:    euckr
TCP port:               3306
Uptime:                 31 days 14 hours 52 min 1 sec

Threads: 11  Questions: 64028  Slow queries: 0  Opens: 415  Flush tables: 1  Open tables: 379  Queri
es per second avg: B.023
--------------

mysql>
```

0.4 MySQL 연결 테스트

- jUnit은 자바 프로그래밍 언어용 유닛 테스트(Unit Test) 프레임워크이다. 이와 관련해서는 다음과 같은 기본적인 사항만 알고 있으면 된다.
  - @Test : 테스트해야 하는 내용을 메소드 안에 작성하고 메소드 위에 @Test 어노테이션을 추가하면, jUnit은 해당 메소드를 테스트용 코드로 간주하고 테스트를 진행할 수 있게 한다.
  - @Before : 모든 테스트 작업에 앞서 준비되어야 하는 내용을 작성해서 메소드에 추가하는 어노테이션이다.
  - @After : 테스트 작업이 끝난 후 자동으로 실행되는 메소드에 추가하는 어노테이션이다.
  - org.junit.Assert.assertxxx : 테스트 중에 발생되는 값을 확신하는 용도로 사용한다.

0.4.1 MySQL 테스트 관련 라이브러리

- MySQL의 JDBC 연결을 위한 드라이버인 MySQL Connector/J라는 JDBC 라이브러리가 필요한다.
- 현재 사용하는 MySQL의 버전이 5.6이하의 경우는 5.x의 버전만을 이용할 수 있으므로 주의해야 한다.

```
// pom.xml 파일 추가
                <dependency>
                        <groupId>mysql</groupId>
                        <artifactId>mysql-connector-java</artifactId>
                        <version>5.1.36</version>
                </dependency>
```

0.4.2 JDK의 버전 변경과 jUnit의 버전 변경

■ 'Properties for ex00Lab > Project Facets > Java Version 1.6 -> 11'으로 변경하고 적용한다.
■ 또한 pom.xml 파일의 내용을 아래와 같이 수정한다.

```
// pom.xml 파일 추가
        <properties>
                <java-version>11</java-version>
                <org.springframework-version>4.3.3.RELEASE</org.springframework-version>
                <org.aspectj-version>1.6.10</org.aspectj-version>
                <org.slf4j-version>1.6.6</org.slf4j-version>
        </properties>

        <!-- Test -->
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>4.12</version>
                <scope>test</scope>
        </dependency>
```

0.4.3 JDBC 연결 테스트 코드 만들기

[ex00/src/test/java/org/zerock/web/MySQLConnectionTest.java]

```
01    package org.zerock.web;
02
03    import java.sql.Connection;
04    import java.sql.DriverManager;
05
06    import org.junit.Test;
07
08    public class MySQLConnectionTest {
09
10            private static final String DRIVER =
11                                    "com.mysql.jdbc.Driver";
12            private static final String URL =
13                                    "jdbc:mysql://127.0.0.1:3306/book_ex";
14            private static final String USER =
15                                    "zerock";
16            private static final String PW =
17                                    "zerock";
18
19
20            @Test
21            public void testConnection() throws Exception{
22
23                    Class.forName(DRIVER);
24
25                    try(Connection con = DriverManager.getConnection(URL, USER, PW)){
26
27                            System.out.println(con);
28
29
30                    }catch(Exception e){
31                            e.printStackTrace();
32                    }
33            }
34    }
```

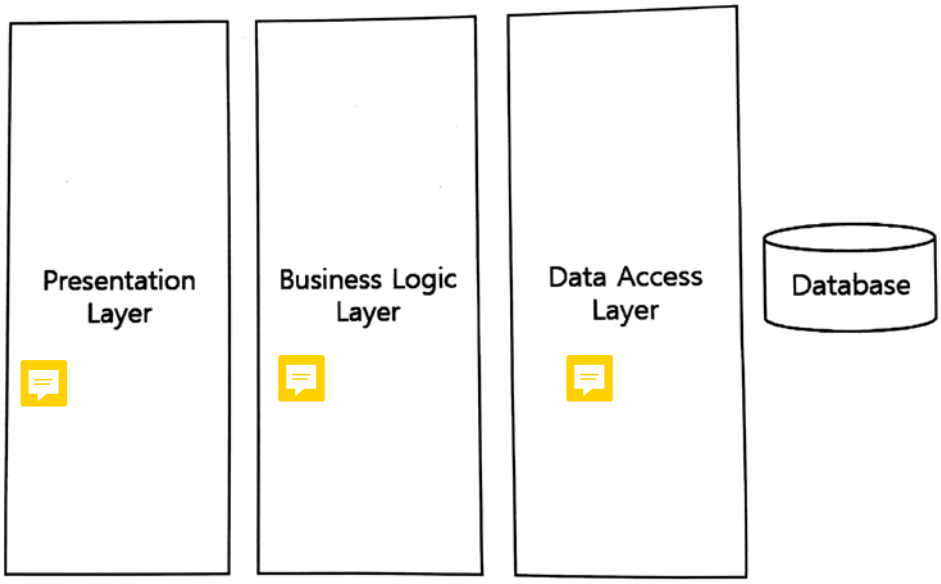
■ 최종 테스는 testConnection() 메소드를 지정하고, jUnit 테스트를 실행시켜서 아래와 같이 확인한다.

```
// eclipse console 메시지 확인
com.mysql.jdbc.JDBC4Connection@1d251891
```

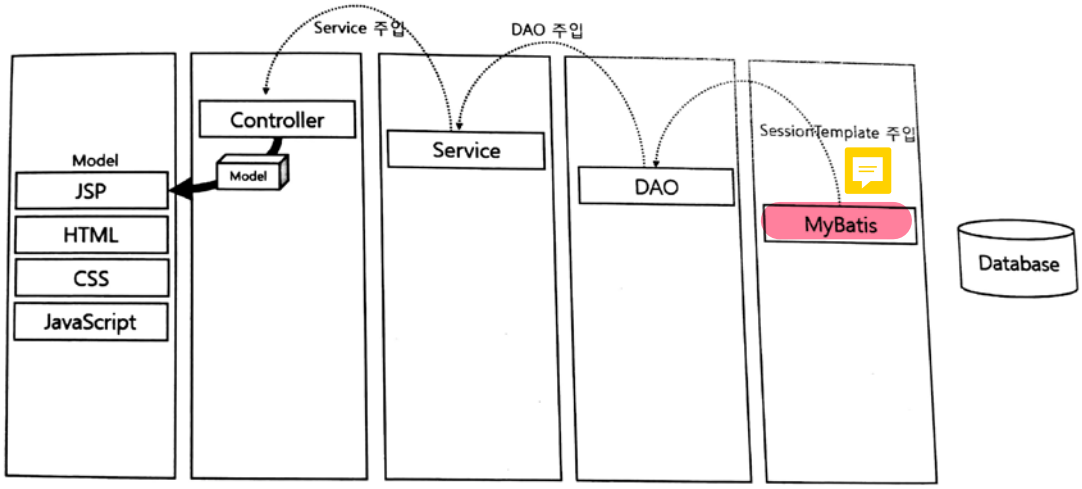
0.5 스프링 + MyBatis + MySQL의 연동

0.5.1 일반적인 스프링 웹 프로젝트의 구성

■ 일반적으로 웹 프로젝트는 3개의 레이어(혹은 tier)로 구성한다.

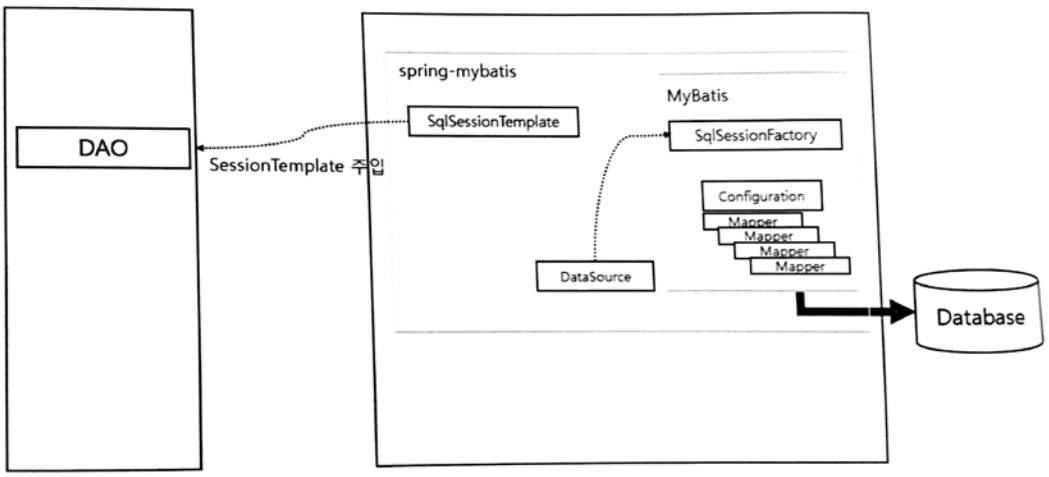


■ 이 과정의 프로젝트 구성
- Presentation Layer: HTML, CSS, JavaScript, JSP와 같은 뷰
- Business Logic Layer: Controller, Service
- Data Acess Layer: DAO(Data Acess Object), MyBatis



0.5.2 MyBatis와의 구성

■ 가장 핵심이 되는 부분의 구성은 아래와 같은 구조를 사용하게 된다.



0.5.3 MyBatis 연동을 위한 준비

■ JDBC를 이용해서 프로그램을 작성하는 것과 비교하면 MyBatis는 다음과 같은 편리함을 제공한다.
- 간결한 코드의 원리
- SQL 문의 분리운영
- Spring과의 연동으로 자동화된 처리
- 동적 SQL을 이용한 제어 기능

■ spring-jdbc, spring-test, MyBatis, mybatis-spring 추가: pom.xml에 다음과 같은 프레임워크, 라이브러리를 추가한다.

