스프링 개요

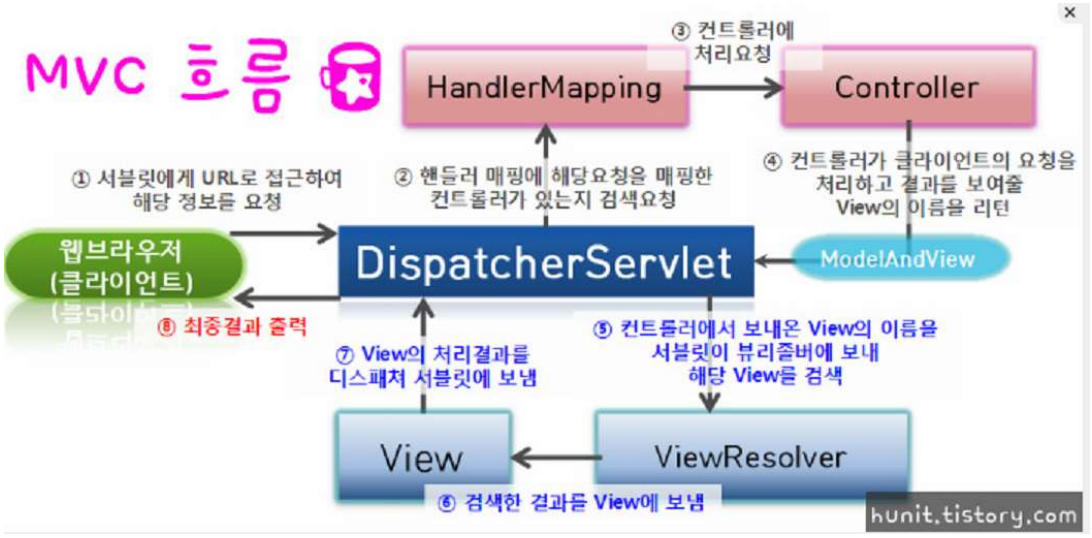
1. 웹사이트 개발을 **쉽고, 편리하게 개발할 수 있도록 지원해주는 오픈소스(공개된 의미) 프레임워크(외부 라이브러리에 의존)**로 경량급 애플리케이션 프레임워크

2. 기존 서블릿 MVC 자바언어 개발에 비해서 **개발 기간이 단축되고,코드 작성이 줄어들면서 유지 보수가 쉽다.**

3.스프링은 **로드 존슨이 2003년 오픈소스 프레임워크 프로젝트로 개발하기 시작함. 2004년 3월에 스프링 1.0이 발표되었다.**

4. 스프링은 철저히 MVC 패턴을 따르기 때문에 **개발자 코드영역과 UI 개발자 영역이 철저히 분리됨.** 그러므로 유지 보수가 뛰어나고, **개발자와 UI를 담당하는 분들과 출동 염려가 현격히 사라짐**.

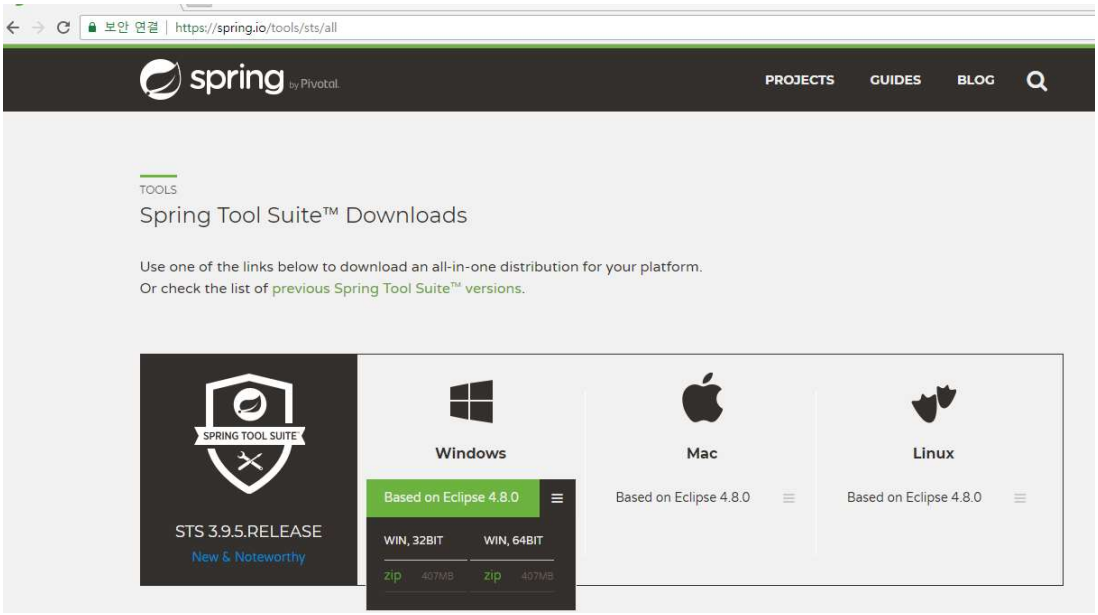
스프링 MVC 구조(여기서는 MVC에서 컨트롤과 뷰만 표시됨)



1. 스프링 개발툴 STS(SPring Tool Suite)

가. 윈도우를 사용한다면 현재 자신의 운영체제가 32비트인지, 64비트인지를 명확히 알고 있어야 한다. 32비트 jdk가 설치되었어면 STS개발툴 32비트를 다운받으면 되고,64비트 JDK가 설치되었다면 64비트 STS개발툴을 다운받으면 된다.

