

61 76 103
114 131

Spring 3.x 기초

1. 프레임워크란? [프레임워크이란 잘 정의된 약속된 구조의 클래스의 집합을 의미한다.]

사전적 의미로는 "어떠한 것을 이루는 뼈대, 기본 구조"를 뜻한다.

소프트웨어에서의 프레임워크란 소프트웨어의 특정 문제를 프로그램으로 쉽게 그리고 편리하게 개발할 수 있도록 미리 뼈대를 이루는 클래스와 인터페이스를 제작하여 이것들을 모아둔 것이라고 할 수 있다.

다시 말해 어떤 영역의 API 들을 사용하기 편리한 형태로 포장해 놓은 것이라 할 수 있다.

- 프레임워크의 장점- 개발자의 할 일을 줄여준다.

- 정해진 틀 안에서 코딩하기 때문에 프로그램의 가독성이 높아지므로 유지보수가 용이하다.
- 선언적인 방법을 도입하여 프로그램의 유지보수가 용이하다.

- 프레임워크의 단점

- 일관성의 가치를 지나치게 높게 평가한 나머지 코딩 패턴에 너무나 많은 구속이 따른다.
- 코드가 길어지고 복잡해질 수 있다.

2. 스프링 설치

- ① <http://maven.springframework.org/release/org/springframework/spring/> 사이트에 접속해서 다운로드 받는다.

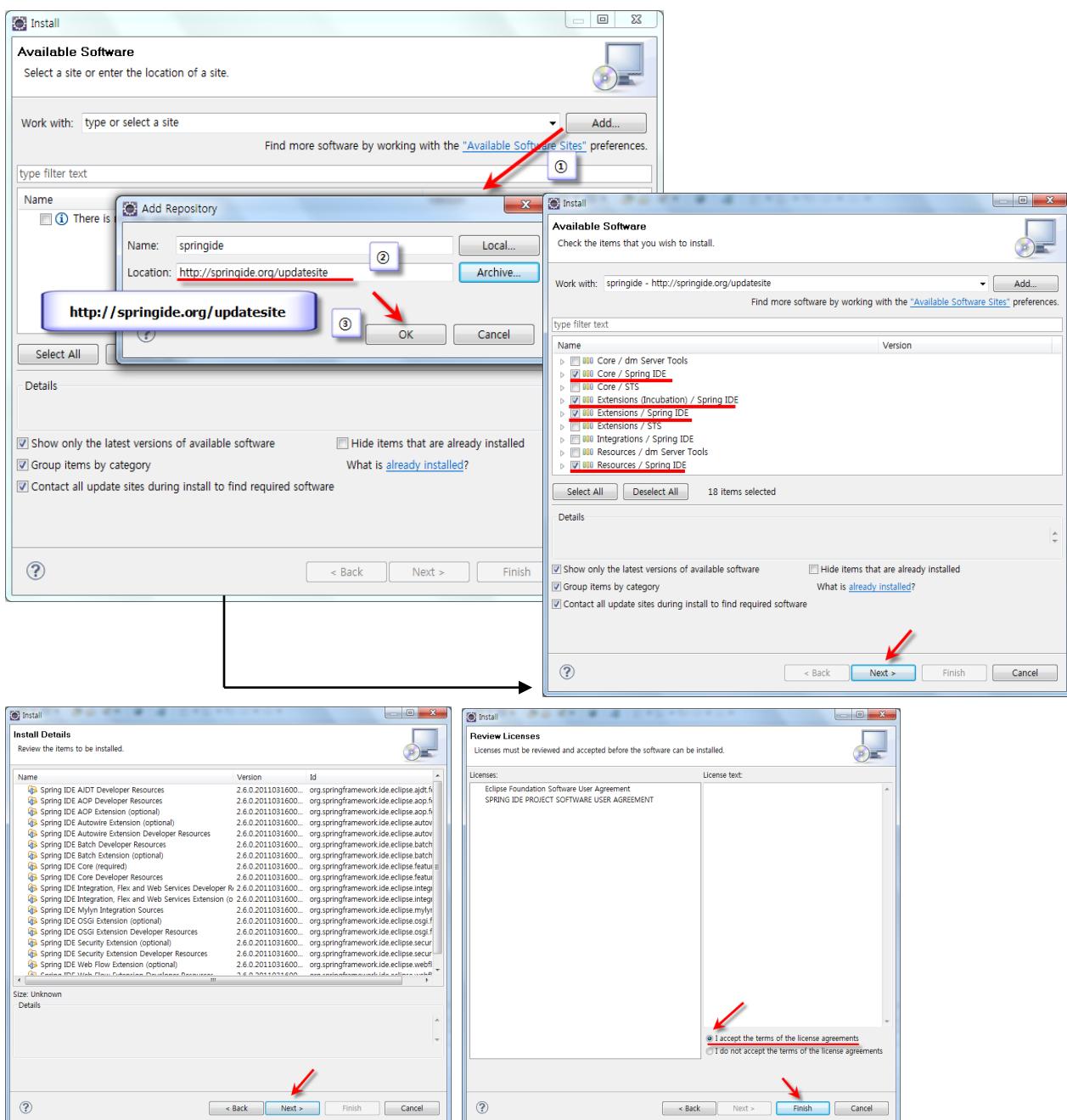
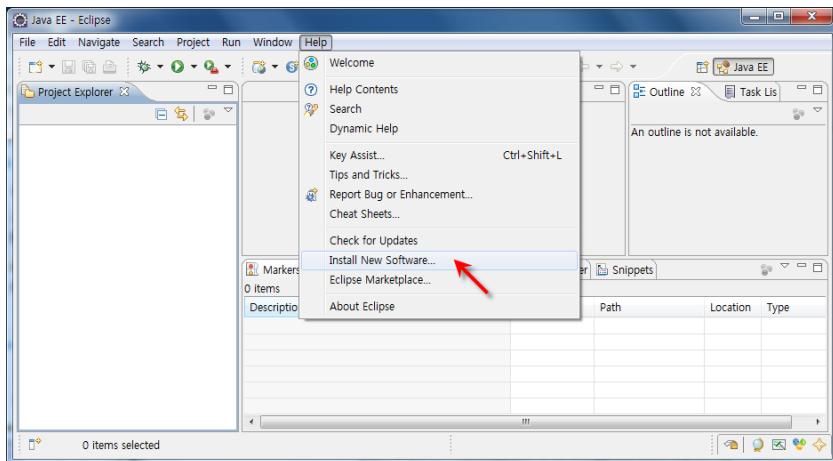
Name	Last modified	Size
...		
spring-framework-3.2.2.RELEASE-dist.zip	13-Mar-2013 21:09	47.93 MB
spring-framework-3.2.2.RELEASE-dist.zip.asc	13-Mar-2013 21:11	183 bytes
spring-framework-3.2.2.RELEASE-dist.zip.asc.md5	13-Mar-2013 21:11	32 bytes
spring-framework-3.2.2.RELEASE-dist.zip.asc.sha1	13-Mar-2013 21:11	40 bytes
spring-framework-3.2.2.RELEASE-dist.zip.md5	13-Mar-2013 21:09	32 bytes
spring-framework-3.2.2.RELEASE-dist.zip.sha1	13-Mar-2013 21:09	40 bytes
spring-framework-3.2.2.RELEASE-docs.zip	13-Mar-2013 21:11	24.18 MB
spring-framework-3.2.2.RELEASE-docs.zip.asc	13-Mar-2013 21:12	183 bytes
spring-framework-3.2.2.RELEASE-docs.zip.asc.md5	13-Mar-2013 21:12	32 bytes
spring-framework-3.2.2.RELEASE-docs.zip.asc.sha1	13-Mar-2013 21:12	40 bytes
spring-framework-3.2.2.RELEASE-docs.zip.md5	13-Mar-2013 21:11	32 bytes
spring-framework-3.2.2.RELEASE-docs.zip.sha1	13-Mar-2013 21:11	40 bytes
spring-framework-3.2.2.RELEASE-schema.zip	13-Mar-2013 21:13	206.39 KB
spring-framework-3.2.2.RELEASE-schema.zip.asc	13-Mar-2013 21:13	183 bytes
spring-framework-3.2.2.RELEASE-schema.zip.asc.md5	13-Mar-2013 21:13	32 bytes
spring-framework-3.2.2.RELEASE-schema.zip.asc.sha1	13-Mar-2013 21:13	40 bytes
spring-framework-3.2.2.RELEASE-schema.zip.md5	13-Mar-2013 21:13	32 bytes
spring-framework-3.2.2.RELEASE-schema.zip.sha1	13-Mar-2013 21:13	40 bytes

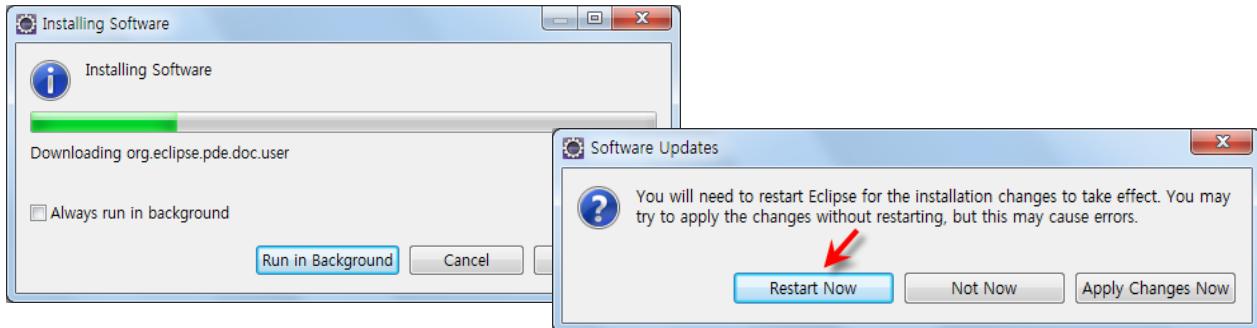
Artifactory Online Server at maven.springframework.org Port 80

C:\₩에 spring-framework-3.2.2.RELEASE-dist.zip 압축을 푸다(spring-framework-3.0.2.RELEASE.zip)

또는 <http://docs.spring.io/downloads/nightly/release-download.php?project=SPR>

② 현재 사용하는 이클립스에 스프링 플러그인을 설치한다.





1. 스프링의 기본 기능과 구조(애플리케이션)

스프링은 콘솔 어플리케이션이나 Servlet/JSP를 사용한 웹 어플리케이션, 또는 스윙과 같은 GUI 어플리케이션 등 모든 자바 어플리케이션에서 이용할 수 있다.

세 개의 예제를 통해서 기능 및 구조에 대한 기본개념을 알아본다.

어플리케이션의 main()메소드를 실행하면 화면에 "Hello, ○○!" 또는 "○○씨, 안녕하세요!"라는 메시지를 출력한다.

첫 번째(sample1)와 두 번째(sample2)에서는 스프링을 사용하지 않고 세 번째(sample3)에서는 스프링을 사용한다.

예제 설명

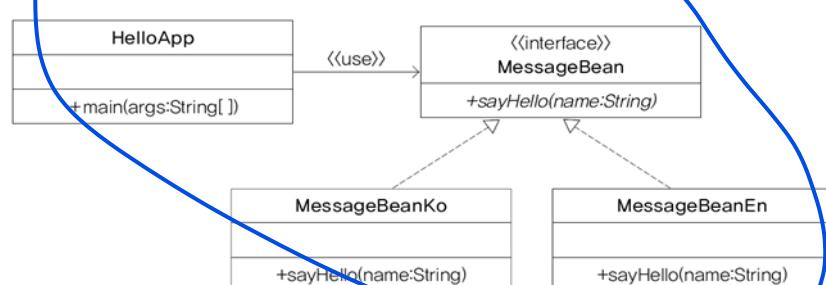
첫 번째 예제는 단순한 샘플 어플리케이션으로 sample1패키지에 MessageBean과 HelloApp라는 2개의 클래스가 있다.



MessageBean 클래스는 멤버로 sayHello()메소드를 한 개 가지며, 이 메소드는 '이름'을 String형 인수로 받아서 화면에 'Hello, 000! '라고 출력한다.

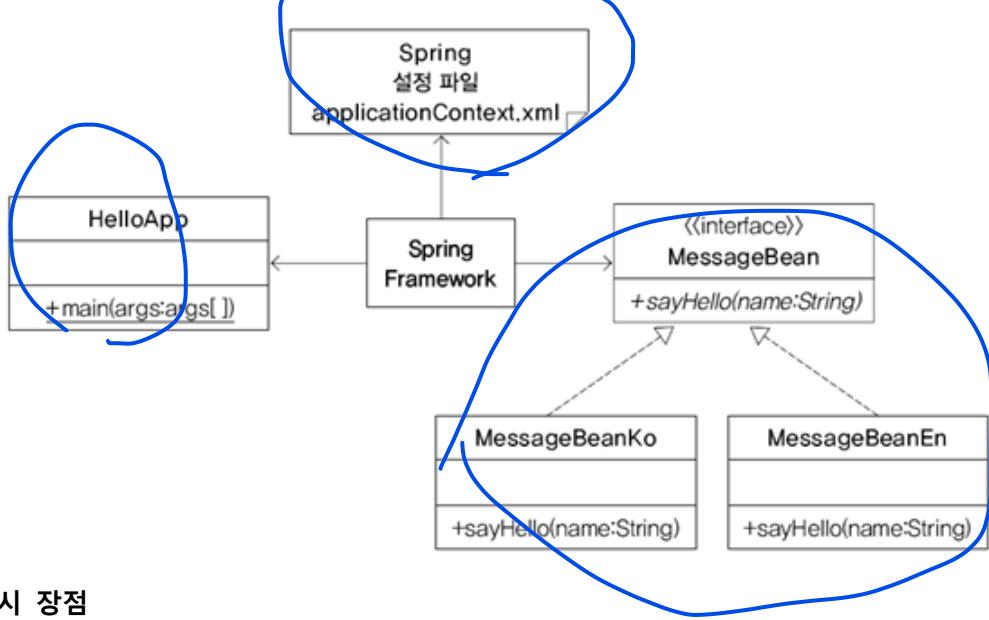
HelloApp 클래스는 main()메소드 안에서 MessageBean의 인스턴스를 생성하여 sayHello()메소드를 호출할 뿐이다.

두 번째 샘플 어플리케이션은 인터페이스를 이용하여 클래스 사이의 의존관계를 약하게 설정하고 있다. sample2 패키지에 MessageBean 인터페이스를 마련하고, 이 인터페이스를 구현한 두 개의 클래스 (MessageBeanEn과 MessageBeanKo)를 작성하였다.



MessageBeanKo 클래스의 sayHello()메소드는 메시지를 한글로, MessageBeanEn 클래스의 sayHello()메소드는 메시지를 영어로 출력한다.

세 번째 예제에서는 스프링을 사용한다. 이 예제에서는 스프링이 MessageBean 클래스의 인터페이스를 생성한다. MessageBean 인터페이스의 구현 클래스는 MessageBeanKo란 이름과 MessageBeanEn이라는 이름으로 두 개를 작성하여 HelloApp 클래스에서 이용할 수 있도록 한다. 인터페이스의 생성에 관한 정보는 스프링 설정 파일로부터 모두 읽어 들인다.



스프링 사용 시 장점

스프링을 사용하면 '실행시 필요한 클래스를 스프링이 준비해준다.'는 스프링의 최대 장점을 누릴 수 있다. 애플리케이션에서 필요한 클래스를 준비하는 방법은 sample1처럼 단순히 new를 사용해 인스턴스를 생성하는 경우도 있고, 디자인 패턴의 Factory Method 패턴을 사용해 JNDI(Java Naming and Directory Interface)를 통해 인스턴스를 취득하는 경우도 있다.

스프링을 사용하면 인스턴스를 준비하는 역할을 스프링이 맡아서 하므로 인스턴스 준비코드가 필요없다. 결국, new를 사용해 인스턴스를 생성하거나 JNDI로 인스턴스를 취득하는 코드를 만들지 않아도 된다.

구체적으로 살펴보면,

new를 사용하면 클래스 이름을 직접 코드에 작성한다. 작성하지 않으면 실제 사용하는 클래스를 몰라도 된다. 이것은 클래스간 결합이 약해져 의존관계가 약해지는 것을 말한다. 결국, 변경이 용이하고, 테스트가 간편하며, 콤파넌트 재사용성이 높은 애플리케이션을 만드는 것이 가능해진다.

이런 방식을 쓰면 의존하는 객체를 Mock Object(유사 객체)로 변경할 때도 코드를 다시 작성하지 않고 스프링 설정 파일만 변경하면 된다.

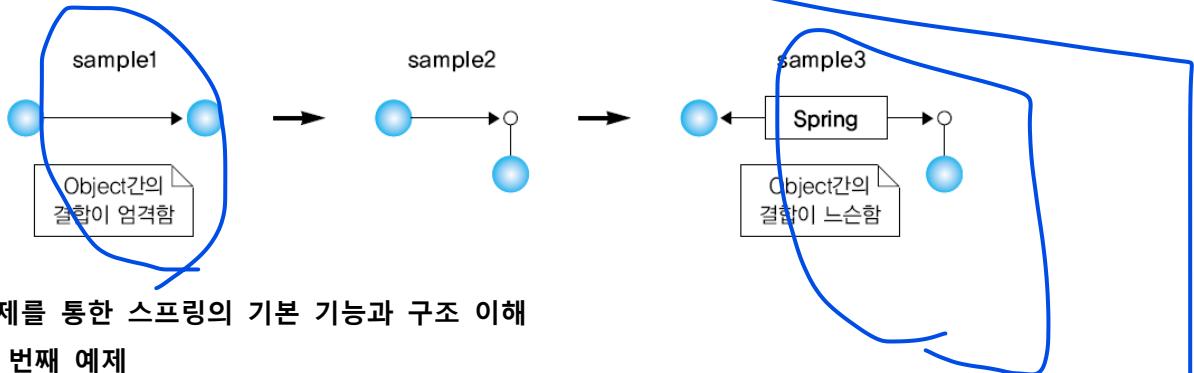
스프링을 활용하기 위한 설계

의존 관계가 약한 클래스를 작성하기 위해서 스프링은 인터페이스를 설계하고, 이렇게 설계한 인터페이스는 의존 관계에 있는 클래스를 불러낼 때 전달 인자로 사용한다.

메소드에서 사용되는 클래스가 어떤 것이든지 간에 전달인자로 클래스의 고유명을 코딩하지 않고 인터페

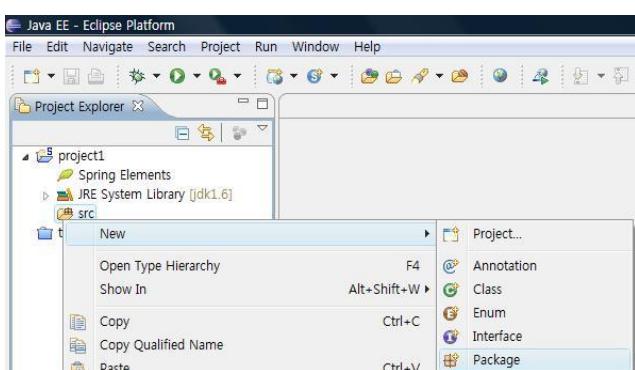
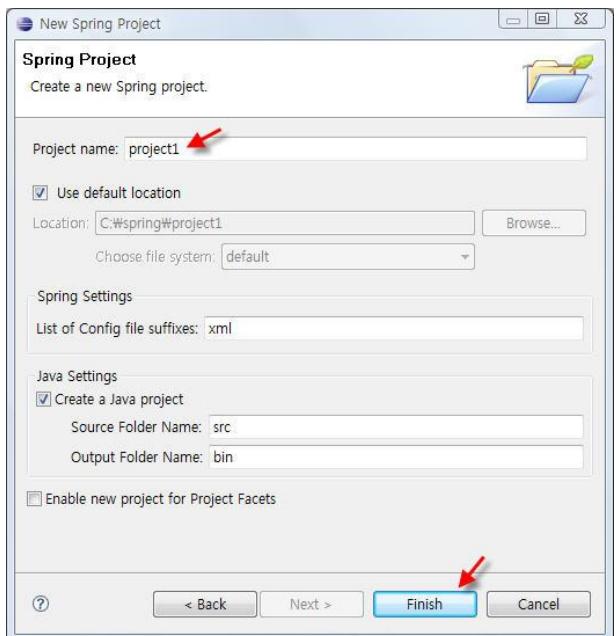
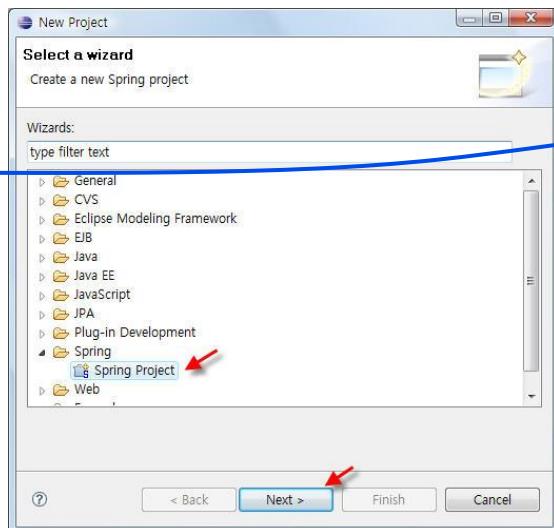
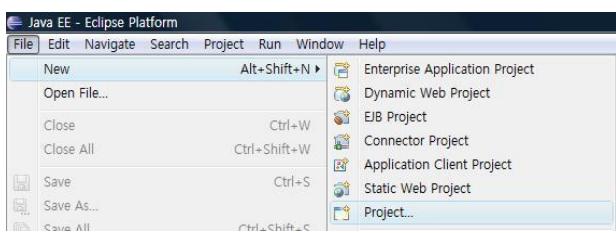
이스를 기술하여 메소드를 정의한다. 이 메소드에 전달될 인스턴스는 스프링 설정 파일에서 결정 하도록 하고, 다양한 클래스에 대해서 테스트할 경우에는 스프링 설정 파일을 변경하여 클래스 사이의 결합 상태를 느슨하게 한다.

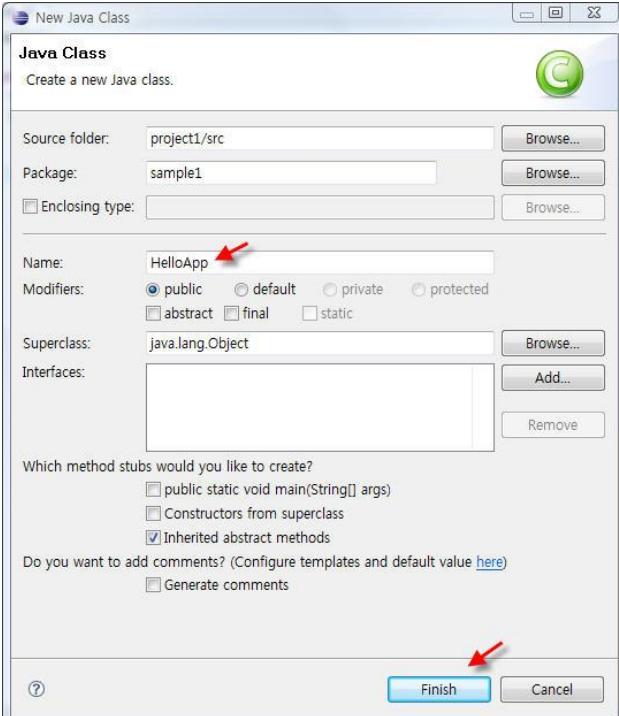
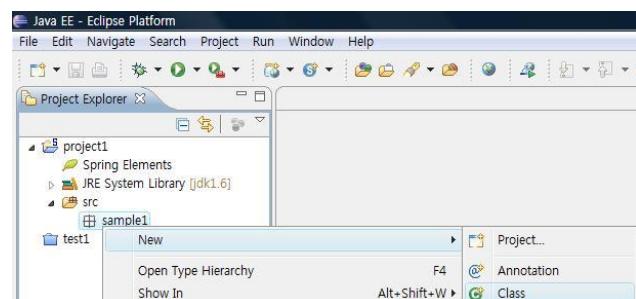
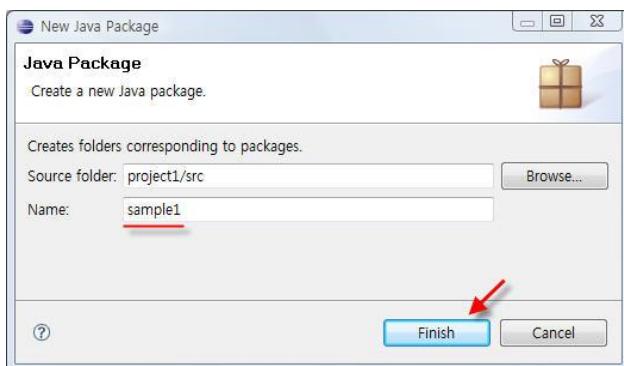
~~sample1에서 작성한 어플리케이션은 인터페이스 베이스로 설계되어 있지 않기 때문에 우선은 sample2에서 인터페이스 베이스 설계로 변경하였고, sample3에서는 인터페이스 베이스 설계로 변경한 sample2에 스프링을 적용하는 내용을 덧붙여서 클래스 사이의 결합 상태를 느슨하게 하였다.~~



- 세 개의 예제를 통한 스프링의 기본 기능과 구조 이해

1) 간단한 첫 번째 예제





MessageBean.java

```
package sample1;
public class MessageBean {
    public void sayHello(String name) {
        System.out.println("Hello, " + name + "!");
    }
}
```

HelloApp.java

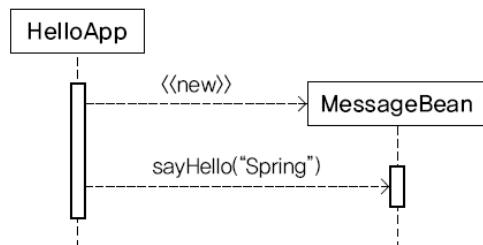
```
package sample1;
public class HelloApp {
    public static void main(String[] args) {
        MessageBean bean = new MessageBean();
    }
}
```

```

        bean.sayHello("Spring");
    }
}

```

[Run As] -> [Java Application] 실행하면 "Hello, Spring!"이 출력됨을 볼 수 있다.

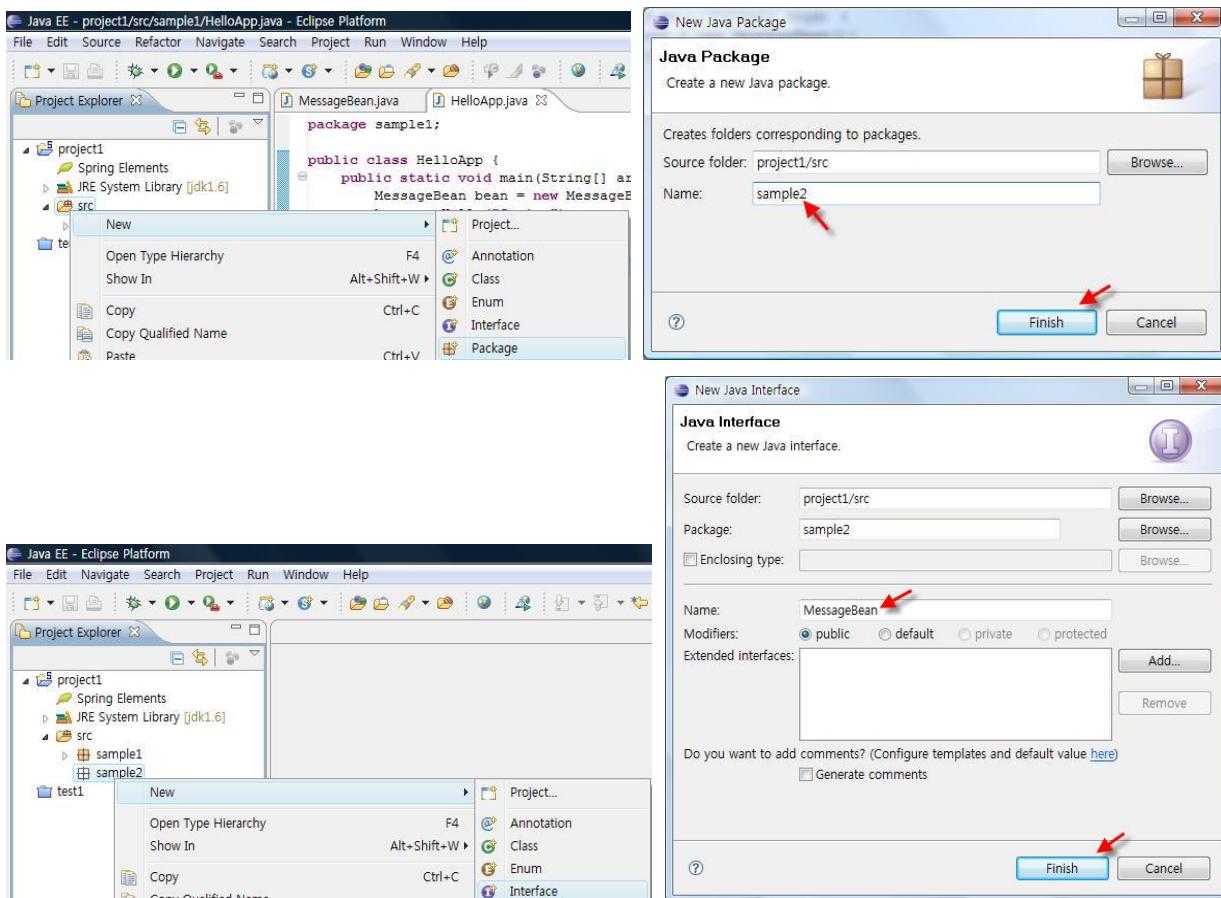


HelloApp 클래스에서 MessageBean 클래스로 직접 인스턴스를 생성하여, sayHello() 메소드를 호출하고 있기 때문에 HelloApp 클래스가 MessageBean 클래스를 강하게 의존하고 있다.

이런 구조는 클래스를 자주 변경해야 하는 유연한 어플리케이션을 구축할 때에는 코드를 대폭 수정해야 하는 문제점이 있다. 만일, MessageBean 클래스 대신 다른 클래스를 HelloApp 클래스에서 사용할 경우, 그 클래스에 sayHello() 메소드가 멤버로 존재한다는 보장이 없기 때문에 변경한 클래스의 메소드로 코드를 수정해야 한다.

2) 두 번째 예제(인터페이스 사용)

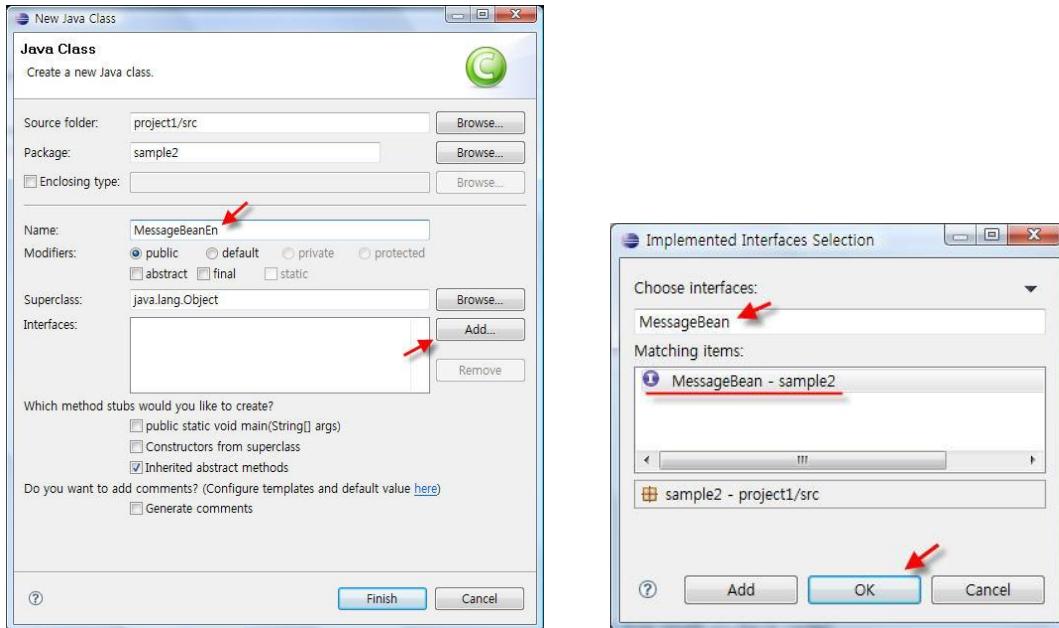
sample2 패키지 생성



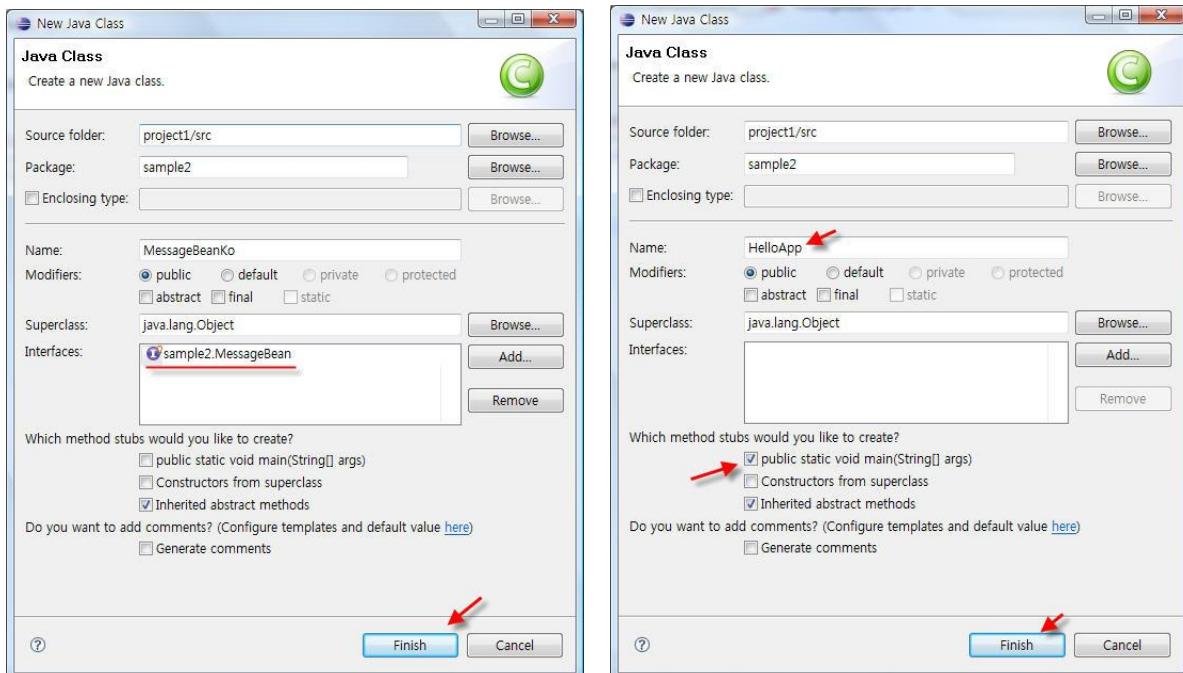
MessageBean.java (인터페이스 생성)

```
package sample2;
public interface MessageBean {
    void sayHello(String name);
}
```

MessageBeanEn.java -> MessageBean 인터페이스 연결



MessageBeanKo.java -> MessageBean 인터페이스 연결



MessageBeanEn.java

```
package sample2;  
public class MessageBeanEn implements MessageBean {  
    @Override  
    public void sayHello(String name) {  
        System.out.println("Hello, " + name + "!");  
    }  
}
```

MessageBeanKo.java

```
package sample2;  
public class MessageBeanKo implements MessageBean {  
    @Override  
    public void sayHello(String name) {  
        System.out.println("안녕하세요! " + name + "씨");  
    }  
}
```

HelloApp.java

```
package sample2;  
public class HelloApp {  
    public static void main(String[] args) {  
        MessageBean bean = new MessageBeanKo();  
        //MessageBean bean = new MessageBeanEn();  
        bean.sayHello("Spring");  
    }  
}
```

한글 메시지가 출력되기 위해 MessageBeanKo 클래스의 인스턴스를 생성한다.

물론 인터페이스로 의존관계를 약하게 설정하여 코드를 변경하지 않아도 되는 부분도 있지만, 여전히 코드를 변경해야 하는 부분이 남아있다.

MessageBean bean = new
bean.sayHello("Spring");

MessageBeanKo();

아직은 수정해야 하는 코드가 존재

수정하지 않아도 되는 코드

3) 세 번째 예제(스프링 사용)

- New Java Package -> sample3을 만든다.

- sample2 패키지에서 사용한 MessageBean.java, MessageBeanEn.java, MessageBeanKo.java 세 개의 파일을 sample3 패키지에 복사한다.

HelloApp.java

```
package sample3;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

@SuppressWarnings("deprecation")
public class HelloApp {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new
                FileSystemResource("applicationContext.xml"));
        MessageBean bean = factory.getBean("messageBean", MessageBean.class); //①
        bean.sayHello("Spring");
    }
}
```

sample3 에서 핵심은 HelloApp 클래스가 사용하는 클래스(MessageBeanEn)의 클래스 이름을 코드에 직접 기술하지 않는다.

①에서 getBean() 메소드의 인수에 지정되어 있는 messageBean은 클래스 이름이 아닌 단순 문자열 식별자이다. 이 식별자에 대응하는 클래스는 외부 정의 파일(스프링 설정 파일)에 정의 되어 있기 때문에 HelloApp 클래스에서 사용하는 클래스를 변경하는 경우 외부 정의 파일만 수정하면 된다.

스프링을 사용해 Bean을 취득하는 경우 기본적으로 다음과 같은 절차를 따른다.

- ① Bean 팩토리를 생성한다.
- ② Bean 팩토리로부터 Bean을 취득한다.

스프링 기능에서 Bean의 생성과 설정, 관리 등의 역할을 담당하는 부분을 "Bean 팩토리"라고 한다.

Bean 팩토리는 org.springframework.beans.factory.xml.XmlBeanFactory 인터페이스로 제공된다.

다음과 같은 메소드가 있다.

• BeanFactory 인터페이스에서 제공하는 메소드

메소드	설명
Boolean containBean(String name)	인수로 지정한 이름의 Bean이 정의되어 있는지 여부를 반환한다.
String[] getAliases(String name)	Bean의 이름에 별칭이 정의되어 있는 경우, 그 별칭을 반환한다.

Object getBean(String name)	인수로 지정된 이름의 Bean 인스턴스를 생성하여 반환한다.
<T>T getBean(String name, Class<T> requiredType)	인수로 지정된 이름의 Bean 인스턴스를 생성하여 반환한다.
Class getType(String name)	인수로 지정된 이름의 Bean의 타입을 반환한다.
Boolean isSingleton(String name)	Bean이 Singleton인지 여부를 반환한다.

스프링에는 BeanFactory 인터페이스를 구현한 여러 개의 클래스가 준비되어 있다. 이들 클래스 중에서도 가장 일반적으로 사용되는 것이 XmlBeanFactory 클래스이다. **XML 설정 파일에서 빈의 생성 및 설정, 관리 정보를 취득하기 위해 사용한다.**

XmlBeanFactory의 생성자에는 Resource 인터페이스를 구현하는 클래스가 여러 개 준비되어 있다. 그러나 XML 설정 파일을 이용할 때는 org.springframework.core.io 패키지에 있는 ClassPathResource 클래스나 FileSystemResource 클래스 이용을 권장한다.

ClassPathResource 클래스를 사용하여 빈 팩토리를 생성하는 예

```
Resource resource = new ClassPathResource("applicationContext.xml");
BeanFactory factory = new XmlBeanFactory(resource);
```

FileSystemResource 클래스를 사용하여 빈 팩토리를 생성하는 예

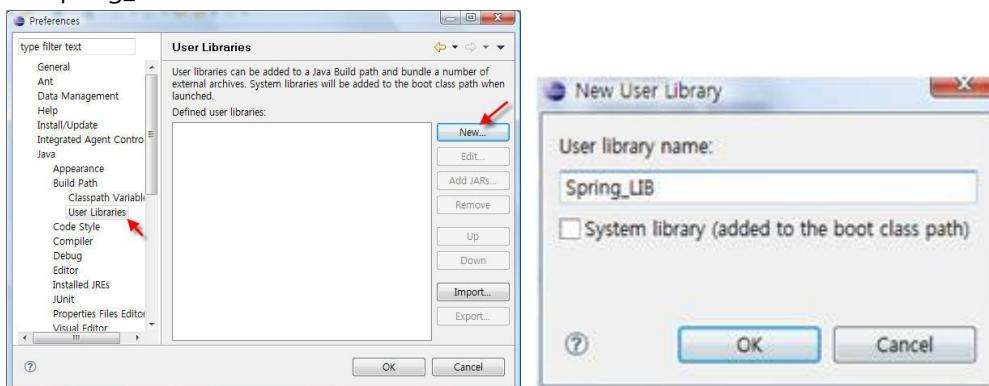
```
BeanFactory factory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));
```

BeanFactory 인스턴스를 생성하고 나면 getBean()메소드를 호출해서 사용하려는 Bean 인스턴스를 취득한다. getBean()메소드의 인수는 XML 정의 파일에서 정의한 Bean의 이름(messageBean) 문자열과 BeanFactory에서 취득할 클래스를 지정한다.

○ 스프링을 위한 jar 파일(User Libraries 추가방법)

애플리케이션에서 스프링 기능을 이용하려면 spring-beans-3.2.2.RELEASE.jar와 spring-core-3.2.2.RELEASE.jar 그리고 commons-logging-1.1.2.jar를 외부 라이브러리로 추가해야 한다.

사용자 라이브러리로 Spring_LIB를 새롭게 만들어서 여기에 spring-beans-3.2.2.RELEASE.jar와 spring-core-3.2.2.RELEASE.jar 그리고 commons-logging-1.1.2.jar를 추가해 놓고, 스프링 기능을 필요로 하는 프로젝트의 Spring_LIB 사용자 라이브러리를 사용한다.



<http://commons.apache.org/proper/commons-logging/>에서 commons-logging-1.1.2-bin.zip 다운받아 압축을 풀다.