```
// pom.xml 파일의 일부
...
                    <!-- lab begin -->
                    <dependency>
                              <groupId>mysql</groupId>
                              <artifactId>mysql-connector-java</artifactId>
                              <version>5.1.36</version>
                    </dependency>
                    <dependency>
                              <groupId>org.mybatis</groupId>
                              <artifactId>mybatis</artifactId>
                              <version>3.4.1</version>
                    </dependency>
                    <dependency>
                              <groupId>org.mybatis</groupId>
                              <artifactId>mybatis-spring</artifactId>
                              <version>1.3.0</version>
                    </dependency>
                    <dependency>
                              <groupId>org.springframework</groupId>
                              <artifactId>spring-jdbc</artifactId>
                              <version>${org.springframework-version}</version>
                    </dependency>
```

```
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-test</artifactId>
                        <version>${org.springframework-version}</version>
                </dependency>
                <dependency>
                        <groupId>org.bgee.log4jdbc-log4j2</groupId>
                        <artifactId>log4jdbc-log4j2-jdbc4</artifactId>
                        <version>1.16</version>
                </dependency>

                <!-- lab end -->
...
```

■ log4j 관련 파일 생성

- src/main/resources/폴더에 두개의 파일(log4j.xml, log4jdbc.log4j2.properties)을 복사해 놓는다.

■ Spring Project에서 root-context.xml 파일의 수정

- src/main/webapp/WEB-INF/spring/root-context.xml 파일은 STS가 스프링 프로젝트를 생성할 때 만들어 주는 파일에서 가장 중요한 파일이다.
- MySQL과의 연결을 담당하는 DataSource 설정한다.

[ex00/src/main/webapp/WEB-INF/spring/root-context.xml]

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <beans xmlns="http://www.springframework.org/schema/beans"
03          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04  xmlns:aop="http://www.springframework.org/schema/aop"
05          xmlns:context="http://www.springframework.org/schema/context"
06          xmlns:jdbc="http://www.springframework.org/schema/jdbc"
07          xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
08          xsi:schemaLocation="http://www.springframework.org/schema/jdbc
09  http://www.springframework.org/schema/jdbc/spring-jdbc-4.3.xsd
10                  http://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-spring-
11  1.2.xsd
12                  http://www.springframework.org/schema/beans
13  http://www.springframework.org/schema/beans/spring-beans.xsd
14                  http://www.springframework.org/schema/context
15  http://www.springframework.org/schema/context/spring-context-4.3.xsd
16                  http://www.springframework.org/schema/aop
17  http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
18
19          <!-- Root Context: defines shared resources visible to all other web components -->
20
21          <!-- lab begin -->
22          <bean id="dataSource"
23                  class="org.springframework.jdbc.datasource.DriverManagerDataSource">
24
25                  <property name="driverClassName"
26  value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
27
28                  <property name="url"
29  value="jdbc:log4jdbc:mysql://127.0.0.1:3306/book_ex"></property>
30
31                  <property name="username" value="zerock"></property>
32                  <property name="password" value="zerock"></property>
33          </bean>
34
35          <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
36                  <property name="dataSource" ref="dataSource" />
37          </bean>
```

```
38
39            <!-- lab end -->
40
41    </beans>
```

[꿀팁] servlet-context.xml과 root-context.xml 차이점

- servlet-context.xml:
  - <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure --> 즉, DispatcherServlet의 request-processing에 관한 내용이 선언되어 있다.
  - url과 관련된 controller나, @(어노테이션), ViewResolver, Interceptor, MultipartResolver 등의 설정을 해준다.

- root-context.xml
  - <!-- Root Context: defines shared resources visible to all other web components --> 즉, 다른 웹 컴포넌트들과 공유하는 자원들을 선언하는 용도로 사용된다.
  - servlet-context.xml 과는 반대로 view와 관련되지 않은 객체를 정의한다. 따라서 Service, Repository(DAO), DB등 비즈니스 로직과 관련된 설정을 해준다.

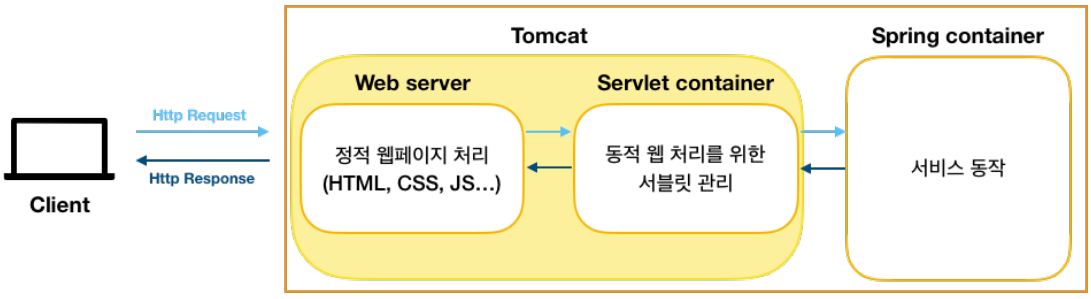
- servlet context는 root context의 bean를 참조할 수 있지만, 반대의 경우는 불가능하다.

[꿀팁] Tomcat + Spring MVC의 동작 과정

- 참고: https://taes-k.github.io/2020/02/16/servlet-container-spring-container/

- Tomcat
  - 일반적으로 탐캣(Tomcat)은 'WAS(Web Application Server)'의 대표적인 미들웨어 서비스로 알려져있다.
  - 하지만 탐캣은 일반적으로 아파치 탐캣(Apache Tomcat)이라 불리며 회사명이자 웹서버의 대표적인 미들웨어인 아파치(Apache)의 기능 일부분을 가져와 함께 사용되면서 웹서버(Web Server)의 기능과 웹 애플리케이션 서버(Web Application Server) 모두를 포함하고 있다.

- Process
  - 일반적으로 자바 웹프로그래밍을 할때 사용하는 Spring + Tomcat 조합으로 서비스를 올리게되면 아래와 같은 구조를 통해 클라이언트와 통신하게 된다.



- Web server
  - 웹서버는 사용자가 웹 브라우저에서 URL 입력했을때 사용자에게 응답을 처리하는 http

통신의 일련의 과정을 진행한다. 이 통신을 위해 소켓 연결 등의 네트워크 처리를 해주며 WAS와 비교하자면 html, css, js 등의 정적 소스에 대한 요청을 처리한다.