나. STS 다운로드 주소 : https://spring.io/tools



다. 되도록 영문 숫자 조합으로 된 폴더에 압축을 푼다. 압축이 푼 폴더에 STS.exe 실행파일이 있다. 이 파일을 실행해서 프로젝트 폴더가 저장될 워크스페이스 폴더를 생성한다.

라. Spring MVC Project를 생성하면 C:\Users\mun\.m2가 생성되고 그 하위 폴더에 repository 폴더가 생성되면서 스프링 관련 라이브러리가 다운로드 된다. 이 때 인터넷 연결이 불안정 하든지 네트워크가 불안하면 제대로 다운로드가 안되어 지고 해당 프로젝트에 에러 표시가 나타난다. 이런 경우는 에러가 없어 질 때 까지 반복적으로 .m2 폴더와 프로젝트를 지우면서 다시 프로젝트 생성해서 라이브러리를 다운받아 설치해야 한다.

마. **반드시 주의해야 할 한글 사용자 이름**

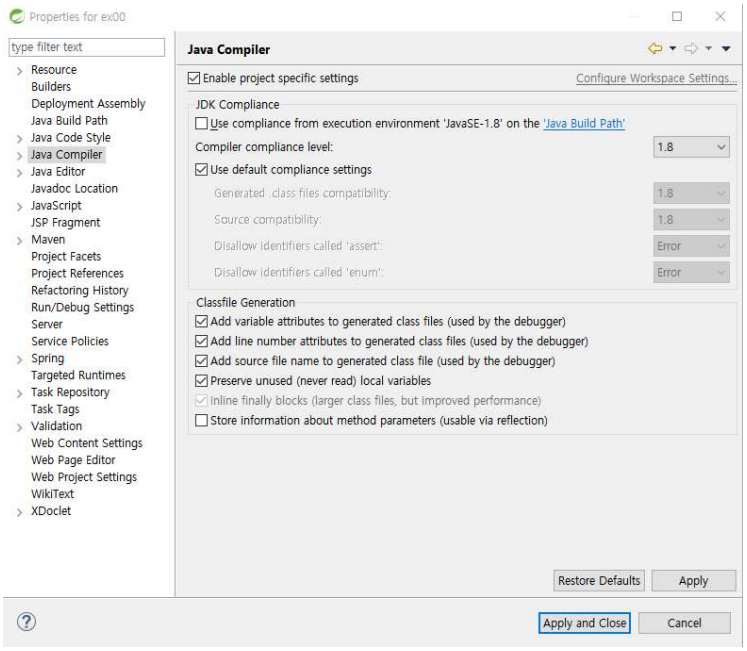
가끔 제대로 스프링 라이브러리가 설치가 안되는 경우는 사용자 이름과 폴더가 한글로 된 경우이다. 이런 경우는 사용자와 폴더를 영문 숫자조합으로 바꾸고 다시 다운받아 설치해야 한다. 그래도 안되면 운영체제를 다시 포맷해서 사용자 이름과 컴퓨터 이름을 반드시 한글이 아닌 영문 숫자조합으로 하고 해당 프로젝트를 생성해야 한다. 그래야 제대로 라이브러리가 다운 받아 설치된다. 이 한글 사용자이름 경로는 오라클에서도 해당 디비 서버를 sql development 클라이언트 연결 툴에서도 연결을 못하는 문제를 발생시킨다.

2. 프로젝트 실행 점검

가. 해당 프로젝트를 선택하고, 'Run' 메뉴의 'Run on Server'를 선택해서 실행한다.

그러면 웹브라우저에 Hello World!라는 문자열을 포함한 해당 내용이 보인다. 그러면 프로젝트가 제대로 만들어 진 것이다.

나. 'Project Facets' 메뉴 조금 위쪽에는 'Java Compiler' 메뉴가 있다. 이를 이용해서 현재 프로젝트의 jdk 버전을 1.8로 맞춘다.