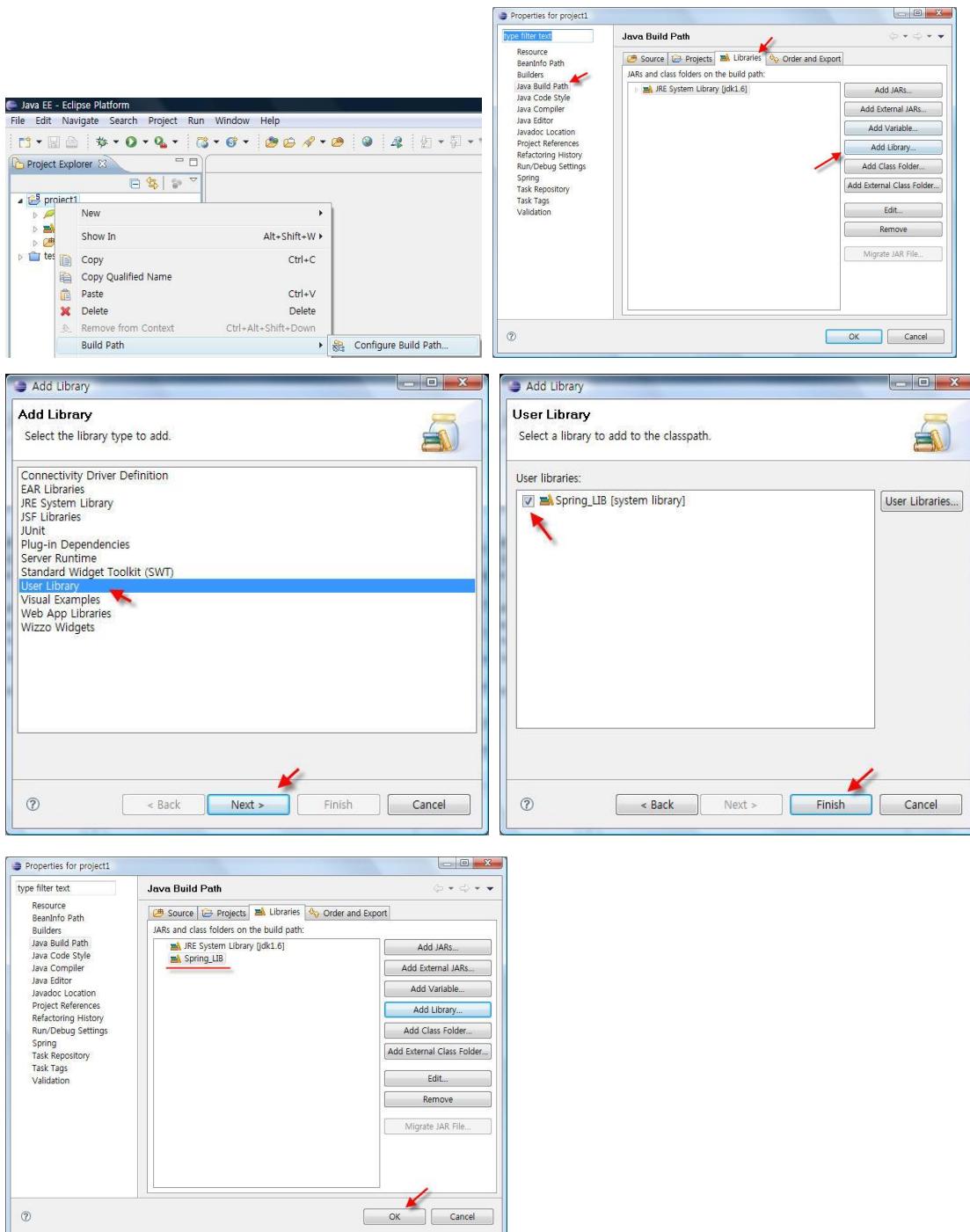
Add JARs 클릭

C:\commons-logging-1.1.2\commons-logging-1.1.2.jar

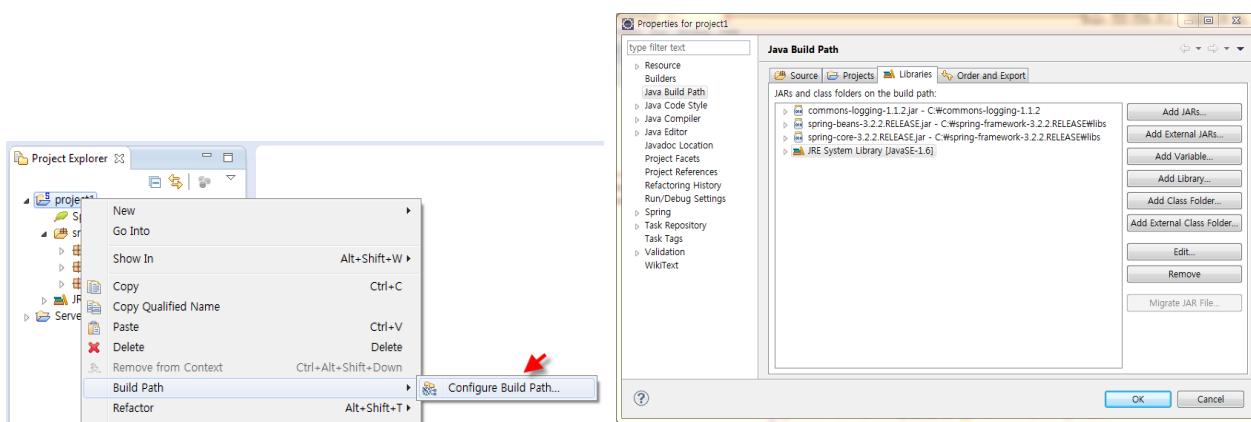
C:\spring-framework-3.2.2.RELEASE\libs\spring-beans-3.2.2.RELEASE.jar

C:\spring-framework-3.2.2.RELEASE\libs\spring-core-3.2.2.RELEASE.jar

프로젝트명 오른쪽 버튼 클릭 -> Build Path -> Configure Build Path..

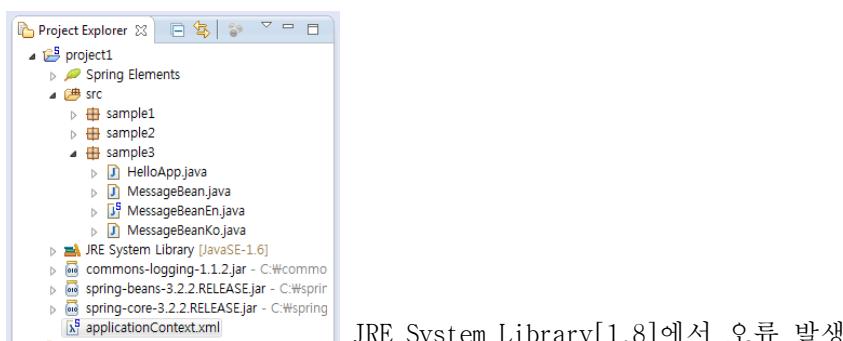
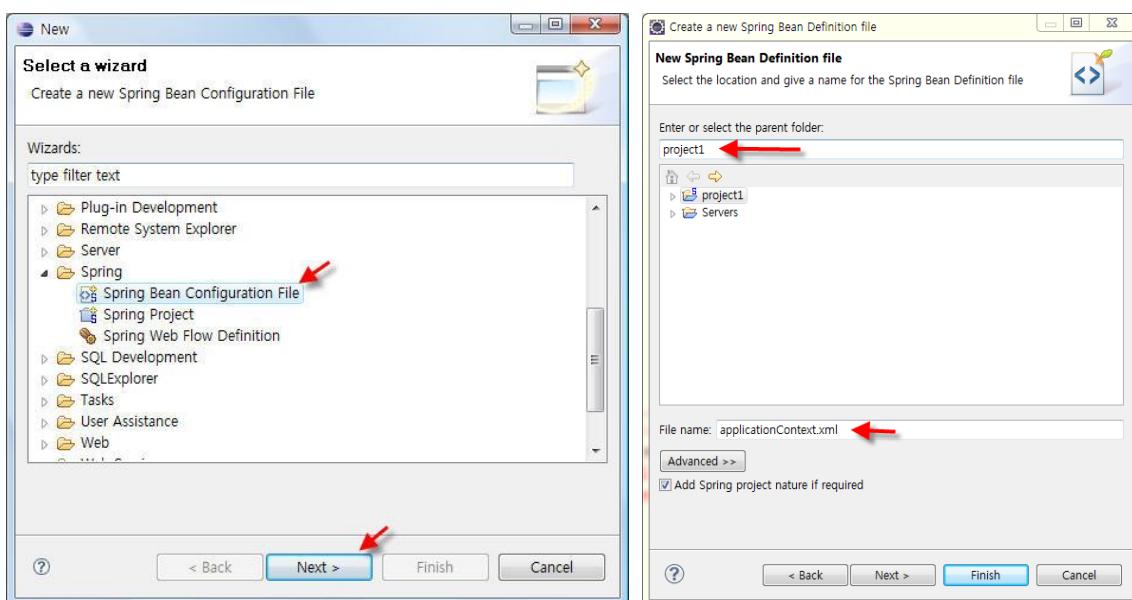


⑤ 스프링을 위한 jar 파일(Libraries 추가방법)



⑥ 스프링을 이용해서 문자열 출력하는 프로그램 작성

- 스프링 설정 파일(applicationContext.xml)을 생성하기 위해 File -> New -> Other..



applicationContext.xml

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

```

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="messageBean" class="sample3.MessageBeanEn" />
</beans>

```

스프링은 기본적으로 인스턴스를 싱글톤으로 생성한다. 즉, 같은 인스턴스를 취득하는 BeanFactory의 getBean를 이용하지 않는 경우 XML 설정 파일에서 <bean>의 singleton 속성을 false로 한다.

```
<bean id="messageBean" class="sample3.MessageBeanEn" singleton="false" />
```

○ XML 설정 파일

스프링에서는 XML 설정 파일을 사용하여 빈을 관리 할 수 있다. XML 설정 파일은 빈 팩토리의 설정 파일을 정의하기 위한 XML 파일로서 XmlBeanFactory 인스턴스를 생성할 때 생성자에 이 파일의 정보를 전달한다.

설정 파일명은 다른 이름을 사용할 수 있다.

루트 요소는 <beans>이며, 그 밑 요소인 <bean>요소를 이용하여 빈에 관한 정의를 기술한다.

- <bean> 태그 요소

속성	의미	디폴트 값
id	Bean에 대해 붙이는 식별자(고유값)	-
name	id에 비교하여 별칭(Alias). 복수 정의가 가능	-
class	Bean의 클래스 명, 완전한 형태의 클래스 이름을 기술한다.	-
parent	Bean 정의를 계승할 때 지정하는 새로운 Bean의 id	-
abstract	Bean 클래스가 추상 클래스인지 여부	false
singleton	Bean이 singleton으로서 관리될지 안될지	false
lazy-init	Bean의 자연 로딩을 행할지 안 할지	false
autowire	구현 클래스를 자동적으로 연결	no
dependency-check	의존 관계 확인 방법	none
depends-on	이 Bean이 의존하는 Bean의 이름, 먼저 초기화된 것이 보증된다.	
init-method	Bean의 초기화 때에 실행시킬 메소드	
destroy-method	Bean 컨테이너의 종료 시에 실행시킬 메소드	

일반적인 자바 코드에서는 인스턴스를 생성하기 위해서 다음과 같이 기술한다.

sample3.MessageBeanEn	messageBean = new sample3.MessageBeanEn();	
패키지명.클래스명	레퍼런스 변수	패키지명.클래스명

인스턴스 생성을 위한 코드를 XML 설정 파일에서 할 경우에는 <bean>요소를 사용한다.

```
<bean id="messageBean" class="sample3.MessageBeanEn"/>
```

레퍼런스 변수 패키지명.클래스명

id 속성에 지정한 값은 레퍼런스 변수에 해당하며, class 속성에는 id 속성에 의해 설정된 레퍼런스 변수의 인스턴스가 되는 클래스명(패키지명.클래스명)을 기술한다.

이렇게 설정 파일에서 빈의 이름을 messageBean으로 정의하고 BeanFactory 인스턴스를 생성했다면 getBean() 메소드의 전달 인자로 문자열 형태인 빈의 이름을 지정하여 호출하면 빈 인스턴스를 취득할 수 있다.

```
MessageBean bean = factory.getBean("messageBean", MessageBean.class);
```

그리고 name 속성을 사용하여 id 속성으로 붙여진 이름의 별칭(Alias)을 정의할 수 도 있다.

이 별칭은 복수 정의가 가능하며, 각각의 별칭을 스페이스 또는 콤마, 세미콜론으로 구분한다.

```
<bean id="messageBean" name="a b c" class="sample3.MessageBeanKo"/>
<bean id="messageBean" name="a,b,c" class="sample3.MessageBeanKo"/>
<bean id="messageBean" name="a;b;c" class="sample3.MessageBeanKo"/>
<bean id="messageBean" name="a;b,c" class="sample3.MessageBeanKo"/>
```

이와 같은 별칭을 정의한 경우, getBean() 메소드의 인수로 지정한 이름에 id 속성 값인 빈으로 이름은 물론, name 속성 값인 별칭을 사용할 수 있다.

```
MessageBean bean = factory.getBean("messageBean", MessageBean.class);
MessageBean bean = factory.getBean("a", MessageBean.class);
MessageBean bean = factory.getBean("b", MessageBean.class);
MessageBean bean = factory.getBean("c", MessageBean.class);
```

getBean() 메소드의 인수로 전달한 문자열(messageBean)은 XML 설정 파일(applicationContext.xml)에서 정의된 빈 이름이다.

```
<bean id="messageBean" class="sample3.MessageBeanKo"/>
```

빈의 이름을 "messageBean"으로 주었기 때문에 빈을 취득한다.

```
MessageBean bean = factory.getBean("messageBean", MessageBean.class);
```

빈을 이용해 메소드를 호출하면 된다.

```
bean.sayHello("Spring");
```

HelloApp.java 클래스 생성

```
package sample3;
```

```

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

@SuppressWarnings("deprecation")
public class HelloApp {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));
        MessageBean bean = factory.getBean("messageBean", MessageBean.class);
        bean.sayHello("Spring");
    }
}

```

○ 스프링의 특성

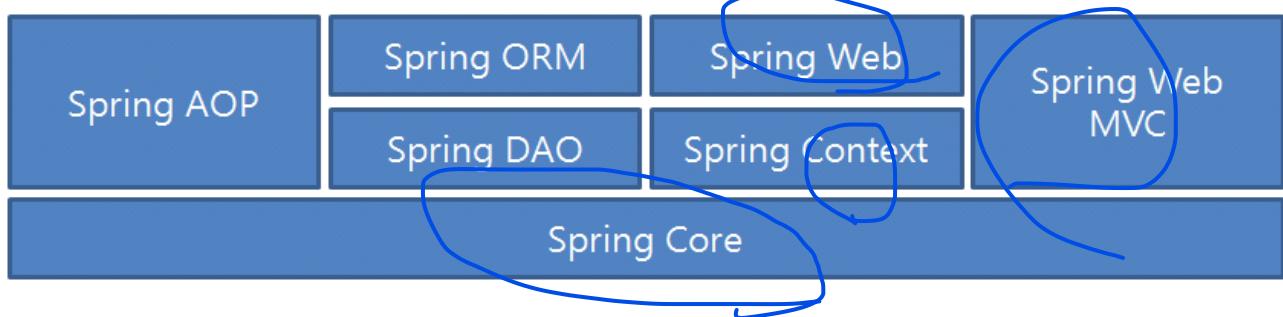
1) 스프링의 장점

『필요한 인스턴스를 스프링에서 미리 생성해 준다.』

클래스 사이의 결합(loosely coupled)을 느슨하게 할 수 있어 클래스 간의 의존 관계가 약해진다.

- 스프링은 '어플리케이션 프레임워크'로 불리며, 웹 어플리케이션은 물론, 위에서 살펴본 예제처럼 콘솔 어플리케이션이나 스윙과 같은 GUI 어플리케이션 등 어떤 어플리케이션에도 적용 가능한 프레임워크이다.
- 스프링은 EJB와 같이 복잡한 순서를 거치지 않아도 간단하게 이용할 수 있기 때문에 '경량(Lightweight) 컨테이너'라고도 부른다.
- 스프링은 Dependency Injection(DI)과 Aspect Oriented Programming(AOP)을 가장 중점적인 기술로 사용되지만, 이외에도 여러 가지 기능을 제공하고 있다.

2) 스프링을 구성하는 모듈



2. 스프링의 DI

프로그래밍에서 의존성(Dependency)이란, 어떤 클래스가 자신의 임무를 다하기 위해 필요한 값(필드 값)이나 사용할 다른 클래스와의 관계를 말한다. 주입(Injection)이란, 어떤 클래스의 인스턴스에 대해 외부로부터 '의존성'을 설정하는 것을 말한다.

DI는 「Dependency Injection」의 (의존주입)약어이며 객체 사이의 의존 관계가 자기 자신이 아닌 외부에 의해서 설정된다는 개념이다. 스프링에서는 설정 파일을 사용하여 손쉽게 객체간의 의존 관계를 설정하기에 이를 DI 컨테이너라 한다. DI 컨테이너는 어떤 클래스가 필요로 하는 값이나 인스턴스를 자동으로 생성, 취득하여 연결시켜주는 역할을 한다. 이렇게 하면 필요한 인스턴스를 생성, 취득하는 코드를 직접 만들지 않아도 된다. 그 결과 클래스간 관계가 느슨한 결합이 되어 의존성이 약해진다. DI 컨테이너가 인스턴스를 생성하도록 하려면 프로그램 소스 내부에서 new로 직접 생성하지 않고 설정 파일에서 필요로 하는 클래스의 정보를 설정해주어야 한다.

스프링은 각 클래스 간의 의존 관계를 관리하기 위한 2가지 방법

- Constructor Injection
- Setter Injection

2개의 인젝션이 어떤 것인지 두 개의 클래스를 예로 의존 관계를 알아본다.

Foo 클래스의 멤버로 Bar 클래스의 레퍼런스 변수가 필드로 존재한다면, Foo 클래스가 Bar 클래스를 참조하기 때문에 Bar 클래스에 의존하고 있다고 할 수 있다.

Foo.java

```
public class Foo {  
    private Bar bar;  
}
```

Foo

----->

Bar

1) Constructor Injection

'Constructor Injection'란 생성자를 통해서 의존 관계 연결시키는 것이다.

Foo.java

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

생성자 Foo()를 사용하여 클래스 Foo와 Bar 사이의 의존 관계를 연결하면 Foo는 언제든지 Bar 인스턴스를 참조 할 수 있게 된다.

Bar 인스턴스가 어느 곳에도 생성 되지 않았다. 이것은 이미 Bar 인스턴스가 존재한다는 가정을 한 것이고 이것이 가능한 것은 스프링의 설정 파일 때문이다.

① XML 설정 파일의 <constructor(컨스트럭터)-arg> 요소

생성자를 통해서 의존 관계를 연결하기 위해서는 스프링의 XML 설정 파일에서 생성된 Bar 인스턴스를 <bean> 요소의 자식 요소인 <constructor-arg> 요소를 연결한다.

<constructor-arg>의 요소의 속성

속성	설명
index	constructor의 몇 번째의 인수에 값을 전달 할 것인지를 지정한다.
type	constructor의 어느 데이터형의 인수에 값을 전달할 것인지를 지정한다.
ref	자식 요소 <ref bean="빈 이름" /> 대신에 사용할 수 있다.
value	자식 요소 <value>값</value> 대신에 사용할 수 있다.

construct injection에서는 <bean> 요소의 자식 요소로서 <constructor-arg> 요소를 사용한다.

객체를 전달할 경우에는 ref 요소를 사용한다.

```
◆ Foo.java
public class Foo{
    private Bar bar;
    public Foo(Bar bar){
        this.bar = bar;
    }
}

◆ applicationContext.xml
<bean id = "foo" class = "Foo" >
    <constructor-arg>
        <ref bean = "bar"/>
    </constructor-arg>
</bean>

<bean id = "bar" class = "Bar" />
```

② 전달 인자가 두 개인 생성자에 연결 관계 설정

```
public Foo(int a, String b) {...}
```

객체를 전달할 경우에는 ref 요소를 사용하여 값을 지정하지만, 기본 데이터 타입일 경우에는 <constructor-arg>요소의 자식요소인 <value>요소를 사용하여 실제로 의존관계를 연결시키기 위한 지정하거나, <constructor-arg>요소의 value 속성값으로 의존관계를 연결할 수 있다.

생성자의 연결 인자가 두 개일 경우에는 index 속성을 사용한다.

```
<bean id="foo" class="Foo">
    <constructor-arg index="0">
        <value>25</value>
    </constructor-arg>
    <constructor-arg index="1" value="Hello">
    </bean>
```

<value>요소를 사용하여 실제로 의존 관계를 연결시키기 위한 값을 지정 한다.

value 속성 값으로 의존 관계를 연결한다.

다음과 같이 type속성을 사용하여 지정하는 것도 가능하다.

```
<bean id="foo" class="Foo">
    <constructor-arg type="int" value="25">
    <constructor-arg type="java.lang.String" value="Hello">
</bean>
```

2) Setter Injection

'Setter Injection'이란 클래스 사이의 의존 관계를 연결시키기 위해서 setter 메소드를 이용하는 방법을 말한다. 아래와 같은 Bar 타입 인수를 가지는 설정 메서드 setBar()를 통해 Bar에 대한 인스턴스 참조를 지정한다.

Foo.java

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

Foo 클래스의 멤버로 존재하는 Bar형 레퍼런스 변수 bar에 대한 setter는 setBar()메소드가 된다. setBar()메소드를 호출할 때 전달해 준 인수를 메소드 내부에서 Foo클래스의 멤버로 존재하는 Bar 형의 레퍼런스 변수에 대입하고 있다. 이로서 Foo는 setBar() 메소드를 통해서 Bar 인스턴스를 참조할 수 있게 된다.

Setter Injection에서는 <property> 요소를 사용한다. <property> 요소에서는 name 속성을 이용하여 값의 의존 관계를 연결시킬 대상이 되는 필드 값을 지정한다.

<property> 요소의 속성

속성	설명
ref	자식 요소 <ref bean="빈 이름" /> 대신에 사용할 수 있다.
value	자식 요소 <value>값</value> 대신에 사용할 수 있다.

```
<bean id="food" class="Foo">  
    <property name="bar" ref="bar"> </property> → a  
    </bean>  
    <bean id="bar" class="Bar" /> → b
```

setter 메소드를 사용하여 값을 전달 받아야 할 경우에는 <property> 요소를 사용해야 한다.

기본 데이터 대신 빈 객체를 전달해야 할 경우에는 <ref> 요소를 사용해야 한다.

a는 Foo 클래스의 setBar 메소드의 전달 인자로 사용하기 위해서 ref 속성값으로 bar를 지정하였다.

b는 bar는 Bar 클래스로 선언한 빈 객체이다.

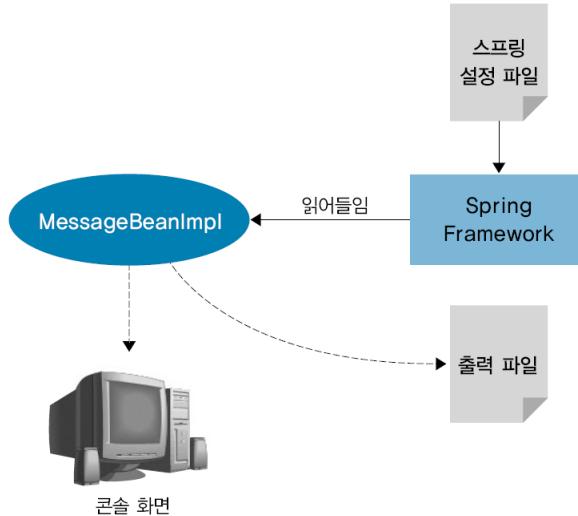
3) DI 패턴 이해를 위한 예제

기존의 프로그램 방식에서는 의존 관계에 있는 객체를 객체가 직접 생성하였다면, 스프링에서는 객체를 직접 생성하지 않고 설정 파일을 대신 사용하여 객체를 생성하는 DI 패턴을 사용한다.

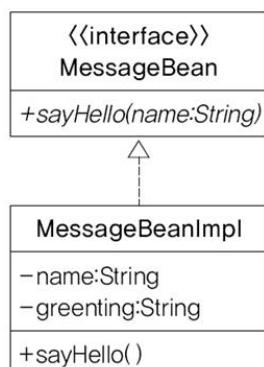
만일 의존 관계에 있는 클래스를 변경해야 할 경우에 기존 프로그램 방식은 클래스가 사용된 곳을 일일이 찾아서 수행해야 한다. 하지만, 스프링의 DI 패턴을 사용한다면 설정 파일에서 클래스 명을 한 번만 수정해 주면 된다.

이전 예제에서 인사말이 영어 혹은 한글 중에서 하나만 출력하는 인터페이스 MessageBean을 구현한 클래스로 MessageBeanEn과 MessageBeanKo 두 개를 사용하였다.

이 예제를 변경하여 '이름'과 '인사말'을 스프링 설정 파일로부터 얻어오도록 하여 다양한 이름과 인사말이 출력되도록 한다. 출력 메시지를 콘솔이 아닌 다른 파일에도 기능이 추가되게 한다.



MessageBean을 구현한 클래스 MessageBeanImpl 하나로 다양한 메시지를 출력하기 위해서 인사말을 저장할 멤버로 greeting 필드를 추가하고, 다양한 이름이 출력되도록 하기 위해서 이름을 저장할 멤버로 name 필드를 추가한다.



▲ 클래스



▲ 파일 구성

<DI 패턴을 이용한 예제 실습>

'project2'란 이름으로 [Spring Project]를 생성하고 스프링 기능을 사용하기 위해 사용자 라이브러리 'Spring_LIB'를 추가한다.

'sample1' 패키지를 생성한 후에 MessageBean 인터페이스를 생성한다.

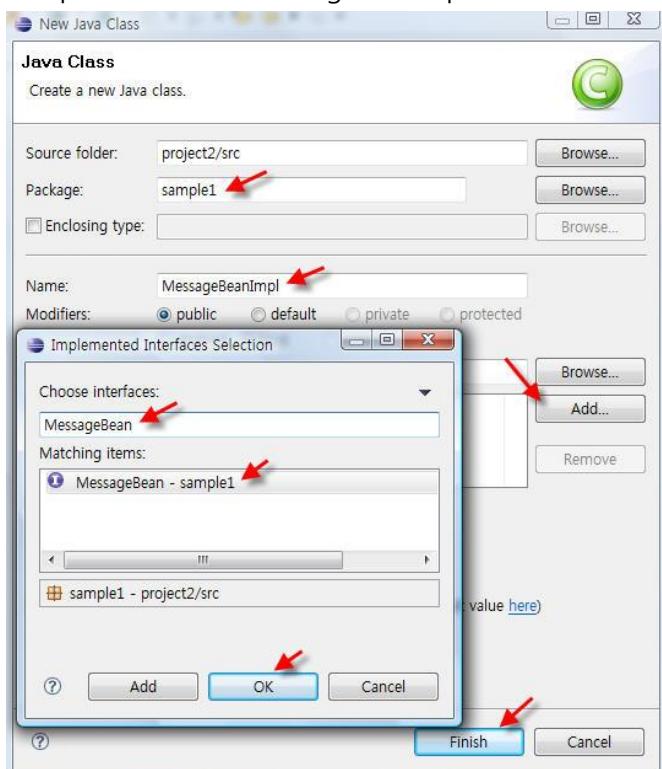
MessageBean.java

```

package sample1;
public interface MessageBean {
    void sayHello();
}

```

'sample1' 패키지에 MessageBeanImpl 클래스를 생성한다.



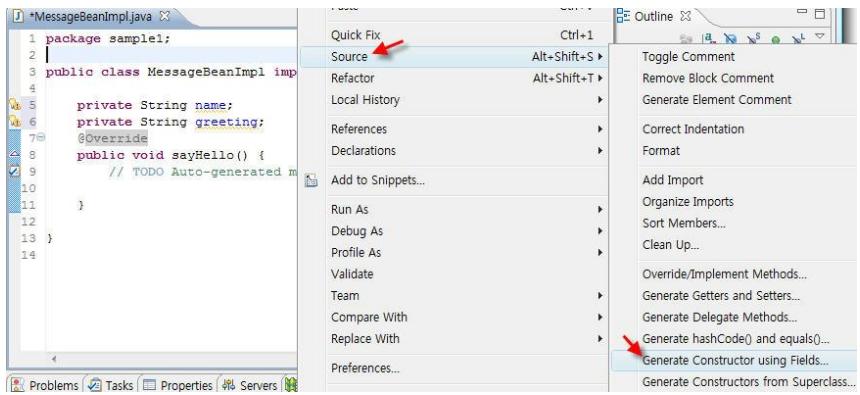
MessageBeanImpl.java

```

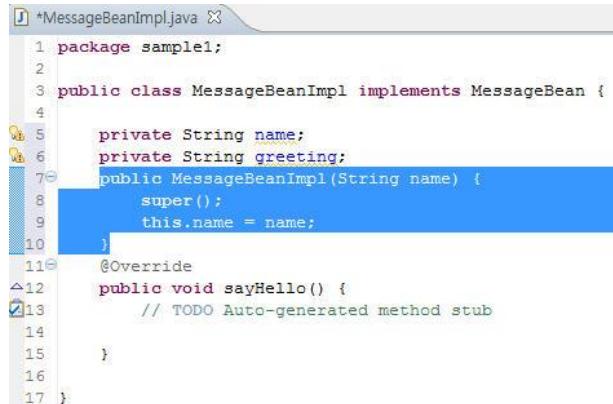
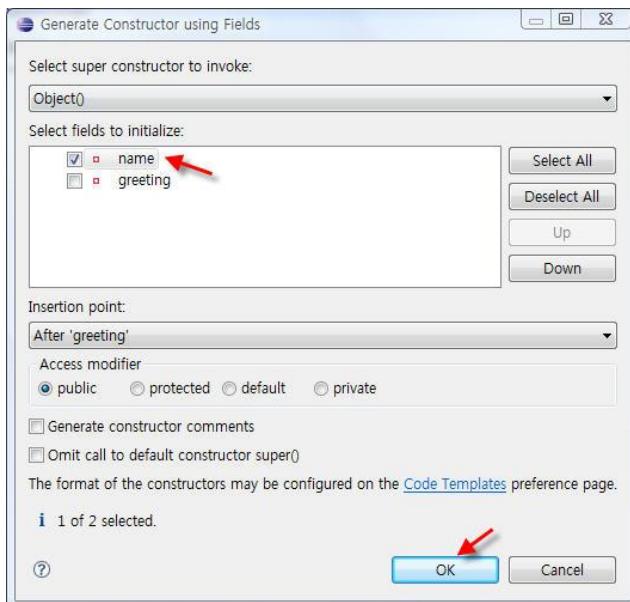
package sample1;
public class MessageBeanImpl implements MessageBean {
    private String name;
    private String greeting;
    @Override
    public void sayHello() {
        // TODO Auto-generated method stub
    }
}

```

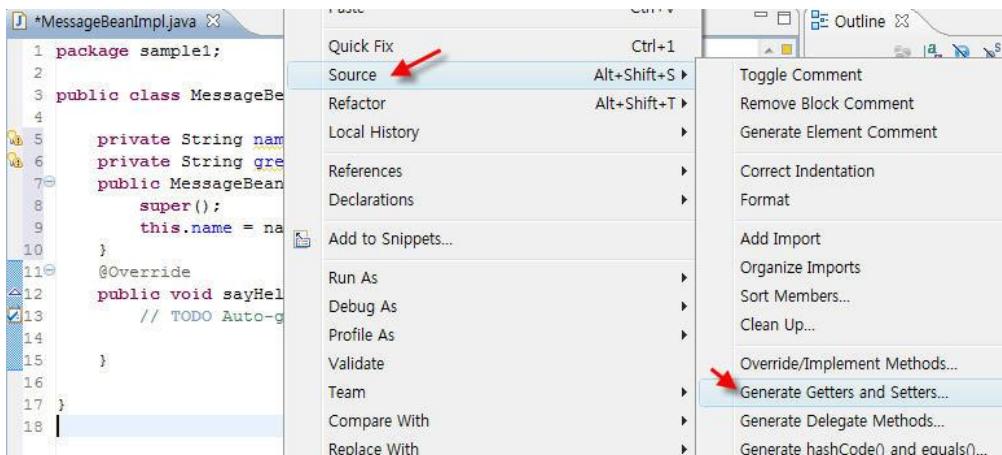
클래스 MessageBeanImpl에 name과 greeting 필드를 추가한 후에 전달 인자를 갖는 생성자를 추가하기 위해서 MessageBeanImpl 클래스를 정의한 내부의 바로 가기 메뉴에서 [Source] -> [Generate Constructor using Fields..]를 선택



name을 전달 인자로 갖는 생성자를 작성하기 위해서 name 필드만 선택한다.



name 필드에 대한 setter를 만들기 위해서 MessageBeanImpl 클래스를 정의한 내부의 바로 가기 메뉴에서 [Source] -> [Generate Getters and Setter..]를 선택



Select getters and setters to create:

- greeting
- name

Insertion point:
After 'name'

Sort by:
Fields in getter/setter pairs

Access modifier:
 public protected default private
 final synchronized

Generate method comments

The format of the getters/setters may be configured on the [Code Templates](#) preference page.

i 2 of 4 selected.

OK Cancel

Code Editor (MessageBeanImpl.java):

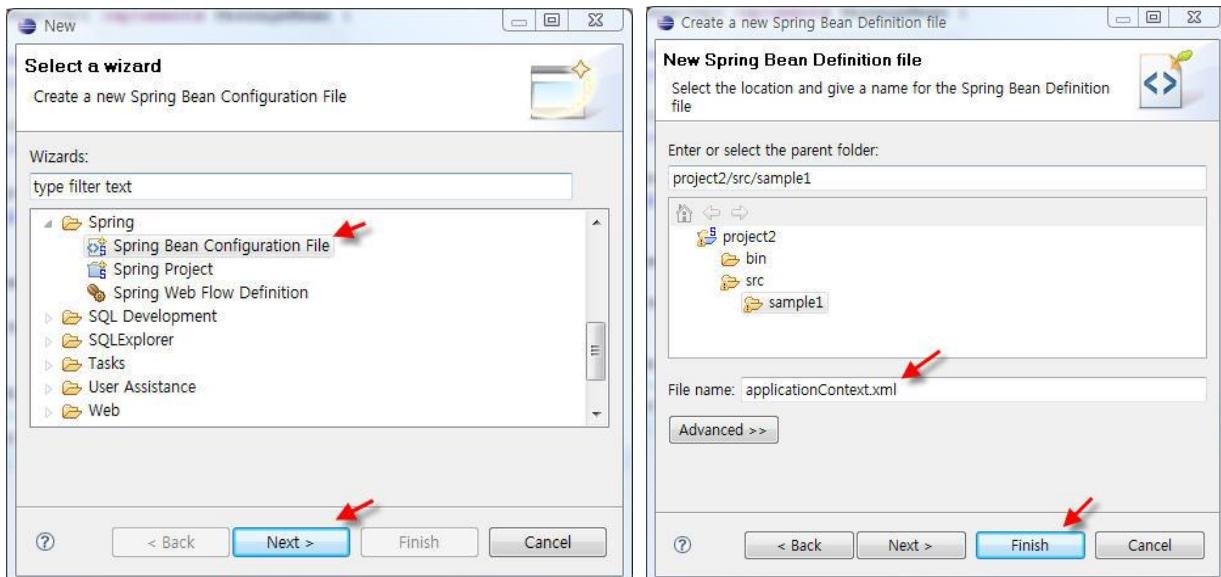
```

1 package sample1;
2
3 public class MessageBeanImpl implements MessageBean {
4
5     private String name;
6     private String greeting;
7
8     public String getGreeting() {
9         return greeting;
10    }
11    public void setGreeting(String greeting) {
12        this.greeting = greeting;
13    }
14
15    public MessageBeanImpl(String name) {
16        super();
17        this.name = name;
18    }
19    @Override
20    public void sayHello() {
21        // TODO Auto-generated method stub
22    }
23
24 }
25
  
```

오버 라이딩된 sayHello()에 소드를 추가한다.

System.out.println(greeting+name+"!");

스프링 설정 파일 applicationContext.xml 을 생성하기 위해서 [New]->[Other..]



applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/spring-beans.xsd">
    <bean id="messageBean" class="sample1.MessageBeanImpl">
        <constructor-arg>—————
        <value>Spring</value>
    
```

MessageBeanImpl 클래스
스로 빈 인스턴스 생성

생성자의 전달 인자에 값을 전달

```

</constructor-arg>
<property name="greeting">
    <value>Hello, </value>
</property>
</bean>
</beans>

```

greeting 프로퍼티에 값을 전달

HelloApp.java 생성 다음과 같이 입력한다.

```

package sample1;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

@SuppressWarnings("deprecation")
public class HelloApp {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));
        MessageBean bean = factory.getBean("messageBean", MessageBean.class);
        bean.sayHello();
    }
}

```

applicationContext.xml에서 이름이 messageBean인 빈 인스턴스를 얻어온다.

HelloApp.java를 실행하고 결과 확인한다.



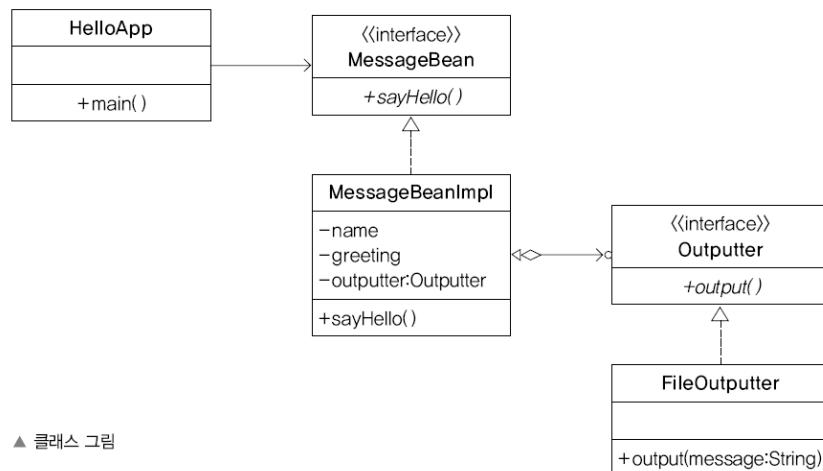
Problems Tasks Properties Servers
<terminated> HelloApp (3) [Java Application] C:\Java\HelloApp
2009. 5. 12 오후 12:34:02 org.springframework
정보: Loading XML bean definitions from
Hello, Spring!

<다른 Bean 참조를 주입하는 예제>

sayHello() 메소드로 파일이나 데이터베이스 및 메일 등 여러 대상으로 메시지를 출력할 수 있도록 하기 위한 인터페이스 베이스를 설계한다.

Outputter라는 인터페이스를 작성하여 output() 메소드를 정의한다. 이 Outputter 인터페이스를 구현하는 FileOutputStreamer 클래스를 작성한다. FileOutputStreamer 클래스의 output() 메소드에서는 메시지를 파일에 출력하는 처리를 설정한다. MessageBeanImpl 클래스는 Outputter형의 오브젝트를 이용한다.

<다양한 장치로 메시지를 출력하는 빈 작성>



'project2_01'란 이름으로 [Spring Project]를 생성하고 스프링 기능을 사용하기 위해 사용자 라이브러리 'Spring_LIB'를 추가한다.

sample1 패키지 생성하고 MessageBean 인터페이스를 추가한 후 sayHello()메소드를 정의 한다.

MessageBean.java

```
package sample1;

public interface MessageBean {
    void sayHello();
}
```

Outputter 인터페이스를 만들고 output() 메소드를 추가한다.

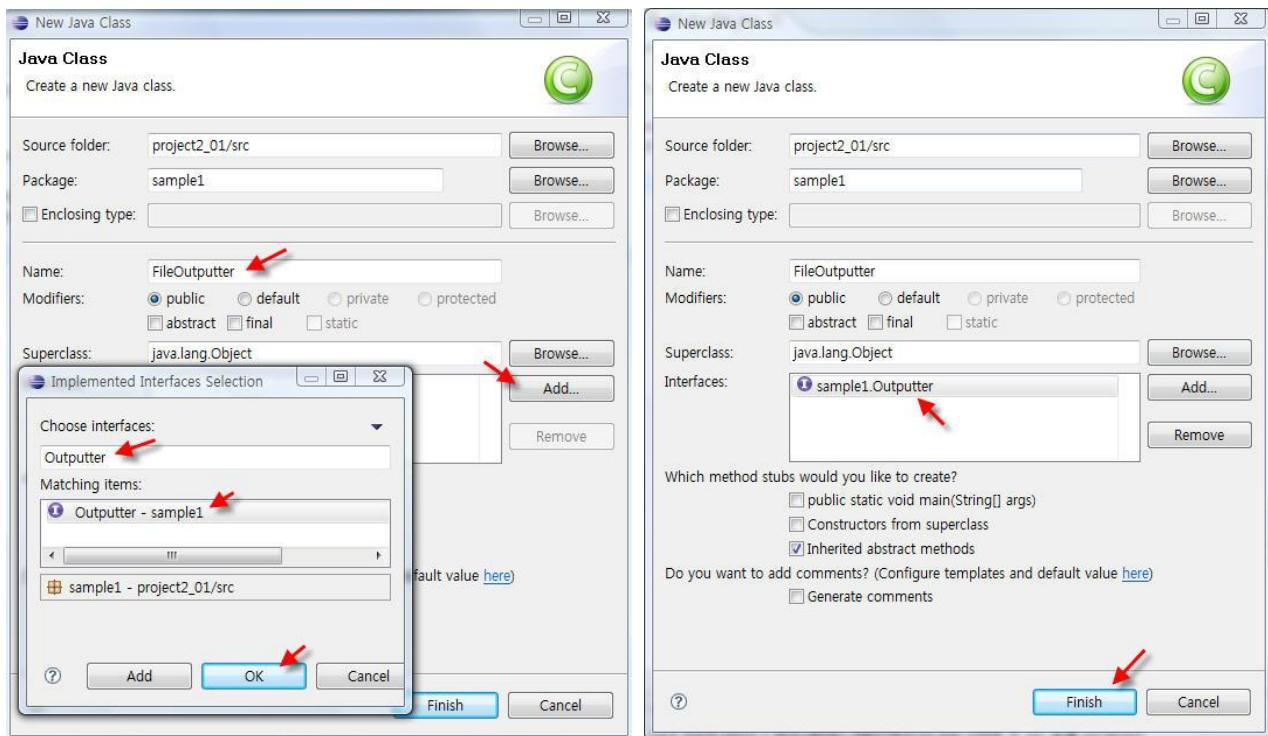
Outputter.java

```
package sample1;

import java.io.IOException;

public interface Outputter {
    void output(String message) throws IOException;
}
```

Outputter 인터페이스를 구현하는 FileOutputStream 클래스 작성



FileOutputter.java

```

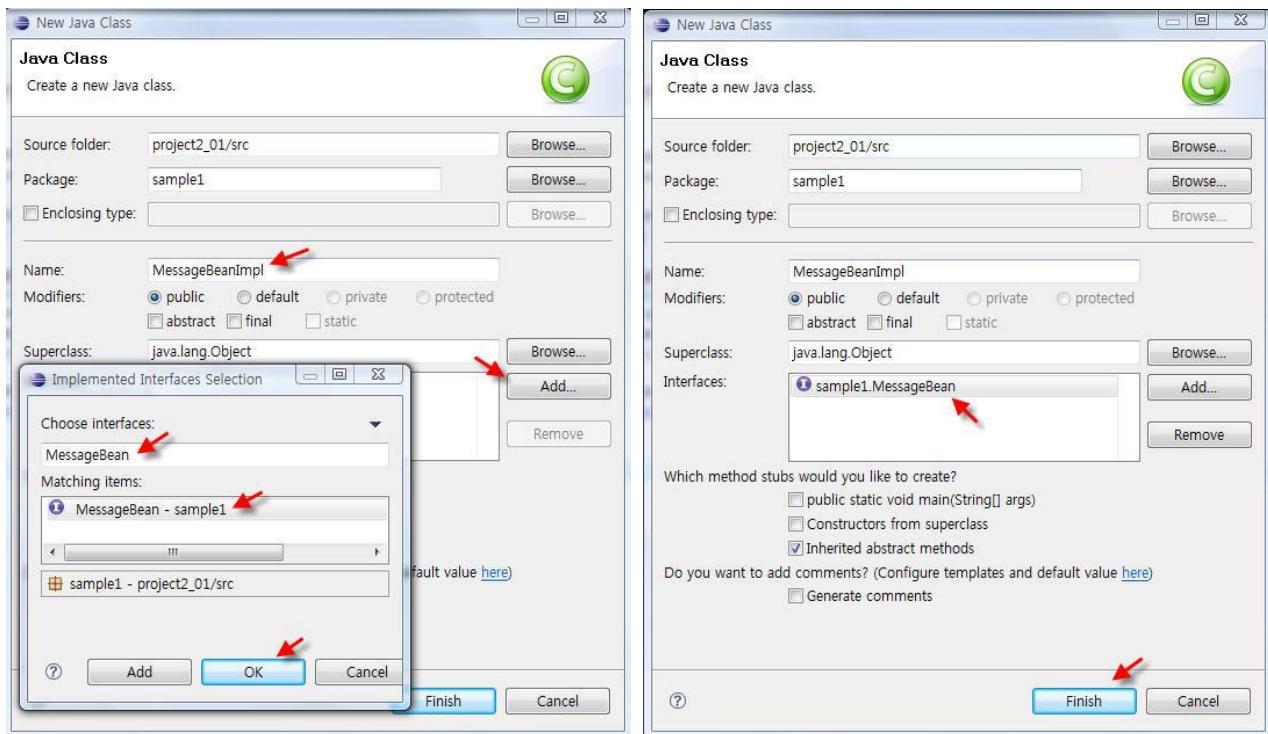
package sample1;
import java.io.FileWriter;
import java.io.IOException;

public class FileOutputter implements Outpuuter {
    private String filePath; // 출력 파일에 대한 경로와 파일 이름을 저장할 필드

    // 메시지를 파일에 출력하기 위한 내용
    public void output(String message) throws IOException {
        FileWriter out = new FileWriter(filePath);
        out.write(message);
        out.close();
    }
    // filePath 필드에 대한 setter 추가
    public void setFilePath(String filePath) {
        this.filePath = filePath;
    }
}

```

MessageBean 인터페이스를 구현한 MessageBeanImpl 클래스를 생성한다.



MessageBeanImpl.java

```

package sample1;
import java.io.IOException;
public class MessageBeanImpl implements MessageBean {
    private String name;
    private String greeting;
    private Outputer outputter;

    @Override
    public void sayHello() {
        String message = greeting + name + "!";
        System.out.println(message);
        try {
            outputter.output(message);
        } catch(IOException e) {
            e.printStackTrace();
        }
    }
    //생성자
    public MessageBeanImpl(String name) {
        this.name = name;
    }
}

```

```

    }
    //setter
    public void setGreeting(String greeting) {
        this.greeting = greeting;
    }
    //setter
    public void setOutputter(Outputter outputter) {
        this.outputter = outputter;
    }
}

```

스프링 설정 파일 applicationContext.xml을 생성한다.

applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="messageBean" class="sample1.MessageBeanImpl" >
        <constructor-arg>
            <value>Spring</value>
        </constructor-arg>

        <property name="greeting">
            <value>Hello, </value>
        </property>

        <property name="outputter">
            <ref local="outputter" />
        </property>
    </bean>

    <bean id="outputter" class="sample1.FileOutputter">
        <property name="filePath">
            <value>out.txt</value>
        </property>
    </bean>
</beans>

```

MessageBeanImpl 클래스의 빈 정의와 FileOutputter 클래스의 빈 정의로 구성된다.