- 정적 소스를 별도로 처리하는 이유는 단순히 정적 파일만 전달 하면 되는 서비스에서 굳이 동적 컨텐츠를 처리하는 WAS에 부담을 줄 필요는 없기때문에 WAS와 업무를 나누어 처리하여 각 서비스의 목적에 따른 성능상의 장점을 살리기 위한 것이다.
- 또한 웹서버는 클라이언트와의 연결을 WAS로 전달하여 WAS가 클라이언트와 직접 통신하지 못하도록 중계역할을 해주어 WAS에게 클라이언트와의 연결에대한 독립성을 보장해주며, 보안적으로도 한단계 안전 할 수 있도록 해준다.
- 종류: Apache HTTTP Server, Microsoft IIS 등

■ Servlet

- 일반적으로 웹 프로그래밍을 한다고 하면 정의된 클라이언트의 요청에 대해 상응하는 결과를 return해 주어야 하는데 웹 페이지 혹은 결과값을 동적으로 생성 해 주기위한 역할을 하는 자바 프로그램을 서블릿 이라고 한다.
- 아래는 java8에서 제공하는 Servlet 인터페이스 이다.

```
// javax.servlet.Servlet.java
public interface Servlet {
    public void init(ServletConfig config) throws ServletException;
    public ServletConfig getServletConfig();
    public void service(ServletRequest req, ServletResponse res)
            throws ServletException, IOException;
    public String getServletInfo();
    public void destroy();
}
```

- 위의 인터페이스를 보면 서블릿의 라이프사이클을 확인할 수 있다.

  init > service > destroy

- javaEE로 웹서비스를 직접 구현할때에는 서블릿을 만들기 위해 위의 Servlet 인터페이스의 구현체를 직접 만들어 사용하지만, 스프링MVC 에서는 Dispatcher Servlet이라는 모든 요청을 담당하는 서블릿을 두고 컨트롤러에 위임을 하여 요청을 처리한다.

- 이와같은 프론트 컨트롤러 디자인 패턴이 적용된 SpringMVC를 통해 개발자는 별도의 서블릿 개발없이, Controller의 구현만으로도 동적인 response를 클라이언트에게 줄 수 있다.

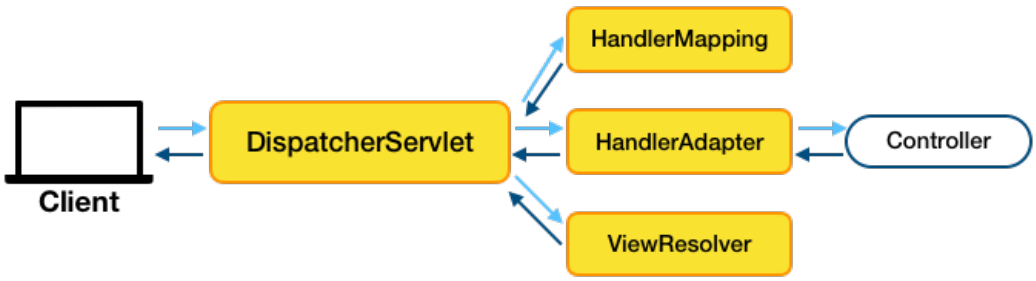
■ DispatcherServlet

- 다음은 Spring framework에 구현되어있는 DispatcherServlet.java 이다.

```
// package org.springframework.web.servlet;
@SuppressWarnings("serial")
public class DispatcherServlet extends FrameworkServlet {
    ...
```

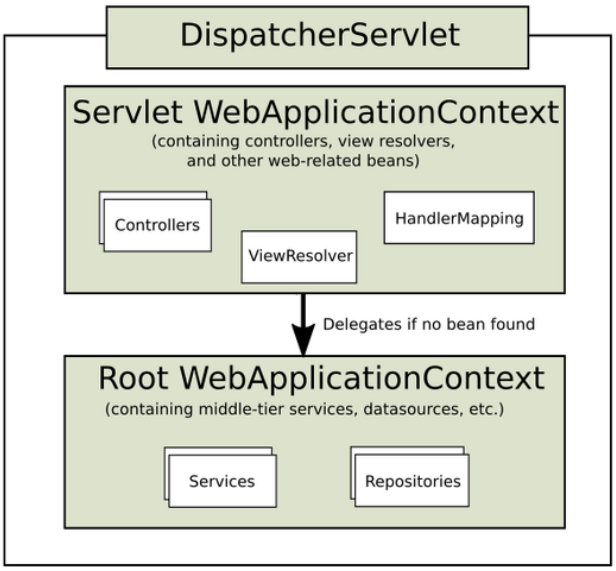
```
          public void setDetectAllHandlerMappings(boolean detectAllHandlerMappings) {
                    this.detectAllHandlerMappings = detectAllHandlerMappings;
          }

          protected void initStrategies(ApplicationContext context) {
                    initMultipartResolver(context);
                    initLocaleResolver(context);
                    initThemeResolver(context);
                    initHandlerMappings(context);
                    initHandlerAdapters(context);
                    initHandlerExceptionResolvers(context);
                    initRequestToViewNameTranslator(context);
                    initViewResolvers(context);
                    initFlashMapManager(context);
          }

          @Override
          protected void doService(HttpServletRequest request, HttpServletResponse response) throws
Exception {
             ...
          }

          protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws
Exception {
             ...
          }

          protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
             ...
          }

        protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
              ...
          }

    ...
}
```

- SpringMVC에서 제공해 주고 있는 DispatcherServlet은 [FrameworkServlet.java > HttpServlet.java > Servlet.java] 를 상속받아 구현한 서블릿이다. 이 클래스 내부에 여러 핸들러(Handler), 어댑터(Adapter), 리졸버(Resolver) 등을 가지고 클라이언트의 요청에 따라 개발자가 정의해 둔 내용을 응답 해 줄수있도록 front-controller의 역할을 하고 있음을 확인 하실 수 있다.
  - HandlerMapping: Client로 부터 들어온 Request를 분석하여 매핑된 Controller가 있는지 확인한다.
  - HandlerAdapter: 매핑 대상 Controller에게 Request 처리요청을 보낸다.
  - ViewResolver: Controller에서 view를 return 했을경우 해당하는 view를 찾아 client에게 return 한다.



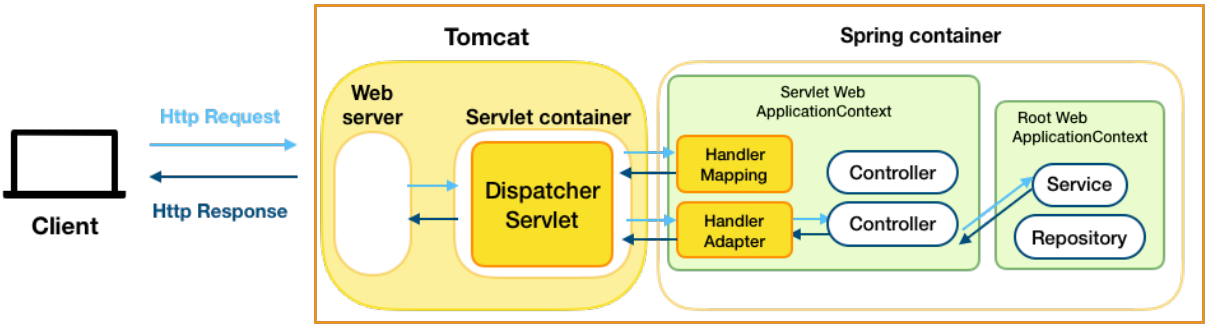
■ Servlet container
- 톰캣의 메인 기능이라고 할 수 있는 서블릿 컨테이너(Servlet container)의 역할을 알아보자.
- 첫째, 서블릿 컨테이너라는 말 그대로 서블릿을 관리하는 역할을 하게 된다. 위의 서블릿을 알아보면서 서블릿 라이프사이클(init > service > destroy)을 알아보았는데 서블릿 클래스의 로드, 초기화, 호출, 소멸까지의 라이프사이클을 직접적으로 관리해주는 역할을 하는것이 바로 서블릿 컨테이너이다. 서블릿으로 구현된 DispatcherServlet 역시 서블릿 컨테이너에서 수행된다.
- 둘째, 상단 Process 그림에서 보셨던것 처럼 서블릿 컨테이너는 웹서버와 통신을 통해 클라이언트의 request를 전달받아 동적 서비스를 response를 해야하는데, 해당 통신을 위해 소켓을 만드는등의 역할을 진행한다.
- 셋째, 클라이언트로부터 request를 받을때마다 쓰레드를 생성해 요청을 처리한다. 해당 쓰레드는 서블릿 컨테이너에서 쓰레드풀을 별도로 관리하여 실행하게 된다.

■ Spring container
- 아래 그림은 Spring Document에서 제공하는 것으로 DispatcherServlet 내부에 Servlet WebApplicationContext와 Root WebApplicationContext가 동작하는것으로 보이지만 이 두 ApplicationContext가 바로 process 그림에서 봤던 스프링 컨테이너(Spring container)에서 동작하는 컨텍스트라고 이해하면 된다.
- 서블릿 컨테이너는 서블릿의 생명주기를 관리했다면, 스프링 컨테이너는 Java object인 빈(Bean)의 라이프 사이클 관리하여 Spring 프레임워크의 특징인 IOC(제어역전)와 DI(의존성주입)을 제공해주는 역할을 한다.