3. 스프링 MVC 프로젝트 템플릿 구조

가. src/main/java : 패키지 이하 원본 자바파일

나. src/main/resources : mybatis 관련 xml파일 경로

다. src/test/java : 자바 테스트 경로

라. src/main/webapp/resources : html,css,javascript,jQuery,이미지 등 리소스 자원 경로

마. src/main/webapp/WEB-INF/spring : 스프링 관련 중요 설정파일

바. pom.xml : 메이븐 설정 파일

4. pom.xml 수정 내용

```
<?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
https://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>org.zerock</groupId>
	<artifactId>controller</artifactId>
	<name>ex00</name>
	<packaging>war</packaging>
	<version>1.0.0-BUILD-SNAPSHOT</version>

	<properties>
		<java-version>1.8</java-version>

<org.springframework-version>5.0.7.RELEASE</org.springframework-version>
		<org.aspectj-version>1.9.0</org.aspectj-version>
		<org.slf4j-version>1.7.25</org.slf4j-version>
	</properties>

	<dependencies>

	    <!-- 오라클 JDBC 드라이버 -->
		<dependency>
			<groupId>oracle.jdbc</groupId>
			<artifactId>jdbc</artifactId>
			<version>12.2.0.1</version>
			<scope>system</scope>

<systemPath>${project.basedir}/src/main/webapp/WEB-INF/lib/ojdbc8.jar</systemP
ath>
		</dependency>

			<!-- json 라이브러리 추가 -->
			<dependency>
				<groupId>com.fasterxml.jackson.core</groupId>
				<artifactId>jackson-databind</artifactId>
				<version>2.9.4</version>
```

```
                        </dependency>

                        <!-- 브라우저가 받는 데이터가 XML이라는 것을 확인할 수 있는 라이브러리 추가.
                        JSON데이터가 XML형태로 브라우저에서 보이면 매핑주소.json으로 입력하면
                        기존 xml데이터가 json데이터인 키,값 쌍으로 변경되어서 보여진다. -->
                        <dependency>
                         <groupId>com.fasterxml.jackson.dataformat</groupId>
                         <artifactId>jackson-dataformat-xml</artifactId>
                         <version>2.9.5</version>
                        </dependency>

                         <!-- mybatis-spring 부터 추가 -->
                        <dependency>
                                <groupId>org.mybatis</groupId>
                                <artifactId>mybatis-spring</artifactId>
                                <version>1.3.2</version>
                        </dependency>

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

                        <!-- spring jdbc -->
                        <dependency>
                                <groupId>org.springframework</groupId>
                                <artifactId>spring-jdbc</artifactId>

<version>${org.springframework-version}</version>
                        </dependency>

                        <!-- spring test -->
                        <dependency>
                                <groupId>org.springframework</groupId>
                                <artifactId>spring-test</artifactId>
```

```
<version>${org.springframework-version}</version>
                        </dependency>

                        <!-- 스프링 트랜잭션 적용 -->
                        <dependency>
                                <groupId>org.springframework</groupId>
                                <artifactId>spring-tx</artifactId>

<version>${org.springframework-version}</version>
                        </dependency>

                        <!-- aop aspectj 문법-->
                        <dependency>
                                <groupId>org.aspectj</groupId>
                                <artifactId>aspectjrt</artifactId>
                                <version>${org.aspectj-version}</version>
                        </dependency>

                        <dependency>
                                <groupId>org.aspectj</groupId>
                                <artifactId>aspectjweaver</artifactId>
                                <version>${org.aspectj-version}</version>
                        </dependency>

                        <!-- 끝 -->

                <!-- Spring -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context</artifactId>
                        <version>${org.springframework-version}</version>
                        <exclusions>
                                <!-- Exclude Commons Logging in favor of
SLF4j -->
                                <exclusion>
                                        <groupId>commons-logging</groupId>
                                        <artifactId>commons-logging</artifactId>
                                 </exclusion>
                        </exclusions>
```

```
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>${org.springframework-version}</version>