빈 객체의 outputter를 참조해야 할 경우는 <ref> 요소를 사용하여 참조하는 빈의 이름을 지정한다.

어플리케이션을 실행하면 어플리케이션이 'out.txt'라는 파일을 작성하여 메시지를 파일로 출력할 수 있도록 filePath 프로퍼티의 값으로 'out.txt'로 지정한다.

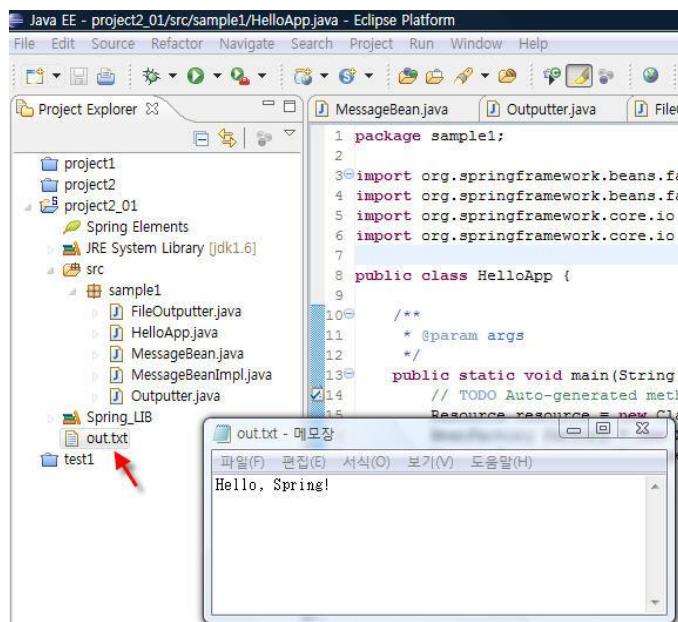
HelloApp.java 클래스 생성한다.

HelloApp.java

```
package sample1;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

@SuppressWarnings("deprecation")
public class HelloApp {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));
        MessageBean bean = factory.getBean("messageBean", MessageBean.class);
        bean.sayHello();
    }
}
```

[결과 확인]



4) Bean 라이프 사이클

의존 주입 컨테이너에서 핵심인 BeanFactory가 하는 역할은 Bean의 생성과 관리이다.

BeanFactory에 의해 관리되는 Bean의 라이프 사이클과 Bean 관리를 위한 설정을 알아본다.

BeanFactory에 의해 Bean이 생성되고, 그 Bean을 사용할 수 있기까지 호출되는 메서드를 확인하는 예제 애플리케이션을 만든다.

참고) null값 처리

<constructor-arg>나 <property> 요소에서 비어있는 <value>요소를 작성하면 String 타입의 필드에는 공문자("")가 설정된다.(숫자 타입 필드일 경우는 예외를 발생한다.)

```
<constructor-arg>
  <value></value>
</constructor-arg>

공문자가 아니라 명시적으로 'null'값을 설정하려면 <value>대신에 <null> 요소를 사용한다.

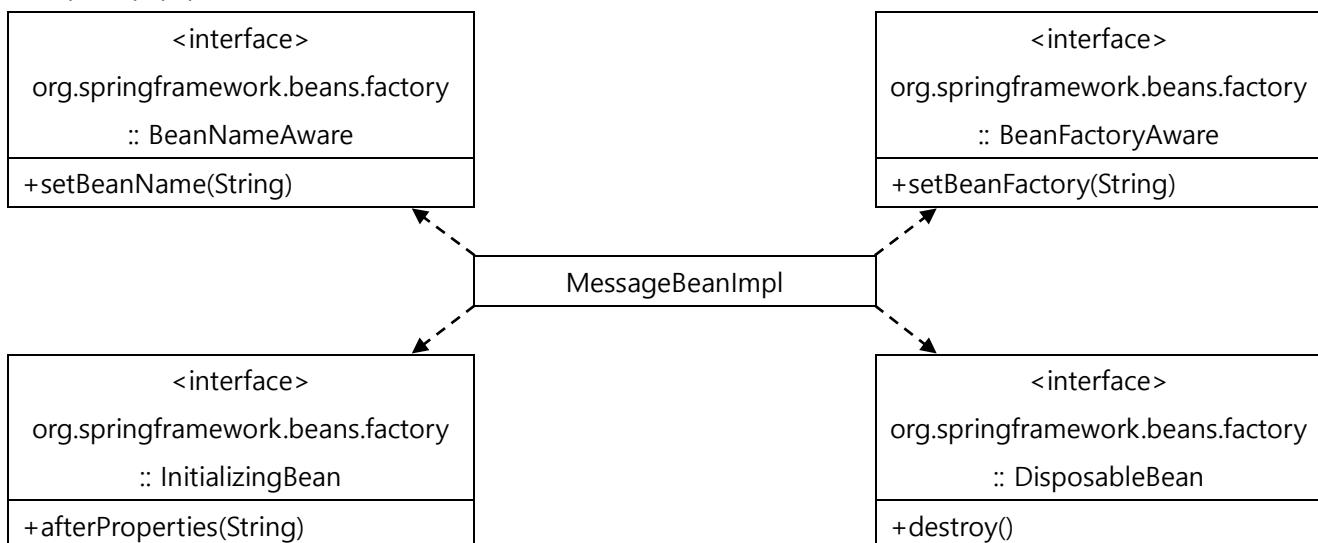
<constructor-arg>
  <null />
</constructor-arg>
```

예제를 실행하면 Bean 생성자나 설정 메서드, 그 밖에 다른 메서드가 스프링에 의해 호출된다. 각 메서드가 호출된 시점에 콘솔에 로그를 출력한다.

이 예제는 Bean의 라이프사이클과 스프링이 호출하는 메서드 호출 시점과 순서를 이해하는 것이 핵심이다. 실행 대상 Bean인 MessageBeanImpl 클래스는 org.springframework.beans.factory 패키지에서 제공하는 4가지 인터페이스를 구현하고 있다.

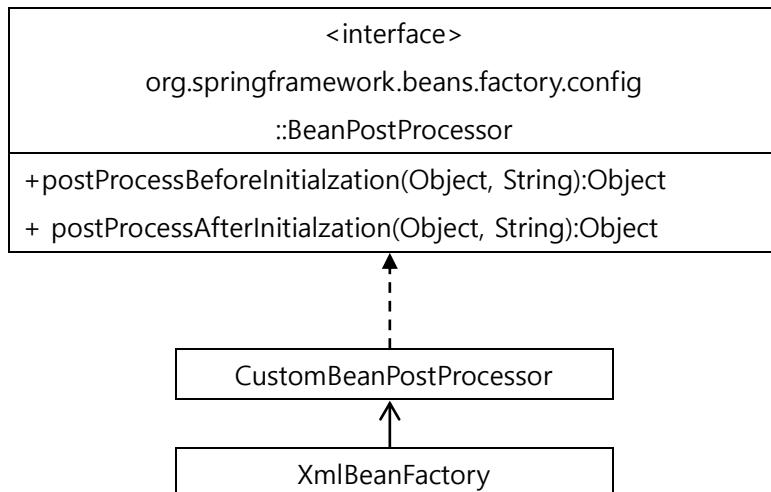
- BeanNameAware
- BeanFactoryAware
- InitializingBean
- DisposableBean

클래스 다이어그램



예제에서는 org.springframework.beans.factory.config 패키지의 BeanPostProcessor 인터페이스를 구현하는 CustomBeanPostProcessor 클래스를 만들어 BeanFactory에 등록한다.

클래스 다이어그램



BeanPostProcessor 인터페이스에 정의된 2가지 메서드 postProcessBeforeInitialization() 메서드와 postProcessAfterInitialization() 메서드는 Bean을 초기화 하기 전과 후에 스프링이 각각의 메서드를 호출하는 구조이다.

applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="messageBean" class="sample1.MessageBeanImpl" init-method="init">
        <property name="greeting"><value>Hello, </value></property>
    </bean>
</beans>
  
```

Bean의 모든 프로퍼티 지정이 끝난 후 호출되는 메서드를 '사용자 구현 초기화 메서드'라고 한다.

스프링의 관리 대상인 Bean에서 이 초기화 메서드를 구현하려면 임의의 메서드를 만들어 <bean> 요소의 init-method 속성에 그 메서드 이름을 지정한다. 여기서는 init-method 속성에 'init'라고 했으므로 MessageBeanImpl 클래스의 init()메서드가 사용자 구현 초기 메서드이다.

마찬가지로 Bean이 소멸될 때 스프링이 호출하는 메서드를 '사용자 구현 소멸 메소드'라고 하며 <bean> 요소의 destroy-method 속성에 지정한다.

HelloApp.java

```

package sample1;
import org.springframework.beans.factory.xml.XmlBeanFactory;
  
```

```

import org.springframework.core.io.FileSystemResource;

@SuppressWarnings("deprecation")
public class HelloApp {
    public static void main(String[] args) {
        XmlBeanFactory factory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));
        factory.addBeanPostProcessor(new CustomBeanPostProcessor()); //①
        MessageBean bean = (MessageBean)factory.getBean("messageBean");
        bean.sayHello();
    }
}

```

①에서 Bean의 초기화 전과 후에 처리를 실행하는 CustomBeanPostProcessor 클래스의 인스턴스를 생성하고, XmlBeanFactory 클래스의 addBeanPostProcessor() 메소드를 사용하고, BeanFactory와 관련을 맺고 있다. 이를 통해 CustomBeanPostProcessor 클래스의 postProcessorBeforeInitialization() 메서드와 postProcessorAfterInitialization() 메서드가 Bean의 초기화 전후에 스프링에 의해 자동으로 호출된다.

MessageBean.java

```

package sample1;
public interface MessageBean {
    void sayHello();
}

```

MessageBeanImpl.java

```

package sample1;
import org.springframework.beans.factory.*;

public class MessageBeanImpl implements MessageBean, BeanNameAware, BeanFactoryAware,
InitializingBean, DisposableBean {

    private String greeting;
    private String beanName;
    private BeanFactory beanFactory;

    public MessageBeanImpl() {
        System.out.println("① Bean의 생성자 실행");
    }

    public void setGreeting(String greeting) {

```

```

        this.greeting = greeting;
        System.out.println("② 세터 메서드 실행");
    }

    public void setBeanName(String beanName) {
        System.out.println("③ Bean명 지정");
        this.beanName = beanName;
        System.out.println(" -> " + beanName);
    }

    public void setBeanFactory(BeanFactory beanFactory) {
        System.out.println("④ BeanFactory 지정");
        this.beanFactory = beanFactory;
        System.out.println(" -> " + beanFactory.getClass());
    }

    public void init() {
        System.out.println("⑦ 초기화 메서드 실행");
    }

    public void destroy() {
        System.out.println("종료");
    }

    public void afterPropertiesSet() {
        System.out.println("⑥ 프로퍼티 지정 완료");
    }

    public void sayHello() {
        System.out.println(greeting + beanName + "!");
    }
}

```

CustomBeanPostProcessor.java

```

package sample1;
import org.springframework.beans.factory.config.*;

public class CustomBeanPostProcessor implements BeanPostProcessor {
    public Object postProcessBeforeInitialization(Object bean, String beanName) {

```

```

        System.out.println("⑤ 초기화 전 Bean에 대한 처리 실행");
        return bean;
    }

    public Object postProcessAfterInitialization(Object bean, String beanName) {
        System.out.println("⑧ 초기화 후 Bean에 대한 처리 실행");
        return bean;
    }
}

```

BeanFactory(XmlBeanFactory 인스턴스)가 Bean(MessageBeanImpl 인스턴스)을 생성하고, Bean이 사용 가능한 상태가 될 때까지 흐름은 아래와 같다.

- ① Bean의 인스턴스화(생성자 호출)
- ② 필드값 지정
- ③ setBeanName() 메서드 호출(BeanNameAware 인터페이스를 구현하고 있는 경우)
- ④ setBeanFactory() 메서드 호출(BeanFactoryAware 인터페이스를 구현하고 있는 경우)
- ⑤ BeanPostProcessor의 postProcessBeforeInitialization() 메서드 호출(BeanFactory에 BeanPostProcessor 클래스가 관련되어 있는 경우)
- ⑥ postProcessAfterInitialization() 메서드 호출(InitializingBean 인터페이스를 구현하고 있는 경우)
- ⑦ 사용자 구현 초기화 메서드 호출(사용자 구현 초기화 메서드를 정의하고 있는 경우)
- ⑧ BeanPostProcessor의 postProcessAfterInitialization() 메서드 호출(BeanFactory에 BeanPostProcessor 클래스가 관련되어 있는 경우)

컨테이너가 종료될 때는 다음과 같은 순서로 메서드가 호출된다.

- ① destroy() 메서드 호출(DisposableBean 인터페이스를 구현하고 있는 경우)
- ② 사용자 구현 소멸 메소드 호출(사용자 구현 소멸 메소드를 정의하고 있는 경우)

결과

The screenshot shows the Eclipse IDE's Console view with the following output:

```

Markers Properties Servers Data Source Explorer Snippets Console
<terminated> HelloApp (3) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (2013. 5. 10. 오후 5:14:30)
2013. 5. 10 오후 5:14:30 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
정보: Loading XML bean definitions from file [C:\spring3_exam\project2_02\applicationContext.xml]
① Bean의 생성자 실행
② 세터 메서드 실행
③ Bean명 지정
-> messageBean
④ BeanFactory 설정
-> class org.springframework.beans.factory.xml.XmlBeanFactory
⑤ 초기화 전 Bean에 대한 처리 실행
⑥ 프로퍼티 지정 완료
⑦ 초기화 메서드 실행
⑧ 초기화 후 Bean에 대한 처리 실행
Hello, messageBean!

```

참고) Autowiring

BeanFactory에는 '오토와이어링'이라는 기능이 있다. 어떤 Bean이 참조하는 형태와 실제 사용할 클래스

를 자동으로 연결해 준다. 예를 들어, 다음과 같이 클래스 A가 클래스 B에 의존하는 관계가 있다고 한다.

A -----> B

A.java

```
public class A {  
    B b;  
    public void setB(B b){  
        this.b = b;  
    }  
}
```

applicationContext.xml 에 다음과 같이 설정한다.

```
<bean id="a" class="A">  
    <property name="b"> <ref bean="B" /> </property>  
</bean>
```

하지만 Autowiring 기능을 사용하면 위와 같은 설정을 기술할 필요가 없다. 다음과 같이 기술하면 스프링이 자동으로 B클래스로의 의존성을 해결해 준다.

```
<bean id="a" class="A" autowire="byname" />
```

Autowiring 설정 방법은 2가지이다. 하나는 <beans> 요소의 defualt-autowire 속성을 이용하는 방법이다. 다른 하나는 <bean> 요소의 autowire 속성을 사용하는 것이다.

<beans> 요소의 defualt-autowire 속성을 사용하면, 그 자식 요소인 모든 <bean>요소에 설정이 적용된다. 각각의 <bean>마다 Autowiring을 설정하고 싶다면, 개별적으로 <bean>요소의 autowire 속성을 사용하면 된다.

defualt-autowire 속성과 autowire 속성에 지정 가능한 값은 다음과 같다.

defualt-autowire 속성과 autowire 속성에 지정 가능한 값

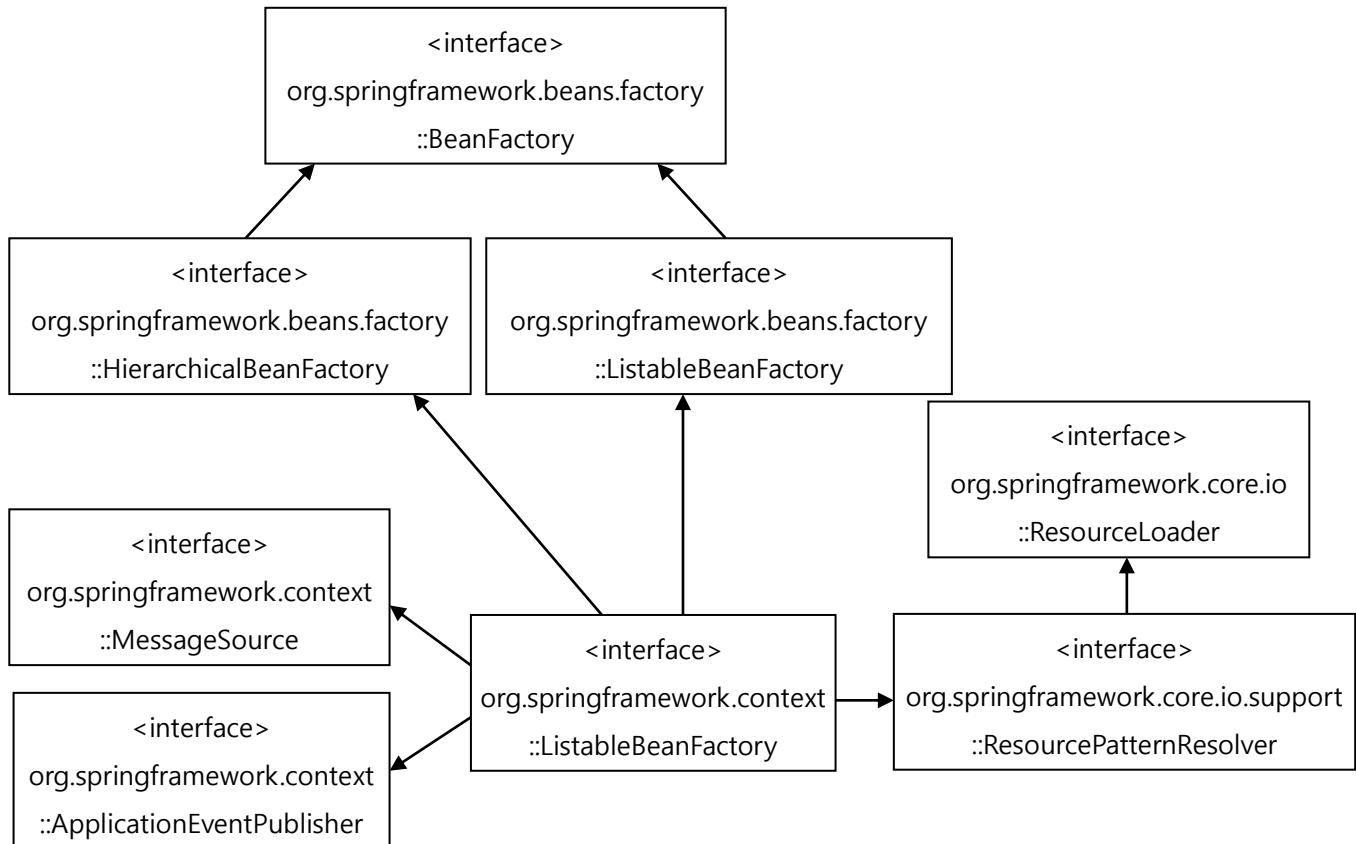
값	의미
No	Autowiring을 하지 않는다. 기본 값
byName	Bean 필드 이름과 같은 이름의 id를 갖는 Bean을 연결해준다.
byType	Bean 필드에 대한 setter 메서드의 인수 타입을 사용해 연결해준다.
constructor	생성자에 대해 byType을 사용해 연결해준다.
autodetect	constructor인지 byType인지 자동으로 판단한다. 기본 생성자가 정의되어 있지 않은 경우는 constructor를, 정의되어 있는 경우에는 byType을 사용해 연결해준다.

5) ApplicationContext 인터페이스

BeanFactory 인터페이스의 하위 인터페이스인 org.springframework.context.ApplicationContext 인터페이스는 BeanFactory에 몇 가지 편리한 기능을 추가한다.

spring-context-3.2.2.RELEASE.jar 와 spring-expression-3.2.2.RELEASE.jar 파일의 추가가 필요하다.

클래스 다이어그램



구체적으로 다음과 같은 기능을 BeanFactory에 추가한다.

- 메시지 국제화
- 자원 접근 수단 간소화
- 이벤트 처리
- 복수 컨텍스트 적재

HelloApp 클래스의 코드를 수정해서 ApplicationContext를 사용한다. BeanFactory를 XmlBeanFactory 인터페이스에서 ApplicationContext 인터페이스로 변경한다. ApplicationContext 인터페이스의 구현 클래스는 `org.springframework.context.support.FileSystemXmlApplicationContext`를 사용하는데, 이 클래스의 생성자는 `ApplicationContext.xml`을 직접 인수로 지정할 수 있다. 변경할 코드는 다음과 같다.

```
XmlBeanFactory factory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));
```

위 코드를 아래와 같이 변경한다.

```
ApplicationContext factory = new FileSystemXmlApplicationContext("applicationContext.xml");
```

ApplicationContext는 BeanPostProcessor를 자동으로 읽어들인다. 이 때문에 CustomBeanPostProcessor 클래스의 인스턴스를 생성하고 addBeanProcessor() 메소드를 호출하는 코드를 제거한다.

대신에 CustomBeanPostProcessor 클래스의 Bean 정의를 applicationContext.xml에 기술한다.

우선 아래의 코드를 제거한다.

```
factory.addBeanProcessor(new CustomBeanPostProcessor());
```

대신 applicationContext.xml에 아래와 같이 CustomBeanPostProcessor의 Bean 정의를 추가한다.

```
<bean class = "sample1.CustomBeanPostProcessor" />
```

XmlBeanFactory에서 ApplicationContext로 변경한 HelloApp1 클래스 코드와 applicationContext1.xml은 다음과 같다.(HelloApp1.java와 applicationContext1.xml 추가)

HelloApp1.java

```
package sample1;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class HelloApp1 {
    public static void main(String[] args) {
        ApplicationContext factory = new FileSystemXmlApplicationContext("applicationContext1.xml");
        MessageBean bean = (MessageBean)factory.getBean("messageBean");
        bean.sayHello();
    }
}
```

applicationContext1.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"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="messageBean" class="sample1.MessageBeanImpl" init-method="init">
        <property name="greeting"><value>Hello, </value></property>
    </bean>

    <bean class="sample1.CustomBeanPostProcessor" />

```

```
</beans>
```

실행 후 결과는 같다.

참고) 어노테이션에 의한 Autowiring

스프링2.0 부터 어노테이션에 의한 설정을 도입하였다. 지금까지 설정 파일에 기술했던 것을 어노테이션을 사용해서 설정할 수 있다. 스프링 3.0에서는 다양한 어노테이션을 제공한다.

'Autowiring'에서 소개한 기능을 어노테이션을 이용해서 설정할 수 있다.

Autowiring을 설정하는 어노테이션은 '@Autowired' 이다. @Autowired 어노테이션을 설정 메소드에 기술한다. project2_01 프로젝트의 MessageBeanImpl 클래스의 설정 메소드 setOutputter()에 어노테이션을 붙인다.

```
@Autowired  
public void setOutputter(Outputter outputter) {  
    this.outputter = outputter;  
}
```

@Autowired 어노테이션을 사용하면 Autowiring은 'byType', 즉 Bean의 타입을 사용해서 연결하는 방법으로 실행된다. 그리고 설정 파일 applicationContext.xml에 <context:annotation-config> 요소를 추가한다.

```
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xmlns:context="http://www.springframework.org/schema/context"  
       xsi:schemaLocation="http://www.springframework.org/schema/beans  
                           http://www.springframework.org/schema/beans/spring-beans.xsd  
                           http://www.springframework.org/schema/context  
                           http://www.springframework.org/schema/context/spring-context-3.0.xsd">  
  
    <context:annotation-config />  
  
    <bean id="messageBean" class="sample1.MessageBeanImpl" >  
        <constructor-arg>  
            <value>Spring</value>  
        </constructor-arg>  
  
        <property name="greeting">  
            <value>Hello, </value>  
        </property>  
  
        <property name="outputter">  
            <ref local="outputter" />  
        </property>  
    </bean>  
</beans>
```

```

        </property>
    </bean>

    <bean id="outputter" class="sample1.FileOutputter">
        <property name="filePath">
            <value>out.txt</value>
        </property>
    </bean>
</beans>

```

<context:annotation-config> 요소를 사용하려면 context 스키마가 필요하기 때문에 <beans> 요소를 변경한다.

MessageBeanImpl 클래스의 outputter 프로퍼티는 @Autowired 어노테이션에 의해 자동으로 스프링의 의존 관계를 설정하기 때문에 설정 파일에서 제거한다. @Autowired 어노테이션을 적용하려면 BeanFactory가 아니라 ApplicationContext를 사용해야 한다.

```
XmlBeanFactory factory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));
```

위 코드를 다음과 같이 변경해야 한다.

```
ApplicationContext factory = new FileSystemXmlApplicationContext("applicationContext.xml");
```

실행하려면 다음의 라이브러리를 추가해야 한다.

spring-context-3.2.2.RELEASE.jar

spring-expression-3.2.2.RELEASE.jar

실행 결과는 같다.

@Autowired 어노테이션에는 required 한 가지 속성 밖에 없다.

속성	설명
required	의존 관계 설정이 필수 인지 여부를 true/false로 지정한다. 기본값은 true이다.

기본 required 속성은 true이기 때문에 @Autowired 어노테이션을 붙이면 의존 관계 설정이 필수가 됩니다. 해당하는 Bean이 없다면, 스프링 애플리케이션이 실행될 때 예외를 발생시킨다.

'injection of autowired dependencies failed' 에러가 출력된다.

설정 메서드 뿐만 아니라 필드에도 적용할 수 있다.

@Autowired

```
private Outputter outputter;
```

이렇게 하면 설정 메서드를 기술할 필요가 없다.

bean 태그의 기본적인 사용법

```
<beans>
    <!-- 클래스 Y에 클래스 X를 Autowired로 인젝션한다. -->
    <bean id="오브젝트명X" class="패키지명.클래스명X" />
    <bean id="오브젝트명Y" class="패키지명.클래스명Y" autowire="byType" />

    <!-- 클래스 B에 클래스 A를 명시적으로 인젝션한다. -->
    <bean id="오브젝트명A" class="패키지명.클래스명A" />
    <bean id="오브젝트명B" class="패키지명.클래스명B">
        <property name="인스턴스명" ref="오브젝트명A" />
    </bean>

    <!-- 클래스 B에 클래스 A를 명시적으로 인젝션한다. -->
    <bean id="오브젝트명C" class="패키지명.클래스명C">
        <property name="인스턴스명" value="문자열1" />
        <property name="인스턴스명" value="문자열2" />
    </bean>
</beans>
```

참고) 의존 관계 설정 확인

어노테이션에 의한 Autowiring에서 @Autowired 어노테이션 이외에, 설정 파일에서도 의존 관계를 설정하고 있는지 여부를 애플리케이션을 실행할 때 확인할 수 있다. 설정을 확인하려면 <bean> 요소의 dependency-check 속성을 사용한다.

dependency-check 속성에 지정 가능한 값

값	의미
none	확인하지 않는다.(기본값)
simple	기본 자료형과 문자열, 컬렉션 타입 필드 설정을 확인해 준다.
object	참조 타입 필드 설정을 확인한다.
all	모든 필드의 설정을 확인한다.

Bean의 필드에 지정하는 값이나 클래스 참조 설정을 applicationContext.xml에 기술하지 않아도, 예외는 발생하지 않는다. 이런 경우 private 타입이라면 제로가, 참조 타입이라면 null이 설정된다.
dependency-check 속성을 이용해서, 설정에서 부족한 부분을 빨리 찾을 수 있다.

3. 스프링의 AOP

AOP란, 기능을 핵심 비즈니스 로직과 공통 모듈로 구분하고, 핵심 로직에 영향을 미치지 않고 사이사이에 공통 모듈을 효과적으로 잘 끼워넣도록 하는 개발 방법이다. 공통 모듈은 보안 인증, 로깅 같은 요소들이 해당된다

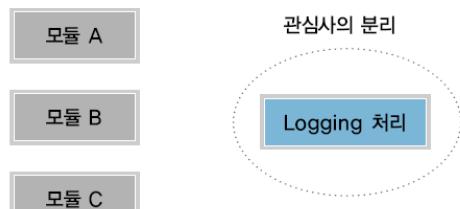
관점 지향 프로그래밍(Aspect Oriented Programming, AOP)은 객체지향 프로그래밍의 뒤를 잇는 또 하나의 프로그래밍 언어 구조라 할 수 있다. AOP와 관련된 가장 중요한 개념은 'AOP는 결코 OOP의 자리를 대신하기 위한 기술이 아니다.' 라는 것이고 오히려 AOP는 OOP를 더욱 OOP답게 만들어 준다.

1) 에스펙트 지향과 횡단 관점 분리

Aspect 지향에서 중요한 개념은 「횡단 관점의 분리(Separation of Cross-Cutting Concern)」이다. 이에 대한 이해를 쉽게 하기 위해서 은행 업무를 처리하는 시스템을 예를 들어 보겠다.

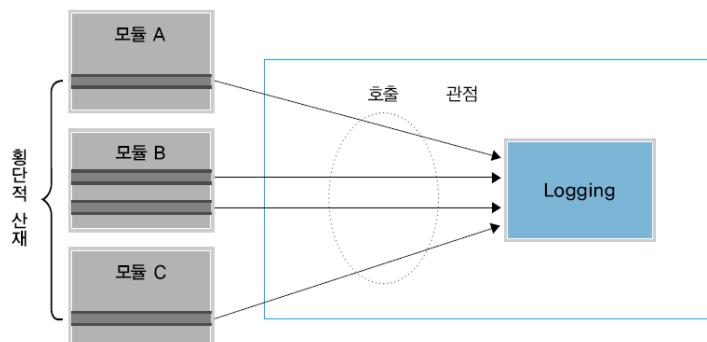
은행 업무 중에서 계좌이체, 이자계산, 대출처리 등은 주된 업무(핵심 관점, 핵심 비즈니스 기능)로 볼 수 있다. 이러한 업무(핵심 관점)들을 처리하는데 있어서 「로깅」, 「보안」, 「트랜잭션」등의 처리는 어플리케이션 전반에 걸쳐 필요한 기능으로 핵심 비즈니스 기능과는 구분하기 위해서 공통 관심 사항(Cross-Cutting Concern)이라고 표현한다.

오브젝트 지향에서는 이들 업무들을 하나의 클래스라는 단위로 모으고 그들을 모듈로부터 분리함으로써 재이용성과 보수성을 높이고 있다.



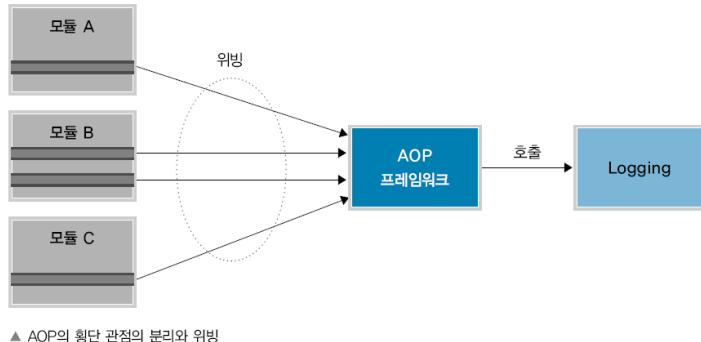
▲ 기존의 객체 지향에서 관점의 분리

오브젝트 지향에서는 로깅이라는 기능 및 관련하는 데이터 자체는 각 모듈로부터 분리하는 것으로 성공했지만 그 기능을 사용하기 위해서 코드까지는 각 모듈로부터 분리할 수 없다. 그렇기 때문에 분리한 기능을 이용하기 위해서 코드가 각 모듈에 횡단으로 산재하게 된다.



▲ 횡단적으로 산재하는 '기능의 호출'

AOP에서는 분리한 기능의 호출도 포함하여 「관점」으로 다룬다. 그리고 이러한 각 모듈로 산재한 관점을 「횡단 관점」라 부르고 있다. AOP에서는 이러한「횡단 관점」까지 분리함으로써 각 모듈로부터 관점에 관한 코드를 완전히 제거하는 것을 목표로 한다.



▲ AOP의 횡단 관점의 분리와 위빙

AOP에서는 분리된 관점이 AOP 프레임워크에 의해 관리되고, 필요한 순간에 각 모듈로 삽입된다. 분리된 관점을 여러 차례 모듈에 삽입하는 것을 AOP에서는 위빙(Weaving:엮기)이라 한다.

Advice를 핵심 로직 코드에 삽입하는 것을 Weaving(위빙)이라 부른다.

AOP에서는 각 모듈에 분리된 기능을 이용하기 위한 코드를 기술할 필요가 없다. 즉, AOP에서는 각 모듈의 개발자들이 횡단 관점에 대해 전혀 알 필요가 없도록 하는데 목적이 있다.

이와 같이 분리된 관점은 객체 지향보다 독립성이 훨씬 높으며, 재활용을 하거나 유지 보수를 하기에 편리하다. 또한 기존 소프트웨어의 코드에 손을 대지 않고도 새로운 기능을 추가하는 것도 가능해졌다.

스프링 AOP에서의 용어

스프링을 사용한 AOP에서는 다음과 같은 용어를 이해해야한다.

- 애스펙트(Aspect) : 횡단관심사의 동작과 그 횡단관심사를 적용하는 소스코드 상의 포인트를 모은 것.
하나 또는 그이상의 어드바이스(동작)와 포인트컷(동작을 적용하는 조건)을 조합한 것이다.(Advisor)
- 어드바이스(Advice)
- 조인포인트(Joinpoint)
- 포인트컷(Pointcut)

- Advice

관점으로 분리되고 실행시 모듈에 위빙된 구체적인 처리를 AOP에서는 Advice라고 한다. Advice를 어디에서 위빙하는지는 뒤에 나오는 Pointcut이라는 단위로 정의한다. 또한, Advice가 위빙되는 인스턴스를 '대상객체'라고 한다. Joinpoint에서 실행되는 코드이다. 로그출력이나 트랜잭션 관리 등의 코드

스프링의 Advice 타입으로 아래의 5개를 제공한다.

Advice 타입	설명
Around Advice	Joinpoint 앞과 뒤에서 실행되는 Advice
Before Advice	Joinpoint 앞에서 실행되는 Advice
AfterReturning Advice	Joinpoint 메서드 호출이 정상적으로 종료된 뒤에 실행되는 Advice
AfterThrowing Advice	예외가 던져질 때 실행되는 Advice

- Joinpoint

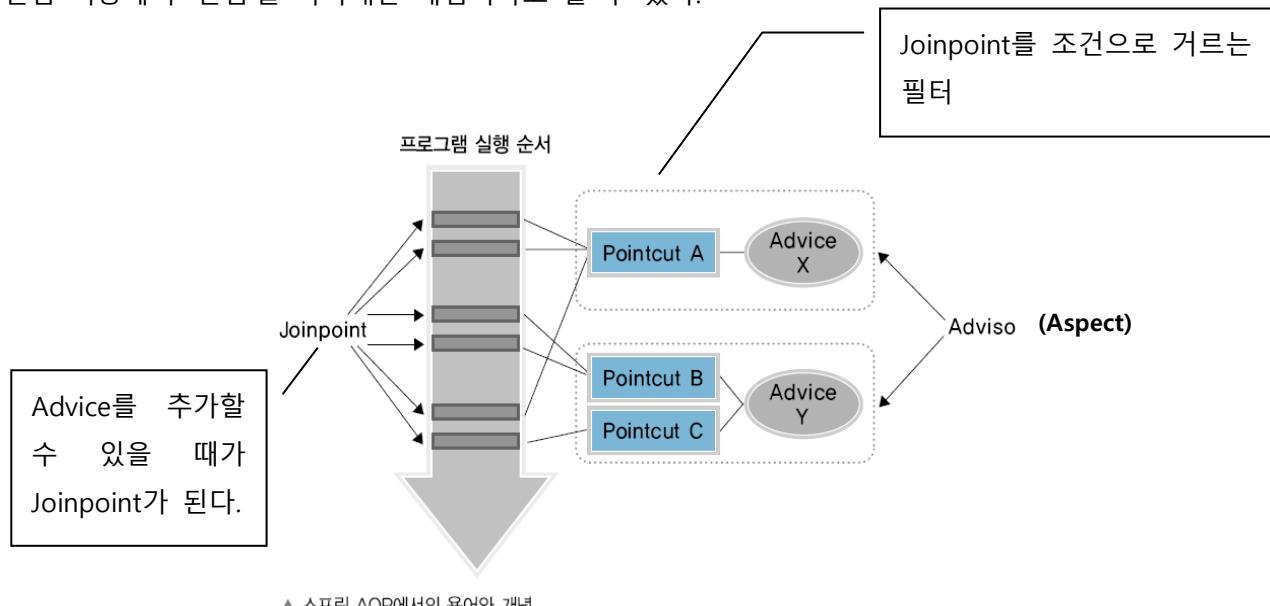
실행시 처리 플로우에서 Advice를 Weaving하는 포인트를 Joinpoint라고 한다. 구체적으로는 '메서드 호출'이나 '예외 발생'이라는 포인트를 Joinpoint로 정의한다. 개발자가 만들어 넣을 수 없는 AOP의 사양.

- Pointcut

하나 또는 복수의 Joinpoint를 하나로 묶는 것을 Pointcut이라고 한다. Advice의 Weaving 정의는 Pointcut을 대상으로 설정한다. 하나의 Pointcut에는 복수 Advice를 연결할 수 있다. 반대로 하나의 Advice를 복수 Pointcut에 연결하는 것도 가능하다. Joinpoint와 Advice의 중간에 있으면서 처리가 Joinpoint에 이르렀을 때 Advice를 호출할지 선별한다.

- Aspect(Advisor)

Advice와 Pointcut을 하나로 묶어 다루는 것을 Aspect라고 한다. Aspect는 스프링 AOP에만 있는 것인데, 관점 지향에서 '관점'을 나타내는 개념이라고 할 수 있다.



예를 들어,

1) 자금 이체를 하는 프로그램을 작성한다고 생각해본다.

출금 계좌와 입금 계좌, 그리고 이체 금액을 입력받아 함수를 수행할 것이다. 하지만, 이것으로 프로그램이 끝나지 않는다.

해킹방지하기 위한 보안 프로그램 점검코드 / 사용자 인증을 점검하는 코드 / 상대편 은행에서 적절하게 처리되었는지 여부의 점검코드 / 시스템 상의 로그도 남겨야 하는 코드 작성등. 구현하려는 핵심 이체기능 뿐만 아니라 보안, 인증, 로그, 트랜잭션 처리 같은 복잡한 다른 측면들을 다루는 코드가 훨씬 길어지게 될 것이다.

또 다른 예로,

2) 일반적으로 하나의 소프트웨어는 복수 모듈로 구성되는데, 어떤 관심사가 이런 모듈에 산재하는 것을

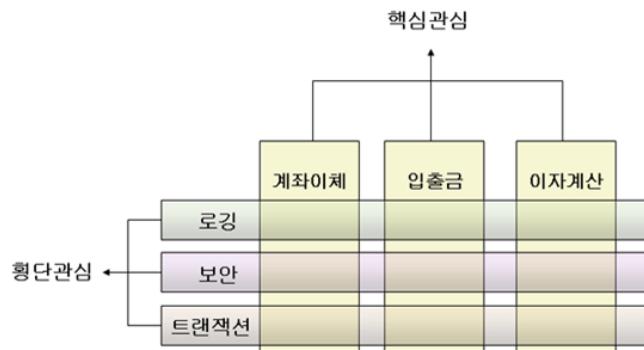
'횡단적 관심사'라 한다. 횡단적 관심사의 구체적인 예로 <로그 출력>을 들 수 있다. 로그 출력은 소프트웨어의 동작이나 사용자의 조작, 그 밖의 다양한 정보를 기록으로 출력하기 위한 중요한 관심사이다. 그러나 이런 로직이 다양한 모듈에 횡단적으로 산재해 있다는 것이 핵심 로직을 방해하는 걸림돌로 작용한다.

이때 구현하려고 하는 비즈니스 로직을-> Primary(Core) Concern

보안, 로그, 인증과 같이 시스템에 산재된 로직들을-> Cross-cutting concern 이라 한다.

AOP는 Cross-cutting concern을 어떻게 Core Concern과 분리시킬 것인가에 대한 새로운 패러다임이라고 할 수 있다.

기술적 용어로 표현하자면, Advice(Cross-cutting concern 코드)와 Primary Concern 코드를 Pointcut (Join Point가 둑인 것) 정보를 이용해서 Weaving(조합) 하는 것이 AOP가 이 문제를 다루는 방법이다.

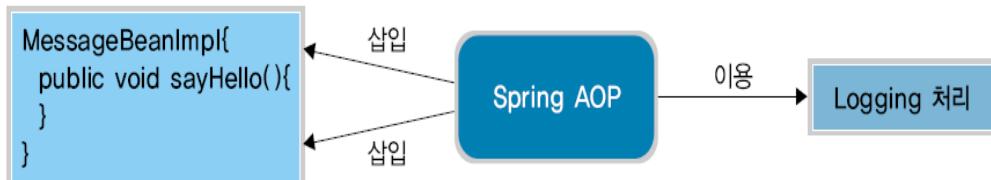


위 그림에서 봤을 때 여러가지 문제가 발생 할 수 있겠지만, 몇가지만, 생각해 보더라도 중복코드(소스가 비즈니스에 집중되어 있지 않다), 지저분한 코드, 생산성 저하(개발자가 비즈니스 이외의 다른 코드를 타이핑해야 한다), 변화가 어렵다거나 하는 것들이 발생할 것이다.(결합도 문제)

위의 그림에서 보듯이, 소스코드 안에 삽입된 채로 개발해야 했던 것들을 횡적 관심을 통해 추출하여 공통의 영역 안에서 애초에 기대했던 역할을 해내도록 추려내는 것이 AOP 사상의 핵심이다.

2) AOP를 이용한 logging 구현 예제

이 예제에서는 AOP 구조를 활용하여 메소드 트레이스 정보의 Logging 처리를 MessageBeanImpl의 sayHello() 메소드 호출 전후에 삽입한다. 로깅 처리 자체 및 그 호출은 MessageBeanImpl에는 기술하지 않는다. 스프링이 제공하는 기능인 「스프링 AOP」가 그 역할을 담당한다.



▲ 예제 개요 그림

sayHello() 메소드가 핵심 로직이고 이 메소드를 멤버로 갖는 MessageBeanImpl은 타겟 클래스가 된다. LoggingAdvice 클래스가 로깅 처리를 담당한다.