■ 총정리
- 클라이언트의 request부터 response 받는 흐름은 아래와 같다.
- Server 실행 단계
  1. Web server init
  2. Root WebApplicationContext 로딩
  3. Web server start
- Client 호출 단계
  1. Client -> Web server 으로 request 보냄
  2. 동적 Web server -> Servlet container로 전달
  3. Servlet container 쓰레드 생성
  4. DispatcherServlet init (서블릿 생성 안되어 있을경우)

5. 생성된 쓰레드에서 DispatcherServlet service() 메서드 호출
6. HandlerMapping을 통해 매핑 컨트롤러 조회
7. HandlerAdapter를 통해 매핑 컨트롤러에 request 전달
8. 개발자가 구현한 Controller -> Service -> Repository … 동작



0.5.4 MyBatis와 스프링 연동

■ SqlSessionFactory 객체 설정: MyBatis와 스프링 연동 작업에서의 핵심은 Connection을 생성하고, 처리하는 SqlSessionFactory의 존재이다. SqlSessionFactory는 데이터베이스와의 연결과 SQL의 실행에 대한 모든 것을 가진 가장 중요한 객체이다.

```
// root-context.xml 파일의 수정
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
</bean>
```

■ mybatis-config.xml 파일의 추가

[ex00/src/main/resource/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>

</configuration>
```

■ MyBatis의 연결 테스트

[ex00/src/test/java/org/zerock/web/MyBatisTest.java]

```
package org.zerock.web;

import javax.inject.Inject;

import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
```

```
13  //@ContextConfiguration(locations ={"file:src/main/webapp/WEB-INF/spring/**/*.xml"})
14  @ContextConfiguration(locations ={"file:src/main/webapp/WEB-INF/spring/root-context.xml"})
15  public class MyBatisTest {
16
17          @Inject
18          private SqlSessionFactory sqlFactory;
19
20          @Test
21          public void testFactory(){
22
23                  System.out.println(sqlFactory);
24
25          }
26
27          @Test
28          public void testSession()throws Exception{
29
30                  try(SqlSession session = sqlFactory.openSession()){
31
32                          System.out.println(session);
33
34                  }catch(Exception e){
35                          e.printStackTrace();
36                  }
37
38          }
39  }
```

```
// Run As -> JUnit Test 를 실행한 결과
INFO : org.springframework.web.servlet.handler.SimpleUrlHandlerMapping - Mapped URL path [/resources/**] onto
handler 'org.springframework.web.servlet.resource.ResourceHttpRequestHandler#0'
INFO : org.springframework.jdbc.datasource.DriverManagerDataSource - Loaded JDBC driver:
net.sf.log4jdbc.sql.jdbcapi.DriverSpy
org.apache.ibatis.session.defaults.DefaultSqlSessionFactory@234a38
org.apache.ibatis.session.defaults.DefaultSqlSession@d13246
```

0.6 Oracle DB 연결 테스트

0.6.1 오라클 설치

- OTN 가입 : http://www.oracle.com/kr

- Oracle Database 11g Release 2 다운로드
  - http://www.oracle.com/technology/software/products/database/index.html

- Oracle Database 11g Release 2 설치
  - 전역 데이터베이스 이름(SID): **myoracle**
  - 데이터베이스 암호: "**sys**"로 설정한다.
  - 설치 디렉토리: **D:\prod\oracle**

- 주요 오라클 서비스 (Microsoft Windows 버젼인 경우)
  - OracleServiceMYORACLE : OracleService + SID명 형태로 구성된 서비스로 설치한 오라클 기본 서비스이며 오라클 사용 시 반드시 맨 먼저 시작되어야 한다.
  - OracleOraDb11g_home1TNSListener : 리스너 서비스로 이것 역시 반드시 시작되어야 한다. 리스너란 네트워크를 통해 클라이언트(오라클을 사용하려는 사용자)와 오라클 서버와의

연결을 담당하는 관리 프로그램이다.

- OracleDBConsolemyoracle : EM(Enterprise Manager)을 사용할 경우 시작해야 하는데, EM은 설치한 오라클을 관리하는 프로그램이다.

[꿀팁] 오라클 네트워크 접속 설정

```
// 1. d:\oracle\app\product\11.2.0\dbhome_1\NETWORK\ADMIN\listener.ora 파일 설정
# listener.ora Network Configuration File: D:\oracle\app\product\11.2.0\dbhome_1\network\admin\listener.ora
# Generated by Oracle configuration tools.

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = CLRExtProc)
      (ORACLE_HOME = D:\oracle\app\product\11.2.0\dbhome_1)
      (PROGRAM = extproc)
      (ENVS = "EXTPROC_DLLS=ONLY:D:\oracle\app\product\11.2.0\dbhome_1\bin\oraclr11.dll")
    )
  )

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
      (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    )
  )

ADR_BASE_LISTENER = D:\oracle\app


// 2. d:\oracle\app\product\11.2.0\dbhome_1\NETWORK\ADMIN\tnsnames.ora 파일 설정
# tnsnames.ora Network Configuration File: D:\oracle\app\product\11.2.0\dbhome_1\network\admin\tnsnames.ora
# Generated by Oracle configuration tools.

LISTENER_MYORACLE =
  (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))


ORACLR_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
    (CONNECT_DATA =
      (SID = CLRExtProc)
      (PRESENTATION = RO)
    )
  )

MYORACLE =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = myoracle)
    )
  )

ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
```

```
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = myoracle)
    )
  )
```

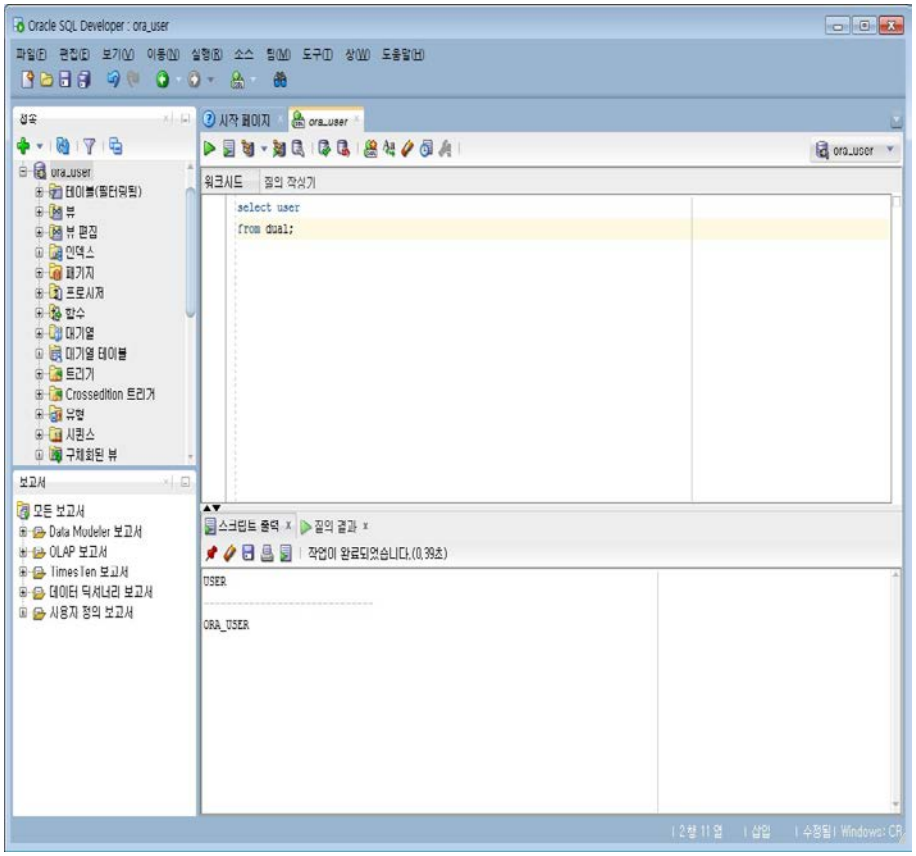
[꿀팁] 8080 포트 변경

- 오라클 11g의 경우 기본적으로 8080 포트를 이용해서 웹 환경으로도 데이터베이스를 접근할 수 있다. 문제는 웹 개발 시 많이 사용하는 Tomcat의 기본 포트가 8080이기 때문에 동시에 오라클과 Tomcat이 8080 포트를 사용하는 문제를 가진다.
- 이 문제를 해결하기 위해 오라클의 포트를 아래와 같이 변경해 주는 것이 좋다.