		</dependency>

		<!-- AspectJ -->
		<dependency>
			<groupId>org.aspectj</groupId>
			<artifactId>aspectjrt</artifactId>
			<version>${org.aspectj-version}</version>
		</dependency>

		<!-- 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>
		<dependency>
			<groupId>log4j</groupId>
			<artifactId>log4j</artifactId>
			<version>1.2.15</version>
			<exclusions>
				<exclusion>
					<groupId>javax.mail</groupId>
					<artifactId>mail</artifactId>
```

```
                        </exclusion>
                        <exclusion>
                                <groupId>javax.jms</groupId>
                                <artifactId>jms</artifactId>
                        </exclusion>
                        <exclusion>
                                <groupId>com.sun.jdmk</groupId>
                                <artifactId>jmxtools</artifactId>
                        </exclusion>
                        <exclusion>
                                <groupId>com.sun.jmx</groupId>
                                <artifactId>jmxri</artifactId>
                        </exclusion>
                </exclusions>
                <scope>runtime</scope>
        </dependency>

        <!-- @Inject -->
        <dependency>
                <groupId>javax.inject</groupId>
                <artifactId>javax.inject</artifactId>
                <version>1</version>
        </dependency>

        <!-- Servlet
        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>servlet-api</artifactId>
                <version>2.5</version>
                <scope>provided</scope>
        </dependency>-->

        <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>javax.servlet-api</artifactId>
                        <version>3.1.0</version>
                        <scope>provided</scope>
        </dependency>

        <dependency>
```

```
                        <groupId>javax.servlet.jsp</groupId>
                        <artifactId>jsp-api</artifactId>
                        <version>2.1</version>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>jstl</artifactId>
                        <version>1.2</version>
                </dependency>

                <!-- Test -->
                <dependency>
                        <groupId>junit</groupId>
                        <artifactId>junit</artifactId>
                        <version>4.12</version>
                        <scope>test</scope>
                </dependency>
        </dependencies>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-eclipse-plugin</artifactId>
                <version>2.9</version>
                <configuration>
                    <additionalProjectnatures>

<projectnature>org.springframework.ide.eclipse.core.springnature</projectnature>
                    </additionalProjectnatures>
                    <additionalBuildcommands>

<buildcommand>org.springframework.ide.eclipse.core.springbuilder</buildcommand>
                    </additionalBuildcommands>
                    <downloadSources>true</downloadSources>
                    <downloadJavadocs>true</downloadJavadocs>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
```

```
                <version>2.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <compilerArgument>-Xlint:all</compilerArgument>
                    <showWarnings>true</showWarnings>
                    <showDeprecation>true</showDeprecation>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.2.1</version>
                <configuration>
                    <mainClass>org.test.int1.Main</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

5. 스프링에 대한 간단한 소개

가. 스프링 MVC 프로젝트에서는 기본적으로 프로젝트 생성시 지정하는 패키지의 마지막 경로가 웹애플리케이션의 중간 경로가 된다. 예를 들면 org.zerock.controller 패키지를 작성했다면 중간 경로가 /controller이 된다. 최초 실행 경로는 브라우저 상에서 http://localhost:8081/controller가 된다.

나. 프레임워크란 말 그대로 '뼈대나 근간'을 이루는 코드들의 묶음이라고 볼 수 있다. 프레임워크의 최대 장점은 개발에 필요한 구조를 이미 코드로 만들어 놓았기 때문에 실력이 부족한 개발자라 하더라도 반쯤 완성한 상태에서 필요한 부분을 조립하는 형태의 개발이 가능하다는 점이다.

다. 의존성이란 어떤 객체가 혼자 일을 처리할 수 없다는 것을 의미한다.
라. 의존성 주입의 종류
 1) 생성자를 통한 의존성 주입 2)setter() 메서드를 통한 의존성 주입

마. AOP는 Aspect Oriented Programming의 약어로서 반복적인 코드를 줄이고,핵심 비즈니스 로직(개발코드)에만 집중할수 있게 해준다.
이 aop를 통한 데이터 베이스 트랙잭션 처리를 하게 한다. 스프링은 이런 트랜잭션의

관리를 애노테이션이나 xml로 설정할 수 있다.

6. JDBC 연결 자바 테스트 코드 만들기

jUnit을 이용한 JDBC의 연결 코드 작성은 src/test/java밑에 org.zerock.controller패키지 하위에 OracleConTest.java파일을 작성한다.