- 사용자 라이브러리에 스프링 AOP를 위한 jar 파일 포함시킨다.

spring-aop-3.2.2.RELEASE.jar

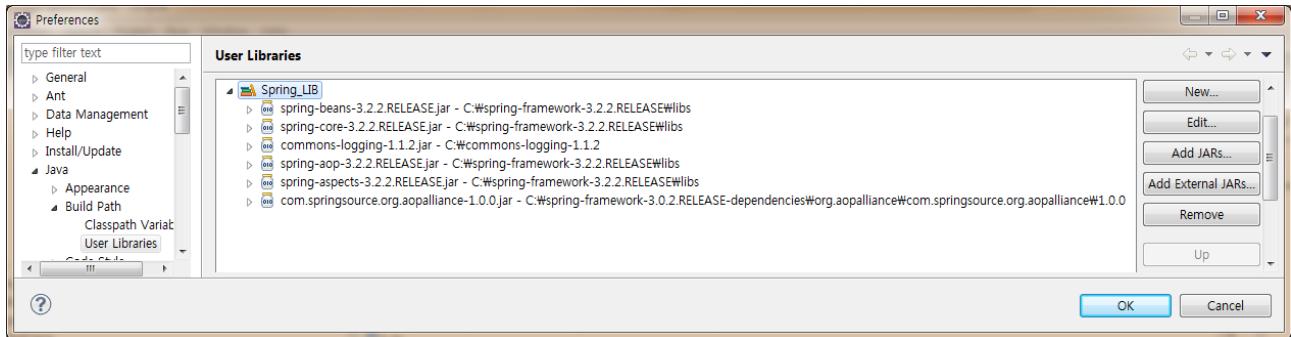
spring-aspects-3.2.2.RELEASE.jar

<http://docs.spring.io/downloads/nightly/release-download.php?project=SPR>에서

spring-framework-3.0.2.RELEASE-dependencies.zip

.\org.aopalliance\com.springsource.org.aopalliance\1.0.0\com.springsource.org.aopalliance-1.0.0.jar

사용자 라이브러리를 추가하기 위해 [Window]->[Preferences] 선택



- Target이 되는 MessageBeanImpl 클래스

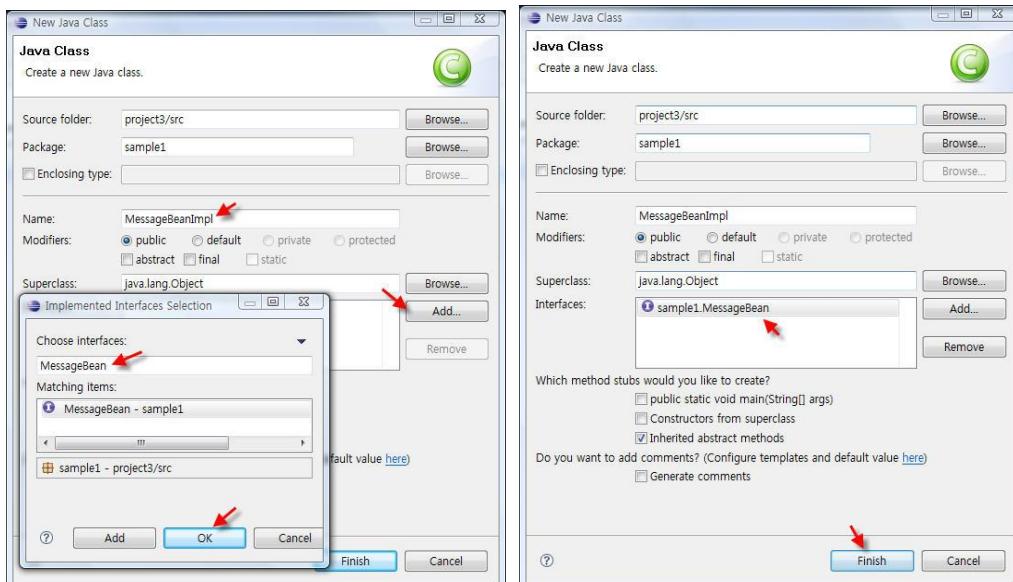
'project3'란 이름으로 [Spring Project]를 생성하고 스프링 기능을 사용하기 위해 사용자 라이브러리 'Spring_LIB'를 추가한다.

sample1 패키지 생성하고 MessageBean 인터페이스를 추가한 후 sayHello()메소드를 정의 한다.

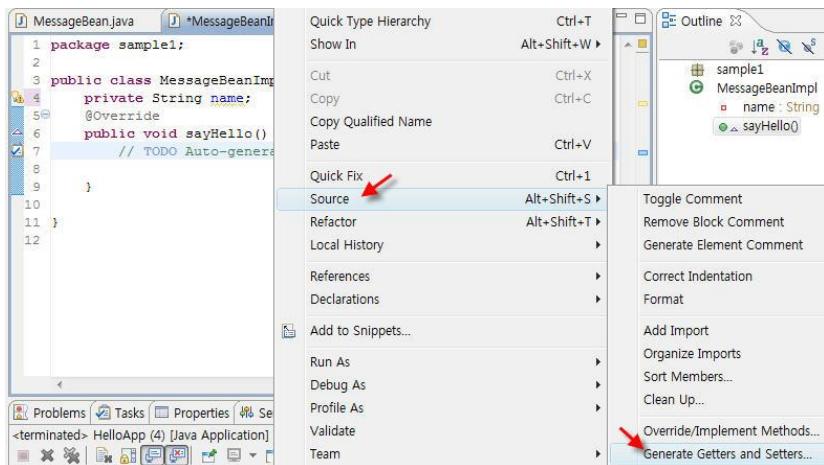
MessageBean.java

```
package sample1;
public interface MessageBean {
    void sayHello();
}
```

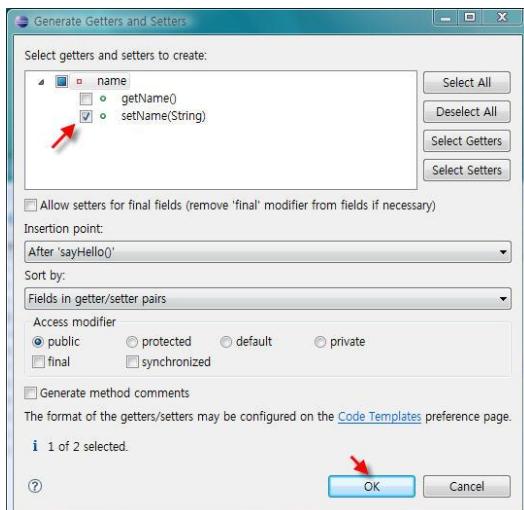
MessageBean 인터페이스를 구현한 MessageBeanImpl 클래스를 생성한 후에 name 필드를 추가한다.



name 필드에 대한 setter를 만들기 위해서 MessageBeanImpl 클래스를 정의한 내부의 바로 가기 메뉴에서 [Source] -> [Generate Getters and Setter..]를 선택



setter를 추가하기 위해 setName(string)만을 선택한다.



sayHello() 메소드에 **Advice**가 실행하는 Proxy에 처리 시간을 포함하기 때문에 Thread.sleep()메소드를 사용하여 의도적으로 처리 시간을 느리게 한다.

MessageBeanImpl.java

```
package sample1;

public class MessageBeanImpl implements MessageBean {

    private String name;

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

    public void sayHello() {
```

```

try {
    Thread.sleep(5000);
} catch(InterruptedException e) {}

System.out.println("Hello, " + name + "!");
}
}

```

스프링에서 AOP를 구현하는 과정은 다음과 같다.

- ① Advice 클래스를 작성한다.
- ② 설정 파일에 Pointcut을 설정한다.
- ③ 설정 파일에 Advice와 Pointcut을 묶어 놓는 Advisor를 설정한다.
- ④ 설정 파일에 ProxyFactoryBean 클래스를 이용하여 대상 객체에 Advisor를 적용한다.
- ⑤ getBean() 메소드로 빈 객체를 가져와 사용한다.

- ① Advice 클래스를 작성

Advice는 언제 어떤 공통 관심 사항을 실행할지에 대한 정보를 갖고 있다.

• 스프링에서 제공하는 Advice 종류

Around Advice	Joinpoint 전과 후에 실행될 Advice org.aopalliance.intercept.MethodInterceptor
Before Advice	Joinpoint 전에 실행될 Advice org.aopalliance.intercept.MethodInterceptor
After Returning Advice	Joinpoint 가 되는 메소드 호출이 정상적으로 종료된 후에 실행될 Advice org.aopalliance.intercept.MethodInterceptor
Throws Advice	예외가 Throws될 때에 실행될 Advice org.springframework.aop.AfterReturningAdvice
Introduction	클래스에 대한 인터페이스와 설치를 추가하는 특수한 Advice org.springframework.aop.IntroductionInterceptor

Joinpoint는 클래스의 인스턴스 생성 시점, 메소드 호출 시점 및 예외 발생 시점과 같이 어플리케이션을 실행할 때 특정 작업이 시작되는 시점을 의미한다. 이 예제는 Advice 중에서 Around Advice를 사용한다.

- MethodInterceptor 인터페이스 구현 클래스 작성

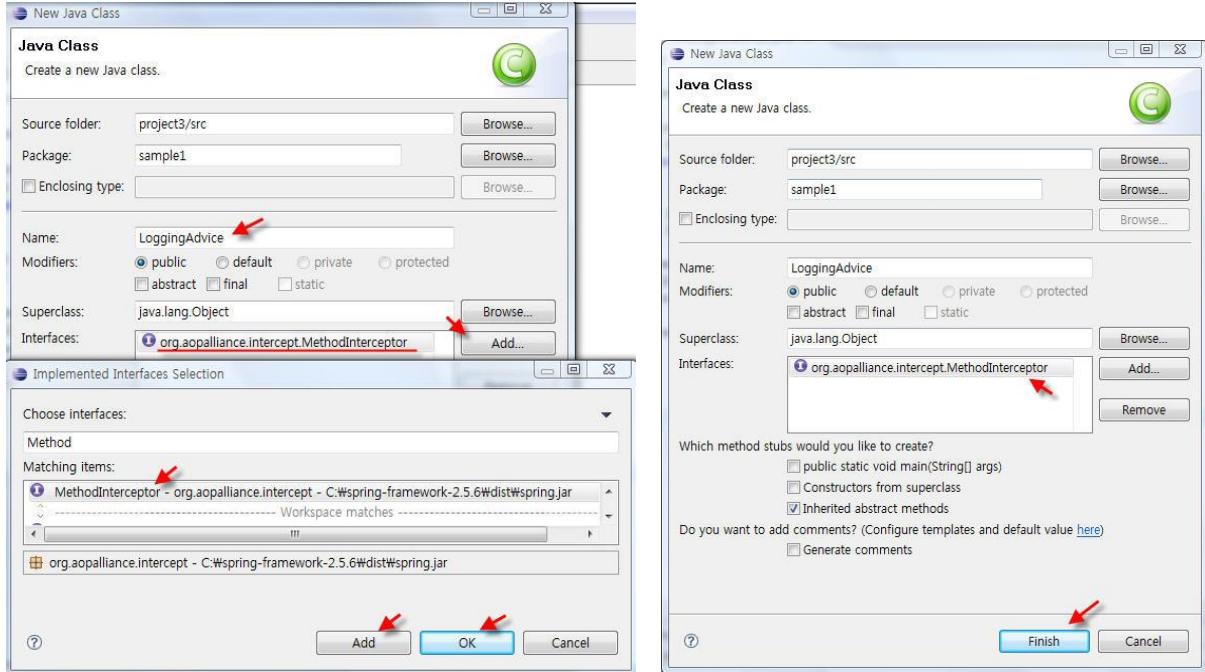
Around Advice는 MethodInterceptor 인터페이스 형태로 제공된다. MethodInterceptor 인터페이스는 AOP Alliance에서 정의한 인터페이스이다. MethodInterceptor 인터페이스는 invoke()라는 하나의 추상 메소드를 갖는다.

MethodInterceptor 인터페이스에 정의되어 있는 invoke() 메서드는 대상인 메서드 호출시에 가동되는 메서드이다.

- Advice가 되는 클래스 작성

MethodInterceptor 인터페이스를 구현한 LoggingAdvice 클래스는 MessageBeanImpl의 sayHello() 메소드 호출 전후에, Proxy 처리를 위빙하는 'Around Advice'로 작성한다.

org.aopalliance.intercept.MethodInterceptor 선택



LoggingAdvice.java

```
package sample1;  
import org.aopalliance.intercept.MethodInterceptor;  
import org.aopalliance.intercept.MethodInvocation;  
import org.springframework.util.StopWatch;  
  
public class LoggingAdvice implements MethodInterceptor {  
    public Object invoke(MethodInvocation invocation) throws Throwable {  
        String methodName = invocation.getMethod().getName();  
        StopWatch sw = new StopWatch();  
  
        sw.start(methodName);  
  
        System.out.println("[LOG] METHOD: " + methodName + " is calling.");  
        Object rtnObj = invocation.proceed();  
  
        sw.stop();  
        System.out.println("[LOG] METHOD: " + methodName + " was called.");  
    }  
}
```

```

        System.out.println("[LOG] 처리시간 " + sw.getTotalTimeMillis() / 1000 + "초");
        return rtnObj;
    }
}

```

- Advice 빈 설정

Advice는 스프링에서 빈 형태로 제공 된다. 그러므로 지금까지처럼 XML 설정 파일에서 <bean> 요소를 사용하여 빈 객체를 생성한다.

```
<bean id="loggingAdvice" class="sample1.LoggingAdvice" />
```

② Pointcut을 설정한다.

```

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

```

JdkRegexpMethodPointcut 클래스로 Pointcut 빈 객체를 정의하고 JdkRegexpMethodPointcut 클래스의 pattern 속성으로 Pointcut을 사용할 메소드의 패턴을 지정한다.

.*sayHello.*를 설정하면 'sayHello'가 포함된 메소드의 호출을 Pointcut으로 지정하게 된다.

③ Aspect(Advisor)를 설정한다.

Pointcut을 정의했다면 어떤 Advice를 어떤 Pointcut에 설정할지를 결정해야 한다.

JdkRegexpMethodPointcut 클래스를 이용하면 Advice와 Pointcut을 하나로 묶을 수 있다.

```

<bean id="helloAdvisor" class="org.springframework.aop.support.DefaultPointcutAdvisor">
    <property name="advice">
        <ref local="loggingAdvice" />
    </property>
    <property name="pointcut">
        <bean class="org.springframework.aop.support.JdkRegexpMethodPointcut">
            <property name="pattern">
                <value>.*sayHello.*</value>
            </property>
        </bean>
    </property>

```

```
</bean>
```

'helloAdvisor'란 Advisor는 'loggingAdvice'란 이름의 Advice에 'helloPointcut'란 이름의 Pointcut을 적용하고 있다.

④ProxyFactoryBean 클래스를 이용하여 대상 객체에 Advice를 적용

Advice를 적용할 대상 객체인 타겟을 설정하고 target 객체를 생성하기 위한 Bean을 정의한다.

```
<bean id="targetBean" class="sample1.MessageBeanImpl">
    <property name="name">
        <value>Spring</value>
    </property>
</bean>
```

MessageBeanImpl 클래스로 target 인스턴스를 'targetBean'이란 이름으로 생성하였다.

스프링은 Advice 적용 대상이 되는 객체(Target)를 직접 접근하기보다는 프록시를 사용하여 간접적으로 접근한다. 프록시 객체는 ProxyFactoryBean 클래스를 사용한다.

```
<bean id="proxy" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="target">
        <ref local="targetBean" />
    </property>

    <property name="interceptorNames">
        <list>
            <value>helloAdvisor</value>
        </list>
    </property>
</bean>
```

- target 속성

Advice 적용 대상 객체(Target)를 지정한다. 이미 선언해 둔 MessageBeanImpl인 인스턴스인 'targetBean'을 ref 속성값으로 주어 타겟으로 지정한다.

- interceptorNames 속성

대상 객체(Target)에 적용할 Advice 또는 Advisor를 지정한다. 여기서는 helloAdvice를 지정한다.

스프링 AOP에서는 프록시를 이용하여 메소드 호출의 인터럽트(어떤 처리가 호출되었을 때에 그 처리가 실행되기 전에 끼어들어 다른 처리를 실행하는 것)를 실현하고 타겟인 MessageBeanImpl의 인스턴스인 sayHello를 포함한 메소드가 호출되기 전과 후에 Advice 객체인 loggingAdvice의 invoke()메소드가 호출된다.

스프링 설정 파일 applicationContext.xml을 생성한다.

applicationContext.xml

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

<!-- 타깃이 되는 클래스 MessageBeanImpl 로 targetBean이라는 이름으로 빈 객체를 생성한다 --&gt;
&lt;bean id="targetBean" class="sample1.MessageBeanImpl"&gt;
    &lt;property name="name"&gt;
        &lt;value&gt;Spring&lt;/value&gt;
    &lt;/property&gt;
&lt;/bean&gt;

<!-- MessageBeanImpl로 위빙되는 Advice로 클래스인 LoggingAdvice로 loggingAdvice라는 빈
객체를 생성한다 --&gt;
&lt;bean id="loggingAdvice" class="sample1.LoggingAdvice" /&gt;

<!-- 프록시 인스턴스인 Proxy를 ProxyFactoryBean 클래스로 정의하고, 'target' 속성에 위빙 대상
이 되는 빈인 targetBean을 그리고 interceptorNames속성에는 타깃에 적용되는 Advice인
helloAdvisor를 지정한다 --&gt;
&lt;bean id="proxy" class="org.springframework.aop.framework.ProxyFactoryBean"&gt;
    &lt;property name="target"&gt;
        &lt;ref local="targetBean" /&gt;
    &lt;/property&gt;
    &lt;property name="interceptorNames"&gt;
        &lt;list&gt;
            &lt;value&gt;advisor&lt;/value&gt;
        &lt;/list&gt;
    &lt;/property&gt;
&lt;/bean&gt;

<!-- Advisor의 정의이다. 스프링 AOP에서는 Advice와 Pointcut를 하나로 묶는 것을 Advisor로 부르
고 있다. 여기서는 DefaultPointcutAdvice 클래스로 Advisor를 구현하였다. 'advice'속성에
loggingAdvice를 지정하고, 'pointcut' 속성에는 스프링이 제공하는 JdkRegexpMethodPointcut 클래
스를 지정하고 있다. JdkRegexpMethodPointcut 클래스의 'pattern' 속성의 값으로 .*sayHello.*를 지
정하여 이름에 .*sayHello.*에 포함된 클래스를 Pointcut 대상이 되도록 하였다.--&gt;
&lt;bean id="advisor" class="org.springframework.aop.support.DefaultPointcutAdvisor"&gt;</pre>
```

```

<property name="advice">
    <ref local="loggingAdvice" />
</property>
<property name="pointcut">
    <bean class="org.springframework.aop.support.JdkRegexpMethodPointcut">
        <property name="pattern">
            <value>.*sayHello.*</value>
        </property>
    </bean>
</property>
</beans>

```

⑤ getBean() 메소드로 빈 객체를 가져와 사용한다.

HelloApp.java 클래스 생성한다.

HelloApp.java

```

package sample1;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class HelloApp {
    public static void main(String[] args) {
        BeanFactory factory = new XmlBeanFactory(new FileSystemResource("applicationContext.xml"));
        MessageBean bean = (MessageBean)factory.getBean("proxy");
        bean.sayHello();
    }
}

```

Bean을 취득할 getBean() 메소드 호출로 MessageBeanImpl 클래스가 아닌 프록시가 되는 ProxyFactoryBean 클래스를 인수로 지정한다. 이 프록시에 의해 메소드가 Intercept(메소드 호출 사이에 끼어들어 그 호출 직전에 어떠한 처리를 삽입하여 실행하는 것)된다.

[실행 결과]



The screenshot shows the Eclipse IDE's Console view with the following output:

```

2013. 5. 13 오후 4:41:36 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
정보: Loading XML bean definitions from file [C:\spring3_exam\project3\applicationContext.xml]
[LOG] METHOD: sayHello is calling.
Hello, Spring!
[LOG] METHOD: sayHello was called.
[LOG] 처리시간 5초

```

2) 실전 AOP 예제

실질적인 AOP로서 POJO(Plain Old Java Object)라는 aop 스키마를 활용한 설정 파일로 활용한다. 스프링 2.0 이후 도입된 XML 스키마 기반인 스키마 설정을 활용하면 AOP에서 Pointcut 설정등을 설정 파일에 간결하게 기술할 수 있다.

이 애플리케이션을 실행하면 앞의 예제와 같은 결과이다. 하지만 앞의 예제와는 다르게 로그 출력 클래스는 MethodInterceptor 인터페이스를 구현하지 않고 만든다. POJO의 로그 출력 클래스를 Advice로서 이용하는 것이 핵심이다.

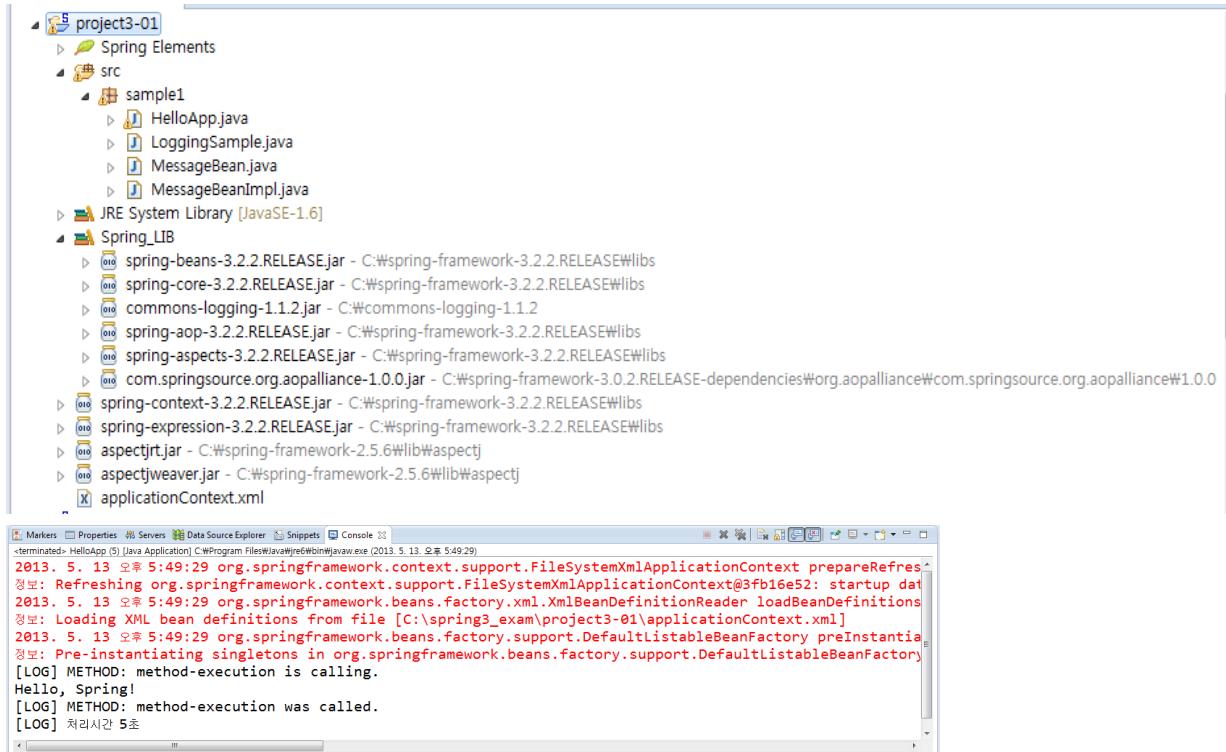
앞의 예제와 같이 AOP 구조를 이용해서 메서드 추적 정보를 출력한다. 로그 출력 처리가 있는 LoggingSample 클래스는 인터페이스 구현을 사용하지 않는다.

- AspectJ 스타일 AOP의 사용

AspectJ는 제록스 팔로알토 연구소에서 개발했던 AOP구현으로, 현재 eclipse.org 개발 프로젝트에서 관리하고 있다.

C:\spring-framework-2.5.6\lib\aspectj 에 있는 aspectjrt.jar, aspectjweaver.jar 라이브러리를 추가한다.

파일 구성과 결과



① applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

<bean id="loggingSample" class="sample1.LoggingSample" />

<aop:config>
    <aop:aspect id="logAspece" ref="loggingSample">
        <aop:pointcut expression="execution(* sayHello())" id="logPointCut"/>
        <aop:around pointcut-ref="logPointCut" method="logAround"/>
    </aop:aspect>
</aop:config>

<bean id="targetBean" class="sample1.MessageBeanImpl">
    <property name="name">
        <value>Spring</value>
    </property>
</bean>
</beans>

```

AspectJ 스타일로 AOP를 설정하려면 aop 스키마가 필요하므로 <beans> 요소에 aop 스키마를 기술한다. <aop:config> 요소를 사용해서 AOP를 설정한다. <aop:config> 요소의 자식 요소로서 <aop:aspect> 요소가 있다. 여기에 Advice를 정의한다. ref 속성에 LoggingSample 클래스를 참조하고 있다.

<aop:pointcut> 요소에서 Pointcut을 정의하고, <aop:around> 요소에서 이 Advice가 Around Advice인 것을 기술한다. <aop:around>요소의 point-ref 속성은 이 Advice를 실행하는 Pointcut을 앞에서 정의한 Pointcut에 설정하고 method 속성은 Advice 실행처리를 logAround() 메소드로 설정한다.

<aop:pointcut> 요소의 expression 속성에는 새로운 기법으로 값이 기술되어 있다. 이것은 AspectJ의 Pointcut 정의 기법이다. sayHello() 메서드 실행을 Pointcut으로 정의한다는 의미이다.

- Pointcut 정의 구문

메서드 실행을 Pointcut으로 할 경우, <aop:pointcut> 요소의 expression 속성값은 다음과 같이 기술한다.

execution(접근수준 반환값의 타입 패키지 이름, 클래스 이름, 메서드 이름(파라미터 타입, 파라미터 타입...) throws 예외 타입)

execution은 이미 정의되어 있는 Pointcut이며, 메서드나 생성자 실행을 Pointcut으로 한다. 반환값, 메서드 이름, 파라미터 타입 외에는 객체이기 때문에 생략 가능하다. Pointcut에는 와일드카드를 사용할 수 있

다.

- Pointcut 정의에 사용할 수 있는 와일드카드

와일드카드	설명
*	'.'을 포함하지 않는 임의의 문자열
..	'.'을 포함하는 임의의 문자열
+	하위 클래스 또는 하위 인터페이스

execution 등 정의되어 있는 Pointcut으로는 다음과 같은 것이 있다. 각각의 Pointcut에서 기술할 수 있는 내용이 다르므로 주의해야 한다.

- AspectJ에 정의된 Pointcut

포인트컷	설명
execution	Bean의 조건에 맞는 메서드나 생성자의 실행을 Pointcut으로 한다.
within	Bean이 조건에 설정한 타입이라면, 메서드의 실행을 Pointcut으로 한다.
this	Bean이 조건에 설정한 타입에 대입할 수 있는 인스턴스라면 메서드 실행을 Pointcut으로 한다.
target	대상이 되는 객체가 조건에 설정된 타입에 대입할 수 있는 인스턴스라면 메서드 실행을 Pointcut으로 한다.
args	메서드의 인수가 조건에 설정한 타입에 대입할 수 있는 인스턴스라면, 메서드의 실행을 Pointcut으로 한다.

몇 가지 예를 들어보면

execution(public * set*(..))

public 메서드이면서 set으로 시작되는 모든 메서드를 대상으로 한다.

execution(* sample1.*.*(..))

sample1 패키지에 있는 모든 클래스의 모든 메서드를 대상으로 한다.

execution(* sample1..*.*(..))

sample1 패키지와 그 하위 패키지에 있는 모든 클래스의 모든 메서드를 대상으로 한다

Pointcut 지정에는 논리연산자를 사용할 수 있다.

execution(public * set*(..)) **or** execution(* sample1..*.*(..))

'public 메서드이면서 set으로 시작되는 모든 메서드' 또는 'sample1 패키지에 있는 모든 클래스의 모든 메서드'를 대상으로 한다.

- AspectJ에서 사용할 수 있는 논리연산자

논리 연산자	키워드
&&	and

	or
!	not

여기에서 설명한 것을 이용해서 Pointcut을 정의한다. 물론, AspectJ에 정의되어 있는 Pointcut을 모두 이용할 수 있는 것은 아니기 때문에 주의해야 한다.(call이나 get, initialization 등은 사용할 수 없다.)

② LoggingSample.java

```
package sample1;

import org.aspectj.lang.ProceedingJoinPoint;
import org.springframework.util.StopWatch;

public class LoggingSample {
    public Object logAround(ProceedingJoinPoint pjp) throws Throwable {
        String methodName = pjp.getKind();
        StopWatch sw = new StopWatch();

        sw.start(methodName);

        System.out.println("[LOG] METHOD: " + methodName + " is calling.");
        Object rtnObj = pjp.proceed();

        sw.stop();

        System.out.println("[LOG] METHOD: " + methodName + " was called.");
        System.out.println("[LOG] 처리시간 " + sw.getTotalTimeMillis() / 1000 + "초");

        return rtnObj;
    }
}
```

인터페이스로 구현되지 않는 POJO로 만든다.

logAround() 메서드는 Advice의 처리로서 실행되는 메서드이다. Around Advice로서 동작시킬 메서드는 첫 번째 파라미터가 ProceedingJoinPoint 타입이어야 한다.

Advice로 실행할 메서드 중 ProceedingJoinPoint 클래스의 proceed() 메서드를 호출한다. proceed() 메서드는 AOP에서 대상 객체의 메서드 호출을 시작하는 메서드이다. 대상 객체의 메서드가 인수를 갖는다면 필요한 값을 Object 배열로 proceed() 메서드에 넘긴다. proceed() 메서드는 반드시 한 번만 호출한다. proceed() 메서드는 여러 번 호출하거나, 한 번도 호출하지 않으면 안된다.

③ HelloApp.java

```
package sample1;
```

```

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

public class HelloApp {
    public static void main(String[] args) {
        ApplicationContext factory = new FileSystemXmlApplicationContext("applicationContext.xml");
        MessageBean bean = (MessageBean)factory.getBean("targetBean");
        bean.sayHello();
    }
}

```

참고) 어노테이션으로 AOP 설정(project3_01)

AOP 설정을 설정 파일에 기술하고 싶은 내용 중 몇 가지는 어노테이션으로 설정한다.

설정 파일 전체를 없애는 것은 불가능하다. 설정 파일을 아래와 같이 변경한다.

```

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

    <aop:aspectj-autoproxy />
    <bean id="loggingSample" class="sample1.LoggingSample" />
    <bean id="targetBean" class="sample1.MessageBeanImpl">
        <property name="name">
            <value>Spring</value>
        </property>
    </bean>
</beans>

```

어노테이션을 사용해 AOP를 설정하는 경우 <aop:aspect-autoproxy /> 요소를 기술해야 한다.

<aop:config> 요소 내용을 어노테이션으로 설정하므로 <aop:config> 요소를 삭제한다. 그리고 어드바이스인 LoggingSample 클래스에 어노테이션을 기술한다.

```

package sample1;
import org.aspectj.lang.ProceedingJoinPoint;

```

```

import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.util.StopWatch;

@Aspect
public class LoggingSample {

    @Around("execution(* sayHello())")
    public Object logAround(ProceedingJoinPoint pjp) throws Throwable {
        ...이하 생략
    }
}

```

포인트컷은 애노테이션 괄호안에 ("execution(* sayHello())") 형식으로 기술한다.

@Aspect 어노테이션을 어드바이스로 사용하는 클래스에 기술한다. 그리고 Advice 처리를 맡은 메서드에 @Around 어노테이션을 기술한다. @Around 어노테이션 값에 Pointcut 정의를 작성한다.

Pointcut 정의는 정의 파일과 같은 방법으로 작성한다. 결과는 같다.

- AspectJ의 어노테이션

어노테이션	설명
@Aspect	Advice인 클래스에 마크를 부여하는 어노테이션
@Around	Around Advice인 메서드에 부여하는 어노테이션
@Before	Before Advice인 메서드에 부여하는 어노테이션
@After	After Advice인 메서드에 부여하는 어노테이션
@AfterReturning	After Returning Advice인 메서드에 부여하는 어노테이션
@AfterThrowing	After Throwing Advice인 메서드에 부여하는 어노테이션

설정 파일에서 정의한 경우와 기능적으로는 차이가 없다.

샘플

MyFirstAspect.java

```

package sample.aop;

@Aspect
@Component
public class MyFirstAspect {

    @Before("execution(* getProduct(String))")
    public void before(JoinPoint jp) {
        // 메소드 시작 시에 Weaving하는 Advice
    }
}

```

```

        System.out.println("Hello Before! *** 메소드가 호출되기 전에 나온다!");
        Signature sig = jp.getSignature();
        System.out.println("----->메소드 이름을 취득한다:" + sig.getName());
        Object[] o = jp.getArgs();
        System.out.println("-----> 가인수 값을 취득한다:" + o[0]);
    }
}

```

```

@After("execution(* getProduct(String))")
public void after() {
    // 메소드 종료 후에 Weaving하는 Advice
    System.out.println("Hello After! *** 메소드가 호출된 후에 나온다!");
}

```

```

@AfterReturning(value = "execution(* getProduct(String))", returning = "product")
public void afterReturning(JoinPoint jp, Product product) {
    // 메소드 호출이 예외 송출 없이 종료했을 때 호출되는 Advice
    System.out.println("Hello AfterReturning! *** 메소드를 호출한 후에 나온다! ");
    // System.out.println("-----> return value = " + ret);
    Signature sig = jp.getSignature();
    System.out.println("-----> 메소드 이름을 취득한다:" + sig.getName());
    Object[] o = jp.getArgs();
    System.out.println("----->가인수 값을 취득한다:" + o[0]);
}

```

```

@Around("execution(* getProduct(String))")
public Product around(ProceedingJoinPoint pjp) throws Throwable {
    // 메소드 호출 전후에 Weaving하는 Advice
    System.out.println("Hello Around! before *** 메소드를 호출하기 전에 나온다!");

    // Signature sig = pjp.getSignature();
    // System.out.println("-----> aop:around 메소드 이름을 취득한다:" + sig.getName());
    Product p = (Product) pjp.proceed();
    // msg = msg + ": 결과에 멋대로 추가해버린 hoge!";
    System.out.println("Hello Around! after *** 메소드를 호출한 후에 나온다!");
    return p;
}

```

```
@Around("execution(* hoge())")
```

```

public String around2(ProceedingJoinPoint pjp) throws Throwable {
    // 메소드 호출 전후에 Weaving하는 Advice
    System.out.println("***pre proceed");

    Signature sig = pjp.getSignature();
    System.out.println("Sig: " + sig.getName());
    String msg = (String) pjp.proceed();
    // msg = msg + "fuga";
    System.out.println("***post proceed");
    return msg;
}

@AfterThrowing(value = "execution(* getProduct(String))", throwing = "ex")
public void afterThrowing(Throwable ex) {
    // 메소드 호출이 예외를 내보냈을 때 호출되는 Advice
    System.out.println("Hello Throwing! *** 예외가 생기면 나온다");
    System.out.println("exception value = " + ex.toString());
}
}

```

applicationContext.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.1.xsd
           http://www.springframework.org/schema/aop
           http://www.springframework.org/schema/aop/spring-aop-3.1.xsd">
    <context:annotation-config />
    <context:component-scan base-package="sample" />
    <b>aop:aspectj-autoproxy />
</beans>

```

Spring Web MVC

1. 스프링 프레임워크

스프링 프레임워크(spring Framework)는 자바 플랫폼을 위한 오픈 소스 애플리케이션 프레임워크이며 간단히 스프링(spring)이라고도 불린다. 동적인 웹 사이트를 개발하기 위한 다양한 서비스를 제공하고 있고, 대한민국 공공기관의 전자정부 표준 프레임워크 기반 기술로서 사용되고 있다.

이 프레임워크는 2003년 6월에 최초로 아파치 2.0 라이선스로 공개되었으며, 현재 표준 프레임워크 기반 기술로 다양한 실무 프로젝트 개발에 적용되고 있다.

Spring Framework

※ 스프링은 다음과 같은 특징이 있다.

- 경량 컨테이너로서 자바 객체를 직접 관리한다. 각각의 객체 생성, 소멸과 같은 생명 주기를 관리하며, 스프링으로부터 필요한 객체를 얻어올 수 있다.
- 스프링은 POJO(Plain Old Java Object): 경량의 자바 객체 또는 별도로 종속되지 않는 자바 객체를 의미) 방식의 프레임워크이다. 일반적인 J2EE에 비하여 기존에 존재하는 라이브러리 등의 지원이 용이하고 객체가 가볍다.
- 스프링은 제어 반전(Inversion Of Control)을 지원한다. 컨트롤의 제어권이 사용자가 아니라 프레임워크에서 필요한 사용자의 코드를 호출한다.
- 스프링은 의존성 주입(Dependency Injection: 객체 사이의 의존 관계가 자기 자신이 아닌 외부에 의해서 설정된다는 개념)을 지원한다. 각각의 계층이나 서비스들 간에 의존성을 설정 파일을 통하여 프레임워크가 서로 연결시켜준다.
- 스프링은 관점 지향 프로그래밍(Aspect-Oriented Programming)을 지원한다.
- 스프링은 영속성과 관련된 다양한 API를 지원한다. JDBC, iBatis(MyBatis)나 Hibernate 등 완성도 높은 데이터베이스 처리 라이브러리와 연결할 수 있는 인터페이스를 제공한다.
- 스프링은 확장성이 높다. 스프링 프레임워크에 통합하기 위해 간단하게 기존 라이브러리 사용이 가능하기 때문에 수많은 라이브러리가 이미 스프링에서 지원되고 있고 스프링에서 사용되는 라이브러리를 별도로 분리하기도 용이하다

1) 프레임워크란? [프레임워크이란 잘 정의된 약속된 구조의 클래스의 집합을 의미한다.]

사전적 의미로는 "어떠한 것을 이루는 뼈대, 기본 구조"를 뜻한다.

소프트웨어에서의 프레임워크란 소프트웨어의 특정 문제를 프로그램으로 쉽게 그리고 편리하게 개발할 수 있도록 미리 뼈대를 이루는 클래스와 인터페이스를 제작하여 이것들을 모아둔 것이라고 할 수 있다. 다시 말해 어떤 영역의 API 들을 사용하기 편리한 형태로 포장해 놓은 것이라 할 수 있다.

- 프레임워크의 장점- 개발자의 할 일을 줄여준다.
 - 정해진 틀 안에서 코딩하기 때문에 프로그램의 가독성이 높아지므로 유지보수가 용이하다.
 - 선언적인 방법을 도입하여 프로그램의 유지보수가 용이하다.
-
- 프레임워크의 단점

- 일관성의 가치를 지나치게 높게 평가한 나머지 코딩 패턴에 너무나 많은 구속이 따른다.
- 코드가 길어지고 복잡해질 수 있다.

2. 스프링 프레임 워크 모듈 구성

스프링 프레임워크의 모듈은 Core, Container, Date Access/Integration, Web, AOP, Aspects, Instrumentation, test의 7개 카테고리에 20여개의 모듈로 구성되어 있다.

1) 코어 컨테이너

코어 컨테이너(Core Container)는 4개의 Core, Beans, Context, EL(Expression Language) 모듈로 구성되어 있다.

- Core 모듈과 Beans 모듈은 프레임워크의 기반이 되는 가장 핵심 부분으로 IoC와 DI 기능을 제공한다.
- Context 모듈은 Core 모듈과 Beans 모듈에서 제공하는 기반하에 구성되었다. Context 모듈은 Beans 모듈의 기능을 상속받고, 리소스 로딩, 서블릿 컨테이너와 같은 컨텍스트의 생성 기능들을 함께 제공한다.
- EL(Expression Language) 모듈은 런타임에서 객체 그래프를 조회하고 조작할 수 있는 강력한 표현언어 기능을 제공한다.

2) 데이터 접근/통합

데이터 접근/통합(Data Access/Integration) 계층은 5개의 JDBC, ORM, OXM, JMS, 트랜잭션 모듈들로 구성되어 있다.

- JDBC 모듈은 JDBC 추상화 계층을 제공하여 데이터베이스의 종류에 따른 JDBC 관련 코딩의 예러 코드를 대신 다루어준다.
- ORM 모듈은 iBatis(MyBatis), JPA, JDO, 하이버네이트(Hibernate)와 같이 잘 알려진 객체-관계 매핑 API에 대한 통합 계층을 제공한다.
- OXM 모듈은 JAXB, Castor, XMLBeans, JiBX, XStream과 같은 객체/XML 매핑 구현을 지원하는 계층을 제공한다.
- JMS(Java Messaging Service) 모듈은 메시지의 생성과 소비 기능을 제공한다.
- Transaction 모듈은 특별한 인터페이스와 POJO(Plain Old Java Object)의 클래스에 대한 트랜잭션 관리 기능을 제공한다.

3) 웹

웹(web) 계층은 4개의 Web, Web-Servlet, Web-Struts, Web-Portlet 모듈로 구성되어 있다.

- Web 모듈은 멀티파트 파일업로드, 서블릿 리스너와 웹 지향적인 애플리케이션 컨텍스트를 사용한 IoC 컨테이너의 초기화 등 기본적인 웹 지향적인 통합기능을 제공한다.
- Web-Servlet 모듈은 웹 애플리케이션에 필요한 스프링 MVC(Model-View-Controller) 구현을 제공하며, JSP에 대한 뷰 연동을 지원한다.
- Web-Struts 모듈은 스프링 애플리케이션과 스트럿츠의 연동 기능을 제공하며, 스프링 3.0부터 폐기되었다.
- Web-Portlet 모듈은 포틀릿 환경에서 사용되는 MVC 구현과 웹-서블릿 모듈 기능의 미러(mirror) 기

능을 제공한다.

4) AOP와 Instrumentation

- AOP 모듈은 AOP Alliance 규약에 호환되는 관점-지향 프로그래밍 구현체로서 메서드 인터셉트와 포인트 컷을 정의하여 기능별로 깔끔하게 분리하도록 할 수 있다.
 - Aspects 모듈은 AspectJ와의 통합을 제공한다.
 - Instrumentation(인스투르멘테이션) 모듈은 Instrumentation을 지원하는 클래스와 특정 애플리케이션 서버에서 사용되는 클래스 로더(class loader) 구현체를 제공한다.

5) 테스트

- 테스트(Test) 모듈은 JUnit 또는 TestNG를 사용하여 스프링 컴포넌트의 테스트를 지원한다.

3. 스프링에서 제공하는 jar 파일

스프링 프레임워크 3.2.4의 20개로 구성된 jar 파일이며, "WEB-INF/lib" 폴더에 위치한다. jar 파일에서 제공되는 기능은 다음과 같다.

1) 내부 모듈

jar 파일명	설명
spring-aop-*jar	AOP Alliance에 호환되는 AOP 구현 기능을 제공한다.
spring-aspects-*jar	AspectJ와 통합 기능을 제공한다.
spring-beans-*jar	BeanFactory 인터페이스를 통한 구현기능을 제공한다.
spring-context-*jar	코어와 빈 모듈을 확장해서 이벤트 처리, 리소스 로딩, 서블릿 컨테이너를 위한 컨텍스트 등의 추가 기능을 제공하고, applicationContext 인터페이스를 통해 구현한다.
spring-context-support-*jar	spring-context 모듈 확장, 메일, 스케줄링 기능을 제공한다.
spring-core-*jar	DI 기능의 프레임워크의 기반을 제공한다.
spring-expression-*jar	스프링 표현 언어(SpEL) 지원 클래스가 제공된다.
spring-instrument-*jar	Instrument 지원 클래스가 제공된다.
spring-instrument-tomcat-*jar	톰캣 서버의 Instrument 지원 클래스를 제공한다.
spring-jdbc-*jar	JDBC 지원 기능을 제공한다.
spring-jms-*jar	JMS(Java Message Service)의 메시지 생성과 수신 기능을 제공한다.
spring-orm-*jar	Hibernate, iBatis 등 ORM API를 위한 통합 레이어를 제공한다.
spring-oxm-*jar	객체와 XML 매핑을 지원하는 추상 레이어를 제공한다.
spring-test-*jar	JUnit 등의 스프링 컴포넌트의 단위 테스트를 지원한다.
spring-tx-*jar	스프링의 트랜잭션 관리 기능을 제공한다.
spring-web-*jar	파일업로드 등 웹 통합 기능과 원격자원 기능 등 웹 관련 기능을 제공한다.
spring-webmvc-*jar	스프링 MVC 프레임워크 기능을 제공한다.

2) 외부 의존 모듈

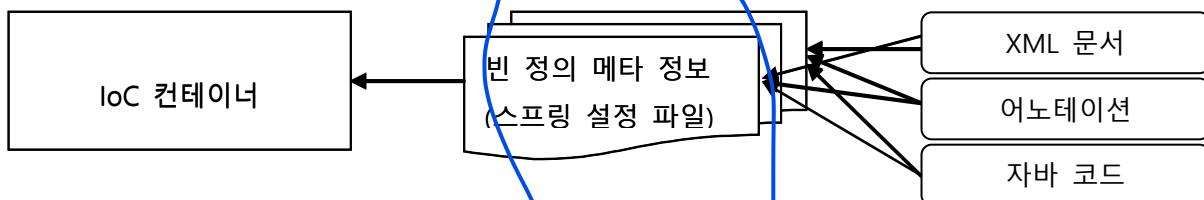
Hibernate, iBatis(MyBatis) 등 여러 외부 라이브러리를 사용한다.

spring-framework-3.x.x.RELEASE-dependencies.zip 파일에서 제공한다.

4. IoC(Inversion of Control)

IoC(Inversion of Control)란 “역 제어 또는 제어의 역전”으로 해석되며, 객체의 생명주기를 관리하고 의존성 주입(Dependency Injection)을 통해 각 계층이나 서비스들 간의 의존성을 맞춰두는 스프링에서 가장 핵심되는 기능이다.

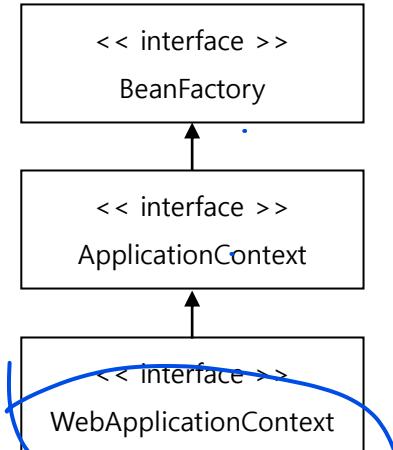
스프링 프레임워크에서는 객체의 생성, 소멸, 의존성에 관한 생명주기를 컨테이너가 관리한다. 개발자는 객체 생성에 신경 쓰지 않고 코드 작성만 하면 된다. 그러나 컨테이너가 어떻게 알 수 있을까? 그것은 xml 파일의 형태나 어노테이션, 자바 프로퍼티 파일 형태의 설정 파일에 빈을 정의하기 때문에 가능하다. 스프링에서 설정 파일은 DI 방식을 사용하며, 빈 정의(Bean Definition) 정보는 클래스 사이의 의존 관계를 IoC 컨테이너가 자동으로 설정한다. 스프링에서 애플리케이션의 중요 부분을 형성하고 스프링 IoC 컨테이너에 의해 관리되는 객체는 빈(bean) 정의 메타 정보로 참조된다. 빈들과 각각의 의존성은 IoC 컨테이너에 의해 사용되는 설정 메타 정보로 반영된다.



자바는 클래스가 부모 클래스 또는 다른 클래스로부터 상속받는 관계이다. 스프링에서는 컨테이너에 의해 빈이 생성될 때 의존성 주입이 정반대이기 때문에 제어의 역전이라고 명명하였다.

1) IoC 컨테이너

스프링 프레임워크의 IoC 컨테이너를 위한 기본 패키지는 org.springframework.beans와 org.springframework.context이다. 스프링 IoC 컨테이너의 역할을 수행하는 BeanFactory 인터페이스와 ApplicationContext 인터페이스, WebApplicationContext 인터페이스가 있다.



① BeanFactory

BeanFactory 인터페이스는 빈 객체를 관리하고, 빈 객체간의 의존 관계 설정 기능을 제공하는 가장 기본적인 컨테이너이다. Resource 구현 클래스이다.

클래스	설명
org.springframework.core.io.FileSystemResource	파일 시스템의 특정 파일에서 정보를 읽음
org.springframework.core.io.InputStreamResource	입력 스트림으로부터 정보를 읽음
org.springframework.core.io.ClassPathResource	클래스 패스에 있는 자원에서 정보를 읽음
org.springframework.core.io.UrlResource	특정 URL로부터 정보를 읽음
org.springframework.core.io.ServletContextResource	웹 애플리케이션의 루트 경로를 기준으로 지정한 경로의 자원에서 정보를 읽음

외부 자원의 설정 정보를 읽어 빈 객체를 생성하는

org.springframework.beans.factory.xml.XmlBeanFactory 클래스와 인터페이스를 사용하여 다양한 종류의 자원을 동일한 방식으로 표현하는 org.springframework.core.io.Resource 인터페이스가 제공된다. 특정 자원의 설정 정보로 XmlBeanFactory 객체를 생성한 후 getBean() 메서드로 빈을 가져와서 사용한다.

Resource 구현 클래스를 이용한 BeanFactory 객체 생성의 예는 다음과 같다.