```
ks@ubuntu:~$ sqlplus system/sys
SQL> select dbms_xdb.gethttpport() from dual;

DBMS_XDB.GETHTTPPORT()
----------------------
                  8080

SQL> exec dbms_xdb.sethttpport(9090);
SQL> select dbms_xdb.gethttpport() from dual;

DBMS_XDB.GETHTTPPORT()
----------------------
                  9090
```

0.6.2 SQL Developer 설치

- 오라클에서 제공하는 무료 개발도구
- 다운로드: http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html



0.6.3 프로젝트의 JDBC 연결

(1) DB 스키마 생성

■ 실습용 계정을 생성한다.

```
// system 계정으로 접속하여 실행한다.
create user book_ex identified by book_ex;
grant connect, resource to book_ex;
```

■ book_ex 스키마 생성한다.

```
// WEB-INF/sql/ddl.sql
// book_ex 계정으로 접속하여 실행한다.

--alter table board drop constraint board_num_pk;
--drop table board;

create table board (
    num          number(10) not null,
    name         varchar2(20) not null,
    title        varchar2(100) not null,
    content     clob null,
    readcount   number(10) default 0 not null,
    writedate   date not null
);
```

```
alter table board add constraint board_num_pk primary key ( num );

insert into board values (1,'유재석','제목 테스트1','내용 테스트1',0,sysdate);
insert into board values (2,'이광수','제목 테스트2','내용 테스트2',1,sysdate);

commit;

select * from board;
```

(2) 프로젝트 생성

■ 프로젝트명: ex000racleLab

(3) OJDBC 라이브러리 추가

```
// pom.xml의 일부

	</properties>

	<!-- lab1 begin -->
	<repositories>
		<repository>
			<id>oracle</id>
			<url>http://maven.jahia.org/maven2</url>
		</repository>
	</repositories>
	<!-- lab1 end -->

	<dependencies>
		<!-- lab2 begin -->
		<!-- 오라클 JDBC 드라이버 -->
		<dependency>
			<groupId>com.oracle</groupId>
			<artifactId>ojdbc6</artifactId>
			<version>12.1.0.2</version>
		</dependency>

		<!-- lab2 end -->

	</dependencies>
```

0.6.4 로깅(Logging)

■ Loggging이란 프로그램 개발 중이나 완료 후 발생할 수 있는 오류에 대해 디버깅하거나 운영 중인 프로그램 상태를 모니터링 하기 위해 필요한 정보(로그)를 기록하는 것이다.

■ Java의 주요 Logging Framework
- native java.util.logging: 별로 사용하지 않는다.
- Log4J: 몇 년 전까지 사실상 표준으로 사용했다.
- Logback: Log4J 개발자가 만든 Log4J의 후속 버전, 현재 많은 프로젝트에서 사용되고 있다.
- SLF4J(Simple Logging Facade for Java): Log4J 또는 Logback과 같은 백엔드 Logger

- Tinylog: 사용하기 쉽게 최적화된 Java용 최소형 프레임워크
- Lombok: Lombok의 @Log4j 혹은 @Slf4j를 달아주면 log 필드가 생기고, 편하게 logger를 불러서 로깅할 수 있다.

■ Debbugger와 차이
- 장점
  - Logging은 응용 프로그램 실행에 대한 정확한 컨텍스트(이벤트 순서)를 제공한다.
  - 일단 코드에 삽입되면 logging output이 만들어질 때 사용자 개입이 필요 없다.
  - 로그 출력은 나중에 살펴볼 수 있도록 영구 매체에 저장할 수 있다.
- 단점
  - 출력문이 들어가기 때문에 응용 프로그램 속도를 늦출 수 있다.
  - 너무 장황할 수 있다. (오버해드)
  - 고급 사용은 구성을 확실히 알아야 한다.

■ Plain Output (System.out.println())
- 장점
  - 높은 유연성
  - 우선순위 level 이상의 출력 메시지를 선택할 수 있다. (trace, debug, info, warn, error)
  - 모든 모듈, 특정 모듈 또는 클래스에 대해 메세지를 출력할 수 있다.

(1) SLF4J(Simple Logging Facade for Java)

■ pom.xml 파일에 관련 라이브러리를 추가한다.

[pom.xml의 일부]

```
        <properties>
                <java-version>1.8</java-version>
                <org.springframework-version>4.1.7.RELEASE</org.springframework-version>
                <org.aspectj-version>1.6.10</org.aspectj-version>
                <!-- <org.slf4j-version>1.6.6</org.slf4j-version> -->
                <org.slf4j-version>1.7.25</org.slf4j-version>
        </properties>

...(생략)...

                <!-- Logging -->
                <dependency>
                        <groupId>org.slf4j</groupId>
                        <artifactId>slf4j-api</artifactId>
                        <version>${org.slf4j-version}</version>
                </dependency>
                <dependency>
                        <groupId>org.slf4j</groupId>
                        <artifactId>jcl-over-slf4j</artifactId>
                        <version>${org.slf4j-version}</version>
                        <scope>runtime</scope>
                </dependency>
                <dependency>
                        <groupId>org.slf4j</groupId>
                        <artifactId>slf4j-log4j12</artifactId>
                        <version>${org.slf4j-version}</version>
                        <scope>runtime</scope>
                </dependency>
```

[ex00Oracle/src/test/java/util/SLF4JLogger.java]

```
package util;

import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class SLF4JLogger {
	Logger logger = LoggerFactory.getLogger(SLF4JLogger.class);

	@Test
	public void print() {

		logger.info("Hello World");
	}

}
```

```
// 실행 결과
INFO : util.SLF4JLogger - Hello World
```

(2) Lombok

- Lombok는 자바에서 작성해야 하는 boilerplate code(ex. getter/setter, constructor, toString)를 어노테이션을 통해서 자동으로 생성해 주는 라이브러리이다. 코드 자체가 더 간결해져 가독성도 높아지고 더 빠르게 개발할 수 있는 장점이 있다. 또한 @Log4j 혹은 @Slf4j를 달아주면 log 필드가 생기고, 편하게 logger를 불러서 로깅할 수 있다.

- 자주 사용하는 어노테이션
  - @Log4, @Slf4j: log를 출력할 수 있도록 도와준다. 이 어노테이션은 log라는 변수를 자동으로 만들어 주는 아주 편리한 기능이다.
  - @Getter, @Setter: 클래스 필드에 대한 getter와 setter 메서드를 생성해 준다. @ToString: 클래스의 toString 메서드를 자동으로 생성해 준다.
  - @Data: 여러 어노테이션들(@ToString, @EqualsAndHashCode, @RequiredArgsConstructor, @Getter, @Setter)이 적용되는 어노테이션이다.
  - @NonNull: 메서드나 생성자 인자에 추가하면 Lombok가 null 체크 구문을 생성해 준다.
  - @Builer: 다수의 필드를 가지는 복잡한 클래스의 경우, 생성자 대신에 @Builder 어노테이션을 사용하면 자동으로 해당 클래스에 빌더를 추가해 준다.

- pom.xml 파일에 lombok 라이브러리를 추가한다.
  - STS(eclipse)에 lombok 설치: https://countryxide.tistory.com/16 참조

[pom.xml의 일부]

```
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<version>1.18.0</version>
			<scope>provided</scope>
		</dependency>
```

■ 실습

[ex00Oracle/src/test/java/util/LombokLogger.java] @Log4j 어노테이션의 사용 예

```
package util;

import org.junit.Test;

import lombok.extern.log4j.Log4j;

@Log4j
public class LombokLogger {
	@Test
	public void print() {
		log.info("Hello World");
	}
}
```

```
// JUnit Test의  실행 결과
INFO : util.LombokLogger - Hello World
```

[ex00Oracle/src/test/java/util/LombokBuilder.java] @Builder 어노테이션의 사용 예

```
package util;

import java.util.List;

import org.junit.Test;

import lombok.Builder;
import lombok.Getter;
import lombok.Setter;
import lombok.Singular;
import lombok.ToString;
import lombok.extern.log4j.Log4j;

@Log4j
public class LombokBuilder {
	@Test
	public void print() {
		User user1 =
User.builder().id(1L).username("dale").password("1234").score(70).score(80).build();
		User user2 = User.builder().id(2L).username("hong").password("hong").build();
		log.info(user1.toString());
		log.info(user2.toString());
		user2.setPassword("1234");
		log.info(user2.toString());
	}
}