```
package org.zerock.controller;

import java.sql.Connection;
import java.sql.DriverManager;

import org.junit.Test;

public class OracleConTest {

	private static final String DRIVER=
			"oracle.jdbc.OracleDriver";
	private static final String URL=
			"jdbc:oracle:thin:@127.0.0.1:1521:xe";
	//오라클 접속 주소, 1521은 포트번호, xe는 데이터베이스
	//명
	private static final String USER="week";
	//오라클 접속 사용자
	private static final String PW="week";//접속 비번

	@Test
	public void testCon() throws Exception{
		Class.forName(DRIVER);//오라클 jdbc 드라이버 클래
		//스 로드
		try(Connection con=
	DriverManager.getConnection(URL,USER,PW)){
			System.out.println(con);
		}catch(Exception e) {e.printStackTrace();}
/* jdk 1.7이후에 추가된 try-with구문이다. 이것은 명시적인
 * close()를 하지 않아도 알아서 close() 해준다.
 * AutoCloseable 인터페이스를 구현한 타입이어야 한다.
 */
	}
```

```
}
```

7. 스프링+mybatis+oracle 설정

가. mybatis는 ibatis프레임웍이 단종되고 나온 것이다. 이 둘은 사촌관계이다. 지금도 단종된 ibatis가 현업에서 많이 사용된다. 스프링을 이용한 개발중에서 국내에서 가장 많이 쓰이는 형태는 mybatis와의 연동작업을 통해서 SQL문 처리에 대한 개발 생산성을 높이는 형태로 사용되는 것이다.

나. 스프링 MVC프로젝트 실행 순서
컨트롤러-> 서비스 -> DAOImpl -> mybatis를 통한 쿼리문 실행이고 컨트롤러를 통해서 뷰페이지 즉 jsp 파일로 이동한다.

다. mybatis 프레임웍의 장점
간결한 코드로 sql문을 처리한다. sql문을 따로 분리운영할 수 있다. 스프링과의 연동으로 자동화된 처리가 가능하다.

8. Spring 프로젝트에서 root-context.xml 파일의 수정

src/main/webapp/WEB-INF/spring/root-context.xml 파일은 STS가 스프링 MVC 프로젝트를 생성할 때 만들어 주는 중요한 파일이다. 이 파일에서 스프링 프레임워크에 다양한 설정을 하기 위해서는 네임스페이스 탭을 통한 XML네임스페이스 경로를 추가하는 것이 좋다.

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

```
                http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
                http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

        <!-- Root Context: defines shared resources visible to all other web
components -->
        <aop:aspectj-autoproxy />
        <!-- aop 자동 프록시 설정.  -->

        <!-- spring jdbc=> Data Source -->
        <bean id="dataSource"
        class=
"org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"
    value="oracle.jdbc.OracleDriver"/>
    <property name="url"
    value="jdbc:oracle:thin:@127.0.0.1:1521:xe"/>
    <property name="username" value="week"/>
    <property name="password" value="week" />
</bean>

    <!-- mybatis와 spring 연결설정과 mybatis환경설정 -->
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
     <property name="dataSource" ref="dataSource" />
     <!-- setter 메서드를 통한 의존성 주입(DI) -->
     <property name="configLocation"
     value="classpath:/mybatis-config.xml" />
<!-- classpath경로가 src/main/resources 이다. 이 경로에
mybatis-config.xml파일을 읽어들임. 이 파일의 기능은 도메인
데이터저장빈 클래스 객체명을 관리해준다. -->
<!-- 매퍼태그 경로 설정 -->
      <property name="mapperLocations"
      value="classpath:org/zerock/mappers/*/*.xml"/>
   </bean>

   <!-- 트랜잭션 설정 -->
   <bean id="transactionManager"
```

```
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
                <property name="dataSource" ref="dataSource"></property>
        </bean>
        <tx:annotation-driven />
        <!-- 트랜잭션 설정 애너테이션 ,
        @Transactional 애너테이션을 이용한 트랜잭션
        관리가 가능. -->

    <bean id="sqlSession"
class="org.mybatis.spring.SqlSessionTemplate"
destroy-method="clearCache">
     <constructor-arg
     name="sqlSessionFactory" ref="sqlSessionFactory"/>
     <!-- constructor-arg 의 뜻은 생성자를 통한 의존성
     주입 -->
</bean>

    <!-- DAOImpl의 @Repository이 설정된 부분을 스프링에
    오토 스캔하여 등록하게 함. -->
    <context:component-scan
    base-package="org.zerock.persistence" />

    <!-- 비지니스 오토스캔 추가. 이 패키지의 역할은 첫번째
    로 컨트롤과 모델 dao를 연결하는 중간 매개체 역할을 한
    다. 둘째 고객의 추가요구사항을 반영할 때 개발로직코드를
    유연하게 작성가능하게 한다. 셋째 스프링의 AOP를 통한
    트랜잭션 적용 핵심이다. -->
    <context:component-scan
    base-package="org.zerock.service" />

    <aop:config></aop:config>
        <!-- aop 설정=>AOP를 애너테이션 말고
        XML방식으로 설정할 때 사용 -->
</beans>
```

9. DataSource 테스트 진행
src/test/java 폴더내에서 DataSourceTest.java 파일을 작성한다.

```
package org.zerock.controller;

import java.sql.Connection;
```

```
import javax.inject.Inject;
import javax.sql.DataSource;

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)
@ContextConfiguration(
locations= {"file:src/main/webapp/WEB-INF/spring/*.xml"})
public class DataSourceTest {

	@Inject //자동의존성 주입->ds참조변수에 객체 주소를
	//주입해서 실제 사용할 수 있게 의존성 주입
	private DataSource ds;
	/* 스프링에서 의존성 주입방법
	 *  1.setter() 메서드를 통한 의존성 주입
	 *  2.생성자를 통한 의존성 주입
	 */

	@Test
	public void testCon() throws Exception{
		try(Connection con=ds.getConnection()){
			System.out.println(con);
		}catch(Exception e) {e.printStackTrace();}
	}
}
```

10. mybatis 연결

가. MyBatis는 SQL 매핑 프레임워크를 별도의 설정 파일을 통해 관리 할 수 있다.

나. 'src/main/resources' 경로에 mybatis-config.xml 파일 하나를 생성해 둔다. 이 파일은 자바 저장빈 클래스 별칭 객체를 관리한다.

11. mybatis 연결 테스트

모든 설정이 끝났다면 정상적으로 스프링에서 mybatis와 오라클 연결이 제대로 되었는지를 확인해 볼 필요가 있다. mybatis의 설정이 온전하게 되었는 지를 테스트하는 코드를 작성한다.