```
Resource re = new FileSystemResource("xml 설정 파일명");
XmlBeanFactory f = new XmlBeanFactory(re);
클래스명 dao = (클래스명)f.getBean("xml 설정 파일의 빈_id");
```

② ApplicationContext

ApplicationContext 인터페이스는 BeanFactory 인터페이스의 하위 인터페이스로 BeanFactory 기능, AOP, 메시지 자원 핸들링, 이벤트 위임, XML 스키마 확장 설정 등 웹 애플리케이션의 전사적 중심의 기능을 제공한다.

- org.springframework.context.support.ClassPathXmlApplicationContext
구현 클래스는 클래스 패스에 위치한 XML 파일로부터 설정 정보를 로딩한다.
- org.springframework.context.support.FileSystemXmlApplicationContext
구현 클래스는 파일 시스템에 위치한 XML 파일로부터 설정 정보를 로딩한다.
- org.springframework.context.support.XmlWebApplicationContext
구현 클래스는 웹 애플리케이션에 위치한 XML 파일로부터 설정 정보를 로딩한다.

③ WebApplicationContext

웹 애플리케이션을 위한 ApplicationContext로 하나의 웹 애플리케이션 마다 한 개 이상의 WebApplicationContext를 가질 수 있다. 구현 클래스는 ApplicationContext 인터페이스와 동일하다.

2) 설정 메타 데이터

설정 메타 데이터는 스프링 IoC 컨테이너에 “인스턴스화, 설정, 그리고 애플리케이션내의 객체 조합”하는 방법이며, 가장 공통적으로 XML 형식을 사용한다. XML 기반의 메타 데이터는 스프링 2.5에서 도입한 어노테이션 기반의 설정과 스프링 3.0에서 도입한 자바 기반의 설정 방법이 있다. 설정 메타 데이터는 빈으

로 정의하며, 가장 상위레벨 요소로 <beans>와 내부에 <bean>으로 설정된다. 빈으로 설정하는 것은 애플리케이션에 관한 서비스 레이어 객체, 데이터 접근 객체(DAO) 등이 있다. 다음은 XML 기반의 기본적인 설정 데이터의 구조이다.

```
<?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:beans="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <bean class= "..." id="..." ><!-- 추가 빈의 객체나 설정 --></bean>
    ...
</beans>
```

3) 빈 정의

빈이란 스프링 IoC 컨테이너가 관리하는 객체를 말하며, 애플리케이션의 객체이다. 빈은 IoC 컨테이너에게 제공한 빈 정의 설정 메타 정보에 의해서 생성된다. 빈 정의란 객체를 생성하는 방법이며, XML 기반의 설정 메타 정보는 <bean> 요소로 id와 class의 필수 속성으로 정의한다. id는 빈을 구분하기 위한 문자열이며, class는 빈의 클래스명을 기술한다. 메타 데이터는 다음과 같은 정보를 포함한다.

- 패키지명을 포함한 클래스명 : 정의된 빈의 실제 구현 클래스
- 빈의 설정 요소 : 빈의 동작 상태와 참조에 관한 정보
- 생성된 객체에 관한 기타 설정 값.

빈은 id, class, name, scope, constructor-arg, property, autowire 등의 속성으로 정의한다.

```
<beans>
    <bean id="service" class= "com.service.Service" />
    <bean id="listController" class= "com.controller.ListController" />
</beans>
```

5. DI(Dependency Injection)

DI(Dependency Injection)란 “의존성 주입”으로 스프링 컨테이너가 지원하는 핵심 개념이며, DI는 객체 사이의 의존 관계를 객체 자신이 아닌 스프링 컨테이너가 수행한다. 웹 애플리케이션에서 구성 요소간의 종속성을 소스 코드에서 설정하지 않고, 외부의 설정 파일 등을 통해 주입하도록 하는 설계 패턴이다. IoC 컨테이너는 어떤 클래스가 필요로 하는 인스턴스를 자동으로 생성/취득하여 연결시켜주는 역할을 한다. IoC 컨테이너가 인스턴스를 자동 생성하게 하려면 설정 파일에서 해당 클래스 정보와 설정 메타 정보를

설정해야만 한다. 스프링은 설정 파일이나 어노테이션으로 객체간의 의존 관계를 설정하고 클래스 간에 DI의 의존성 주입 방법을 사용한다. 설정 메타 정보의 주입 방법은 2가지가 있다.

생성자 기반의 주입	생성자를 통해 의존 관계를 설정한다. XML 설정 파일에서 <bean>의 하위 요소로 <constructor-arg>를 사용한다.
Setter 기반의 주입	setter 메서드로 클래스 사이의 의존관계를 설정한다. XML 설정 파일에서 <bean>의 하위 요소로 <property>를 사용한다

1) 생성자 기반의 DI

생성자 기반의 DI는 의존 관계를 나타내는 다수의 인수로 생성자를 호출하는 컨테이너에서 클래스간의 의존관계를 설정하여 인스턴스를 참조할 수 있도록 한다. 생성자 기반의 DI는 <bean>의 하위 요소로 <constructor-arg> 속성이 있다.

속성	설명
index	Constructor의 몇 번째의 인수에 값을 전달할 것인지를 지정
type	Constructor의 어느 데이터형의 인수에 값을 전달할 것인지를 지정
ref	자식 요소 <ref bean = "bean 명" /> 대신 사용 가능
value	자식 요소 <value>값</value> 대신 사용 가능

- <constructor-arg> 요소의 ref 속성으로 빈 객체를 전달하는 DI의 예이다.

```
<bean id="bbsService" class="com.service.BbsService">
    <constructor-arg ref="bbsDAO" />
</bean>
```

- <constructor-arg> 요소의 type 속성으로 생성자 인수의 타입을 지정한다.

```
<bean id="calcBean" class="com.util.CalcBean">
    <constructor-arg type="int" value="100" />
</bean>
```

- <constructor-arg> 요소의 index 속성으로 생성자 인수를 명시적으로 지정한다.

```
<bean id="calcBean" class="com.util.CalcBean">
    <constructor-arg index= "0" value="10" />
    <constructor-arg index= "1" value="20" />
</bean>
```

2) Setter 기반의 DI

Setter 기반의 DI는 빈을 인스턴스화 하기 위해 인수가 없는 생성자나 인수가 없는 static 팩토리 메서드를 호출한 뒤에 빈의 setter 메서드를 호출하여 클래스간의 의존관계를 설정한다. Setter 기반의 DI는 <bean>의 하위 태그로 해당 클래스의 값을 설정하는 <property>와 하위 속성이 있다.

```
<property name="필드명" value="값" />
```

속성	설명
name	값의 의존 관계를 설정시킬 대상이 되는 필드명 지정
ref	객체 전달. <ref bean = "bean 명" /> 으로 표기.
value	값을 전달. <value>값</value> 으로 표기

- <property> 요소로 클래스 간의 의존관계 설정 – 설정하는 클래스는 <ref> 속성으로 표기한다.

```
<bean id="bbsDAO" class="com.dao.BbsDAO" />
<bean id="bbsService" class="com.service.BbsService" >
    <property name="dao"><ref bean="bbsDAO"></property>
</bean>
<bean id="listController" class="com.controller.ListController" >
    <property name="service"><ref bean="bbsService"></property>
</bean>
```

- <property> 요소의 value 속성

```
<bean id="dataSource" class="org.apache.~.dbcp.BasicDataSource" >
    <!--setDriverClassName(String) 메서드 호출 -->
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:orcl" />
    <property name="username" value="scott" />
    <property name="password" value="tiger" />
</bean>
```

- 이너 빈(inner beans)

<property>나 <constructor-arg> 내부에 <bean>으로 정의한 빈을 이너 빈(inner bean) 또는 내부 빈이라고 부른다.

```
<bean id="outer" class="..." />
    <!-- 대상 빈에 대한 참조를 인라인으로 정의 -->
    <property id="target">
        <bean class="com.exam.Person" ><!-- 이너 빈 -->
            <property name="name" value="hong gil dong" />
            <property name="age" value="25" />
        </bean>
    </property>
</bean>
```

- 부모 빈 참조

동일한 빈 설정 정보가 중복될 때 부모 빈을 설정하고, 부모 빈 설정 정보를 참조하여 재사용할 수 있다.

부모 빈을 참조하는 빈에 지정된 프로퍼티의 값을 설정한다. 가장 일반적인 방법은 <ref>의 parent 속성으로 대상 빈을 지정한다.

```
<!-- 부모 컨텍스트 -->
<bean id="bbsService" class="com.service.BbsService" />
<!-- 자식 컨텍스트 -->
<bean id="bbsService" <!-- 부모 빈명과 동일 -->
    class="org.springframework.aop.framework.ProxyFactoryBean" >
        <property name="target"><ref parent="bbsService"></property>
        <!-- 다른 설정과 의존성 추가 -->
    </bean>
```

다음과 같이 dataSource의 설정 빈을 참조하여 다른 빈에 재사용할 수 있다.

```
<bean id="dataSource" class="org.apache.~.DriverManagerDataSource" >
    <property name="driverClassName" value="${jdbc.driverClass}" />
    <property name="url" value=" ${jdbc.url}" />
    ...
</bean>
<bean id="transactionManager" class="org.~DataSourceTransactionManager" >
    <property name="dataSource" ref="dataSource" />
</bean>
<bean id="sqlSessionFactory" class="org.~SqlSessionFactoryBean" >
    <property name="dataSource" ref="dataSource" />
    ...
</bean>
```

- 컬렉션(Collection)

<property> 또는 <constructor-arg>의 하위 태그로 자바 컬렉션 타입인 <list>, <set>, <map>, <props> 태그로 프로퍼티와 인수를 설정할 수 있다.

태그	타입	설명
<list>	java.util.List, 배열	List 타입이나 배열에 값 목록을 전달할 때
<set>	java.util.Map	Map 타입에 <키, 값> 목록을 전달할 때
<map>	java.util.Set	Set 타입에 값 목록을 전달 할 때
<props>	java.util.Properties	Properties 타입에 <프로퍼티이름, 프로퍼티값>목록을 전달할 때

컬렉션에는 <list>, <set>, <map>, <props>가 있으며, 컬렉션에 값을 설정하는 태그는 다음과 같다.

태그	설명	태그	설정
<ref>	<bean>으로 등록된 객체	<bean>	임의의 빈
<value>	기본값, 스트링 타입	<null>	널(null)

컬렉션을 설정하는 일반 형식은 다음과 같다.

```
<bean id="id명" class="클래스명..." >
    <property name="item명">
        <list> | <set> | <map> | <props>
            <ref bean="값1" > | <entry key="1" value-ref="값1" /> | <prop key="키명1">값1</prop>
            ...
            <ref bean="값n" > | <entry key="n" value-ref="값n" /> | <prop key="키명n">값n</prop>
        </list> | </set> | </map> | </props>
    </property>
</bean>
```

- <list> 태그는 List 타입이나 배열에 저장될 객체 목록을 <ref>, <value> 태그로 값을 설정한다.
- <set> 태그는 <list>와 같은 방법으로 값을 설정한다.
- <map> 태그는 Map 계열의 컬렉션 객체들을 <entry> 태그를 이용하여 value 값을 맵에 설정한다.
- <entry> 태그는 <키, 값>으로 표현하며, <key>/<value>, <key-ref>/<value-ref>, <key-type>/<value-type> 속성으로 설정한다.
- <props> 태그는 "java.util.Properties" 문자열 값을 설정하며, <props>의 <prop key="키명">에 키값을 설정한다.

3) XML 네임스페이스를 이용한 프로퍼티 설정

XML 네임스페이스는 <property> 대신에 간단하게 설정하는 방법이다. XML 네임스페이스를 사용할 경우에는 xmlns:p="http://www.springframework.org/schema/p"가 <beans>에 선언되어야 한다.

- p 네임스페이스는 <property> 요소 대신 bean 요소의 속성으로 표기한다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    ...
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        p:driverClassName="oracle.jdbc.driver.OracleDriver"
        p:url="jdbc:oracle:thin:@127.0.0.1:1521:orcl"
        p:username="scott"
        p:password="tiger" />
    </beans>
```

- c 네임스페이스는 <constructor-arg> 대신 인라인 속성으로 설정한다.

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

```

xsi:schemaLocation="http://www.springframework.org/schema/beans
...
http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<bean id="data" class="com.test.Data" >
    <constructor-arg ref="doc">
        <constructor-arg value="data12@naver.com">
    </bean>
    <!-- 'c-namespace' declaration -->
    <bean id="data" class="com.test.Data" c:bar-ref="doc" c:email="data12@naver.com" />
</beans>

```

6. 의존 관계 자동 설정

의존 관계 자동 설정은 자동 설정 모드를 통하여 빈을 설정하는 방법이다. 프로퍼티나 생성자 인수를 지정하는 일을 현저하게 줄일 수 있기 때문에 개발할 때 매우 유용한 방법이다. XML 기반의 설정 메타 데이터를 사용할 때 <bean> 요소의 autowire 속성으로 “모드”를 지정한다. 자동 설정 모드는 다음과 같이 5 가지가 존재한다.

속성	설명
no	기본값이며 자동 설정하지 않는다.
byName	프로퍼티 이름에 의한 자동 설정
byType	프로퍼티 타입을 이용한 자동 설정
constructor	생성자 파라메타 타입을 이용한 자동 설정
autodetect	constructor와 byType을 이용한 자동 설정

1) byName

빈의 프로퍼티명과 일치하는 빈의 “name”이나 “id”가 있으면 자동 설정한다. 전달되는 빈 객체가 setXXX() 메서드로 받아야 한다. 프로퍼티명으로 전달하기 때문에 소스 코드의 프로퍼티명이 변경되면 설정 파일도 변경해야 한다.

2) byType

빈의 프로퍼티 타입과 일치하는 빈의 “name”이나 “id”가 있으면 자동 설정한다.

3) constructor

constructor는 byType과 동일하게 타입을 이용하여 의존관계를 자동 설정한다.

4) autodetect

autodetect는 constructor 방식을 먼저 적용하고, constructor 방식을 적용할 수 없을 경우 byType 방식을 적용하여 의존관계를 자동 설정한다.

5) 자동 설정의 한계와 단점

- <property>와 <constructor-arg> 설정의 명시적 의존성을 항상 자동 설정을 재정의한다. 원시 타입, 문자열, 클래스 같은 간단한 프로퍼티들은 자동 설정할 수 없다.
- 자동 설정은 명시적인 설정보다 정확하지 않다.
- 설정 정보는 스프링 컨테이너에서 문서 생성 같은 도구에서는 사용할 수 없다.
- 컨테이너에서 다수의 빈 정의들은 자동 설정되는 setter 메서드나 생성자 인수로 지정된 타입과 일치하는 것을 찾는다. 단일 값을 갖는 의존성에 대해 모호할 경우 임의로 처리되지 않으며, 유일한 빈 정의를 찾지 못하면 예외가 발생한다.

7. 클래스 자동 검색과 컴포넌트

클래스패스(classpath)에 위치한 클래스들을 검색하여 어노테이션이 붙은 클래스를 빈으로 자동 설정하는 기능이 제공된다. 스프링 2.0의 @Repository, 스프링 2.5 @Component, @Service, @Controller 어노테이션으로 소스코드의 클래스 선언부에 명시한다.

1) @Component와 stereotype(스테레오타입) 어노테이션

@Repository, @Component, @Service, @Controller 어노테이션은 스프링이 관리하는 모든 컴포넌트에 대한 제너릭 스테레오타입이다. 이 어노테이션들은 애플리케이션의 프리젠테이션 계층, 서비스 계층, 퍼시스턴스 계층에서 사용된다.

@Component : 일반적인 컴포넌트를 설정하는 기본 스테레오 타입.

@Repository : 퍼시스턴스 계층의 DAO 컴포넌트

@Service : 서비스 계층의 서비스 컴포넌트

@Controller : 프레젠테이션 계층의 컨트롤러 컴포넌트

2) 클래스 자동 검색과 빈 설정

스프링은 스테레오타입의 클래스들을 검색하고 대응되는 ApplicationContext의 빈을 자동으로 설정한다.

클래스들을 자동 검색하고 대응하는 빈을 설정하려면 XML에 <context:component-scan> 요소를 포함시켜야 한다. base-package 요소는 클래스에 대한 공통 부모 패키지이다.

```
<context:component-scan base-package="패키지명" />
```

다음은 명시한 기본 패키지명에서 클래스 자동 검색과 빈을 설정하는 예이다.

```
<context:component-scan base-package="com.**.controller" />
<context:component-scan base-package="com.**.service" />
<context:component-scan base-package="com.**.dao" />
```

◆ ant 형식의 경로 패턴에 사용되는 3가지 대체 문자

① ? : ? 위치의 1개 문자 대체

② * : * 위치의 모든 문자 대체

③ ** : ** 위치의 모든 패키지(또는 디렉토리) 대체

3) <context:component-scan>의 하위 요소

@Component, @Repository, @Service, @Controller와 @Compenent가 붙은 애너테이션의 클래스들만 후보 컴포넌트로 탐색된다. <component-scan>의 하위요소로 <context:include-filter>나 <context:exclude-filter>를 추가한다.

- include-filter : 자동 스캔 대상에 포함시킬 클래스
- exclude-filter : 자동 스캔 대상에서 제외시킬 클래스

각 필터 요소들은 type과 expression 속성을 필요로 한다.

include-filter/exclude-filter에서 type속성의 필터 타입

```
<context:include-filter type="" expression="" />
<context:exclude-filter type="" expression="" />
```

필터 타입	설명
annotation	대상 컴포넌트에서 타입 레벨로 표현되는 애너테이션
assignable	대상 컴포넌트가 extend/implement로 할당 가능한 클래스나 인터페이스
aspectj	대상 컴포넌트들과 일치되는 AspectJ 타입 표현식
regex	대상 컴포넌트 클래스명과 일치되는 정규 표현식
custom	org.springframework.core.type.TypeFilter 인터페이스를 구현한 커스텀 구현체

- "com" 하위 패키지에서 자동 검색될 때 "org.springframework.stereotype.Controller"의 애너테이션 태입을 포함시키는 <context:component-scan>의 설정은 다음과 같다.

```
<context:component-scan base-package="com"
    <context:include-filter type="annotation"
        Expression="org.springframework.stereotype.Controller" />
</context:component-scan>
```

8. 애너테이션 기반의 설정

애너테이션 기반의 설정은 XML로 컴포넌트를 설정하지 않고 소스 코드의 클래스나 메소드에 애너테이션으로 선언하는 방법이다. 스프링의 2.0의 @Required, 스프링 2.5에서 @Autowired, @PostConstruct, @PreDestroy, 스프링 3.0에서 @Inject, @Named 애너테이션을 사용한다. 애너테이션 주입은 XML 주입 이전에 실행되기 때문에 두 가지 방법을 사용할 경우 프로퍼티들은 XML 설정으로 치환된다.

애너테이션	설명
@Required	프로퍼티 설정 메소드에 @Required 애너테이션 기술
@Autowired	의존 관계 자동 설정. 설정자, 필드, 메서드에 적용 가능. 빈 객체의 타입으로 자동 주입
@Qualifier	타입이 동일한 빈 객체의 특정 빈 설정
@Resource	어플리케이션에서 필요로 하는 자원 자동 설정. 빈 객체의 name 속성으로 자동 주입
@Inject	@Autowired 애너테이션 대신 사용. required 속성 없음

@Named	@Component 애너테이션 대신 사용
--------	------------------------

애너테이션 기반의 설정 방법을 이용할 경우 스프링 설정 파일의 <brans>내부에 <context:annotation-config>로 빈 객체를 설정해야 인식된다.

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

9. 애너테이션 기반의 설정

1) 자바 프로퍼티 형식의 파일

표준 자바 프로퍼티 형식을 사용하는 분리된 파일의 빈 정의에서 프로퍼티 값을 구체화하기 위해서 PropertyPlaceholderConfigurer를 사용한다. PropertyPlaceholderConfigurer 클래스를 빈으로 설정하면 외부의 프로퍼티 파일에 저장된 정보를 스프링 설정 파일에서 사용할 수 있다.

데이터베이스 서버에 관련된 jdbc.properties 프로퍼티 파일의 예

```
jdbc.driverClassName = oracle.jdbc.driver.OracleDriver
jdbc.url = jdbc:oracle:thin:@192.168.0.33:1521:orcl
jdbc.username = scott
jdbc.password = tiger
```

jdbc.properties 파일을 스프링 설정 파일에서 사용할 경우 PropertyPlaceholderConfigurer 클래스를 빈으로 설정하고, <property>의 value 속성으로 프로퍼티 파일을 설정한다. dataSource가 정의된 XML 기반 설정 메타 데이터는 런타임에서 PropertyPlaceholderConfigurer는 dataSource의 프로퍼티들을 대체할 메타 데이터로 설정한다. 대체할 값을 \${property-name} 형식으로 지정한다.

```
<bean class="org.springframework.~.config.PropertyPlaceholderConfigurer">
    <property name="locations" value="classpath:jdbc.properties" />
<bean />
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
<bean />
```

2) XML 기반의 설정 파일

여러 XML 파일로부터 빈 정의를 설정할 수 있다. <import>요소는 다른 파일의 빈 정의 파일을 "resource"에서 검색하여 로드한다. import된 파일의 최상위에 <beans>가 있는 스프링 스키마나 XML 빈 정의 파일이어야 한다.

```
<import resource="servlet-context.xml" />
```

3) AnnotationConfigWebApplicationContext 사용

AnnotationConfigWebApplicationContext는 스프링 ContextLoaderListener 서블릿 리스너, 스프링 MVC 디스패처서블릿 등을 설정할 때 사용할 수 있다.

전형적인 스프링 MVC 웹 애플리케이션을 설정하는 "web.xml"의 예제이다.

```
<web-app ...>
    <!-- AnnotationConfigWebApplicationContext으로 ContextLoaderListener 설정 -->
    <context-param>
        <param-name>contextClass</param-name>
        <param-value>org.~.AnnotationConfigWebApplicationContext</param-value>
    </context-param>
    <!-- 설정 위치는 컴포넌트 스캔으로 지정한다. -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/폴더명/spring-context.xml</param-value>
    </context-param>
    <!-- ContextLoaderListener로 루트 애플리케이션 시작 -->
    <listener>
        <listener-class>org.springframework.web.~.ContextLoaderListener</listener>
    ....
    </web-app>
```

10. 스프링 웹 MVC

기존 MVC 패턴은 스프링에 적용하고자 설계된 것이 스프링 웹 MVC 모듈이다. 스프링 웹 MVC 모듈은 요청 기반의 웹 MVC 프레임워크로 설정 가능한 핸들러 매핑, 뷰 분석, 파일 업로드를 위한 테마 분석과 함께 핸들러에 요청을 할당하는 디스패처 서블릿 기반으로 설계되었고, 웹 애플리케이션 배치를 쉽게 해주는 많은 기능을 제공한다. 기본 핸들러는 @Controller와 @RequestMapping 어노테이션이며, @RequestMapping을 통하여 유연한 요청 처리가 가능하다.

스프링 웹 MVC 모듈은 모든 웹 클라이언트의 요청을 하나의 서블릿이 받아서 처리하는 프론트 컨트롤러를 사용하여 구현되었고, 프론트 컨트롤러라고 부르는 디스패처 서블릿이 핸들러를 호출하여 동작한다.

1) 스프링 웹 MVC의 주요 구성 요소

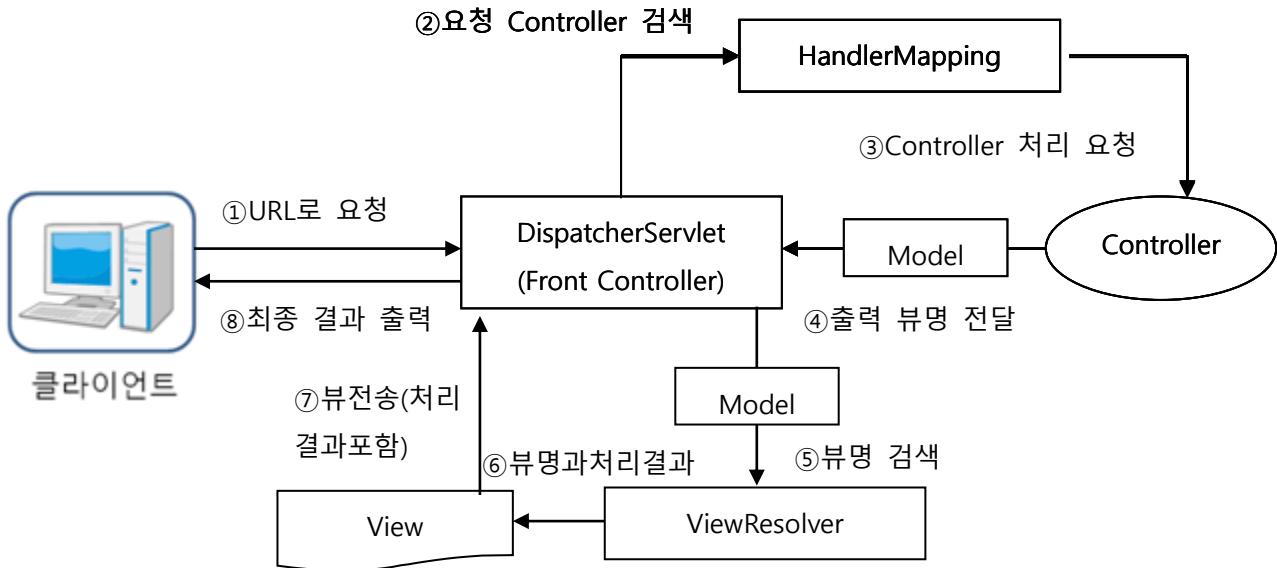
스프링 웹 MVC의 주요 구성요소는 디스패처 서블릿(DispatcherServlet), 핸들러 매핑(HandlerMapping), 컨트롤러(Controller), 뷰리졸버(ViewResolver), 뷰(View)이다.

- 디스패처 서블릿(DispatcherServlet)은 클라이언트의 요청을 전달받아 컨트롤러에게 요청을 전달한다. 컨트롤러가 반환한 결과값을 뷰에 전달하여 적절한 응답을 생성하는 역할을 하며, 이 정보를 "web.xml" 설정 파일에 정의한다.
- 핸들러 매핑은 클라이언트의 요청을 처리할 컨트롤러를 찾는 처리를 한다.
- 컨트롤러는 클라이언트의 요청 처리를 수행하고, 로직을 담당하는 모델에 데이터를 전달하여 처리하게

한다

- 뷰리졸버는 응답할 뷰를 찾는 처리를 한다.
- 뷰는 컨트롤러는 모델로부터 처리 결과를 전달받아 뷰로 처리 결과 화면을 생성하는 역할을 한다.

2) 스프링 웹 MVC의 요청 처리 절차



- ① 클라이언트에서 url 형태로 웹 서버에게 요청한다.
- ② 요청 정보는 프론트 컨트롤러인 디스패처 서블릿에 전달되고, 요청한 컨트롤러가 있는지 핸들러 매핑을 검색한다.
- ③ 핸들러 매핑은 디스패처 서블릿이 요청한 적절한 컨트롤러를 검색하여 컨트롤러에게 처리를 요청한다.
- ④ 컨트롤러는 요청에 대한 비즈니스 로직을 처리하고, 출력할 뷰명과 처리 결과를 디스패처 서블릿에 전달한다.
- ⑤ 디스패처 서블릿은 컨트롤러가 전송한 뷰명을 뷰리졸버를 통하여 뷰를 검색한다.
- ⑥ 뷰리졸버는 해당되는 뷰명에게 처리 결과를 보낸다.
- ⑦ 뷰는 처리 결과가 포함된 뷰를 디스패처 서블릿에게 전송한다.
- ⑧ 디스패처 서블릿은 요청한 최종 결과를 클라이언트에 출력한다.

3) 스프링 웹 MVC의 구현 내용과 순서

스프링 웹 MVC에서 사용자가 구현하는 부분은 컨트롤러와 뷰이며, 웹 애플리케이션 설정파일, 스프링 설정파일, 핸들러 매핑, 뷰리졸버 등은 설정만 하면 된다. 구현 순서는 사용자의 취향이며 중요하지 않다.

웹 애플리케이션에 따라 다음과 같은 구현 내용은 달라질 수 있다.

- 뷰를 JSP 페이지로 작성한다.
- 컨트롤러를 작성한다.
- 웹 애플리케이션 설정 파일(web.xml)에 디스패처 서블릿을 설정한다.
- 스프링 설정 파일에 핸들러 매핑, 컨트롤러, 뷰리졸버를 설정한다.
- 웹 애플리케이션의 데이터를 다루는 자바빈을 생성한다.

- 데이터베이스 연동에 필요한 SQL 구문을 작성한다.
- 데이터베이스를 다루는 DAO 인터페이스와 구현 클래스를 작성한다.
- 컨트롤러와 데이터베이스를 다루는 서비스 로직을 작성한다.
- 데이터베이스 서버 연동에 필요한 프로퍼티 파일을 생성한다.

11. 디스패처 서블릿

디스패처 서블릿(DispatcherServlet)이란 모델(Model)과 컨트롤러(Controller)와 뷰(View)를 조합하여 웹 브라우저로 출력해 주는 역할을 수행하는 클래스이며, 해당 애플리케이션으로 들어오는 모든 요청을 핸들링하는 역할을 담당한다. 디스패처 서블릿(DispatcherServlet)은 서블릿과 같이 웹 애플리케이션의 "WEB-INF" 폴더의 "web.xml" 파일에 사용자 요청을 url을 설정한다.

1) 디스패처 서블릿(DispatcherServlet)과 스프링 설정

디스패처 서블릿(DispatcherServlet)의 설정은 웹 애플리케이션 설정 파일인 "WEB-INF" 폴더의 "web.xml" 파일에 다음과 같이 2가지를 설정한다.

- 클라이언트의 요청을 전달받을 디스패처 서블릿(DispatcherServlet)
- 공통으로 사용할 애플리케이션 컨텍스트(context)

웹 애플리케이션 설정 파일인 "web.xml"에서 <servlet>과 <servlet-mapping>으로 서블릿명과 url 패턴을 설정한다. 디스패처 서블릿(DispatcherServlet) 클래스는 <servlet-name> 요소에 "spring"과 같은 서블릿명을 정의하고 <servlet-mapping> 요소에 사용자 요청 url 패턴을 정의한다. 컨테이너에 로딩되는 스프링 설정 파일명은 "spring-servlet.xml"과 같이 "서블릿명"에 "-servlet.xml"을 추가한다. 이 파일을 "WEB-INF" 폴더에 생성하면 자동으로 컨테이너에 로딩된다. 디스패처 서블릿의 기본 설정은 다음과 같다.

```
<!-- ===== 스프링 관련 설정 ===== -->
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

```
<!-- ===== 스프링 관련 설정 ===== -->
<servlet>
    <servlet-name>springMVC</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
```

```

<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/framework/*.xml</param-value>
</init-param>
</servlet>
<servlet-mapping>
    <servlet-name>springMVC</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>

```

"web.xml"의 <url-pattern>이 *.do로 설정할 경우 클라이언트에서 "*.do"로 끝나는 url 요청 정보가 디스패처 서블릿에 전달된다. 디스패처 서블릿(DispatcherServlet)은 "WEB-INF" 폴더의 "spring-servlet.xml"의 스프링 설정 파일을 초기화한다. 이 설정 파일에 웹 애플리케이션에 필요한 추가적인 기능들을 선언 할 수도 있다. 디스패처 서블릿(DispatcherServlet)이 요청 처리에 필요한 스프링 MVC 구성요소인 핸들러 매핑, 컨트롤러, 뷰리졸버, 뷰 등을 빈으로 설정한다.

12. 컨트롤러

컨트롤러는 스프링 MVC 설계 패턴의 하나이며, 사용자가 구현하는 부분이다. 컨트롤러는 사용자 요청을 해석하고 모델과 뷰를 반환하는 기능을 가진 메서드이다. 기본적이 요소로 컨트롤러의 메서드, 어노테이션, 그리고 사용자에게 요청한 정보를 제공할 모델과 뷰명이 포함되어야 한다. 컨트롤러는 디스패처 서블릿이 요청한 핸들러 매핑에 의해 호출되며 비즈니스 로직을 처리한다. 핸들러 매핑에 필요한 컨트롤러의 설정은 스프링 설정 파일에 빈으로 정의하며 표기 방법도 다양하다. 스프링 3.0 이전의 컨트롤러는 "org.springframework.web.servlet.mvc.Controller" 인터페이스를 구현하여 AbstractController, MultiActionController 등 다양하게 작성하였다. 컨트롤러를 처음 작성할 때 스프링의 다양한 종류의 컨트롤러, 파라미터, 반환타입에 힘들어 한다. 그러나 스프링 3.0부터 좀 간결하게 작성할 수 있는 @어노테이션 기반의 컨트롤러 사용을 권장하고 있다.

종류	설명	용도
Controller, AbstractController	모든 처리 작업 직접 구현	단순 처리
AbstractCommandController	파라메타 값 검증기능 제공	파라미터 매핑
AbstractFormController	폼 전송 지원, 유효성 검사	자바빈 사용
SimpleFormController	폼 출력, 입력 데이터 처리	입력 폼 처리
MultiActionController	다수의 로직을 하나로 구현	다중 액션
UrlFilenameViewController	단순히 요청을 뷰로 전달	정적 뷰 매핑

1) 컨트롤러의 기본 구조

컨트롤러는 일반적으로 다음과 같은 단계로 구현되는 메서드이다. 사용자의 요청을 해석한다. → 요청에 대한 처리를 서비스 로직 등에 위임한다 → 반환된 값으로 모델을 생성한다. → 뷰를 결정한다. → 뷰와 모델을 반환한다 하나의 컨트롤러는 클라이언트 요청 처리에 대해서 다수의 구현 메서드가 필요할 수도

있다. @Controller로 컨트롤러임을 선언하고 @RequestMapping으로 요청 url 패턴을 지정한다. 비즈니스 로직들은 서비스 객체 등에 위임하고 반환된값으로 모델을 생성하여 디스패처 서블릿에 반환한다. 컨트롤러의 기본적인 구조는 다음과 같다.

```
@Controller
public class 컨트롤러명 {
    @RequestMapping("요청url패턴")
    public 반환타입 메서드명(Model 파라메타명...) {
        비즈니스 로직;
        return "뷰명"
    }
}
```

사용 예

```
@Controller
public class HelloWorldController {
    @RequestMapping("hello.do")
    public String helloWorld(Model model) {
        model.addAttribute("message", "Hello World SpringFrameWork!!!");
        return "HelloWorld"
    }
}
```

2) 컨트롤러 @어노테이션

어노테이션이란 메타 데이터를 기술하는 특별한 구문으로, 클래스, 메서드, 변수, 파라메타 선언과 패키지 등의 소스 코드에 삽입하여 사용할 수 있다. 이를 통하여 소스 코드의 가독성을 높일 수 있으며, “@어노테이션” 형태로 표기한다. 스프링 3.0 이전에 트랜잭션 관리, 빈 관리와 의존성 주입 등이 어노테이션이 사용되었다. 스프링 3.0에서는 @Controller의 컨트롤러의 구현을 권장한다.

종류	설명
@Controller	스프링 웹 MVC의 컨트롤러 선언. 클래스 타입에 적용. org.springframework.stereotype.Controller
@RequestMapping	컨트롤러와 매핑되는 url 패턴 정의. http 메서드 등 지원. org.springframework.web.bind.annotation.RequestMapping
@SessionAttribute	세션 설정과 세션 유지 org.springframework.web.bind.annotation.SessionAttribute

① @Controller

@Controller는 특정 클래스의 상단에 명시하여 컨트롤러를 선언한다. @Controller는 org.springframework.stereotype.Controller 패키지가 필요하며, @Controller를 선언한 컨트롤러는 컴포넌트

자동 검색 대상이며, 스프링 설정 파일에서 <context:component-scan base-package="패키지명" /> 태그에 컨트롤러의 패키지명을 선언해야 어노테이션이 선언된 컨트롤러를 자동으로 로딩할 수 있다.

② @RequestMapping

@RequestMapping은 클라이언트의 요청 url 패턴이나 http 요청 메서드에 대해서 컨트롤러의 클래스나 메서드에 선언한다. @RequestMapping은 문자열 url, http 메서드, params() 타입, header(), 메서드 레벨과 매핑한다.

입력타입	매팅 어노테이션	설명
url/POST	@RequestParam	GET/POST/DELETE/PUT으로 호출될 때 전달되는 파라미터 값 단일 http 요청 파라미터를 메서드 파라미터에 전달하는 어노테이션. login() 메서드에 @RequestParam의 "id"와 "pwd"를 전달한다. login(@RequestParam("id") String id, @RequestParam("pwd") String pwd){...}
Path value	@PathVariable	url에 포함된 특정 영역 문자열
Servlet	-	HttpServletRequest, HttpServletResponse를 직접 핸들링하는 기존 서블릿과 같은 코드 사용
Cookie	@CookieValue	쿠키값을 얻거나 설정
Session	@SessionAttribute	세션값을 얻거나 설정
Body	@RequestBody	Request의 Body를 String으로 얻어낼 때 (http 요청 본문 부분이 전달되는 어노테이션)

- 문자열을 이용한 url 매핑

@RequestMapping의 괄호()속에 "url 패턴"을 기술한다. "url 패턴"은 ant 형식과 대체문자 또는 "url 패턴"의 "{}"값을 파라메타로 전달 받을 수도 있다.

"{}"의 변수를 "path variable"이라 부른다.

```
@RequestMapping("/hello") 또는 RequestMapping("/admin/**/user")
@RequestMapping("/user/{userid}")
```

- value와 method를 이용한 HTTP 메서드 매핑

value에 "url 패턴", method에 "RequestMethod.메소드명"을 기술하고, 메서드명은 GET 또는 POST를 기술한다.

```
@RequestMapping(value="/add", method=RequestMethod.GET)
@RequestMapping(RequestMethod.POST)
```

- 타입 레벨과 메서드 레벨의 매핑

클래스와 인터페이스 타입 레벨에 붙은 @RequestMepping은 타입내의 모든 매핑용 메소드의 공통 조건을 지정할 때 사용한다. 메서드 레벨의 매핑은 클래스 레벨의 매핑을 상속한다. 컨트롤러가 "/user" 타입

에 각각 "/add", "/edit" 메소드의 url에 매핑되는 경우는 다음과 같다.

```
@RequestMapping("/user")
public class UserController{
    @RequestMapping("/add")
    public String add() { ... } }
```

③ @SessionAttributes("cmd")

@SessionAttributes("cmd")는 클래스 상단에 선언하여 세션으로 커맨드 객체를 저장하는 애노테이션이며 두 가지 기능이 있다.

- 첫째, 컨트롤러 메서드가 생성하는 모델정보 중에서 @SessionAttributes에 지정한 이름과 동일한 것이 있으면 이를 세션에 저장한다. 부가 이 모델을 참조하여 기존 정보를 폼(form)에 보여주는 기능이다.
- 둘째, @ModelAttribute가 지정된 파라미터가 있을 때 이 파라미터에 전달해줄 오브젝트를 세션으로 가져오는 것이다.

3) 스프링 어노테이션

컨트롤러, 비즈니스 로직이나 DAO 설정 등에 사용하는 스프링 어노테이션이다.

종류	설명
@Service	컨트롤러와 데이터베이스를 처리하는 클래스에 비즈니스 로직이나 트랜잭션을 처리하는 클래스를 두게 되는데 이 역할을 담당하는 클래스를 서비스 클래스로 설정한다. @Service 어노테이션을 선언한 클래스는 자동 검색을 통해 빈으로 자동 설정한다. org.springframework.stereotype.Service
@Repository	DAO의 역할을 담당하여 데이터 베이스와 연동해서 데이터 검색이나 입력, 수정하는 클래스를 빈으로 설정하기 위해서 사용한다. org.springframework.stereotype.Repository
@Component	<context:component-scan> 태그로 클래스를 빈으로 자동 설정한다. org.springframework.stereotype.Component
@Autowired	의존 관계 자동 설정. 생성자, 필드, 메서드에 적용 가능. 즉 생성자, 메서드, 필드에 붙여 스프링에서 자동 주입을 명시한다. org.springframework.beans.factory.annotation.Autowired
@Transactional	트랜잭션 처리시 자동으로 트랜잭션을 제어하는 기능을 제공한다. org.springframework.transaction.annotation.Transactional
@Scope	빈의 범위를 싱글톤이 아닌 request, session, prototype 등으로 설정한다. org.springframework.context.annotation.Scope

4) 컨트롤러의 메서드 파라메타 종류

① @RequestParam

단일 HTTP 요청 파라메타를 메소드 파라메타에 전달하는 애노테이션이다.

```
public String login(@RequestParam("id") String id, @RequestParam("pw") String pw) { ... }
```

② @PathVariable

@RequestMapping의 url 중 괄호 {} path 변수를 @PathVariable로 받는다.

```
@RequestMapping("/user/{userid}")  
Public String view(@PathVariable("userid") String userid) { .... return "뷰명"; }
```

③ @ModelAttribute

클라이언트에서 컨트롤러에게 하나 이상의 값이 전달되는 경우가 있다. 하나 이상의 값을 가져오는 옵젝트 형태로 만들 수 있는 구조적인 정보를 @ModelAttribute 모델이라 부르며, 컨트롤러가 전달 받은 오브젝트 형태의 정보가 @ModelAttribute이다.

컨트롤러에서 폼의 다수 데이터를 UserVO 자바빈에 저장하여 insert() 메서드에서 @ModelAttribute로 받는 경우다.