@Builder
@Getter
@Setter
@ToString
class User {
	private Long id;
	private String username;
	private String password;
	@Singular
	private List<Integer> scores;
}
```

```
// JUnit Test의  실행 결과
INFO : util.LombokBuilder - User(id=1, username=dale, password=1234, scores=[70, 80])
INFO : util.LombokBuilder - User(id=2, username=hong, password=hong, scores=[])
INFO : util.LombokBuilder - User(id=2, username=hong, password=1234, scores=[])
```

[꿀팁] lombok 관련 에러

■ 에러: The attribute onMethod_ is undefined for the annotation type Setter

■ 해결

```
1. lombok.jar 파일을 sts 폴더에 복사한다.
D:\temp>copy lombok.jar D:\dev\eclipse\jee-2020-09\eclipse\

2. eclipse.ini 파일에 아래 내용을 추가한다.
...(생략)...
-Doomph.update.url=http://download.eclipse.org/oomph/updates/milestone/latest
-Doomph.redirection.index.redirection=index:/-
>http://git.eclipse.org/c/oomph/org.eclipse.oomph.git/plain/setups/
-javaagent:D:\dev\eclipse\jee-2020-09\eclipse\lombok.jar

3. exclipse를 재실행하고 메뉴 Project > Clean 한다.
```

■ 참고: https://ojava.tistory.com/131

(3) JDBC

[src/test/java/org.zerock.persistence.JDBCTests.java]

```
01    package org.zerock.persistence;
02
03    import java.sql.Connection;
04    import java.sql.DriverManager;
05
06    import org.junit.Test;
07
08    import lombok.extern.log4j.Log4j;
09
10    @Log4j
11    public class JDBCTests {
12
13              static {
14                        try {
15                                  Class.forName("oracle.jdbc.driver.OracleDriver");
16                        } catch (Exception e) {
17                                  e.printStackTrace();
18                        }
19              }
20
21              @Test
22              public void testConnection() {
23
24                        try (Connection con =
25    DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:MYORACLE", "book_ex",
```

```
26                                          "book_ex")) {
27
28                                  log.info(con);
29                          } catch (Exception e) {
30                                  //fail(e.getMessage());
31                                  e.printStackTrace();
32                          }
33              }
34
35      }
```

```
// JDBCTests.java의 실행결과
INFO : org.zerock.persistence.JDBCTests - oracle.jdbc.driver.T4CConnection@3dd3bcd
```

0.6.5 커넥션 풀 설정

- 일반적으로 여러 명의 사용자를 동시에 처리해야 하는 웹 애플리케이션의 경우 데이터베이스 연결을 이용할 때는 '커넥션 풀(Connection Pool)'을 이용한다.
- 커넥션 풀은 여러 종류가 있고, spring-jdbc 라이브러리를 이용하는 방식도 있지만, 예제는 최근 유행하는 HikariCP(https://github.com/brettwooldridge/HikariCP)를 이용한다.
- pom.xml을 수정해서 HikariCP를 추가한다.

```
// pom.xml의 일부

                    <dependency>
                            <groupId>com.zaxxer</groupId>
                            <artifactId>HikariCP</artifactId>
                            <version>2.7.4</version>
                    </dependency>
```

[WEB-INF/spring/root-context.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
          xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
          xsi:schemaLocation="http://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-
spring-1.2.xsd
                    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">

          <!-- Root Context: defines shared resources visible to all other web components -->

          <!-- HikariCP configuration -->
          <bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
                    <!-- <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"></property>
                              <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:XE"></property>
-->
                    <property name="driverClassName" value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
                    <property name="jdbcUrl"
                              value="jdbc:log4jdbc:oracle:thin:@localhost:1521:MYORACLE"></property>
```

```
                    <property name="username" value="book_ex"></property>
                    <property name="password" value="book_ex"></property>
          </bean>
          <bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource"
                    destroy-method="close">
                    <constructor-arg ref="hikariConfig" />
          </bean>

<!--      <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
                    <property name="dataSource" ref="dataSource"></property>
          </bean> -->

          <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
                    <property name="dataSource" ref="dataSource" />
                    <!-- <property name="configLocation" value="classpath:/mybatis-config.xml"></property> -->
                    <!-- <property name="mapperLocations" value="classpath:mappers/**/*Mapper.xml"></property>
-->
          </bean>

          <mybatis-spring:scan base-package="org.zerock.mapper" />

          <context:component-scan base-package="org.zerock.sample"></context:component-scan>
          <context:component-scan base-package="org.zerock.service"></context:component-scan>

</beans>
```

[src/test/java/org.zerock.persistence.DataSourceTests.java]

```
package org.zerock.persistence;

import static org.junit.Assert.fail;

import java.sql.Connection;

import javax.sql.DataSource;

import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
// @ContextConfiguration(classes= {RootConfig.class})
@Log4j
public class DataSourceTests {

          @Setter(onMethod_ = { @Autowired })
          private DataSource dataSource;

          @Setter(onMethod_ = { @Autowired })
          private SqlSessionFactory sqlSessionFactory;

          @Test
          public void testDataSource() {

                    try (Connection con = dataSource.getConnection()) {
```

```
                                log.info("*** con="+con);

                        } catch (Exception e) {
                                fail(e.getMessage());
                        }
                }

                @Test
                public void testSqlSessionFactory() {

                        try (SqlSession session = sqlSessionFactory.openSession(); Connection con =
session.getConnection();) {

                                log.info("*** session=" + session);
                                log.info("*** con=" + con);

                        } catch (Exception e) {
                                fail(e.getMessage());
                        }

                }

}
```

```
// DataSourceTests.java의 실행결과

...(생략)...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
INFO : org.zerock.persistence.DataSourceTests - ***
session=org.apache.ibatis.session.defaults.DefaultSqlSession@6e950bcf
INFO : org.zerock.persistence.DataSourceTests - *** con=HikariProxyConnection@373378624 wrapping
net.sf.log4jdbc.sql.jdbcapi.ConnectionSpy@74bada02
INFO : org.zerock.persistence.DataSourceTests - *** con=HikariProxyConnection@392904516 wrapping
net.sf.log4jdbc.sql.jdbcapi.ConnectionSpy@74bada02
INFO : org.springframework.context.support.GenericApplicationContext - Closing
org.springframework.context.support.GenericApplicationContext@7f13d6e: startup date [Wed Aug 12 18:06:29 KST
2020]; root of context hierarchy
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown initiated...
INFO : com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Shutdown completed.
```

[꿀팁] HikariCP 세팅시 옵션

- HikariCP설정의 시간 단위는 ms이다.
  - autoCommit: auto-commit설정 (default: true)
  - connectionTimeout: pool에서 커넥션을 얻어오기전까지 기다리는 최대 시간, 허용가능한 wait time을 초과하면 SQLException을 던짐. 설정가능한 가장 작은 시간은 250ms (default: 30000 (30s))
  - idleTimeout: pool에 일을 안하는 커넥션을 유지하는 시간. 이 옵션은 minimumIdle이 maximumPoolSize보다 작게 설정되어 있을 때만 설정. pool에서 유지하는 최소 커넥션 수는 minimumIdle (A connection will never be retired as idle before this timeout.). 최솟값은 10000ms (default: 600000 (10minutes))
  - maxLifetime: 커넥션 풀에서 살아있을 수 있는 커넥션의 최대 수명시간. 사용중인 커넥션은 maxLifetime에 상관없이 제거되지않음. 사용중이지 않을 때만 제거됨. 풀 전체가아닌 커넥션 별로 적용이되는데 그 이유는 풀에서 대량으로 커넥션들이 제거되는 것을 방지하기 위함임. 강력하게 설정해야하는 설정 값으로 데이터베이스나 인프라의 적용된 connection time limit보다 작아야함. 0으로 설정하면 infinite lifetime이 적용됨

(idleTimeout설정 값에 따라 적용 idleTimeout값이 설정되어 있을 경우 0으로 설정해도 무한 lifetime 적용 안됨). (default: 1800000 (30minutes))