```
package org.zerock.controller;
```

```
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;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(
locations= {"file:src/main/webapp/WEB-INF/spring/*.xml"})
public class MybatisTest {

	@Autowired //자동의존성 주입
	private SqlSessionFactory sqlFactory;

	@Test
	public void testFactory() {
		System.out.println(sqlFactory);
	}

	@Test
	public void testSession() throws Exception{
	try(SqlSession sqlSession=sqlFactory.openSession()){
		System.out.println(sqlSession);
		//sqlSession이 mybatis쿼리문 실행객체
	}catch(Exception e) {e.printStackTrace();}
	}
}
```

12. 스프링 프로젝트 servlet-context.xml 파일 내용

가. 스프링 MVC 컨트롤러 실행에 있어서 가장 중요한 정보 몇가지를 볼수 있다.

나. appServlet 폴더내에 존재하는 이 파일은 스프링 MVC 관련 설정만을 분리하기 위해서 만들어진 파일이다.

다. servlet-context.xml 파일 주요 부분 내용

```
<annotation-driven />
	<!-- 클래스 선언에 애노테이션을 이용해서 스프링 컨트
	롤러를 작성할 수 있다. -->
```

```
<resources mapping="/resources/**" location="/resources/" />
    <!-- html,css,javascript,jQuery,이미지 등 관련파일
    이 들어가는 경로이다.-->

<!-- 뷰리졸브 설정->뷰페이지 파일경로와 확장자 -->
        <beans:
bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
                <beans:property name="prefix" value="/WEB-INF/views/" />
                <!-- jsp파일즉 뷰페이지 기본경로 . 기본경로를
                WEB-INF로 설정한 이유는 jsp파일을 직접 웹브라우저
                로 실행할수 없다.보안효과-->
                <beans:property name="suffix" value=".jsp" />
                <!-- jsp파일 기본 확장자이다. -->
        </beans:bean>

        <context:component-scan base-package="org.zerock.controller" />
        <!-- base-package 속성에 설정된 패키지 경로를
        스프링에 자동등록한다. 그 패키지 하위의 스프링 컨트
        롤 클래스를 자동으로 찾아서(오토스캔) 인식하게
        한다.이 부분은 annotation-driven 과 함께 결합해서
        해당 패키지에 애너테이션 처리가 됨.  -->
```

13. 기초적인 스프링 컨트롤러 생성 실습
org.zerock.controller 패키지내에 SampleController를 아래와 같이 작성한다.

```
package org.zerock.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller //@Controller 애너테이션을 사용하면 해당 컨트
//롤 클래스는 스프링에서 인식한다.
public class SampleController {

        @RequestMapping("doA") //get or post로 접근하는 doA
        //매핑주소를 웹주소창에서 실행하게 한다.
        public void doA7() {//리턴타입이 없는 void형이면 매핑
                //주소 doA가 jsp 파일명(doA.jsp) 즉 뷰페이지가 됨
                System.out.println("doA매핑주소가 실행됨");
        }

        @RequestMapping("doB")
```

```
        public void doB() {
                System.out.println("doB매핑주소가 실행됨.");
        }
}
```

14. 컨트롤러에서 메서드 리턴타입이 문자열인 경우는 '문자열+.jsp' 파일을 찾아서 실행하게 된다. SampleController2.java를 작성한다.

```
package org.zerock.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class SampleController2 {

        @RequestMapping("doC") //doC 매핑주소 등록
        public String doC(@ModelAttribute("msg") String msg) {
                //@ModelAttribute("msg")는 msg피라미터 이름에 인자
                //값을 문자열로 전달한다.웹주소 실행 매핑주소값으
                //doC?msg=문자열값 형태의 웹주소창에 노출되는 get
                //방식으로 전달한다.보안성이 안좋다.
                return "result";
                // WEB-INF/views/result.jsp 뷰페이지가 실행
        }
}
```

/WEB-INF/views/result.jsp는 아래와 같이 작성할 수 있다.

```
<%@ page contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> </title>
</head>
<body>
  전달된 피라미터 값:${msg}<%--${출력할 값} 형태의 표현
  언어로 사용 --%>
</body>
</html>
```

15. 데이터 저장빈 클래스를 활용한 컨트롤러와 뷰페이지 작성 실습

먼저 org.zerock.domain패키지에 데이터 저장빈 클래스 ProductVO.java를 작성한다.

```
package org.zerock.domain;

public class ProductVO {

	private String name;//상품명
	private double price;//가격

	public ProductVO(String name,int price) {
		this.name=name;
		this.price=price;//생성자의 주된기능인 멤버변수
		//초기화
	}//생성자를 오버로딩(매개변수가 없는 기본생성자를 묵시
	//적 제공하지 않는다.)

	public String getName() {
		return name;
	}

	public double getPrice() {
		return price;
	}

	@Override
	public String toString() {
		return "ProductVO [name="+name+",price="+
	price+"]";
	}
}
```

org.zerock.controller 패키지에 SampleController3.java를 작성한다.