```
@RequestMapping("/add", method="RequestMethod.POST")  
public User insert(@ModelAttribute UserVO userVO) {  
    userService.insert(UserVO());  
    ... }
```

5) 컨트롤러의 반환 타입

컨트롤러의 메서드가 return으로 반환하는 값의 타입을 선언한다. 반환 타입은 ModelAndView, Model, Map, View, String, void 등이 있다.

객체 반환 타입	설명
ModelAndView	뷰와 모델 정보를 모두 포함하고 있는 반환 타입
Model, Map, ModelMap	뷰에 전달할 객체 정보만 포함하고 있는 반환 타입
String	뷰만 반환. JSP 또는 HTML View 등 호출시 사용
View	뷰 객체를 직접 반환. 해당 뷰 객체를 이용해 뷰 생성
void	반환 타입을 기술하고 않고, 기본으로 자동 생성

① ModelAndView

ModelAndView는 컨트롤러가 디스패처 서블릿에 반환해야 하는 뷰와 모델 정보를 모두 포함하고 있는 객체를 반환한다.

- ModelAndView에서 모델과 뷰를 설정하는 메소드는 3가지이다.
 - setViewName(String 뷰명) : 응답 뷰명 설정
 - addObject(String 뷰명, Object name) : 뷰명과 모델 설정
 - addAllObject(Map map) : 뷰에 전달할 값을 Map에 설정

② String

String 반환 타입은 뷰명만 반환한다.

```
Public String 메서드명 { ... return "뷰명" }
```

③ Model, Map, ModelMap

Model, Map, ModelMap 객체는 String 반환 타입으로 선언된 구현 메서드에서 뷰에 데이터만을 전달하는 객체들이다. Map에는 put() 메서드로 설정하고, Model과 ModelMap은 addAttribute() 메서드로 파라미터를 설정 할 수 있다.

```
@RequestMapping("/test")
public ModelMap test(TestVO testVO, ModelMap modelMap) {
    modelMap.addAttribute("name", "model값");
    return modelMap;
}
```

④ void

메서드에서 HTTP 응답을 처리할 때나 뷰명이 RequestToViewNameTranslator에 의해 내부적으로 자동 생성되는 타입이다.

```
@RequestMapping("요청url")
public void 메서드명(@RequestParam String name, ...){
    객체명.메서드명("name", name);
}
```

⑤ View

뷰 객체로 반환하는 반환 타입이다. 엑셀, 파일 다운로드 등에 사용된다.

13. 핸들러 매핑

핸들러 매핑이란 사용자의 요청이 있을 때 디스패처 서블릿은 어떤 컨트롤러에게 위임할 것인가를 결정하게 되는데 그 요청을 처리하는 컨트롤러의 매핑을 담당하는 인터페이스이다. 핸들러 매핑은 클라이언트의 요청을 어떤 컨트롤러가 처리할 것인가에 대한 정보를 디스패처 서블릿에 반환한다.

1) 핸들러 매핑의 종류와 프로퍼티

스프링 설정 파일에서 핸들러 매핑의 BeanNameUrlHandlerMapping과

DefaultAnnotationHandlerMapping은 기본으로 스프링 MVC에 탑재되어 있기 때문에 특별한 경우가 아니면 별도로 설정할 필요는 없다.

핸들러 매핑 클래스	설명
DefaultAnnotationHandlerMapping	@RequestMapping이 적용된 컨트롤러 메서드와 컨트롤러 url에 매핑. 가장 많이 사용.
SimpleUrlHandlerMapping	url과 컨트롤러의 매핑정보를 빈의 프로퍼티에 넣어준다. 기본 핸들러 매핑이 아니기 때문에 프로퍼티에 매핑 정보를 직접 명시적으로 기술해야 SimpleUrlHandlerMapping 빈을 사용할 수

	있다.
BeanNameUrlHandlerMapping	http 요청을 웹 애플리케이션 컨텍스트 파일에 명시된 빈의 이름으로 맵핑.
ControllerBeanNameHandlerMapping	빈의 id나 빈의 이름을 이용하여 맵핑해주는 핸들러 맵핑.
ControllerClassUrlHandlerMapping	클래스명을 url에 맵핑해 주는 클래스이다.
AbsractUrlHandlerMapping	클라이언트의 요청 url로부터 서블릿 컨텍스트 경로를 기준으로 컨트롤러를 맵핑

핸들러 맵핑을 정의할 때 핸들러를 확장하기 위한 프로퍼티

프로퍼티명	설명
interceptors	인터셉터 목록
defaultHandler	해당 핸들러 맵핑이 없을 때 사용하는 기본 핸들러
order	컨텍스트내 사용 가능한 모든 핸들러 맵핑의 순서 정렬
alwaysUserFullPath	true 설정시 완전 경로 사용 유무. 기본값은 false
urlPathHelper	url 조사시 사용. 디폴트 값은 변경하지 말 것
urlDecode	디코드되지 않은 요청 url과 uri를 반환. 기본값은 false
lazyInitHanders	싱글톤 핸들러의 늦은 초기화 허락. 기본값은 false

① DefaultAnnotationHandlerMapping

@Controller를 사용한 컨트롤러는 컴포넌트 자동 검색 대상으로 스프링 설정 파일에서

<context:component-scan base-package="" /> 태그로 컨트롤러가 저장된 패키지명을 선언해야 컨트롤러를 자동으로 로딩할 수 있다.

DefaultAnnotationHandlerMapping 핸들러 맵핑과 핸들러 어댑터를 함께 설정한다.

```
<bean class="org.springframework.~.DefaultAnnotationHandlerMapping">
    <property name="alwayUsePath" value="true" />
</bean>
<bean class="org.springframework.~.AnnotationMethodHandlerAdapater"
p:alwaysUseFullPath="true">
</bean>
```

```
@Controller
public class 컨트롤러명 { ... }
```

```
<!-- Controller -->
<context:component-scan base-package="패키지명" />
```

② SimpleUrlHandlerMapping

url과 컨트롤러의 맵핑 정보를 빈의 프로퍼티에 넣어준다. 기본 핸들러 맵핑이 아니기 때문에 프로퍼티에

매핑 정보를 직접 명시적으로 기술해야 한다.

③ BeanNameUrlHandlerMapping

HTTP 요청을 웹 애플리케이션 컨텍스트 파일에 명시된 빈의 이름으로 매핑한다. 여러 개의 핸들러 매핑을 사용하는 경우 “order” 프로퍼티로 순서를 명시하는 것이 좋다.

④ ControllerBeanNameHandlerMapping

빈의 아이디나 빈의 이름을 이용하여 매핑해주는 핸들러 매핑이다.

⑤ ControllerClassUrlHandlerMapping

컨트롤러 클래스명을 url에 매핑해주는 클래스이다. 기본적으로 컨트롤러 클래스명을 모두 url로 사용하지만 “Controller” 문자열로 끝날 때는 “Controller”를 제외한 나머지 이름의 첫 글자를 소문자로 변경하여 url에 매핑해 준다.

⑥ AbstractUrlHandlerMapping

클라이언트의 요청 url로부터 서블릿 컨텍스트 경로를 기준으로 컨트롤러 매핑하며 “alwaysUseFullPath” 프로퍼티의 값에 따라서 최종 경로가 결정된다.

14. 뷰리졸버(ViewResolver)와 뷰

스프링 MVC 프레임워크는 뷰를 결정하는 방법이 제공된다. 스프링은 특정 뷰 기술을 위해 특별하게 선언할 필요없이 웹 브라우저에서 모델을 표현할 수 있도록 만들어주는 특별한 두개의 인터페이스 뷰리졸버와 뷰를 제공한다 뷰리졸버는 뷰명과 실제 뷰간의 매핑을 제공하고, 뷰 인터페이스는 준비된 요청을 할당하고 요청을 뷰의 하나에게 처리하도록 넘겨버린다.

1) 뷰리졸버

스프링 MVC 컨트롤러의 모든 핸들러 메서드는 명시적 또는 암시적으로 논리적 뷰명이 결정되어야만 한다. String, ModelAndView 반환 타입과 같은 명시적인 방법과 규칙에 기반 할 수도 있다. 뷰들은 논리적 뷰명에 의해 할당되고, 뷰리졸버에 의해서 결정된다.

뷰리졸버	설명
UrlBasedViewResolver	매핑없이 논리적 뷰명을 url로 처리하는 뷰리졸버
InternalResourceViewResolver	서블릿/JSP 뷰와 JstView 등과 같은 하위 클래스를 지원하는 UrlBasedViewResolver의 하위 클래스
XmlViewResolver	Xml로 작성한 설정 파일을 받는 뷰리졸버. 기본 설정 파일은 /WEB-INF/view.xml

뷰리졸버에는 컨트롤러가 반환하는 뷰로 JSP를 사용할 때 UrlBasedViewResolver를 사용할 수 있다. 뷰리졸버는 뷰명을 url로 해석하고 뷰를 표현하기 위한 요청을 RequestDispatcher로 보낸다. 빈의 접두어가 “/WEB-INF/jsp/”로 설정하고, 접미어가 “.jsp”로 설정하는 뷰리졸버이다.

그래서 만약 “test” 뷰명이 반환될 경우 뷰리졸버는 “/WEB-INF/jsp/test.jsp”로 요청 정보를

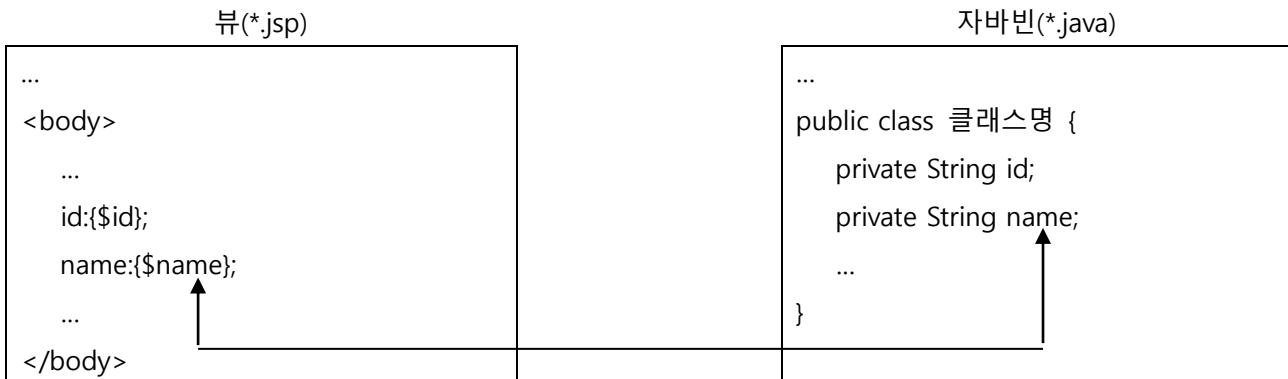
RequestDispatcher로 보내게 된다.

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

2) 뷰와 뷰 작성

뷰란 사용자가 요청 또는 요청 결과를 웹 브라우저에 출력하는 것으로, JSP와 JSTL 등을 이용하여 구현할 수 있다. 뷰 작성할 때 주의 할 점은 화면의 입력 데이터나 응답하기 위한 값들이 자바빈을 통하여 컨트롤러와 뷰에 전송되거나 전달된다. 그래서 뷰에 기술하는 필드명들은 반드시 자바빈의 프로퍼티명과 동일하게 코딩되어야 한다.

또한 스프링 MVC 프레임워크에서는 클라이언트에서 직접 접근할 수 없도록 뷰파일의 저장 위치를 "/WEB-INF" 폴더 내에 위치시키는 것을 강력하게 권장하고 있다.



3) JSP와 JSTL로 작성한 뷰 파일은 "/WEB-INF" 하위 폴더에 저장하고 스프링 설정파일에 InternalResourceViewResolver로 폴더의 위치를 설정한다.

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

4) 스프링 폼 태그

"spring-webmvc-*jar" 파일의 스프링 폼 태그 라이브러리를 사용할 수 있다. 스프링 폼 태그를 사용할 경우에는 JSP 페이지에 taglib를 추가해야 한다.

```
<%@ taglib prefix="form" uri=http://www.springframework.org/tags/form %>
```

스프링 폼 태그는 <form:>로 시작하며 내부 속성을 사용할 수 있다.

스프링 폼 태그	스프링 태그 내부 속성
<form: form>	id, method, action, ModelAttribute, commandName
<form: input>	id, size, path, value
<form: password>	id, path, value
<form: radiobutton>	path, value, label
<form: radioButtons>	path, items, itemValue, itemLabel
<form: checkbox>	path, value, label
<form: checkboxes>	path, items, itemValue, itemLabel
<form: select>	id, path, items
<form: option>	value, label
<form: options>	items, itemValue, itemLabel
<form: textarea>	path, rows, cols
<form: hidden>	path
<form: label>	path, delimiter, cssErrorClass
<form: errors>	path, cssClass

15. 스프링 웹 MVC 개발 환경 구축(JDK 1.7, Tomcat 8)

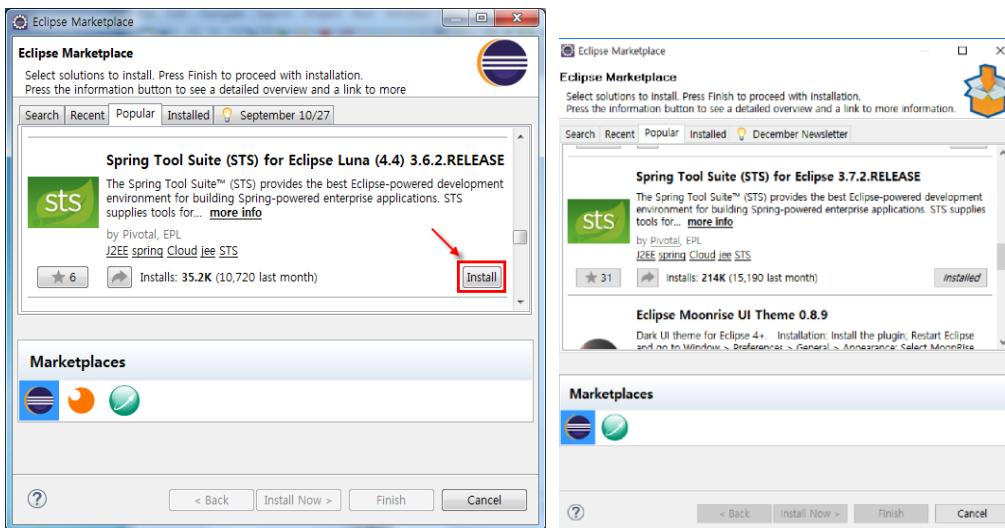
스프링 프레임워크를 이용한 프로그래밍 개발 환경 방법은 매우 중요하다. 스프링 웹 MVC 프로그래밍을 위한 개발 환경 구축은 2가지 방법이 있다.

- 스프링에서 제공하는 이클립스 기반의 스프링 개발 환경에 최적화 되어 있는 개발 툴인 STS(Spring Tool Suite)를 "http://spring.io/tools/sts/all" 사이트에서 다운로드하여 설치한다.
- 이클립스 Luna 개발 도구에 스프링 프레임워크 개발을 위한 스프링 관련 플러그인 STS(Spring Tool Suite)를 추가로 설치한다.

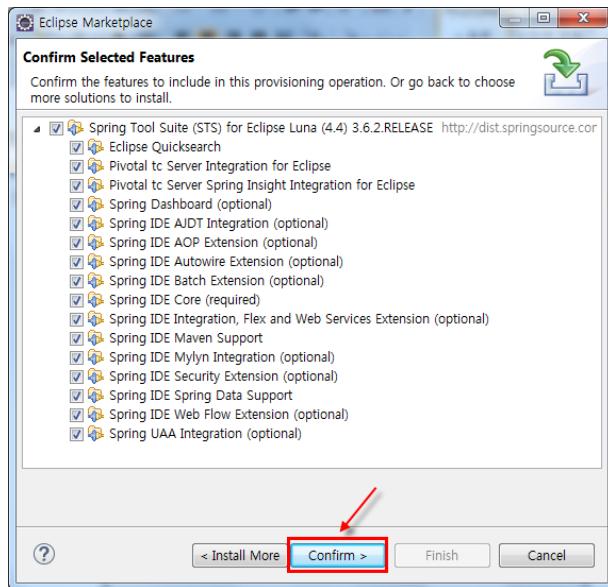
1) 이클립스 Luna에 스프링을 위한 STS 플러그인(3.7.2) 설치

Luna 개발 도구에 STS 플러그인을 설치하는 방법은 다음과 같다.

- Luna 실행 화면에서 [HELP] - [Eclipse Marketplace] - [Popular] 탭 메뉴를 차례로 선택한다.

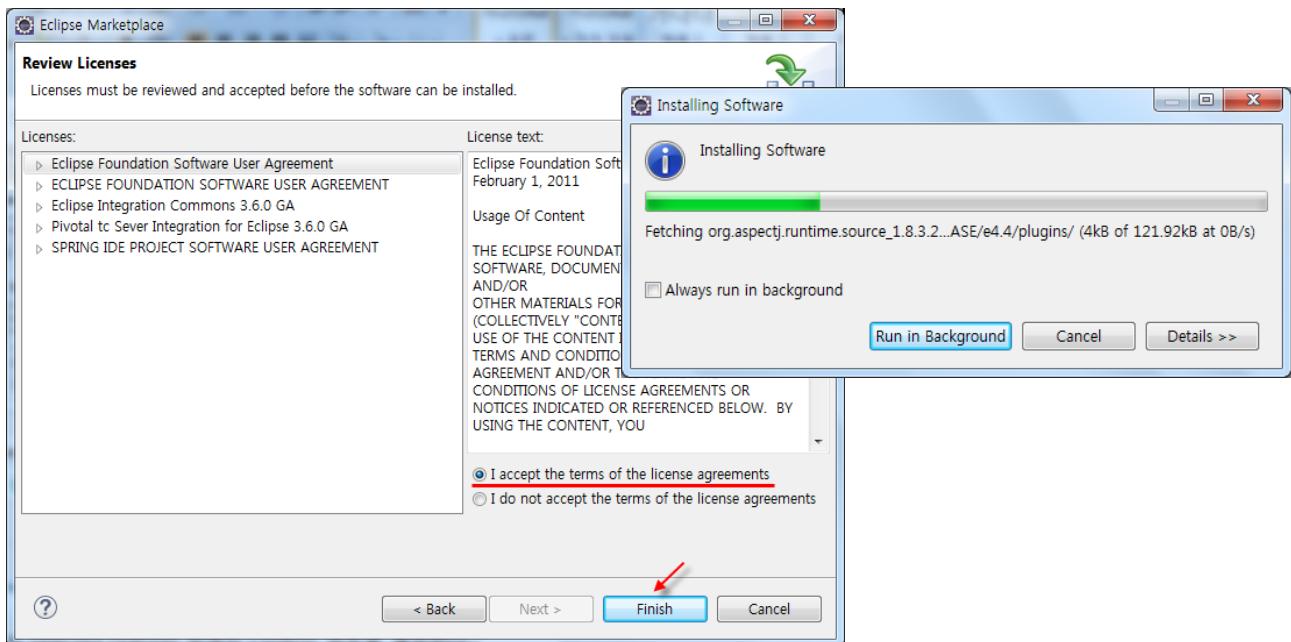


② "Spring Tool Suite (STS) for Eclipse Luna (4.4) 3.6.2.RELEASE"를 찾아 [Install] 버튼을 클릭한다.

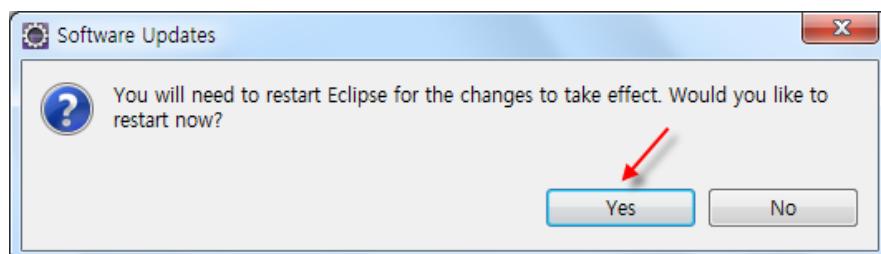


③ Confirm Selected Features 창에서 Confirm 버튼을 클릭한다.

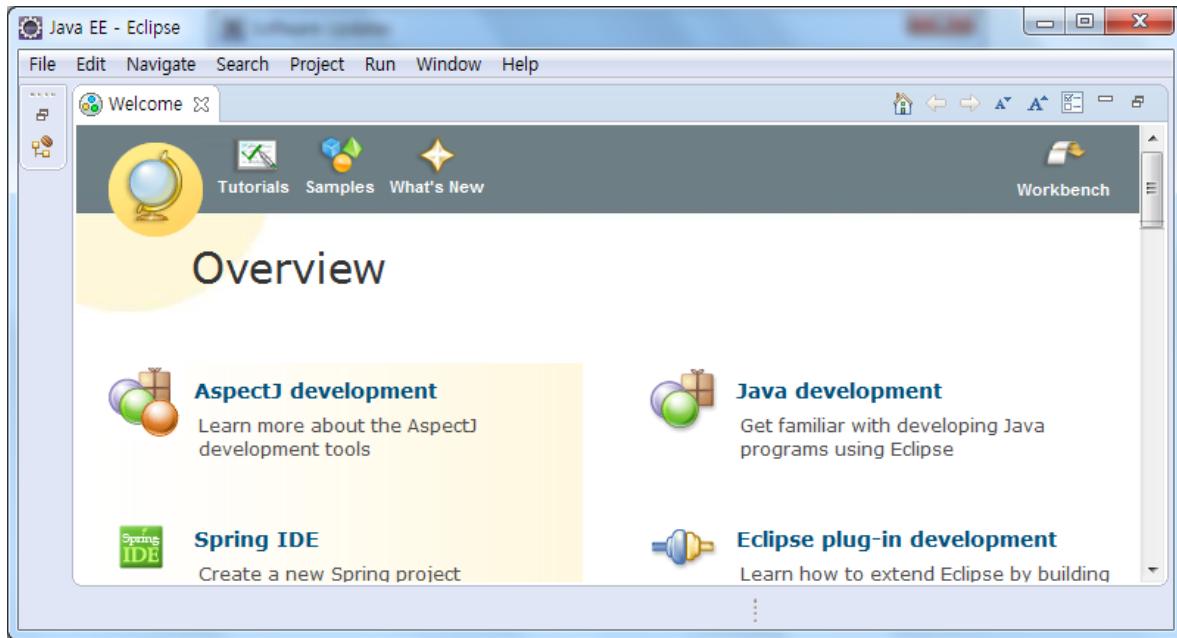
④ Review Licences 화면에서 "동의" 버튼을 클릭하고, [Finish] 버튼을 클릭하면 설치 시작한다. 설치하는 시간이 좀 소요될 수 있으므로 혹 설치가 중단되면 ①부터 다시 시도한다.



⑤ 설치가 완료되면 이클립스 재시작 선택 화면이 나타난다. [Yes] 버튼을 클릭하면 이클립스를 재시작한다.



- ⑥ Luna를 시작하여 “Welcome” 초기 화면이 나타나면 설치가 끝난다.



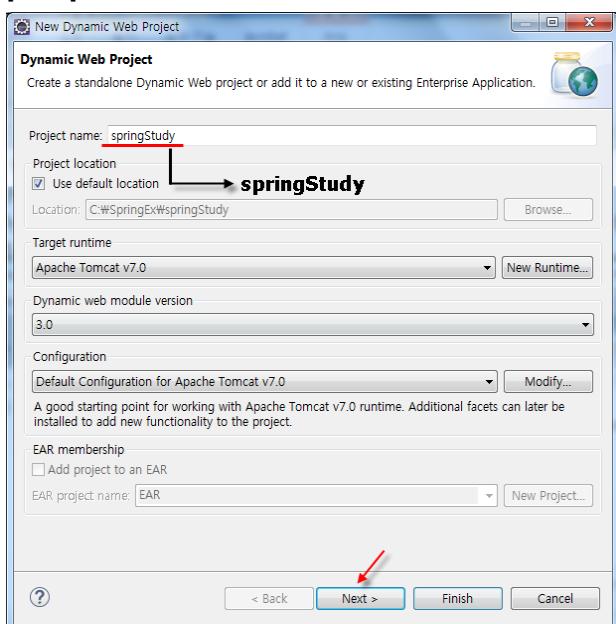
16. 스프링 웹 MVC 예제

스프링 웹 MVC 프로그래밍을 위한 프로젝트와 패키지 생성, 컨트롤러 입력, 폴더 생성과 뷰 입력, 웹 애플리케이션 설정 파일과 스프링 설정 파일을 설정하고 실행하는 과정은 다음과 같다.

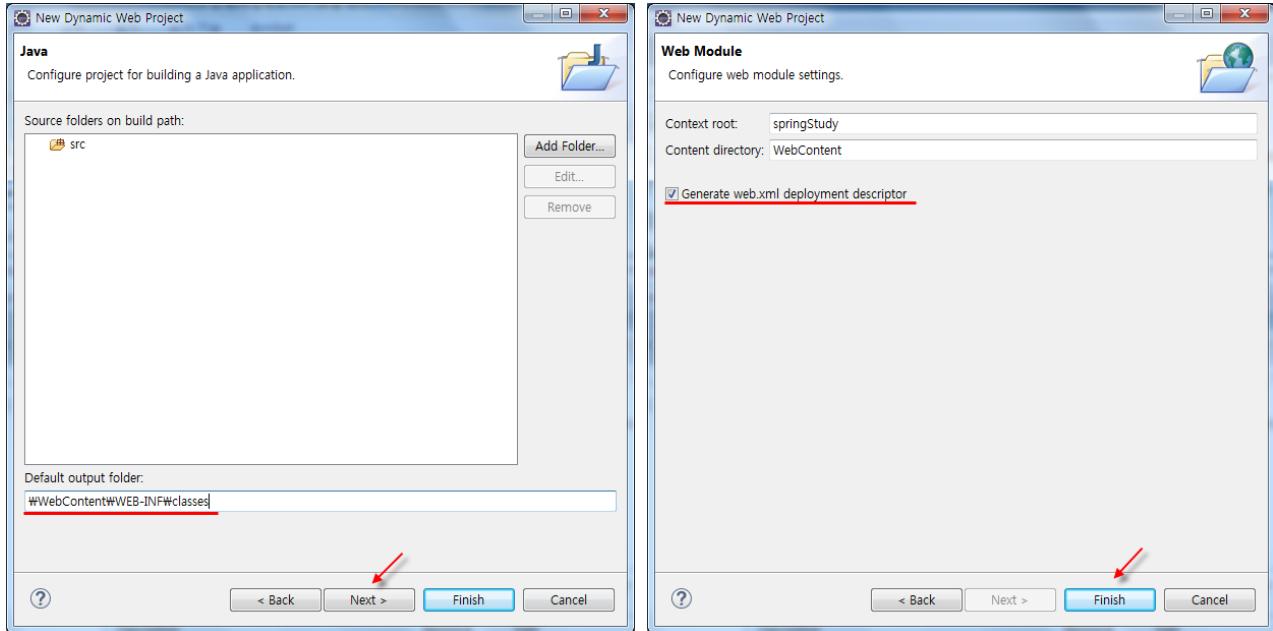
16.1 스프링 웹 MVC를 위한 프로젝트 생성

Server 설정한 후에 스프링 웹 MVC 프로그래밍은 Dynamic Web Project를 생성하면 된다.

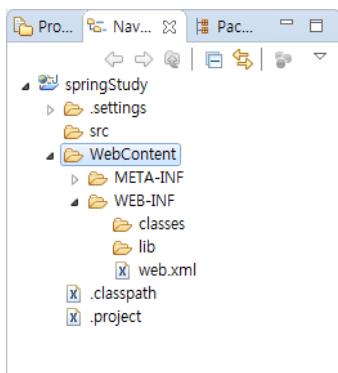
- ① [File] - [New] - [Dynamic Web Project] 메뉴를 선택한다.
- ② “New Dynamic Web Project” 생성 화면이 나타나면 “Project Name”명을 “springStudy”로 입력하고, [Next] 버튼을 클릭한다.



- ③ "New Dynamic Web Project" 생성 화면에서 Java 소스 파일을 경로는 "src"로, Default output folder를 "build\classes" 또는 "WebContent\WEB-INF\classes"로 변경하고 [Next] 버튼을 클릭한다.
- ④ "New Dynamic Web Project" 생성 화면에서 "Generate web.xml deployment descriptor" 체크 박스를 클릭하고, [Finish] 버튼을 클릭한다.

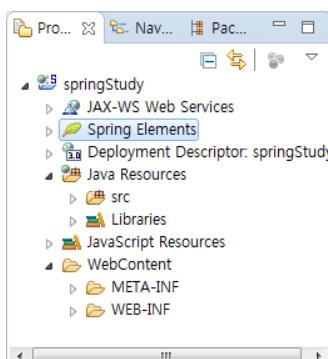


- ⑤ 다음 그림과 같이 "springStudy" 프로젝트가 생성된다.

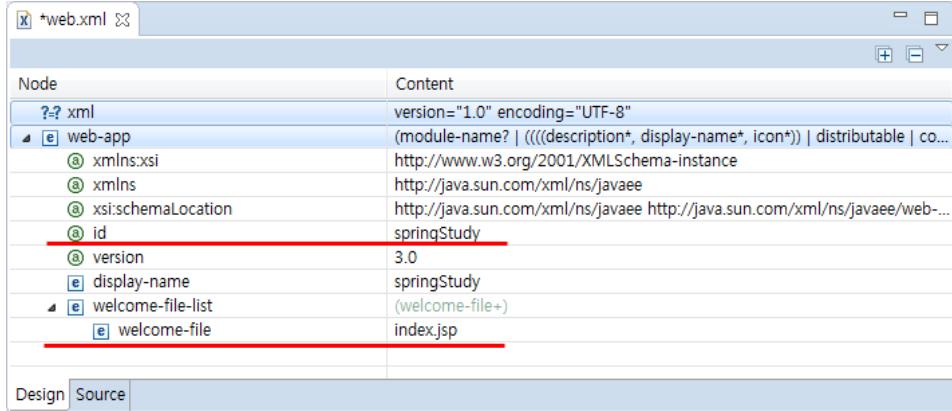


- ⑥ "springStudy" Dynamic Web Project에 스프링 프레임워크를 적용시키기 위해서 "springStudy" 프로젝트를 선택하고, 마우스 오른쪽 버튼을 클릭하고, 팝업창에서 [spring Tools] - [Add Spring Project Nature] 메뉴를 차례로 클릭한다.

- ⑦ "springStudy" 프로젝트명에 "s" 표시가 나타나며, 기본 폴더가 생성된다.



- ⑧ "WEB-INF/" 폴더의 "web.xml" 파일을 열고, "Design" 탭을 클릭하여 "id" 속성의 "Web-App_ID"를 "springStudy"로 수정하고, "Welcome-file-list"의 "index.jsp"를 제외한 나머지 노드를 삭제하여 저장한다.



- ⑨ "springStudy" 스프링 프로젝트에서 스프링 프레임워크의 기능들을 사용하기 위해서는 "WEB-INF/lib" 폴더에 스프링 프레임워크의 jar 파일과 프로그래밍에 관련된 jar 파일들을 다운로드 받는다.

- ⑩ "springStudy"의 "WEB-INF/lib" 폴더에 스프링 프레임워크의 jar 파일과 프로그래밍에 관련된 jar 파일들을 임포트하거나 복사한다.

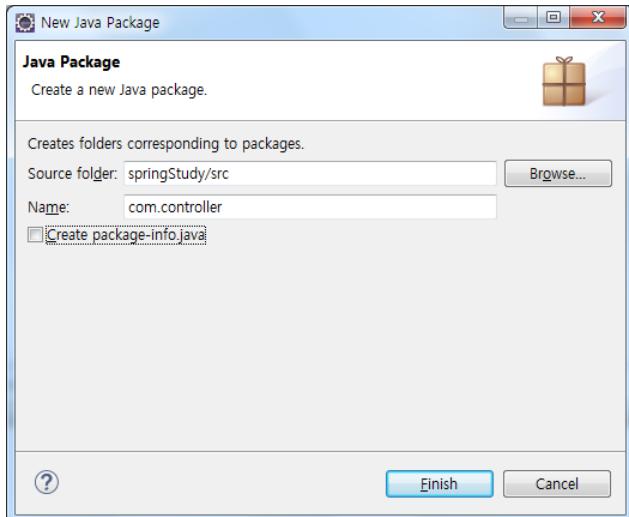
- ⑪ "WEB-INF/lib" 폴더를 클릭하여 jar 파일이 나타나는지 확인한다. jar 파일을 복사한 경우 [Refresh] 버튼을 클릭한다.

16.2 예제 소스 코드의 입력과 실행

"springStudy" 프로젝트에서 "HelloWorld 실행" 링크를 클릭하면 웹 브라우저에 "Hello World Spring!!!"을 출력하는 간단한 스프링 웹 MVC 프로그램을 입력하여 실행한다.

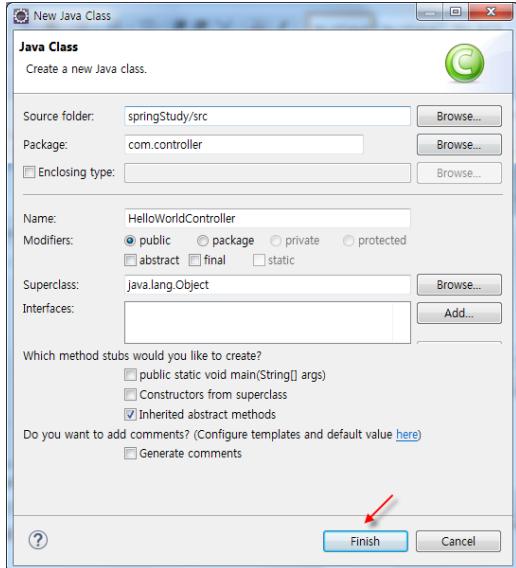
1) 패키지 생성

- ① src 하위 폴더에 "com.controller" 패키지를 생성한다.



2) HelloWorldController.java 소스 코드 입력

- ① "com.controller" 패키지명을 선택하고, 마우스 오른쪽 버튼을 클릭하여 [New] - [Class] 메뉴를 선택하여 Name에 "HelloWorldController" 클래스를 생성한다.



- ② 클래스명 위에 "@Controller"를 입력한다. @Controller 어노테이션을 입력하면, "import org.springframework.stereotype.Controller;"가 추가된다.

※ 참고 : @Con을 입력한 후 "Ctrl + Space"키를 클릭하면 자동완성 키로 추가할 수 있다.

```

1 package com.controller;
2
3 import org.springframework.stereotype.Controller;
4
5 @Controller
6 public class HelloWorldController {
7
8 }

```

- ③ 클래스 내에 @RequestMapping 어노테이션 입력하면, "import org.springframework.web.bind.annotation.RequestMapping;"이 자동으로 추가된다. 그리고 컨트롤러의 본문을 입력하고, 저장한다.

```

3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.RequestMapping;
6
7 @Controller
8 public class HelloWorldController {
9     @RequestMapping("helloWorld")
10    public String helloWorld(Model model) {
11        model.addAttribute("message", "Hello World Spring!!!");
12        return "helloWorld";
13    }
14 }

```

3) 웹 애플리케이션 환경 설정 파일에서 디스패처 서블릿 설정

웹 애플리케이션 설정 파일인 “web.xml” 파일은 WEB-INF 폴더에 생성되어 있다.

WEB-INF/web.xml 웹 애플리케이션 환경 설정 파일에 추가

```
<display-name>springStudy</display-name>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

4) 스프링 설정 파일 생성

“web.xml”的 <servlet-name>에 “spring”로 설정하면 스프링 설정 파일명은 반드시 “spring-servlet.xml” 파일명으로 생성하고 “WEB-INF/” 폴더에 저장한다. 그럼 “web.xml” 문서에 contextConfigLocation 대해 명시할 필요가 없다.

WEB-INF/web.xml 웹 애플리케이션 환경 설정 파일에 추가

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

WEB-INF/spring-servlet.xml 스프링 설정 파일

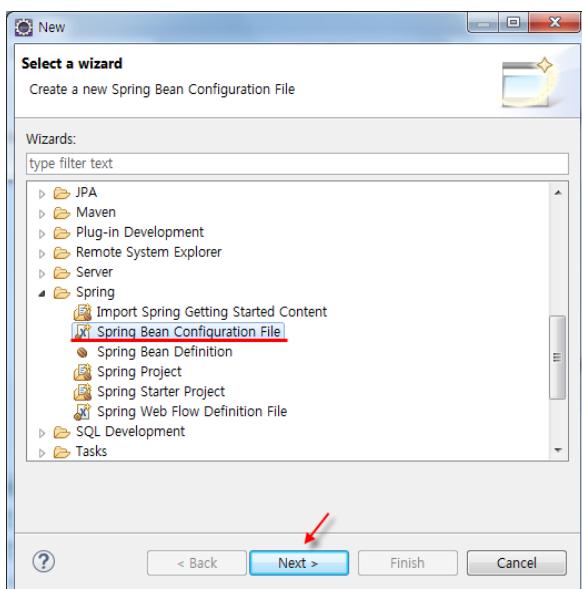
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:c="http://www.springframework.org/schema/c"
       xmlns:context="http://www.springframework.org/schema/context"
```

```

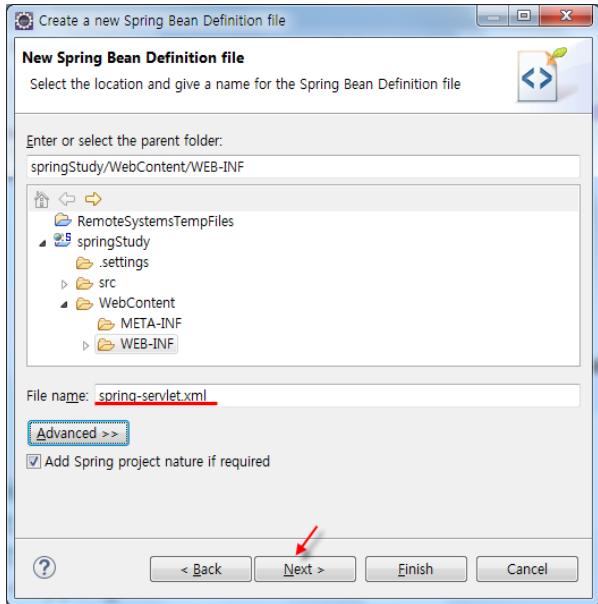
xmlns:lang="http://www.springframework.org/schema/lang"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/spring-lang.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd">
<!-- 어노테이션 설정 -->
<mvc:annotation-driven />
<!-- Controller -->
<context:component-scan base-package="com.controller" />
<!-- ViewResolver -->
<bean class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView">
    </property>
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
</bean>
</beans>

```

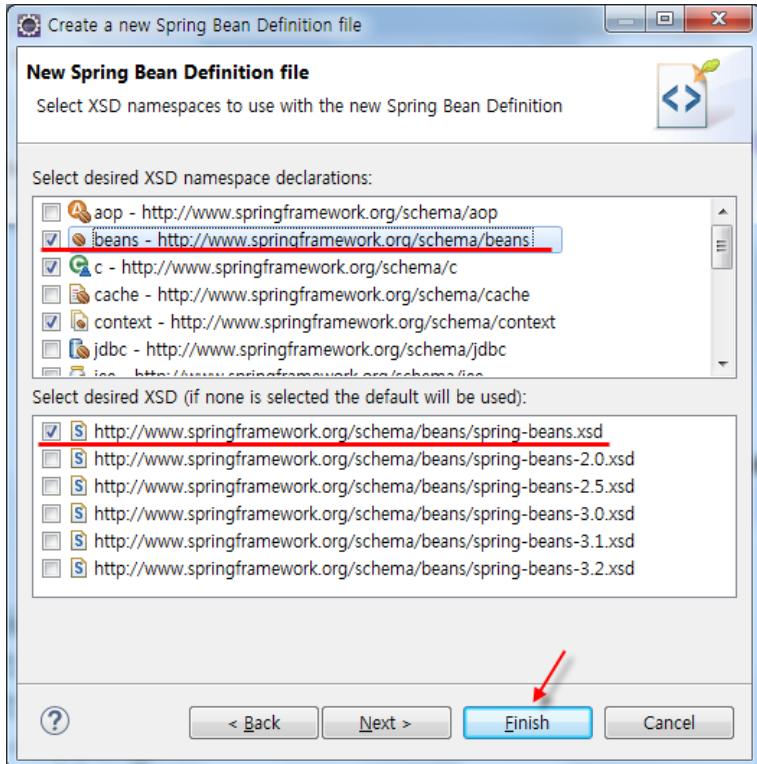
- ① “WEB-INF” 폴더에 [New] - [Other] - [Spring Bean Configuration File]을 차례로 선택한다.



② 파일명으로 “spring-servlet.xml”을 입력하고, [Next] 버튼을 클릭한다.



③ 스프링 설정 정의 파일을 상단 화면에서 XSD namespace를 선택하고, 하단 화면에서 버전의 체크 버튼을 클릭한다. 여러 개의 정의 파일을 선택할 수 있다. “beans”, “c”, “context”, “p”, “mvc”를 선택한 후 [Finish] 버튼을 클릭한다.

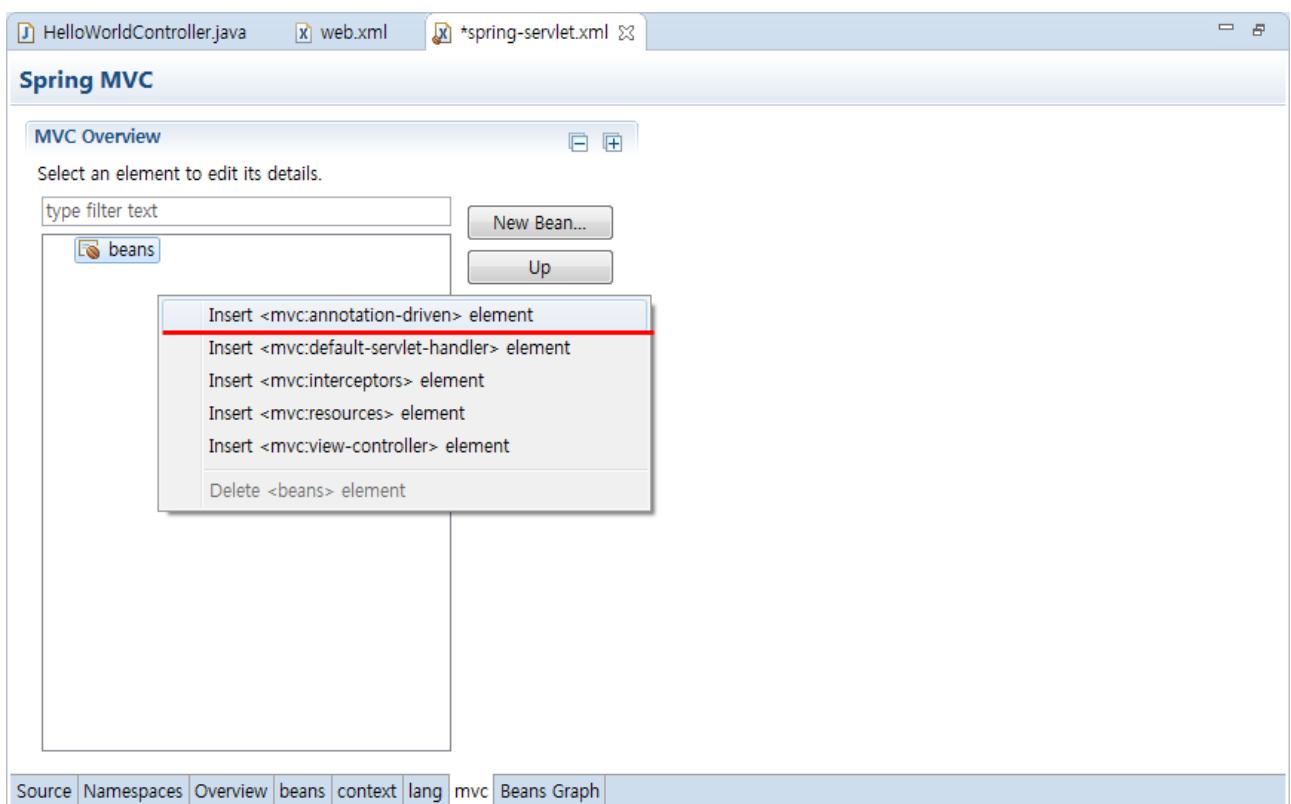


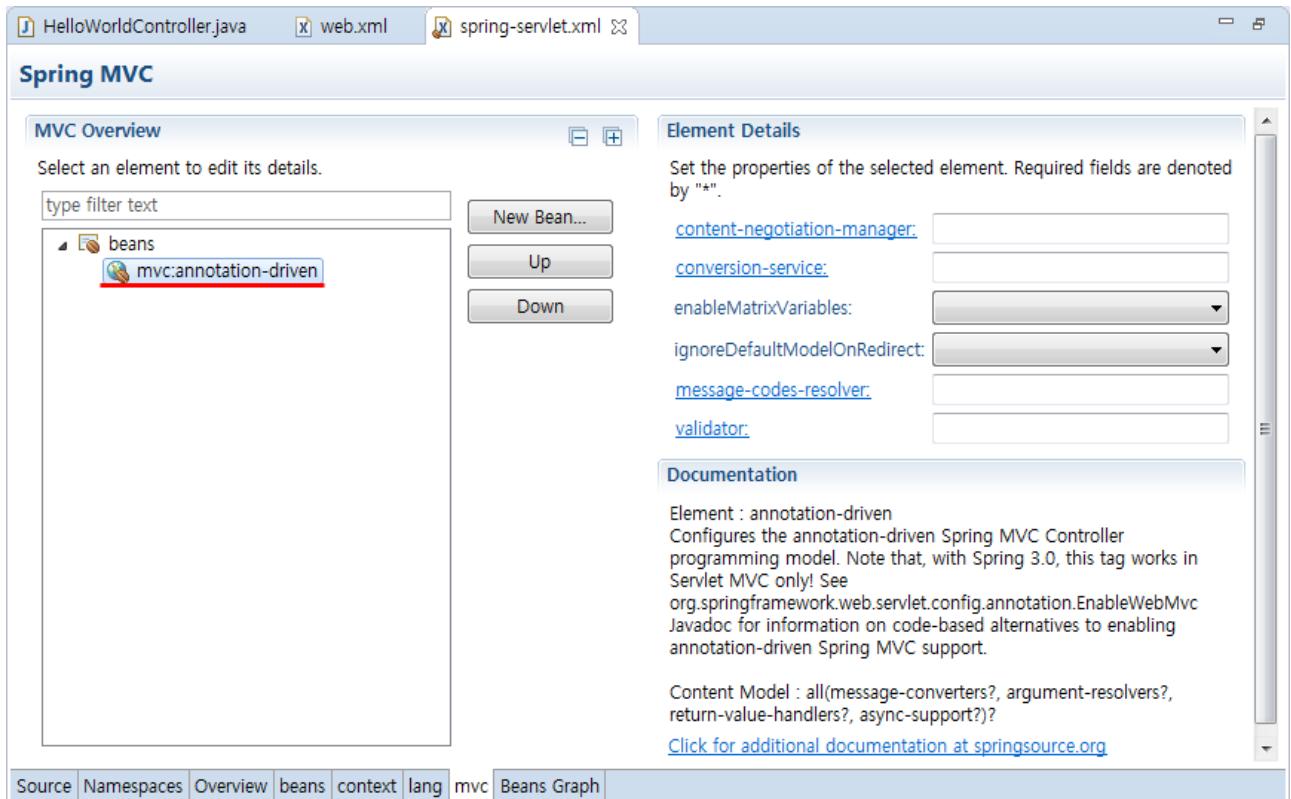
④ 스프링 설정 파일(spring-servlet.xml)의 기본 문서가 나타난다.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:c="http://www.springframework.org/schema/c"
5   xmlns:context="http://www.springframework.org/schema/context"
6   xmlns:lang="http://www.springframework.org/schema/Lang"
7   xmlns:mvc="http://www.springframework.org/schema/mvc"
8   xmlns:p="http://www.springframework.org/schema/p"
9   xsi:schemaLocation="http://www.springframework.org/schema/beans
10    http://www.springframework.org/schema/context http://www.sprin
11    http://www.springframework.org/schema/Lang http://www.sprin
12    http://www.springframework.org/schema/mvc http://www.sprin
13
14 |
15 </beans>
16
```