- connectionTestQuery: JDBC4 드라이버를 지원한다면 이 옵션은 설정하지 않는 것을 추천. 이 옵션은 JDBC4를 지원안하는 드라이버를 위한 옵션임(Connection.isValid() API.) 커넥션 pool에서 커넥션을 획득하기전에 살아있는 커넥션인지 확인하기 위해 valid 쿼리를 던지는데 사용되는 쿼리 (보통 SELECT 1 로 설정) JDBC4드라이버를 지원하지않는 환경에서 이 값을 설정하지 않는다면 error레벨 로그를 뱉어냄.(default: none)
- minimumIdle: 아무런 일을 하지않아도 적어도 이 옵션에 설정 값 size로 커넥션들을 유지해주는 설정. 최적의 성능과 응답성을 요구한다면 이 값은 설정하지 않는게 좋음. default값을 보면 이해할 수있음. (default: same as maximumPoolSize)
- maximumPoolSize: pool에 유지시킬 수 있는 최대 커넥션 수. pool의 커넥션 수가 옵션 값에 도달하게 되면 idle인 상태는 존재하지 않음.(default: 10)
- poolName: 이 옵션은 사용자가 pool의 이름을 지정함. 로깅이나 JMX management console에 표시되는 이름.(default: auto-generated)
- initializationFailTimeout: 이 옵션은 pool에서 커넥션을 초기화할 때 성공적으로 수행할 수 없을 경우 빠르게 실패하도록 해준다. 상세 내용은 한국말보다 원문이 더 직관적이라 생각되어 다음 글을 인용함.

```
// oracle connection(inactive session) kill 방법

$ sqlplus system/sys
SQL> SELECT sid, serial#, status, username FROM v$session;
SID    SERIAL# STATUS   USERNAME
---------- ---------- -------- ------------------------------
        40         65 INACTIVE BOOK_EX
        42         43 ACTIVE   SYSTEM

SQL> SELECT 'ALTER SYSTEM KILL SESSION '''||sid||','||serial#||''' IMMEDIATE;' FROM v$session where
username='BOOK_EX';

'ALTERSYSTEMKILLSESSION'''||SID||','||SERIAL#||'''IMMEDIATE;'
--------------------------------------------------------------------------------
ALTER SYSTEM KILL SESSION '40,65' IMMEDIATE;

SQL> ALTER SYSTEM KILL SESSION '40,65' IMMEDIATE;

System altered.
```

0.7 스프링 + MyBatis + Oracle의 연동

0.7.1 스프링과의 연동 처리

(1) MyBatis 관련 라이브러리 추가

```
// pom.xml의 일부

        </properties>

        <!-- lab1 begin -->
        <repositories>
                <repository>
                        <id>oracle</id>
                        <url>http://maven.jahia.org/maven2</url>
```

```
                <repository>
        </repositories>
        <!-- lab1 end -->

        <dependencies>
                <!-- lab2 begin -->
                <!-- 오라클 JDBC 드라이버 -->
                <dependency>
                        <groupId>com.oracle</groupId>
                        <artifactId>ojdbc6</artifactId>
                        <version>12.1.0.2</version>
                </dependency>

                <!-- 컨넥션 풀을 위한 라이브러리 -->
                <dependency>
                        <groupId>commons-dbcp</groupId>
                        <artifactId>commons-dbcp</artifactId>
                        <version>1.4</version>
                </dependency>

                <!-- MyBatis 라이브러리 -->
                <dependency>
                        <groupId>org.mybatis</groupId>
                        <artifactId>mybatis</artifactId>
                        <version>3.4.6</version>
                </dependency>

                <!-- 마이바티스와 스프링 연동을 위한 라이브러리 -->
                <dependency>
                        <groupId>org.mybatis</groupId>
                        <artifactId>mybatis-spring</artifactId>
                        <version>1.3.2</version>
                </dependency>

                <!-- 스프링에서 JDBC 를 사용하기 위한 라이브러리 -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-jdbc</artifactId>
                        <version>${org.springframework-version}</version>
                </dependency>

                <!-- 스프링에서 트랜잭션 처리를 위한 라이브러리 -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-tx</artifactId>
                        <version>${org.springframework-version}</version>
                </dependency>

                <!-- lab2 end -->

        </dependencies>
```

(2) TimeMapper

[src/main/java/org.zerock.mapper/TimeMapper.java]

```
package org.zerock.mapper;

import org.apache.ibatis.annotations.Select;

public interface TimeMapper {
```

```
	@Select("SELECT sysdate FROM dual")
	public String getTime();

	public String getTime2();

}
```

[src/main/resources/org/zerock/mapper/TimeMapper.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="org.zerock.mapper.TimeMapper">

  <select id="getTime2" resultType="string">
  SELECT sysdate FROM dual
  </select>

</mapper>
```

[WEB-INF/spring/root-context.xml의 일부]

```
...(생략)...

	<mybatis-spring:scan base-package="org.zerock.mapper" />
```

[src/test/java/org.zerock.persistence.TimeMapperTests.java 의 일부]

```
package org.zerock.persistence;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.zerock.mapper.TimeMapper;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
// @ContextConfiguration(classes = { org.zerock.config.RootConfig.class })
@Log4j
public class TimeMapperTests {

	@Setter(onMethod_ = @Autowired)
	private TimeMapper timeMapper;

	//@Test
	public void testGetTime() {
		log.info("/* "+timeMapper.getClass().getName());
		log.info("/* timeMapper.getTime()="+timeMapper.getTime());
	}

	@Test
	public void tetGetTime2() {

		log.info("/* getTime2");
		log.info("/* timeMapper.getTime2()="+timeMapper.getTime2());
```

```
        }

}
```

```
// 실행결과

INFO : org.zerock.persistence.TimeMapperTests - /* getTime2
INFO : jdbc.sqlonly - SELECT sysdate FROM dual

INFO : jdbc.sqltiming - SELECT sysdate FROM dual
 {executed in 701 msec}
INFO : jdbc.resultsettable -
|----------------------|
|sysdate               |
|----------------------|
|2020-08-12 18:59:27.0 |
|----------------------|

INFO : org.zerock.persistence.TimeMapperTests - /* timeMapper.getTime2()=2020-08-12 18:59:27.0
```

(3) BoardMapper

[src/main/java/org.zerock.mapper/BoardMapper.java]

```
package org.zerock.mapper;

import java.util.List;

import org.zerock.domain.BoardVO;

public interface BoardMapper {
        List<BoardVO> selectBoardList() throws Exception;
}
```

[src/main/resources/org/zerock/mapper/BoardMapper.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="org.zerock.mapper.BoardMapper">

        <!-- id 속성은 DAO 의 메소드와 같아야 한다. 여기의 boardVO 는 mybatis-confg.xml 에 Alias 되어
                있습니다. -->
        <select id="selectBoardList" resultType="org.zerock.domain.BoardVO">
                select num, name, title, content, readcount, writedate
                from board
                order by num desc
        </select>

        <select id="selectBoardList2" resultType="org.zerock.domain.BoardVO">
                select num, name, title, content, readcount, writedate
                from board
                where num=2
                order by num desc
        </select>
</mapper>
```

[WEB-INF/spring/root-context.xml의 일부]

```
...(생략)...

	<mybatis-spring:scan base-package="org.zerock.mapper" />
```

[src/test/java/org.zerock.persistence.BoardMapperTests.java 의 일부]

```
package org.zerock.persistence;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.zerock.mapper.BoardMapper;

import lombok.Setter;
import lombok.extern.log4j.Log4j;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("file:src/main/webapp/WEB-INF/spring/root-context.xml")
// @ContextConfiguration(classes = { org.zerock.config.RootConfig.class })
@Log4j
public class BoardMapperTests {

	@Setter(onMethod_ = @Autowired)
	private BoardMapper boardMapper;

	@Test
	public void selectBoardList() throws Exception {
		log.info("/* "+boardMapper.getClass().getName());
		log.info("/* boardMapper.selectBoardList()="+boardMapper.selectBoardList());
	}

}
```

```
// 실행결과

INFO : org.zerock.persistence.BoardMapperTests - /* com.sun.proxy.$Proxy24
INFO : jdbc.sqlonly - SELECT NUM , NAME , TITLE , CONTENT , READ_COUNT , WRITE_DATE FROM TB_BOARD ORDER BY NUM
DESC

INFO : jdbc.sqltiming - SELECT NUM , NAME , TITLE , CONTENT , READ_COUNT , WRITE_DATE FROM TB_BOARD ORDER BY
NUM DESC
 {executed in 409 msec}
INFO : jdbc.resultsettable -
|----|-----|-------|---------|-----------|-----------|
|num |name |title  |content  |read_count |write_date |
|----|-----|-------|---------|-----------|-----------|
|1   |홍길동  |게시물 작성 |[unread] |[unread]   |[unread]   |
|----|-----|-------|---------|-----------|-----------|

INFO : org.zerock.persistence.BoardMapperTests - /* boardMapper.selectBoardList()=[BoardVO [num=1, name=홍길
동, title=게시물 작성, content=글 내용, readCount=null, writeDate=null]]
```