```
package org.zerock.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.zerock.domain.ProductVO;

@Controller
public class SampleController3 {
```

```
	@RequestMapping("/nameprice")
	public String nameprice(Model m) {
		ProductVO p=new ProductVO("신발",120000);//오버로
		//딩된 생성자를 호출하면서 멤버변수값 초기화
		m.addAttribute("p",p);//p키이름에 p객체 저장
		return "shop/pro_name";
		// WEB-INF/views/shop/pro_name.jsp 뷰페이지 파일
		//이 실행
	}
}
```

WEB-INF/views/shop/pro_name.jsp 뷰페이지 파일을 작성한다.

```
<%@ page contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> </title>
</head>
<body>
 상품명=${p.name}<br/>
 상품가격=${p.price}<hr/>
</body>
</html>
```

15. 서버단 스프링 컨트롤러에서 리다이렉트를 해야 하는 경우

가. 가끔은 특정한 컨트롤러 로직을 처리할 때 다른 경로를 호출해야 하는 경우가 있다. 이 경우에는 스프링 MVC의 특별한 문자열인 'redirect:'을 이용하는대 ':'을 이용하는 것에 주의할 필요가 있다.

나. 리다이렉트를 하는 경우 RedirectAttributes라는 클래스를 피라미터로 같이 사용하게 되면 리다이렉트 시점에 원하는 데이터를 임시로 추가해서 넘기는 작업이 가능하다.

먼저 org.zerock.controller 패키지에 다음과 같은 컨트롤러 클래스를 만든다.

```
package org.zerock.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
```

```
@Controller
public class SampleController4 {

	@RequestMapping("/doE")
	public String doE(RedirectAttributes rttr) {
		rttr.addFlashAttribute("msg","this is seven");
		//다른 매핑주소로 msg키이름에 값을 담아서 전달할때
		//사용한다. 서버상에서 실행되기 때문에 웹주소창에
		//전달되는 값이 보이지 않는다. 보안이 좋다.
		return "redirect:/doF";//doE매핑주소가 실행되면
		//다른 매핑주소인 doF로 이동
	}

	@RequestMapping("/doF")
	public void doF(@ModelAttribute("msg") String name) {
		System.out.println("전달된 값:"+name);
	}
}
```

16. JSON 데이터를 생성하는 경우

스프링 MVC의 장점 중 하나는 최근 프로그래밍에서 많이 사용되는 JSON(Javascript Object Notation) 데이터에 대한 처리를 너무나 간단하게 처리할 수 있다는 것이다. 이를 위해서는 pom.xml 에 jackson-databind 라이브러리를 추가해야 한다.

스프링 컨트롤러에서 JSON데이터를 생성하기 위한 작업을 해야 한다. JSON객체를 반환하기 위해서 @ResponseBody 애노테이션을 추가해 주는 작업만 하면 된다. JSON데이터는 키,값 쌍구조로 되어 있다.

org.zerock.controller 패키지에 다음과 같은 컨트롤로 클래스 작업을 한다.

```
package org.zerock.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import org.zerock.domain.ProductVO;

@Controller
public class SampleController5 {

	@RequestMapping("/doJSON")
	public @ResponseBody ProductVO doJSON() {
//@ResponseBody 애노테이션을 사용하면 jsp파일을 만들지 않
```

```
//고도 웹브라우저에 키,값 쌍 의 json데이터를 쉽게 출력할
//수 있다.
                ProductVO p=new ProductVO("수박",15000);
                return p;//json데이터 키이름에 ProductVO빈클래스
                //변수명이 된다.
        }
}
```

17. 스프링 + mybatis

mybatis는 JDBC에서 개발자가 직접 처리하는 PreparedStatement의 ?에 대한 설정이나 ResultSet을 이용한 처리가 이루어지기 때문에 기존 방식에 비해 개발의 생산성이 좋아진다. 국내의 대부분 프로젝트는 XML만을 이용해서 SQL문을 작성하고, 별도의 DAOImpl 만드는 방식을 선호한다. 이 방식의 최대장점은 sql문을 완전히 분리해서 처리하기 때문에 향후에 sql문의 변경이 일어날 때, 대처가 수월하다는 장점이 있다.

가장 먼저 개발에 필요한 테이블을 생성한다.

```
--tbl_member 테이블 만들기
create table tbl_member(
 userid varchar2(50) primary key --회원 아이디
 ,userpw varchar2(50) not null --회원비번
 ,username varchar2(50) not null --회원이름
 ,email varchar2(100) --전자우편
 ,regdate date --가입날짜
 ,updatedate date --수정날짜
);
```

테이블 컬럼명과 일치하는 변수명을 가진 데이터 저장빈 클래스를 만든다.