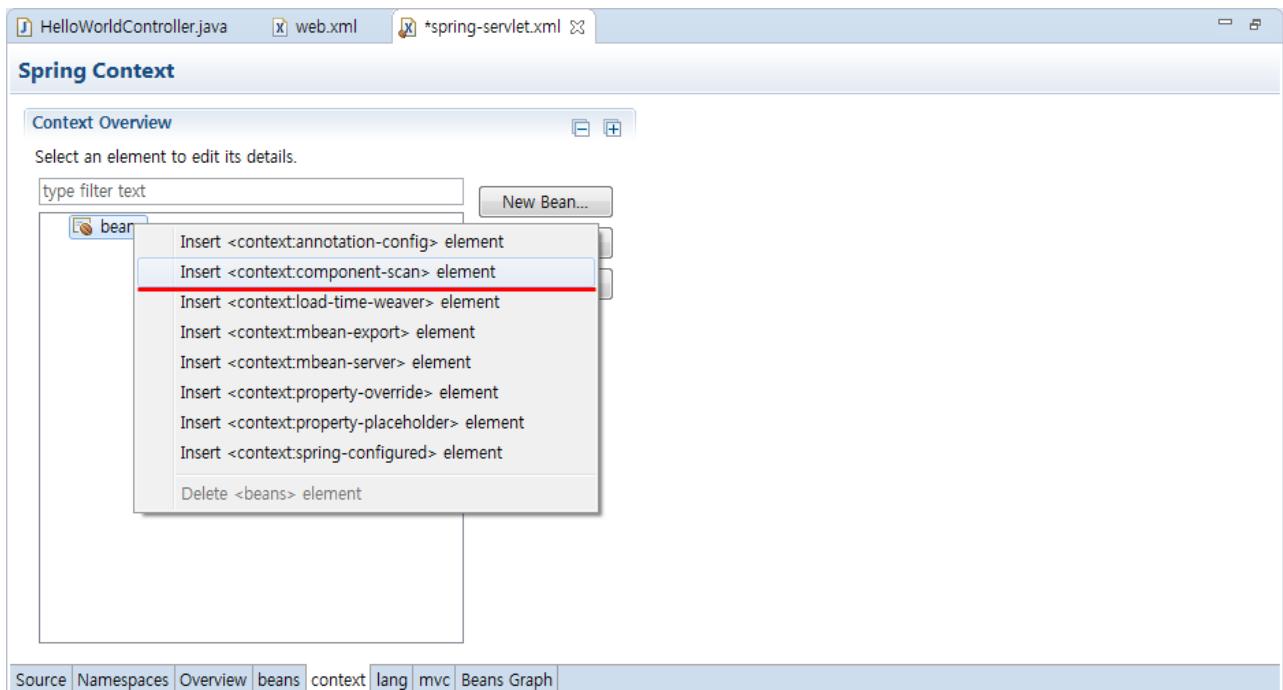
Source Namespaces Overview beans context lang mvc Beans Graph

- ⑤ 화면 하단에 [mvc] 탭을 클릭한 후 “beans”에 마우스를 대고 오른쪽 버튼을 클릭하여 “팝업창”에서 “Insert <mvc:annotation-driven> element”를 클릭한다.

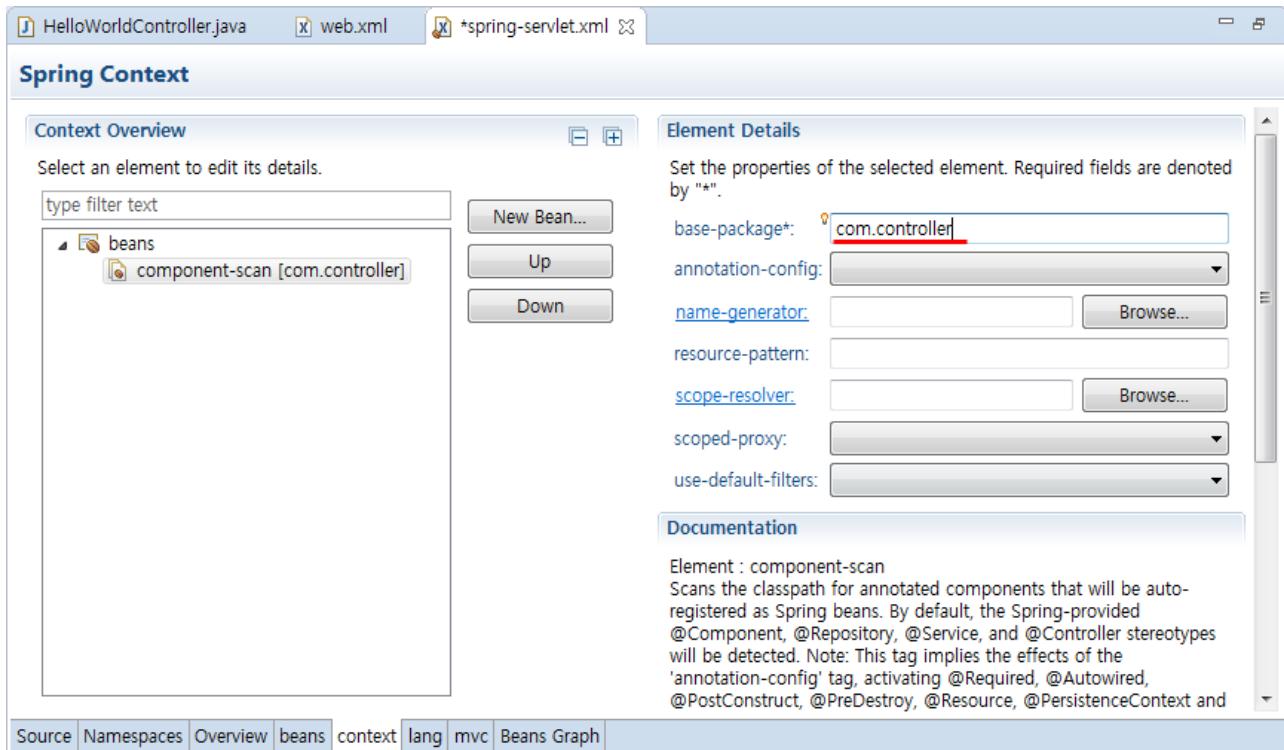




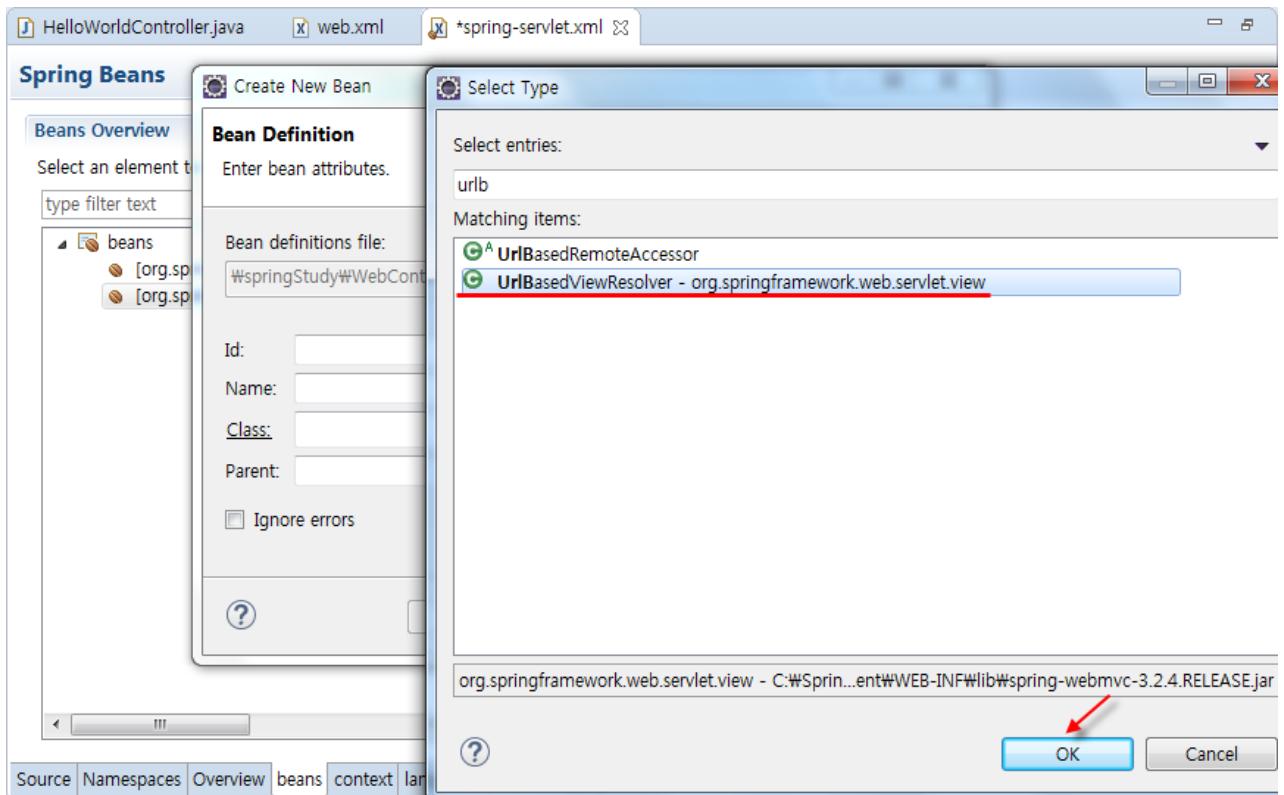
⑥ 컨트롤러의 기본 패키지를 설정하기 위해서 화면 하단에 [Context] 탭을 클릭한다. "beans"에 마우스를 대고 오른쪽 버튼을 클릭하여 "팝업창"에서 "Insert <context:component-scan> element"를 클릭한다.



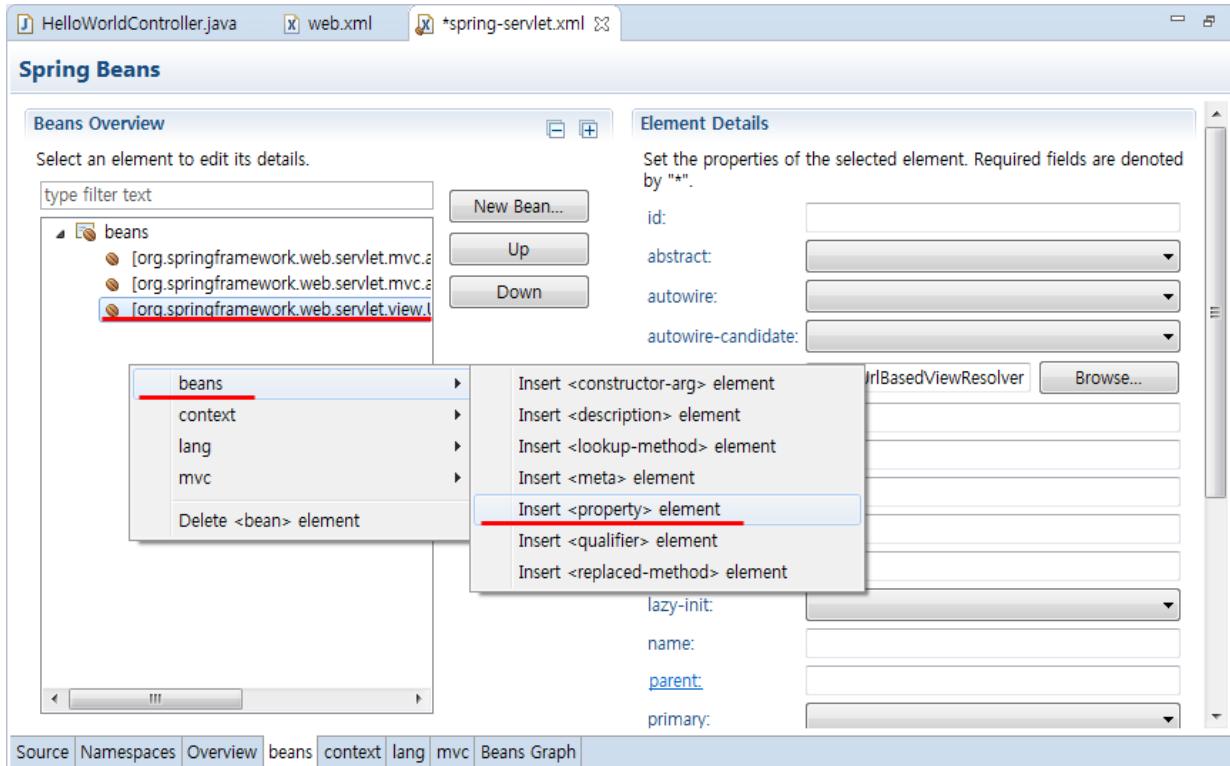
⑦ 오른쪽 화면의 "base-package"에 "com.controller"를 입력한다.



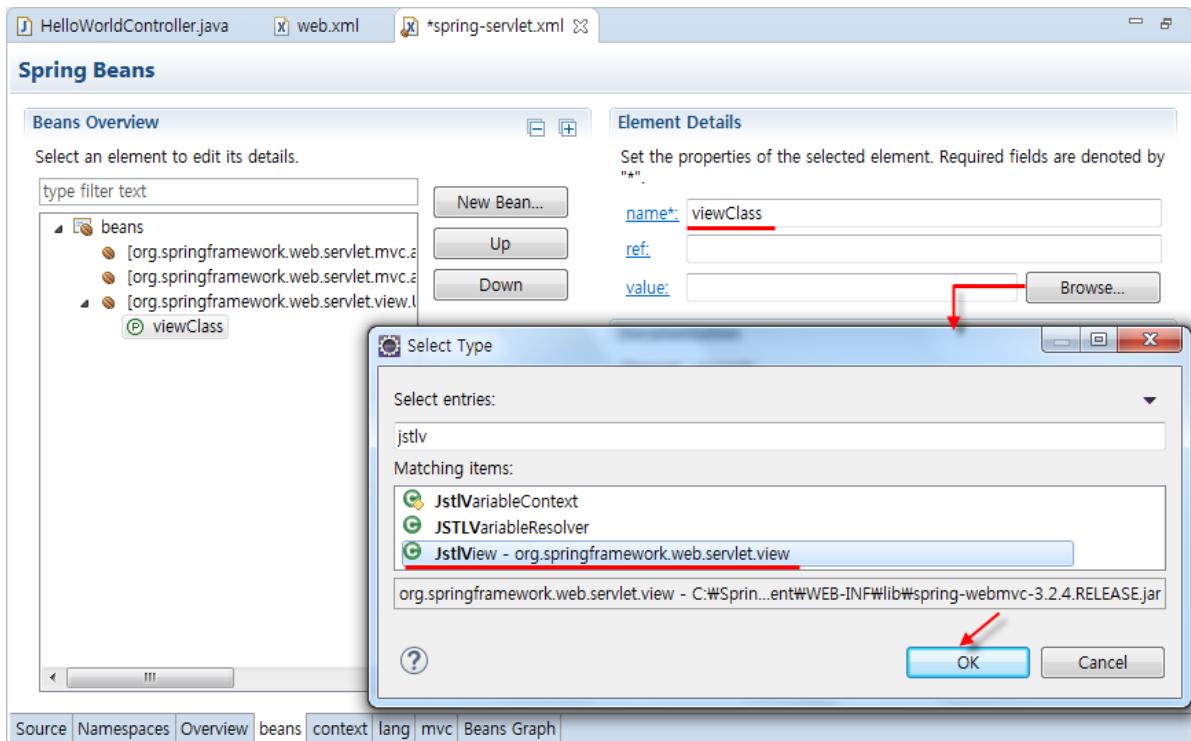
⑧ 뷰리졸버를 입력하기 위해서 [Beans] 탭을 클릭하고, "Class"의 [Browse] 버튼을 클릭하여 "UrlBasedViewResolver"를 입력하여 매칭 아이템을 선택하고 [OK] 버튼을 클릭한다.



⑨ “org.springframework.web.servlet.view.UrlBasedViewResolver”를 선택하고 마우스 오른쪽 버튼의 팝업 창에서 “beans” → “Insert <property> element” 메뉴를 차례로 선택한다.



⑩ “name” 필드에 “viewClass”를 입력하고 “value”에 [Browse] 버튼을 클릭하여 “JstlView”를 검색하여 선택한다. 동일한 방법으로 “beans” → “Insert <property> element” 메뉴를 선택하여 “prefix”와 “/WEB-INF/views/”, “suffix”와 “.jsp”를 입력한다.



Spring Beans

Beans Overview

Select an element to edit its details.

type filter text

- beans
 - [org.springframework.web.servlet.mvc.a
 - [org.springframework.web.servlet.mvc.a
 - [org.springframework.web.servlet.view.l
 - (P) viewClass "org.springframework.we**

New Bean... Up Down

Element Details

Set the properties of the selected element. Required fields are denoted by "*".

name*: viewClass

ref:

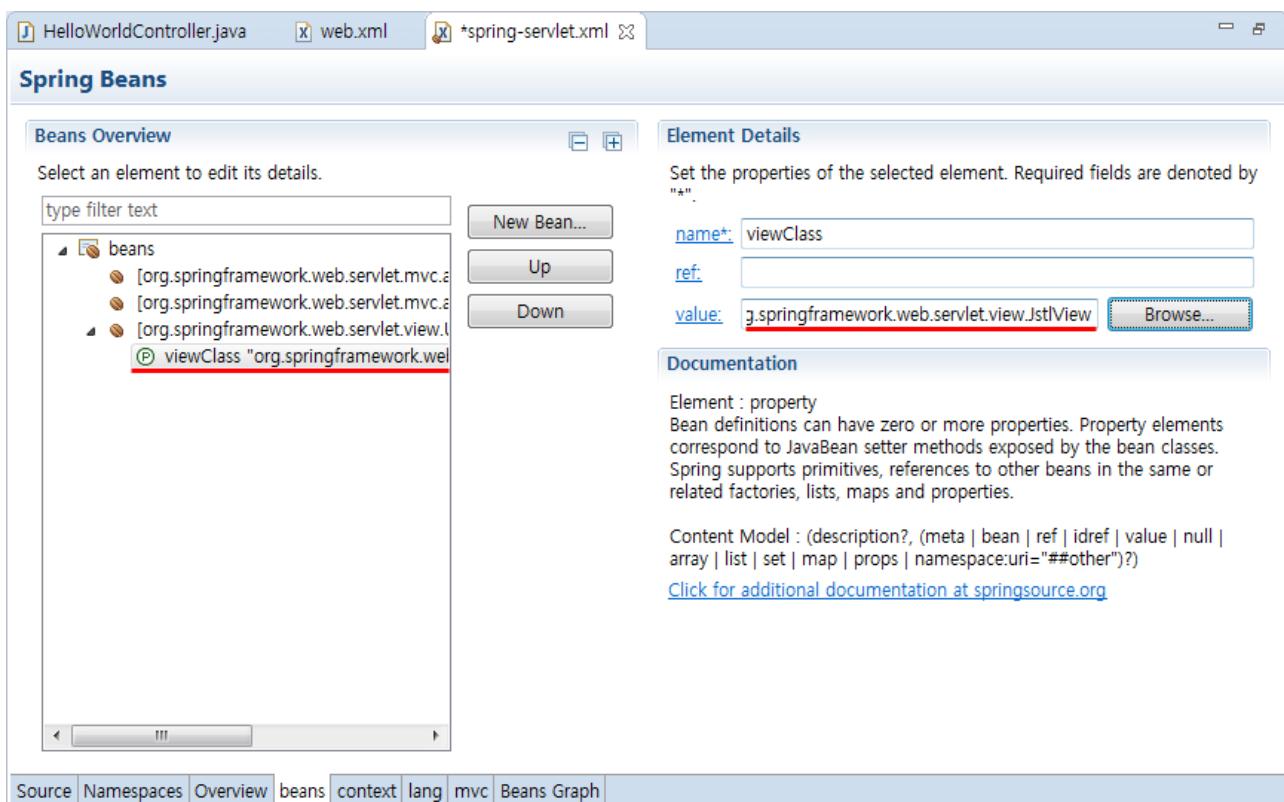
value: `g.springframework.web.servlet.view.JstlView`

Documentation

Element : property
Bean definitions can have zero or more properties. Property elements correspond to JavaBean setter methods exposed by the bean classes. Spring supports primitives, references to other beans in the same or related factories, lists, maps and properties.

Content Model : (description?, (meta | bean | ref | idref | value | null | array | list | set | map | props | namespace:uri="##other")?)
[Click for additional documentation at springsource.org](#)

Source Namespaces Overview beans context lang mvc Beans Graph



Spring Beans

Beans Overview

Select an element to edit its details.

type filter text

- beans
 - [org.springframework.web.servlet.mvc.a
 - [org.springframework.web.servlet.mvc.a
 - [org.springframework.web.servlet.view.l
 - (P) viewClass "org.springframework.we**
 - (P) prefix "/WEB-INF/views/"**
 - (P) suffix ".jsp"**

New Bean... Up Down

Element Details

Set the properties of the selected element. Required fields are denoted by "*".

name*: suffix

ref:

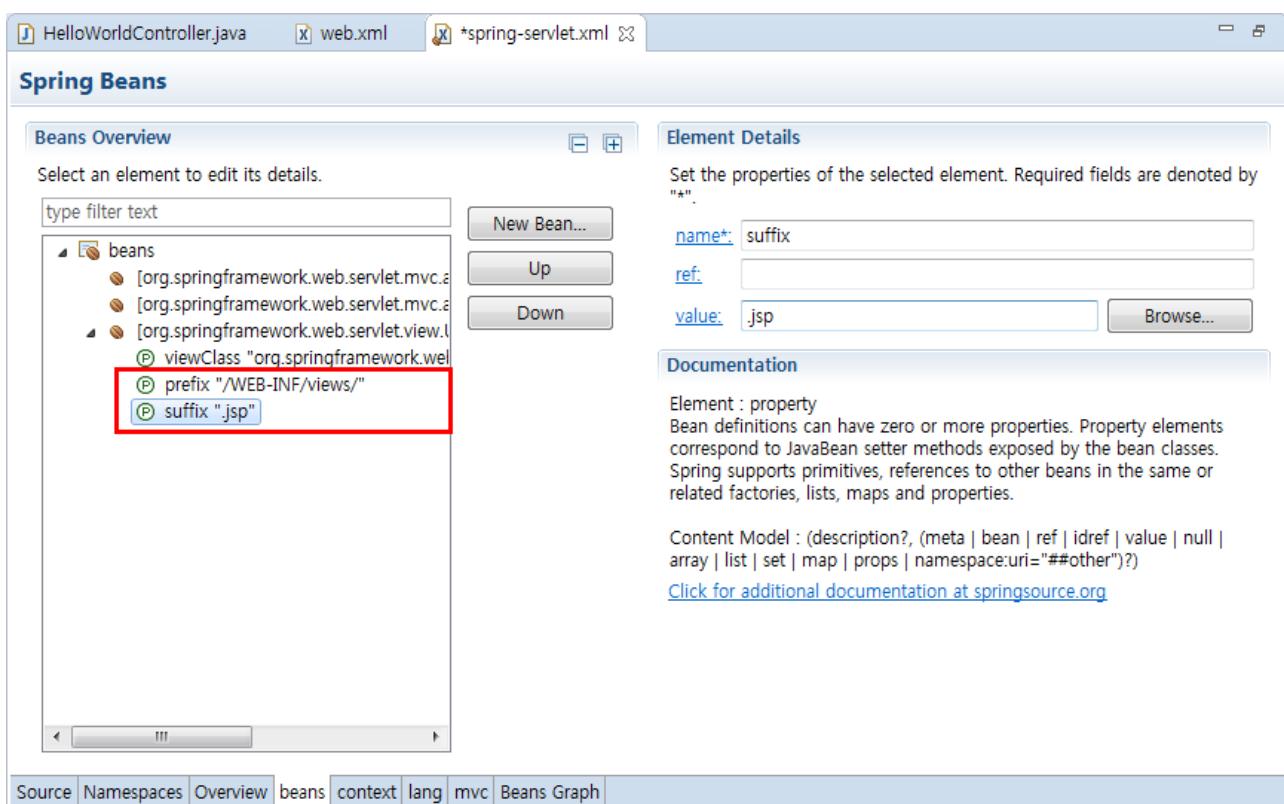
value: `.jsp`

Documentation

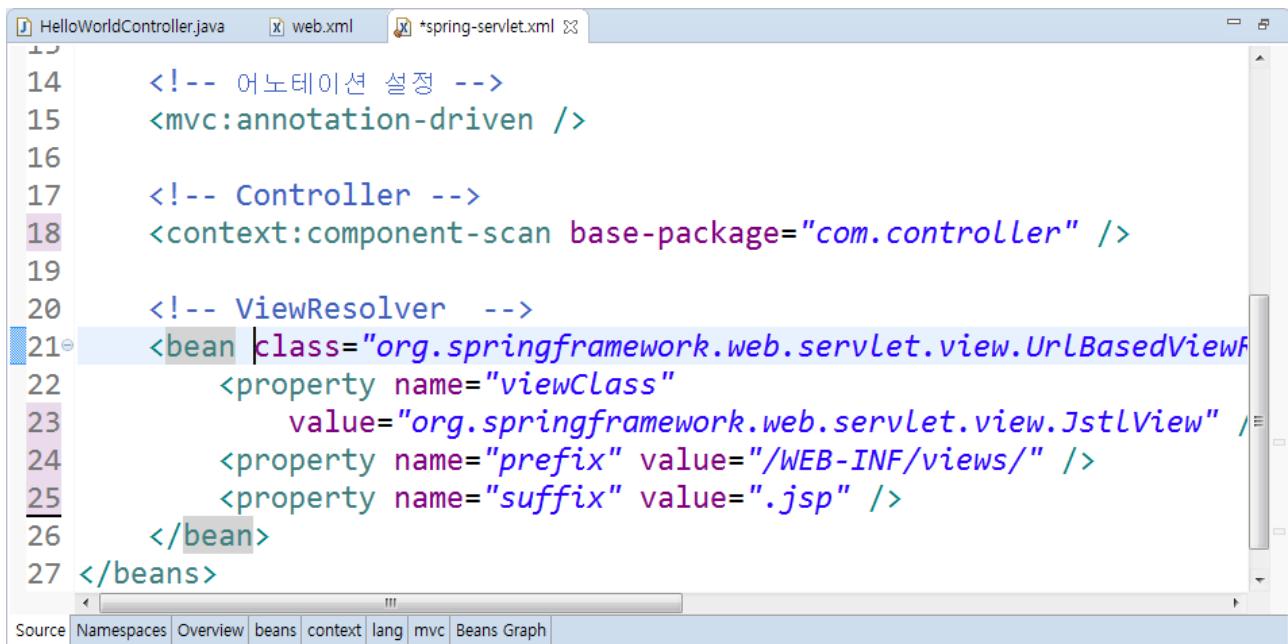
Element : property
Bean definitions can have zero or more properties. Property elements correspond to JavaBean setter methods exposed by the bean classes. Spring supports primitives, references to other beans in the same or related factories, lists, maps and properties.

Content Model : (description?, (meta | bean | ref | idref | value | null | array | list | set | map | props | namespace:uri="##other")?)
[Click for additional documentation at springsource.org](#)

Source Namespaces Overview beans context lang mvc Beans Graph



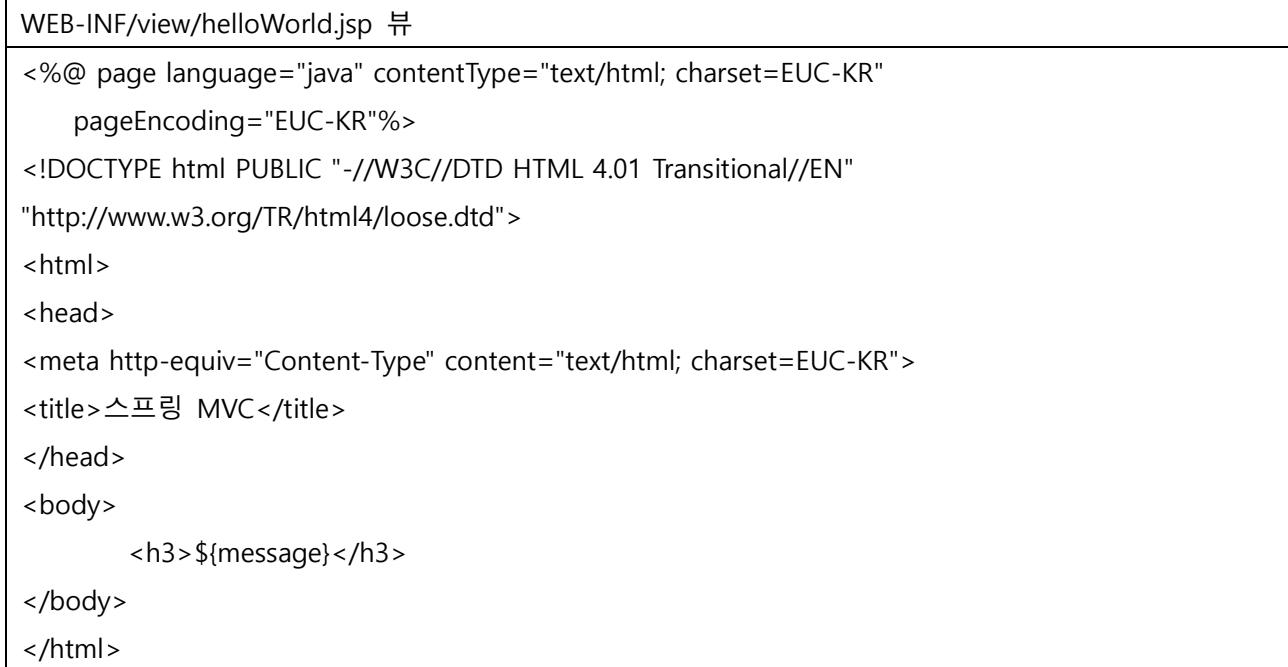
⑪ [source]탭을 클릭하고, spring-servlet.xml 스프링 설정 파일을 참조하여 주석 등을 추가하고, 저장한다.



```
14     <!-- 어노테이션 설정 -->
15     <mvc:annotation-driven />
16
17     <!-- Controller -->
18     <context:component-scan base-package="com.controller" />
19
20     <!-- ViewResolver -->
21     <bean class="org.springframework.web.servlet.view.UrlBasedViewF
22         <property name="viewClass"
23             value="org.springframework.web.servlet.view.JstlView" />
24         <property name="prefix" value="/WEB-INF/views/" />
25         <property name="suffix" value=".jsp" />
26     </bean>
27 </beans>
```

5) 뷰 입력과 저장

- ① "WEB-INF" 폴더에 "view"폴더를 생성한다.
- ② 폴더에 [New] - [JSP file]을 선택하고, 파일명으로 "helloWorld" 입력한 후, [Finish] 버튼을 클릭한다.
- ③ 소스 코드와 같이 "\${message}"를 입력하고, 저장한다.



```
WEB-INF/view/helloWorld.jsp 뷰
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>스프링 MVC</title>
</head>
<body>
    <h3>${message}</h3>
</body>
</html>
```

6) 요청을 위한 JSP 페이지 작성과 실행

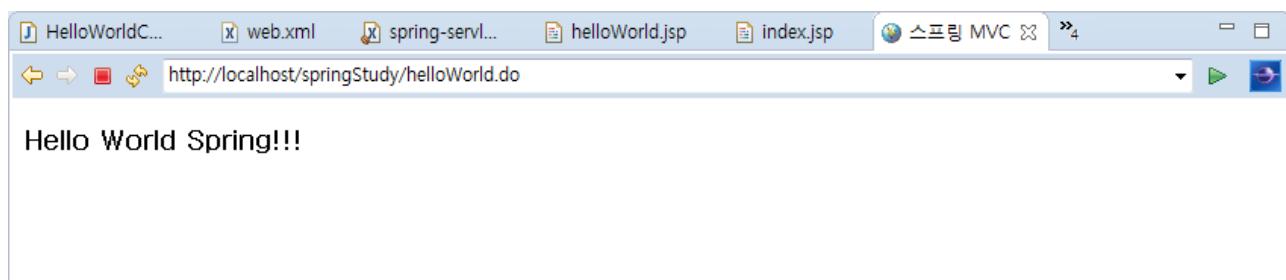
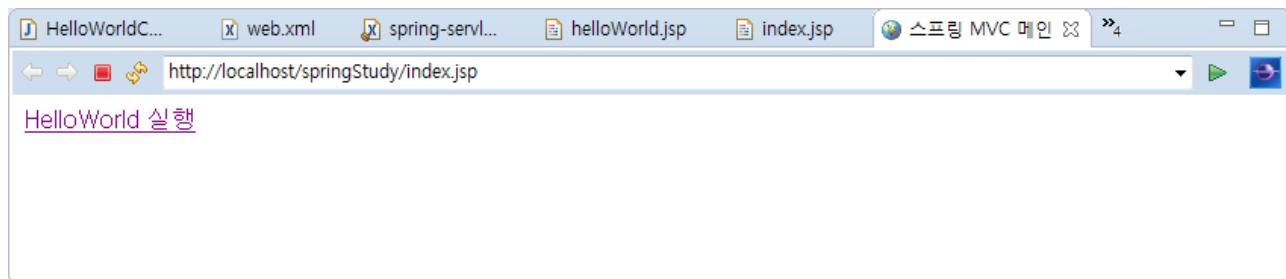
- ① "WebContent" 폴더에서 [New] - [JSP file] 을 선택하고, 파일명으로 "index.jsp"를 입력 한 후 [Finish] 버튼을 클릭한다.

② 소스 코드를 입력하고 저장 버튼을 클릭한다.

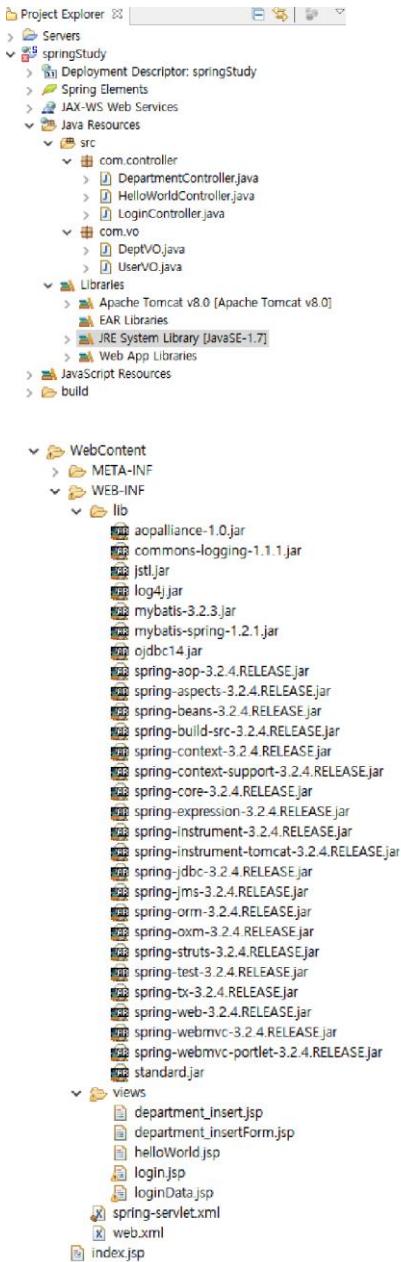
WebContent/index.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
   pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>스프링 MVC 메인</title>
</head>
<body>
   <a href="helloWorld.do">HelloWorld 실행 </a>
</body>
</html>
```

③ "index.jsp" 파일을 실행하여 "HelloWorld 실행"을 클릭하여 실행한다.



17. 스프링 웹 MVC 로그인 예제

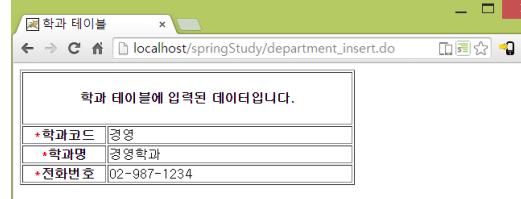
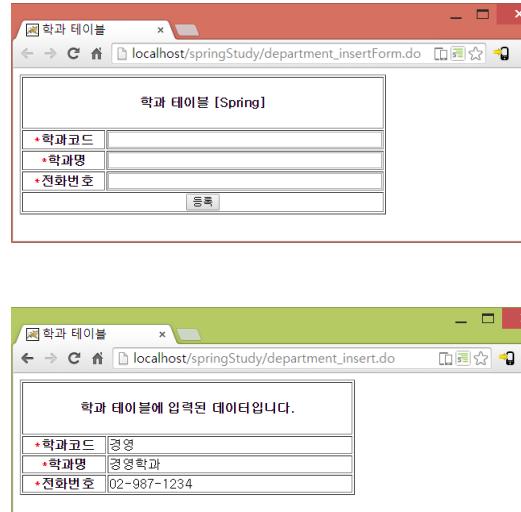
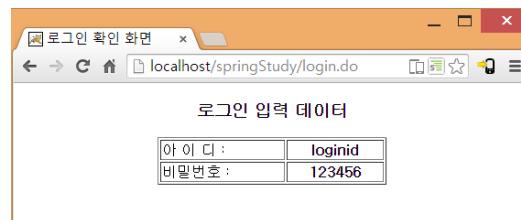
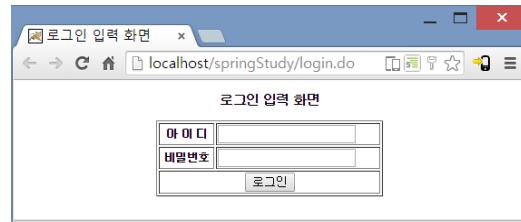


UserVO.java

```
package com.vo;

public class UserVO {
    private String m_uid;
    private String m_pwd;

    public String getM_uid() {
        return m_uid;
    }
}
```



```
}

public void setM_uid(String m_uid) {
    this.m_uid = m_uid;
}

public String getM_pwd() {
    return m_pwd;
}

public void setM_pwd(String m_pwd) {
    this.m_pwd = m_pwd;
}

}
```

LoginController.java

```
package com.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotationModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.vo.UserVO;

@Controller
@RequestMapping(value = "login")
public class LoginController {

    @RequestMapping(method = RequestMethod.GET)
    public String loginForm(Model model) {
        model.addAttribute("userVO", new UserVO());
        return "login";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String onSubmit(@ModelAttribute("userVO") UserVO userVO) {
        System.out.println("onSubmit 구현 메서드입니다.");
        return "loginData";
    }
}
```

login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
```

```

pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"> <title>로그인 입력 화면</title>
<style type="text/css">
div {
    font: 12px "굴림";
}
</style>
</head>
<body>
    <div align="center" class="body">
        <h3>로그인 입력 화면</h3>
        <form:form commandName="userVO" method="POST">
            <table width="250" border="1">
                <tr>
                    <th>아이디</th>
                    <td><form:input path="m_uid" /></td>
                </tr>
                <tr>
                    <th>비밀번호</th>
                    <td><form:password path="m_pwd" /></td>
                </tr>
                <tr>
                    <td colspan="2" align="center"><input type="submit" value="로그인"></td>
                </tr>
            </table>
        </form:form>
    </div>
</body>
</html>

```

loginData.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>

```

```

<html>
<head>
<meta charset="UTF-8">
<title>로그인 확인 화면</title>
</head>
<body>
    <div align="center" class="body">
        <h3>로그인 입력 데이터</h3>
        <table width="250" border="1">
            <tr>
                <td>아이디 :</td>
                <td align="center"><b>${userVO.m_uid}</b></td>
            </tr>
            <tr>
                <td>비밀번호 :</td>
                <td align="center"><b>${userVO.m_pwd}</b></td>
            </tr>
        </table>
    </div>
</body>
</html>

```

DeptVO.java

```

package com.vo;

public class DeptVO {
    private String deptid="";
    private String deptname="";
    private String depttel="";
    public String getDeptid() {
        return deptid;
    }
    public void setDeptid(String deptid) {
        this.deptid = deptid;
    }
    public String getDeptname() {
        return deptname;
    }
}

```

```

public void setDeptname(String deptname) {
    this.deptname = deptname;
}
public String getDepttel() {
    return depttel;
}
public void setDepttel(String depttel) {
    this.depttel = depttel;
}
}

```

DepartmentController.java

```

package com.controller;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.vo.DeptVO;

/* 컨트롤러 */
@Controller
public class DepartmentController {
    @RequestMapping(value = "/department_insertForm.do", method = RequestMethod.GET)
    public String department_insertForm() {
        return "/department_insertForm";
    }
    @RequestMapping(value = "/department_insert.do", method = RequestMethod.POST)
    public String department_insert(DeptVO dvo, Model model) {
        model.addAttribute("dvo", dvo);
        return "/department_insert";
    }
}

```

department_insertForm.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>