0.7.2 샘플 코드

(1) Servcie

[src/main/java/org.zerock.service/BoardServiceImpl.java]

```
package org.zerock.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.zerock.domain.BoardVO;
import org.zerock.mapper.BoardMapper;

@Service
public class BoardServiceImpl implements BoardService {
	@Autowired
	private BoardMapper boardMapper;

	@Override
	@Transactional
	public List<BoardVO> selectBoardList() throws Exception {
		return boardMapper.selectBoardList();
	}

}
```

(2) Controller

[src/main/java/org.zerock.controller/BoardController.java]

```
package org.zerock.controller;

import java.util.List;

import javax.inject.Inject;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.zerock.domain.BoardVO;
import org.zerock.service.BoardService;

/**
 * Handles requests for the application home page.
 */
@Controller
@RequestMapping(value = "/board")
public class BoardController {

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

	@Inject
	private BoardService boardService;

	@RequestMapping(value = "/list")
	public void boardList(Model model) throws Exception {
```

```
                    logger.info("// /board/list");

                    List<BoardVO> list = boardService.selectBoardList();

                    logger.info("// list.toString()=" + list.toString());

                    model.addAttribute("list", list);
          }

}
```

(3) servlet-context.xml

[WEB-INF/spring/appServlet/servlet-context.xml의 일부]

```
...(생략)...

          <context:component-scan base-package="org.zerock.controller" />
```

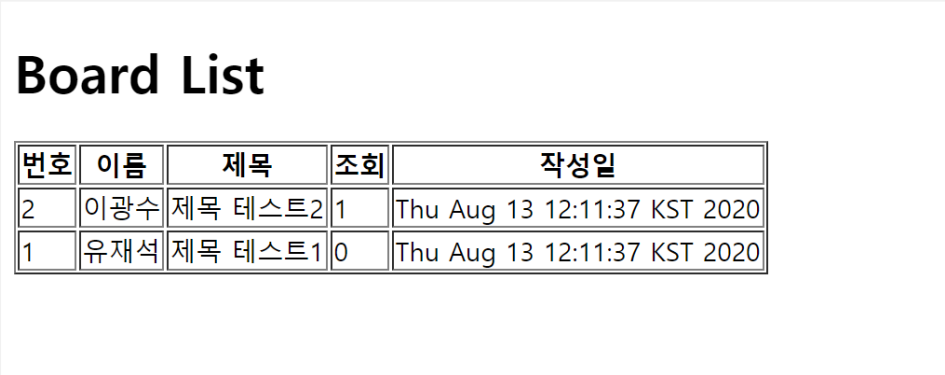
(4) JSP

[WEB-INF/views/board/list.jsp]

```
01    <%@ page contentType="text/html; charset=euc-kr" %>
02    <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
03    <%@ page session="false" %>
04    <!DOCTYPE html>
05    <html>
06    <head>
07    <meta http-equiv="Content-Type" content="text/html; charset=euc-kr"/>
08    <title>Home</title>
09    </head>
10    <body>
11              <h1>Board List</h1>
12              <table border="1">
13              <thead>
14              <tr>
15                        <th>번호</th>
16                        <th>이름</th>
17                        <th>제목</th>
18                        <th>조회</th>
19                        <th>작성일</th>
20              </tr>
21              </thead>
22              <tbody>
23                        <c:forEach var="item" items="${list}">
24                        <tr>
25                                  <td>${item.num}</td>
26                                  <td>${item.name}</td>
27                                  <td>${item.title}</td>
28                                  <td>${item.readCount}</td>
29                                  <td>${item.writeDate}</td>
30                        </tr>
31                        </c:forEach>
32              </tbody>
33              </table>
34    </body>
35    </html>
```

(6) 실행

■ http://localhost:8080/ex00OracleLab/board/list 로 접속한다.



Board List

번호	이름	제목	조회	작성일
2	이광수	제목 테스트2	1	Thu Aug 13 12:11:37 KST 2020
1	유재석	제목 테스트1	0	Thu Aug 13 12:11:37 KST 2020

0.8 스프링 프로젝트 구축 절차 (요약)

0.8.1 설정 순서

```
####################
# 표준 설정
1. Spring Legacy Project 생성: Templates: Spring MVC Project 선택

2. 프로젝트 설정 변경
   a. Project Facets
   b. Java Buil Path 에서 “Junit 5”를 추가한다.
   c. pom.xml 파일의 Java 버전(1.8)과 Spring 버전(4.1.7), maven-compiler-plugin(3.5.1), source(1.8),
target(1.8)로 변경한다.

3. DBMS 연결 테스트
   a. pom.xml 파일에
      - db 라이브러리 추가
      - maven-compiler-plugin 버젼 변경: <version>3.5.1</version>, <source>1.8</source>, <target>1.8<target>
   b. MySQLConnectionTest.java 테스트
      - pom.xml에 jdbc 설정
      - MySQLConnectionTest.java로 연결 테스트

4. 스프링 + MyBatis + MySQL의 연동 테스트
   a. pom.xml에 spring-jdbc, spring-test, MyBatis, mybatis-spring 추가
   b. log4j 파일 복사: src/main/resources/폴더에 두개의 파일(log4j.xml, log4jdbc.log4j2.properties,
logback.xml)을 복사해 놓는다.
   c. root-context.xml 설정: MySQL과의 연결을 담당하는 dataSource, sqlSessionFactory 설정한다.
   d. mybatis-config.xml 파일의 추가
   e. dataSource 연결 테스트: org.zerock.test.DataSourceTest.java
   f. MyBatis의 연결 테스트: src/test/java/org/zerock/web/MyBatisTest.java

5. web.xml에 스프링의 UTF-8 처리 필터 등록
####################


####################
# 웹어플리케이션 설정
6. CSS, JavaScript 준비: bootstrap theme 이용, Context Path 변경

7. 영속 계층
    a. VO, DAO, XML Mapper
```

```
    b. root-context.xml에 SessionFactory, SqlSessionTemplate의 추가
8. 비즈니스 계층
    a. BoardService 등록
    b. root-context.xml에 scan 등록
9. 컨트롤러
    /board/register
10. views(*.jsp) 생성
    <form role="form"의 의미: form에 action을 따로 지정해주지 않으면 현재 경로를 그대로 action의 대상 경로로
잡는다.
######################
```

[꿀팁] sqlsessionfactory 오류

참고: https://jeekchic.tistory.com/2

spring + mybatis 조합에서 mapper.xml 찾지못해 오류일때, classpath뒤에 * 붙여주면 해결된다.

```
// root-context.xml
          <property name="mapperLocations"
                              value="classpath*:mappers/**/*Mapper.xml"></property>
```

0.8.2 개발 순서

```
// Spring 프로젝트 생성 방법
1.  board table 생성
create table board (
          num number(10) not null,
          name varchar2(20) not null,
          title varchar2(100) not null,
          content clob null,
          readcount number(10) default 0 not null,
          writedate date not null
);

2. vo class 생성
public class BoardVO {
          private Integer num;
          private String name;
          private String title;
          private String content;
          private Integer readCount;
          private Date writeDate;

          public Integer getNum() {
                    return num;
          }
          public void setNum(Integer num) {
                    this.num = num;
          }
}

3. mapper 파일 생성
<mapper namespace="org.zerock.mapper.BoardMapper">
          <select id="selectBoardList" resultType="org.zerock.domain.BoardVO">
                    select num, name, title, content, readcount, writedate
```

```
                    from board
                    order by num desc
          </select>

4. dao 인터페이스/구현클래스 생성, or mapper interface 생성
public interface BoardMapper {
          List<BoardVO> selectBoardList() throws Exception;
}

5. service 인터페이스/구현 클래스 생성
@Service
public class BoardServiceImpl implements BoardService {
          @Autowired
          private BoardMapper boardMapper;

          @Override
          @Transactional
          public List<BoardVO> selectBoardList() throws Exception {
                    return boardMapper.selectBoardList();
          }
}

6. controller class 생성
@Controller
@RequestMapping(value = "/board")
public class BoardController {
          @Inject
          private BoardService boardService;

          @RequestMapping(value = "/list")
          public void boardList(Model model) throws Exception {

                    List<BoardVO> list = boardService.selectBoardList();
                    model.addAttribute("list", list);
          }
}


// Spring 프로젝트 설정 방법
1. servlet-context.xml 설정
<context:component-scan base-package="org.zerock.controller" />

2. root-context.xml 설정
<context:component-scan base-package="org.zerock.service"></context:component-scan>
<mybatis-spring:scan base-package="org.zerock.mapper" />
```