```
package org.zerock.domain;



public class MemberVO {
/* mybatis을 사용하기 위해서 테이블 컬럼명과 일치하는 빈
 * 클래스 변수명을 정의한다.
 */
        private String userid;//회원아이디
        private String userpw;//비번
        private String username;//회원명
        private String email;//전자우편
        private String regdate;//가입날짜
        private String updatedate;//수정날짜

        public String getUserid() {
```

```
                return userid;
        }
        public void setUserid(String userid) {
                this.userid = userid;
        }
        public String getUserpw() {
                return userpw;
        }
        public void setUserpw(String userpw) {
                this.userpw = userpw;
        }
        public String getUsername() {
                return username;
        }
        public void setUsername(String username) {
                this.username = username;
        }
        public String getEmail() {
                return email;
        }
        public void setEmail(String email) {
                this.email = email;
        }
        public String getRegdate() {
                return regdate;
        }
        public void setRegdate(String regdate) {
                this.regdate = regdate;
        }
        public String getUpdatedate() {
                return updatedate;
        }
        public void setUpdatedate(String updatedate) {
                this.updatedate = updatedate;
        }
}
```

org.zerock.persistence 패키지에 MemberDAO 인터페이스를 만든다.

```
package org.zerock.persistence;


import org.zerock.domain.MemberVO;
```

```
public interface MemberDAO {

	void insertM(MemberVO m);

}
```

MemberDAO 인터페이스를 구현한 MemberDAOImpl.java 클래스를 작성한다.

```
package org.zerock.persistence;

import javax.inject.Inject;

import org.apache.ibatis.session.SqlSession;
import org.springframework.stereotype.Repository;
import org.zerock.domain.MemberVO;

@Repository //@Repository 애노테이션은 DAO를 스프링에서
//인식하게 한다.
public class MemberDAOImpl implements MemberDAO{

	@Inject //자동의존성 주입(DI설정)
	private SqlSession sqlSession;//mybatis쿼리문 실행객체
	//sqlSession생성

	@Override
	public void insertM(MemberVO m) {
	    this.sqlSession.insert("m_in",m);
	    /* 1.m_in은 member.xml에서 설정한 insert매퍼태그의
	     * 아이디명이다.동일한 아이디명이 있어면 안된다.
	     * 2.mybatis쿼리문 실행메서드
	     *  가.insert():레코드 저장
	     *  나.update():레코드 수정
	     *  다.delete():레코드 삭제
	     *  라.selectOne():단 한개의 레코드만 검색
	     *  마.selectList():하나이상의 레코드를 검색해서
	     *  컬렉션 List로 반환
	     */
	}
}
```

Mybatis에서 sql문을 저장하는 존재를 Mapper라는 용어로 표현한다. 매퍼태그를 작성해

서 쿼리문을 작성한다. org.zercok.mappers.member패키지에 매퍼 태그 xml파일 member.xml을 작성한다. mybatis는 기본적으로 PreparedStatement를 이용해서 처리된다. 개발자가 PreparedStatement에 들어가는 피라미터를 사용할 때는 #{}기호를 이용해서 처리한다.

```
<?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="Member">
 <!-- mybatis쿼리문을 다루는 태그를 매퍼라 한다. -->

  <!-- 회원가입 -->
  <insert id="m_in" parameterType="m">
   <!-- parameterType속성은 전달인자 타입 -->
   insert into tbl_member values(#{userid},#{userpw},
   #{username},#{email},sysdate,sysdate)
   <!-- #{userid}는 자바코드로 m.getUserid()로 표현 -->
  </insert>
 </mapper>
```

스프링에 빈 등록하기

MemberDAOImpl.java 에 @Repository 애노테이션이 설정되더라도 스프링에서 해당 패키지를 스캔하지 않으면 스프링에 빈 객체로 등록되지 못한다. 이 처리는 root-context.xml을 이용해서 아래와 같이 작성한다.

```
<!-- DAOImpl의 @Repository이 설정된 부분을 스프링에
    오토 스캔하여 등록하게 함. -->
    <context:component-scan
    base-package="org.zerock.persistence" />
```

src/test/java 자바 테스트 코드 작성하기

```
package org.zerock.controller;


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.domain.MemberVO;
import org.zerock.persistence.MemberDAO;
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(
locations= {"file:src/main/webapp/WEB-INF/spring/*.xml"})
public class MemberDAOTest {

	@Autowired //자동의존성 주입
	private MemberDAO dao;

	@Test
    public void testInsertMember() throws Exception{
			MemberVO m=new MemberVO();
			m.setUserid("kkkkk");
			m.setUserpw("77777");
			m.setUsername("이순신");
			m.setEmail("sin@naver.com");

			dao.insertM(m);//회원저장
	}
}
```