```

```

<head>
<meta charset="UTF-8">
<title>학과 테이블</title>
<style type="text/css">
.required {
    color: red;
}
</style>
<script type="text/javascript">
    function insertData() {
        if (!validateForm())
            return;
        if (confirm('등록을 진행하시겠습니까?')) {
            deptForm.action = "/springStudy/department_insert.do";
            deptForm.submit();
        }
    }
    function validateForm() {
        if (deptForm.deptid.value.replace(/\s/g, "") == "") {
            alert('학과코드를 입력해주세요.');
            return false;
        }
        if (deptForm.deptname.value.replace(/\s/g, "") == "") {
            alert('학과명을 입력해주세요.');
            return false;
        }
        return true;
    }
</script>
</head>
<body>
    <div>
        <form id="deptForm" name="deptForm" method="post">
            <table border="1">
                <thead>
                    <tr>
                        <td colspan="2" align="center">
                            <h4>학과 테이블 [Spring]</h4>

```

```

        </td>
    </tr>
</thead>
<tbody>
    <tr>
        <th width="100">
            <span class="required">*</span>학과코드</th>
        <td width="300">
            <input type="text" id="deptid" name="deptid" size="52" />
        </td>
    </tr>
    <tr>
        <th>
            <span class="required">*</span>학과명</th>
        <td>
            <input type="text" id="deptname" name="deptname" size="52" />
        </td>
    </tr>
    <tr>
        <th>
            <span class="required">*</span>전화번호</th>
        <td>
            <input type="text" id="depttel" name="depttel" size="52" />
        </td>
    </tr>
    <tr>
        <td colspan="2" align="center">
            <input type="button" id="Insert" value="등록" onclick="insertData()" />
        </td>
    </tr>
</tbody>
</table>
</form>
</div>
</body>
</html>

```

department_insert.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>학과 테이블</title>
<style type="text/css">
.required {
    color: red;
}
</style>
</head>
<body>
<div>
    <table border="1">
        <thead>
            <tr>
                <td colspan="2" align="center">
                    <h4>학과 테이블에 입력된 데이터입니다.</h4>
                </td>
            </tr>
        </thead>
        <tbody>
            <tr>
                <th width="100">
                    <span class="required">*</span>학과코드</th>
                <td width="300">${dvo.deptid}</td>
            </tr>
            <tr>
                <th><span class="required">*</span>학과명</th>
                <td>${dvo.deptname}</td>
            </tr>
            <tr>
                <th><span class="required">*</span>전화번호</th>
                <td>${dvo.depttel}</td>
            </tr>
        </tbody>
    </table>
</div>

```

```

        </tbody>
    </table>
</div>
</body>
</html>
```

spring-servlet.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:c="http://www.springframework.org/schema/c"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="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.xsd
http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/spring-lang.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!-- 어노테이션 설정 -->
    <mvc:annotation-driven />
    <!-- Controller -->
    <context:component-scan base-package="com.controller" />
    <!-- ViewResolver -->
    <bean class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="viewClass"
                 value="org.springframework.web.servlet.view.JstlView" />
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="springStudy" version="3.0">
    <display-name>springStudy</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <!-- ===== 스프링 관련 설정 ===== -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-context.xml</param-value>
    </context-param>

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value></param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    <!-- ===== 한글 처리 설정===== -->
    <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>
```

```
<init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
</init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>
```

index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>스프링 MVC 메인</title>
</head>
<body>
    <!-- <a href="helloWorld.do">HelloWorld 실행</a> -->
    <% response.sendRedirect("login.do"); %>
    <%-- <% response.sendRedirect("department_insertForm.do"); %> --%>
</body>
</html>
```

mybatis-Spring

스프링 프레임워크에서 데이터베이스 연동에 필요한 JDBC, iBatis, Hibernate 등의 라이브러리를 제공한다. 특히 스프링 프레임워크 3.0부터는 아이바티스(iBatis)의 후속 버전으로 강력한 마이바티스(MyBatis)와 마이바티스-스프링 연동 모듈을 적용하여 데이터에 접근과 처리를 하고 있다. (마이바티스는 객체지향 언어인 자바의 관계형 데이터베이스 프로그래밍을 좀더 쉽게 할 수 있도록 도와주는 개발 프레임워크다.)

1. 마이바티스와 마이바티스-스프링

마이바티스(MyBatis)란 XML구문과 어노테이션을 사용한 SQL문이나 저장된 프로시저를 데이터베이스와 자바 등을 연결시켜 주는 역할을 하는 영속성 프레임워크이다. 영속성 프레임워크란 정보에 대한 접근과 저장을 단순화하는 라이브러리를 말한다. JDBC 코드와 수동으로 설정하는 파라미터와 결과 매핑을 없애주고, 데이터베이스에 원시타입(int, String, Date)과 맵 인터페이스, 자바 객체를 설정하고 매핑하기 위해 XML과 어노테이션을 사용할 수 있다.

구분	아이바티스(iBatis)	마이바티스(MyBatis)
스프링 버전	2.x부터 지원	3.x부터 지원
네임스페이스	선택 사용	필수 사용
매핑 구문	XML	XML과 어노테이션
동적 SQL 요소	16개의 XML 엘리먼트	4개의 XML 엘리먼트
스프링 모듈	자체 모듈	별도 모듈
용어 사용	SqlMapConfig / sqlMap	Configuration / sqlMap

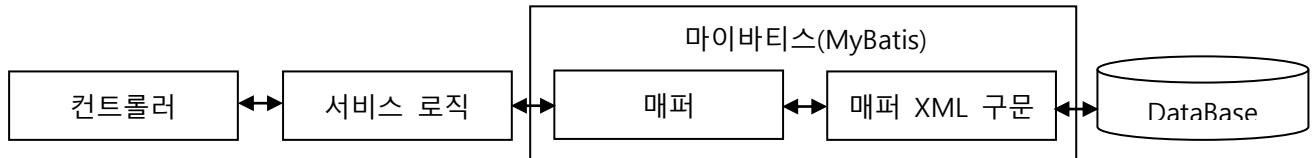
마이바티스(MyBatis)와 스프링 3.x에 통합 지원하는 마이바티스-스프링 라이브러리가 생기게 되었고, 스프링에서 마이바티스-스프링 라이브러리를 사용하면, 마이바티스 객체를 쉽게 생성하고 주입할 수 있으며, 스프링 트랜잭션을 통해 처리할 수 있다. 마이바티스-스프링 연동모듈을 사용하려면 자바 1.5이상의 버전이어야 하고, 마이바티스와 스프링은 각각 버전별로 조금씩 다르다.

마이바티스-스프링	마이바티스(MyBatis)	스프링 프레임워크
1.0.0 그리고 1.0.1	3.0.1에서 3.0.5까지	3.0.0 또는 그 이상
1.0.2	3.0.6	3.0.0 또는 그 이상
1.1.0 또는 그 이상	3.1.0 또는 그 이상	3.0.0 또는 그 이상

스프링 3.0 이상에서 마이바티스(MyBatis) 3.0 이상의 라이브러리와 마이바티스-스프링 1.0 이상 라이브러리 파일이 필요하다. 마이바티스 설치는 "<http://blog.mybatis.org/>" 사이트에서 다운로드 받아 "WEB-INF/lib" 폴더에 복사하면 된다. MyBatis 3.2.4.zip과 MyBatis-spring-1.2.2.zip 파일을 다운로드하여 사용한다.

- MyBatis 3.2.4.zip : 마이바티스 라이브러리
- MyBatis-spring-1.2.2.zip : 마이바티스에서 스프링을 사용할 수 있는 라이브러리

스프링과 마이바티스-스프링을 연동한 데이터에 대한 흐름은 컨트롤러 → 서비스 로직 → 매퍼 → 매퍼 XML 구문순 또는 역순이다.



- 매퍼 XML은 매퍼 XML 요소와 SQL 문으로 작성되는 매퍼 XML 구문이다.
- 매퍼는 인터페이스로 선언하고, 구현 클래스에서 SQL문을 실행한다.
- 서비스 로직은 자바빈으로 구성하고, 데이터베이스를 검색하거나 관리한다.
- 컨트롤러는 스프링 구성요소로 서비스 로직으로부터 데이터를 전달받는다.
- 스프링 설정 파일에서 데이터베이스 서버에 관한 데이터소스(DataSource), 매퍼위치, 트랜잭션, SqlSession, 매퍼 주입에 관하여 설정한다.

2. 매퍼 XML

매퍼 XML은 SQL문을 매퍼 XML 파일로 작성하는 가장 중요한 부분이다. 데이터베이스를 다루는 SQL문은 마이바티스가 제공하는 기능의 XML이나 어노테이션을 통한 매퍼 기법으로 작성된다. 마이바티스의 매퍼 XML 요소이다.

매퍼 XML 요소	설명
<code>cache</code>	해당 네임스페이스를 위한 캐시 설정
<code>cache-ref</code>	다른 네임스페이스를 캐시 설정에 대한 참조
<code>resultMap</code>	데이터베이스 결과 데이터를 객체에 로드하는 방법 정의.
<code>parameterMap</code>	파라미터를 매핑하기 위해서 사용.
<code>insert</code>	매핑된 INSERT문
<code>update</code>	매핑된 UPDATE문
<code>delete</code>	매핑된 DELETE문
<code>select</code>	매핑된 SELECT문

SQL문에서 SELECT문은 select 요소, INSERT문은 insert 요소, UPDATE문은 update 요소, DELETE문은 delete 요소로 매퍼 XML 구문을 만든다. select, insert, update, delete 요소에 사용할 수 있는 공통 속성이 있다.

속성	설명
<code>id</code>	네임스페이스(namespace)의 유일한 구분자 제일 중요
<code>parameterType</code>	구문에 전달될 파라미터의 전체 클래스명이나 별명
<code>parameterMap</code>	외부 parameterMap을 찾기 위한 접근 방법.
<code>flushCache</code>	구문을 호출할 때마다 캐시를 지울지 여부를 설정한다. 디폴트는 false 이다.
<code>timeout</code>	데이터베이스의 요청 결과를 기다리는 최대시간을 설정

statementType	Statement, Prepared, Callable 중 선택. 디폴트는 Prepared이다.
---------------	--

1) select

select 요소는 데이터를 검색하는 SELECT문의 매퍼 구문을 작성한다. 테이블에서 전체 행을 검색하는 SELECT문의 매퍼 구문 형식은 다음과 같다.

```
<select id="구분자" resultType="반환타입">
    select 컬럼명1, 컬럼명2 ... from 테이블명
</select>
```

select 요소는 공통 속성과 속성들로 매퍼 구문을 작성한다.

속성	설명
resultType	반환되는 타입의 전체 클래스명이나 별명.
resultMap	외부 resultMap 참조명
useCache	구문의 결과에 캐시 사용 여부 지정. 디폴트는 true이다.
fetchSize	지정된 수만큼의 결과를 반환하는 값.

```
<select id="listDepartment" resultType="vo.DeptVO">
    Select * from Department
</select>
```

2) insert, update, delete

INSERT문, UPDATE문, DELETE문은 insert, update, delete 매퍼 요소와 공통 속성과 속성을 사용하여 매퍼 구문을 작성한다.

- insert 요소는 테이블에 행을 추가하는 INSERT문의 매퍼 구문을 작성한다.
- update 요소는 테이블의 컬럼 값을 수정하는 UPDATE문의 매퍼 구문을 작성한다.
- delete 요소는 테이블의 행을 삭제하는 DELETE의 매퍼 구문을 작성한다.

속성	설명
useGeneratedKeys	생성 키 사용 여부를 결정한다. 오라클은 sequence를 생성키로 제공한다. 디폴트 값은 false이다.
keyProperty	getGeneratedKeys 메서드나 insert 구문의 selectKey 하위 요소에 의해 리턴된 키를 설정할 프로퍼티를 지정. 디폴트는 설정하지 않는 것이다.
keyColumn	생성키를 가진 테이블의 컬럼명을 설정. 키컬럼이 테이블에 첫번째 컬럼이 아닌 데이터베이스에서만 필요하다.

생성키는 Oracle의 Sequence, MySql의 auto_increment 처럼 자동 증가 필드를 말한다.

3) 파라미터의 위치 지정자(?) 표기

JSP 프로그래밍에서 "SELECT * FROM BOARD WHERE NUM=?"문의 위치 지정자(?)가 있는 SELECT문은 PreparedStatement 객체로 쿼리문을 생성하고 위치 지정자(?)의 값은 "setXXX(위치인덱스, 값)"으로 코딩되었다.

```

String sql = "SELECT * FROM BOARD WHERE NUM=?";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, num);
...

```

파라미터는 마이바티스에서 매우 중요한 요소로, SQL문의 파라메터 값을 받기 위해 "#{}" 기호로 표기하며, {}속에 "값"이나 자바빈의 "프로퍼티명"을 기술한다.

표기법 : #{값 또는 프로퍼티명}

① SELECT문

특정 행을 검색하는 SELECT문의 select 요소의 구문 형식은 다음과 같다.

```

<select id="구분자" parameterType="파라메터타입" resultType="반환타입">
    select 컬럼명1, 컬럼명2 ... from 테이블명 where 컬럼명 = #{값}
</select>

```

② INSERT문

한 행을 추가하는 INSERT문의 insert 요소의 구문 형식은 다음과 같다.

```

<insert id="구분자" parameterType="파라메터타입">
    insert into 테이블명(컬럼명1, 컬럼명2...) values(#{값1}, #{값2} ...)
</insert>

```

③ UPDATE문

특정 또는 일부 행의 컬럼 값을 수정하는 UPDATE문의 update 요소의 구문 형식은 다음과 같다.

```

<update id="구분자" parameterType="파라메터타입">
    update 테이블명 set 컬럼명1 = #{값1}, 컬럼명2 = #{값2} where 컬럼명 = #{값3}
</update>

```

④ DELETE문

특정 행을 삭제하는 DELETE문의 delete 요소의 구문 형식은 다음과 같다.

```

<delete id="구분자" parameterType="파라메터타입">
    delete from 테이블명 where 컬럼명 = #{값1}
</delete>

```

- Department 테이블의 학과코드(#{dept_id})로 특정 행을 검색하는 매퍼 XML 구문을 작성한다.

```

<select id="selectDepartment" parameterType="vo.DeptVO" resultType="vo.DeptVO">
    select *
    from Department
    where Dept_ID = #{dept_id}
</select>

```

- Department 테이블에서 한 행을 추가하는 매퍼 XML 구문을 작성한다.

```
<insert id="insertDepartment" parameterType="vo.DeptVO" >
    insert into Department
    (Dept_ID, Dept_Name, Dept_tel)
    values (#{dept_id}, #{dept_name}, #{dept_tel})
</select>
```

- Department 테이블에서 학과코드(#{dept_id})로 특정 행의 학과명과 전화번호를 수정하는 매퍼 XML 구문을 작성한다.

```
<update id="updateDepartment" parameterType="vo.DeptVO" >
    update Department
    set Dept_Name = #{dept_name}, Dept_tel = #{dept_tel}
    where Dept_ID = #{dept_id}
</select>
```

- Department 테이블에서 학과코드(#{dept_id})로 특정 행을 삭제하는 매퍼 XML 구문을 작성한다.

```
<delete id="deleteDepartment" parameterType="vo.DeptVO">
    delete from Department where Dept_ID = #{dept_id}
</delete>
```

⑤ 데이터베이스가 자동 생성키를 생성한 경우

insert문에서 자동 생성키를 사용하는 기능이 있다. insert는 키(key) 생성과 같은 기능을 추가하기 위한 속성과 하위 요소가 있다.

useGeneratedKeys = "true"로 설정하고 자동 생성키를 적용할 키 컬럼을 keyProperty에 지정한다. INSERT 문에는 키 컬럼은 기술하지 않는다.

```
<insert id="구분자" useGeneratedKeys="true" keyProperty="키컬럼">
    insert into 테이블명(컬럼명1, 컬럼명2...) values(#{값1}, #{값2} ...)
</insert>
```

⑥ 데이터베이스가 자동 생성키를 생성하지 않은 경우

<insert>요소 내에 <selectKey>요소로 키 생성에 필요한 SELECT문을 기술한다. <selectKey>의 속성이다.

속성	설명
keyProperty	selectKey 구문의 결과가 할당될 대상 프로퍼티.
resultType	결과 타입
order	before는 insert문 전 실행. after는 insert문 후 실행.
statementType	Statement, Prepared, Callable 중 선택. 디폴트는 Prepared이다.

order 속성이 before는 먼저 키 값을 생성한 후 INSERT문이 실행되고, after는 INSERT문이 실행된 후에

<selectKey>가 실행된다.

```
<insert id="구분자">
    <selectKey keyProperty="id" resultType="int" order="BEFORE" statementType="PREPARED">
        SELECT문(키 생성 관련)
    </selectKey>
    insert into 테이블명(컬럼명1, 컬럼명2...) values(${값1}, ${값2} ...)
</insert>
```

예제) board 테이블에 게시물이 추가될 때 게시문번호(b_id)를 현재 게시물 번호의 최대값을 구하여 1을 증가하고, null일 때 1로 지정하여 저장하는 XML 매퍼 구문을 작성하여 보자.

```
<insert id="insertBoard" parameterType="com.vo.BoardVO">
    <selectKey keyProperty="bid" resultType="int" order="BEFORE" >
        SELECT CASE WHEN MAX(b_id) IS NULL THEN 1
        ELSE MAX(b_id)+1 END FROM board
    </selectKey>
    INSERT INTO board(b_id, b_name, ...) values(${bid}, ${bname}, ...)
</insert>
```

4) resultMap

모든 컬럼 값의 결과가 HashMap에서 키 형태로 자동 매핑 된다. resultMap의 id, result, constructor, association, collection, discriminator 요소가 있다.

속성	설명
id	기본 키에 해당하는 값을 설정한다.
result	기본 키가 아닌 나머지 컬럼에 대해 맵핑한다.
constructor	생성자를 통해 값을 설정할 때 사용한다.
association	1:1 관계를 처리한다.
collection	1:N 관계를 처리한다.
discriminator	맵핑 과정에서 조건을 지정해서 값을 설정할 때 사용한다.

id나 result 요소는 결과 매핑의 가장 기본적인 형태로 한 개의 컬럼을 한 개의 프로퍼티나 간단한 데이터 타입의 필드에 매핑한다. id와 result에 사용할 수 있는 요소들이다. 테이블 컬럼명과 프로퍼티명이 다른 경우 사용할 수 있다.

속성	설명
property	결과 컬럼에 매핑되는 필드 또는 자바빈과 동일한 프로퍼티
column	데이터베이스 컬럼명이나 컬럼의 별명
javaType	클래스명이나 타입 별명.
jdbcType	jdbc 타입을 명시.

5) 컬럼에 별명을 사용

테이블의 컬럼명과 자바빈의 프로퍼티명이 다를 때 SELECT문의 컬럼명에 별명을 사용하여 일치시킨다.

```
<select id="구분자" resultType="반환타입">
    select 컬럼명1 as 별칭, 컬럼명2 ... from 테이블명 where 컬럼명 = #{값}
</select>
```

6) resultMap의 id와 result 요소

id에 "id명"을 기술하고 <select> 요소의 resultMap에 사용한다. result 요소에 property와 column 속성을 기술한다. 테이블의 컬럼명과 자바빈의 프로퍼티명이 다를 경우 column에 테이블의 컬럼명, property에 자바빈의 프로퍼티명을 기술하여 일치시킨다.

```
<resultMap id="id명" type="타입명">
    <result property="프로퍼티명" column="컬럼명" />
</resultMap>
<select id="구분자" resultType="id명">
    select 컬럼명1, 컬럼명2 ... from 테이블명
</select>
```

3. 동적 SQL

하나의 SQL문을 다양한 형태로 실행하는 기능이 동적 SQL이다.

마이바티스는 if, choose(when, otherwise), trim(where, set), foreach 요소의 동적 SQL문의 작성 방법이다.

예를 들어 student 테이블에서 검색하는 SELECT문이 ① 전체 행을 검색하고 ② "김" 성을 검색하고 ③ 2 학년의 "박"씨 성을 검색하는 3개의 SELECT문은 동적 SQL을 사용하여 한 문장으로 매퍼 XML 구문으로 작성할 수 있다.

- ① SELECT * FROM student
- ② SELECT * FROM student WHERE name LIKE '김%'
- ③ SELECT * FROM student WHERE name LIKE '박%' AND year=2

1) if

<if> 요소는 test의 조건문이 참일 때 연결문자열을 SQL문에 연결하고, 그렇지 않으면 무시된다. 동적 SQL에서 가장 공통적으로 사용되며 SQL문의 WHERE절의 포함 여부를 작성할 수 있다.

```
표기법 : <if test="조건문"> 연결문자열 </if>
```

클라이언트에서 전송된 name이 존재하지 않을 경우 "컴퓨터공학과" 학과의 학생들 전부 출력하고, name이 존재하면 "컴퓨터공학과" 학과 학생 중에서 name으로 검색된 학생들만 출력한다

```
<select id="selectSearch" resultType="studentVO">
    select * from student where dept_name="컴퓨터공학과"
    <if test="name !=null"> and name like #{name} || '%' </if>
</select>
```

2) choose, when, otherwise

choose ~ when ~ otherwise는 if와 함께 가장 공통적으로 제공되는 분기 처리 방식이다. choose 하위의 when 엘리먼트에서 만족하는 조건이 하나라도 나오면 해당 조건의 결과를 동적 SQL에 적용하고 하나라도 없으면 otherwise 엘리먼트의 결과를 동적 SQL에 사용한다.

```
<choose>
    <when test="조건문1"> 연결문자열1</when>
    ...
    <otherwise>연결문자열n</otherwise>
</choose>
```

student 테이블에서 학과코드나 성별로 검색하고, 그렇지 않으면 1학년을 검색하는 경우는 다음과 같다.

```
<select id="selectFind" resultType="studentVO">
    select * from student
    <choose>
        <when test="dept_name!=null"> where dept_name=#{deptName} </when>
        <when test="dept_sex!=null"> where dept_sex=#{deptSex} </when>
        <otherwise> where year=1 </otherwise>
    </choose>
</select>
```

3) trim, where, set

if, choose 등을 사용하여 조건의 결과가 참이 되지 않을 경우 완전하지 못한 SQL문의 요소를 제거하는 trim, where, set이 있다.

- <trim>은 WHERE절에 기술한 AND나 OR의 연산자를 제거한다.

```
표기법 : <trim prefix="WHERE" prefixOverrides="AND | OR"> ... </trim>
```

trim 엘리먼트 속성

- ① prefix : 처리 후 엘리먼트의 내용이 있으면 가장 앞에 붙여준다.
- ② prefixOverrides : 처리 후 엘리먼트 내용 중 가장 앞에 해당 문자들이 있다면 자동으로 지워준다.
- ③ suffix : 처리 후 엘리먼트 내에 내용이 있으면 가장 뒤에 붙여준다.
- ④ suffixOverrides : 처리 후 엘리먼트 내용 중 가장 뒤에 해당 문자들이 있다면 자동으로 지워준다.

- <where>은 SQL문내에 기술한 <where> ~ </where>내의 조건의 모두 참이 되지 않을 경우 <where> ~ </where>의 모든 문자열을 제거한다.

```
표기법 : <where>
    <if test="조건문1"> 연결문자열1</if>
    ...
</where>
```

```

<if test="조건문n"> 연결문자열n</if>
</where>

```

- <set>은 동적인 UPDATE문의 <if>문이 모두 참이 되지 않을 경우 <set> ~ </set>를 문자열을 제거한다.

표기법 : <set>

```

<if test="조건문1"> 연결문자열1</if>
...
<if test="조건문n"> 연결문자열n</if>
</set>

```

다음과 같은 동적인 SELECT문은 <if>의 조건에 따라 부적합한 SELECT문이 생성되어 오류가 발생될 수 있다.

```

<select id="selectFind" resultType="studentVO">
    select * from student
    where  <if test="dept_name!=null"> dept_name=#{deptName} </if>
           <if test="dept_sex!=null"> dept_sex=#{deptSex} </if>
           <if test="year!=null"> year=#{year} </if>
</select>

```

위 SQL문에서 test 조건이 모두 참이 아닐 경우 즉 null이면 where만 남게 된다.

```
select * from student where
```

이 동적인 SQL문은 <where> ~ </where>내에 기술하면 문제를 해결할 수 있다.

```

<select id="selectFind" resultType="studentVO">
    select * from student
    <where> <if test="dept_name!=null"> dept_name=#{deptName} </if>
           <if test="dept_sex!=null"> dept_sex=#{deptSex} </if>
           <if test="year!=null"> year=#{year} </if>
    </where>
</select>

```

<where>의 조건이 참이 되면 "WHERE"를 추가하고, 그렇지 않으면 제거한다.

<set>은 동적인 update문에서 <if>의 조건이 참이 될 때 set을 추가하고 불필요한 콤마(,)를 제거한다.

department 테이블을 수정하기 위한 update문에서 동적인 SQL의 예이다.

```

<update id="updateDepartment">
    update department
    <set> <if test="dept_name!=null"> dept_name=#{deptName} ,</if>
           <if test="dept_tel!=null"> dept_tel=#{deptTel} </if>

```

```

</set>
where dept_id=#{deptId}
</update>

```

<trim> 요소로 제거할 prefix와 suffixOverrides를 기술한다.

```

<update id="updateDepartment">
    update department
        <trim prefix="SET" suffixOverrides=",">
            <if test="dept_name!=null"> dept_name=#{deptName} ,</if>
            <if test="dept_tel!=null"> dept_tel=#{deptTel} </if>
        </trim>
        where dept_id=#{deptId}
</update>

```

4) foreach

SELECT문의 WHERE절에 IN 연산자를 사용할 경우, <foreach>는 컬렉션 명시를 허용하고, item과 index를 사용하여 복수 개의 값을 사용할 수 있다. 복수 개의 값은 open("("), separator(","), close(")")로 구분자를 둘 수도 있다. 파라메타 객체가 마이바티스에 list나 배열로 전달될 때 list와 배열은 "list"와 "array" 키로 사용하고, Map은 "맵명"을 키로 한다.

```

<foreach collection="케" item="아이템" index="인덱스" open="(" close=")" separator=",">
    #{item}
</foreach>

```

foreach 엘리먼트는 다음과 같은 속성을 가진다.

- ① collection : 값 목록을 가진 객체를 설정하면 된다. 파라미터의 타입은 배열이나 List 모두 처리가 가능하다.
- ② item : 목록에서 각각의 값을 사용하고자 할 때 사용하는 속성이다.
- ③ index : 몇 번째 값인지 나타내는 인덱스 값이다. 0부터 시작한다.
- ④ open : 목록에서 값을 가져와서 설정할 때 가장 앞에 붙여 주는 문자를 지정한다. IN절에서는 대부분 (로 시작한다.
- ⑤ close : 목록에서 값을 가져와서 설정할 때 가장 뒤에 붙여 주는 문자를 지정한다. IN절에서는 대부분)로 끝난다.
- ⑥ separator : 목록에서 값을 가져와서 설정할 때 값들 사이에 붙여주는 문자를 지정한다. IN절에서는 값 사이에 쉼표(,)를 붙여준다.

```

<select id="selectPostIn" resultType="domain.blog.Post">
    select *
    from post
    where id in

```

```

<foreach item="item" index="index" collection="list" open="(" separator="" close="")">
    #{item}
</foreach>
</select>

```

department 테이블에서 전체 행이나 검색 조건을 적용하는 매퍼 XML 구문 작성

```

<select id="listDepartment" paramType="vo.DeptVO" resultType="vo.DeptVO">
    select * from department
    <where>
        <if test="department != null and deptname != ''">
            and dept_name like '%' || #{deptname} || '%'
        </if>
    </where>
</select>

```

4. 매퍼

매퍼(mapper)는 매핑된 구문을 주입하기 위한 인터페이스이며, 반드시 인터페이스로 선언해야 한다. 마이바티스의 기본적인 자바 인터페이스는 SqlSession이며, SqlSession 인터페이스는 SQL 매퍼 구문을 실행하고 커밋이나 롤백의 트랜잭션을 관리할 수 있다. SqlSessionFactory 객체가 마이바티스의 전반적인 정보를 가지고 제어한다. SqlSessionFactory 객체를 생성하기 위해 SqlSessionFactoryBuilder 객체를 먼저 선언한다. SqlSessionFactory 객체는 SqlSessionFactoryBuilder 의 build 메소드를 사용해서 생성한다.

마이바티스-스프링에서 마이바티스 API를 직접 사용하는 방법이 있다.

스프링에서 SqlSessionFactroyBean을 이용해서 SqlSessionFactroy를 생성하고, 코드상에서 SqlSessionFactroy를 사용한다.

매퍼 XML 구문 실행의 매퍼 인터페이스

```

public interface DeptMapper {
    public List<DeptVO> listDepartment(DeptVO param);
    public DeptVO selectDepartment(DeptVO param);
    public int insertDepartment(DeptVO param);
    public int updateDepartment(DeptVO param);
    public int deleteDepartment(DeptVO param);
}

```

1) 매퍼 XML 구문의 실행 메서드

실행 메서드들은 매퍼 XML 파일에 정의된 select, insert, update, delete 요소를 실행한다. 메서드명 자체가 그 역할을 설명하도록 명명되었고, id와 파라메터 객체(원시타입, 자바빈, Map)를 가진다.

select 요소를 실행하는 메서드는 selectOne(), selectList(), selectMap() 세 가지가 있다.

- selectOne 메서드는 오직 하나의 객체만을 반환하는 메서드이다. 한 개 이상 또는 null이 반환되면 예외가 발생한다.
- selectList 메서드는 복수 개의 객체를 반환하는 메서드이다.
- selectMap 메서드는 복수 개의 객체를 반환(Map)하는 메서드이다.

insert, update, delete 요소는 insert(), update(), delete()의 실행 메서드가 있으며, 실행한 후 트랜잭션의 수가 반환된다.

5. 마이바티스와 스프링 설정

마이바티스와 스프링을 연동하기 위해서는 다음과 같은 설정이 필요하다.

- 데이터베이스 서버(JDBC 드라이버, url, 계정이름, 암호)의 dataSource
- SqlSessionFactory
- 매퍼 인터페이스
- 트랜잭션
- SqlSession
- Mapper 주입

1) dataSource

dataSource는 데이터베이스 서버에 대한 정보로 JDBC 드라이버, url, 계정이름, 암호 정보가 필수 요소이며, 스프링 설정 파일에 구체적인 정보를 설정하거나 별도의 프로퍼티 파일을 작성하여 호출할 수도 있다.

① 데이터베이스 서버의 프로퍼티 파일

프로퍼티 파일에는 기본적인 드라이버, url, 계정이름, 암호를 추가한다. 일반적으로 파일명은 jdbc와 properties 확장자를 붙인다. 프로퍼티 파일이 저장되는 위치는 src의 classpath 또는 "WEB-INF" 하위폴더에 저장하면 된다. 다음은 *.properties 파일의 일반적인 설정 내용이다.

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

② 스프링 설정 파일에서 dataSource 설정

스프링 설정 파일에 PropertyPlaceholderConfigurer의 빈을 설정하고, <value> 속성으로 프로퍼티 파일이 저장된 “경로:프로퍼티명”을 설정한다.

```
WEB-INF/**/*-context.xml
<bean id="propertyConfigurer"
class= "org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
```

```

<list><value>classpath:jdbc.properties</value></list>
</property>
<property name="fileEncoding" value="euc-kr"/>
</bean>

```

③ dataSource 빈 설정

jdbc.properties에서 읽은 값으로 dataSource를 설정한다. dataSource의 빈은 DriverManagerDataSource 클래스명을 사용하여 빈 설정하고, <property>와 <value> 속성으로 지정한다.

WEB-INF/**/*-context.xml

```

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

```

6. SqlSessionFactoryBean

マイバティス-스프링 연동 모듈의 SqlSessionFactoryBean은 스프링의 팩토리빈 인터페이스를 구현한다. 이 설정은 SqlSessionFactoryBean을 생성하는 것이 아니라 팩토리에서 getXXX() 메서드를 호출 결과를 반환하는 것을 의미한다. 스프링은 애플리케이션 시작 시점에 SqlSessionFactory를 생성하여 저장한다.

스프링 설정 파일에 팩토리 빈을 생성하는 XML 설정은 다음과 같다.

WEB-INF/**/*-context.xml

```

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

```

일반적으로 마이바티스-스프링에서는 SqlSessionFactoryBean이나 sqlSessionFactory를 직접 사용하기 보다는 **sqlSessionDaoSupport**나 **MapperFactoryBean**을 확장하여 다른 DAO에 주입된다.

1) <property> 속성

sqlSessionFactory는 JDBC dataSource의 필수 프로퍼티로, dataSource의 연결을 설정해야만 한다. 마이바티스 XML 설정 파일의 위치를 지정하기위해 사용되는 configLocation을 프로퍼티로 지정 할 수도 있다. 마이바티스 XML 설정 파일이 매퍼 클래스와 동일한 클래스 패스에 있지 않으면 프로퍼티를 설정하여야 한다.

- ① 마이바티스 설정파일에 <mappers> 요소로 XML 파일의 클래스 패스 지정.
- ② 팩토리 빈의 mapperLocations 프로퍼티 사용한다. mapperLocations 프로퍼티는 매퍼의 자원 위치나 마이바티스 XML 매퍼 파일들의 위치를 명시할 때 사용된다.

7. 트랜잭션

마이바티스-스프링 연동 모듈을 사용하면 스프링 트랜잭션에 자연스럽게 연동 될 수 있다. 마이바티스에 종속되는 트랜잭션 관리보다 스프링의 DataSourceTransactionManager로 트랜잭션을 설정할 수 있으며, 3 가지 설정 방법이 있다.

1) 표준 설정 방법

스프링 트랜잭션을 가능하게 하려면 스프링 XML 설정파일에 스프링의 DataSourceTransactionManager를 생성한다. dataSource는 필수 프로퍼티이며, SqlSessionFactoryBean을 생성한 것과 반드시 같아야 한다.

```
<!-- a PlatformTransactionManager is still required -->
<bean id="transactionManager"
      class= "org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

2) 컨테이너 관리 트랜잭션

컨테이너 관리 트랜잭션은 컨테이너 서버의 설정으로 컨테이너 객체들의 트랜잭션을 관리하는 방법이다. 다중 자원 접근을 위한 JTA(Java Transaction API)로 트랜잭션을 관리할 경우 스프링 트랜잭션 네임스페이스를 사용하여 설정할 수 있다.

```
<tx:jta-transaction-manager />
```

3) 프로그램의 트랜잭션 관리

마이바티스 sqlSession은 트랜잭션을 메서드로 제어하지만, 마이바티스-스프링 연동 모듈은 빈을 스프링이 관리하는 sqlSession이나 스프링이 관리하는 매퍼에 주입하여 트랜잭션을 관리한다. 스프링이 관리하는 sqlSession이나 주입된 매퍼 클래스에서는 sqlSession.commit(), sqlSession.rollback() 또는 sqlSession.close() 메서드를 호출할 수가 없다.

8. sqlSession

마이바티스-스프링 연동 모듈은 사용자 빈이 sqlSession에 주입되고 sqlSession은 스프링 트랜잭션 설정에 따라 세션을 자동으로 커밋/롤백한다.

1) sqlSessionTemplate

sqlSessionTemplate은 마이바티스-스프링 연동 모듈의 핵심으로 sqlSession을 구현하고, 코드에서 sqlSession을 대체하는 역할을 한다. sqlSessionTemplate은 여러 개의 DAO나 매퍼에서 공유할 수 있다. SQL을 처리하는 마이바티스 메서드를 호출할 때 sqlSessionTemplate은 sqlSession이 현재의 스프링 트랜잭션에서 사용될 수 있도록 보장하고, 필요한 시점에 세션을 닫고, 커밋과 롤백, 그리고 세션의 생명 주기를 관리한다. sqlSessionTemplate은 <constructor-arg> 생성자로 sqlSessionFactory로 의존 주입할 수 있다.

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

sqlSessionTemplate로 설정된 sqlSession은 <property>로 설정하여 빈에 직접 주입한다.

```
<bean id="deptMapper" class="DeptMapperImpl">
    <property name="sqlSession" ref="sqlSession" />
</bean>
```

2) sqlSessionDaoSupport

sqlSessionDaoSupport는 sqlSession을 제공하는 추상 클래스이다. 서비스 로직의 DAO 구현체에서 **getSqlSession()** 메서드로 SQL문을 처리하는 마이바티스 메서드를 호출할 sqlSessionTemplate을 얻을 수 있다.

```
import org.mybatis.spring.support.SqlSessionDaoSupport;
public class DeptDAOImpl extends SqlSessionDaoSupport implements DeptDAO {
    public DeptVO selectDepartment(DeptVO param) {
        return DeptVO.getSqlSession().selectOne("dao.DeptDAO.selectDepartment");
    }
}
```

9. 매퍼 주입

스프링 설정 파일에 매퍼 주입하고, 검색할 수 있도록 설정한다.

1) XML 설정을 사용하는 매퍼 주입

매퍼는 스프링 설정 파일에 MapperFactoryBean으로 주입한다.

```
<bean id="deptMap" class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface" value="spring.DeptMapper" />
    <property name="sqlSessionFactory" value="sqlSessionFactory" />
</bean>
```

DeptMapper의 인터페이스와 동일한 클래스패스에 마이바티스 XML 매퍼 파일이 있으면 MapperFactoryBean이 자동으로 파싱한다. 매퍼 XML 파일이 동일한 클래스 경로에 있다면 설정 파일에 매퍼를 주입할 필요는 없다.

2) 매퍼의 자동 검색

마이바티스-스프링 연동 모듈은 3가지 설정 방법의 매퍼 자동 검색이 있다. <mybatis:scan>과 @MapperScan은 마이바티스-스프링 연동 1.2.0에서 추가된 기능이며, @MapperScan은 스프링 버전이 3.1 이상이어야 한다.

① <mybatis:scan> 사용

스프링에서 제공하는 <context:component-scan>과 매우 유사한 방법으로 매퍼를 검색한다.

다음과 같이 <mybatis:scan>을 설정한다.

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

"base-package" 속성은 매퍼 인터페이스 파일이 저장된 최상위 패키지를 지정한다. 세미콜론이나 콤마 구분자로 여러 개의 패키지를 설정할 수 있다. 매퍼는 지정된 패키지에서 하위 패키지를 모두 검색한다.

② @MapperScan 어노테이션 사용

@Configuration은 스프링의 자바 설정에서 @MapperScan을 선호하는 방법이며, @MapperScan 어노테이션은 다음과 같다.

```
@Configuration  
@MapperScan("spring.mapper")  
public class AppConfig{ ... }
```

③ 스프링 XML 파일에 MapperScannerConfigurer 설정

MapperScannerConfigurer는 빈처럼 XML 애플리케이션 컨텍스트에 포함하는 방법이다.

MapperScannerConfigurer를 스프링 설정 파일에 추가한다.

```
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  
    <property name="basePackage"> <value>com.spring.dao</value> </property>  
</bean>
```

MapperScannerConfigurer는 예러를 발생시키는 PropertyPlaceholderConfigurer보다 먼저 실행되기 때문에 이 프로퍼티보다 SqlSessionFactoryBeanName 과 SqlSessionTemplateName 프로퍼티 사용을 권장한다.

마이바티스-스프링 연동 모듈과 스프링을 연동하는 스프링 설정 파일을 작성한다. - ①

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans ... >  
    <bean id="propertyConfigurer"  
        class= "org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  
        <property name="locations">  
            <list><value>classpath:jdbc.properties</value> </list>  
        </property>  
        <property name="fileEncoding" value="euc-kr"/>  
    </bean>  
    <!-- DataSource -->  
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
        <property name="driverClassName" value="${jdbc.driverClass}"/>  
        <property name="url" value="${jdbc.url}"/>  
        <property name="username" value="${jdbc.username}"/>  
    </bean>
```

```

<property name="password" value="${jdbc.password}"/>
</bean>
<!-- a PlatformTransactionManager is still required -->
<bean id="transactionManager"
class= "org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
<tx:annotation-driven transaction-manager="transactionManager" />
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mapperLocations" value="classpath:mybatis/dept/*.xml"/>
</bean>
<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg ref="sqlSessionFactory" />
</bean>
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" > <value>com.spring.dept.dao</value> </property>
</bean>
</beans>

```

마이바티스-스프링 연동 모듈과 스프링을 연동하는 스프링 설정 파일을 작성한다. - ②

```

<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
    <!-- DB Connection -->
    <!-- DataSource -->
    <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName" value="java:comp/env/jdbc/Oracle11g" />
    </bean>

    <!-- Mybatis Setting -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="mapperLocations" value="classpath:mybatis/query/*.xml"/>
    </bean>

    <!-- a PlatformTransactionManager is still required -->
    <bean id="transactionManager"
        class= "org.springframework.jdbc.datasource.DataSourceTransactionManager">

```

```

<property name="dataSource" ref="dataSource" />
</bean>

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage"><value>com.spring.*.dao</value></property>
</bean>
</beans>

```

10. 서비스 로직 인터페이스와 구현

매퍼 또는 DAO로 생성된 인터페이스는 데이터 처리만을 담당하고, 데이터를 처리하는 역할은 서비스 로직이다. 컨트롤러에서 데이터를 처리하는 경우에도 서비스 로직이다.

서비스 로직은 데이터베이스 접근 객체인 매퍼를 호출하는 인터페이스를 정의하고, 인터페이스의 구현 클래스에서 처리하도록 권장하고 있다. 이 처리 방법은 관계형 데이터베이스가 변경되더라도 소스 코드에 영향을 주지 않고 적용이 가능하다. 그리고 매퍼로 호출을 전달하므로 마이바티스에 대한 의존성이 없어진다. 서비스 로직의 인터페이스는 매퍼의 인터페이스와 유사한 구조로 작성된다. 구현 클래스에서 메서드의 반환값은 “매퍼명.메서드명(파라미터)”이다.

11. Spring-Mybatis 프로그래밍 예제

1) 테이블 생성

```

-- 부서테이블 생성
create table department(
    dept_id varchar2(10) not null,
    dept_name varchar2(25) not null,
    dept_tel varchar2(12)
);

comment on table department is '부서테이블 정보';
comment on column department.dept_id is '부서번호';
comment on column department.dept_name is '부서명';
comment on column department.dept_tel is '부서별 전화번호';

```

-- 학과테이블 생성

```

create table lesson(
    no      number      not null,
    l_num   varchar2(2 byte) not null,
    l_name  varchar2(20 byte) not null,
    constraint lesson_pk primary key(no)
);

```

```

comment on table lesson is '학과테이블 정보';
comment on column lesson.no is '순번';
comment on column lesson.l_num is '학과번호';
comment on column lesson.l_name is '학과명';

```

```

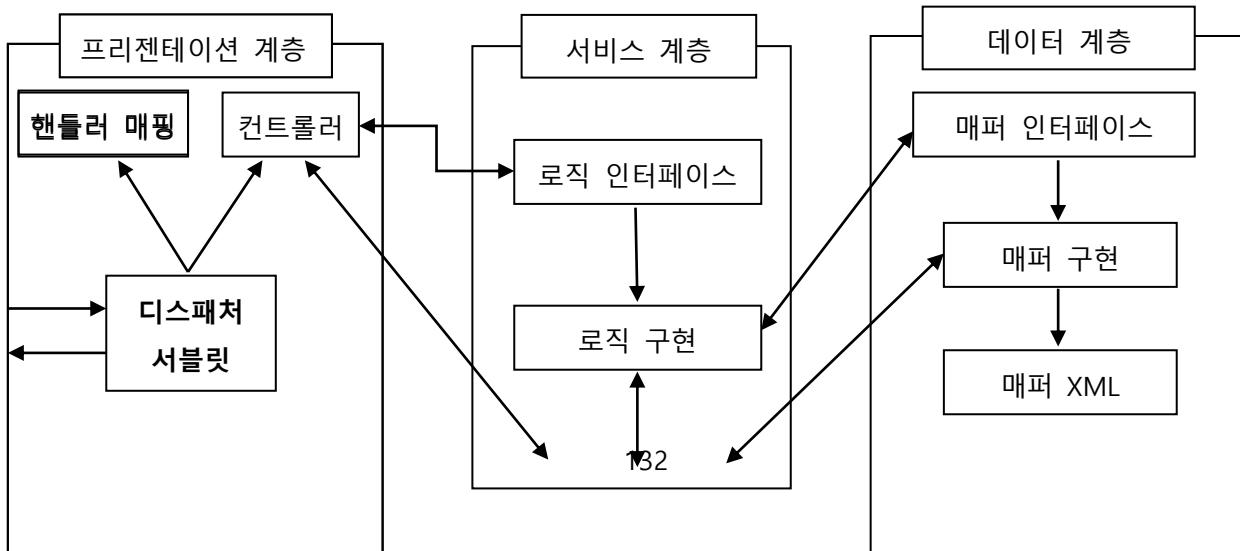
create sequence lesson_seq
start with 21
maxvalue 99999999999999999999999999999999
minvalue 1
nocycle
cache 20
noorder;

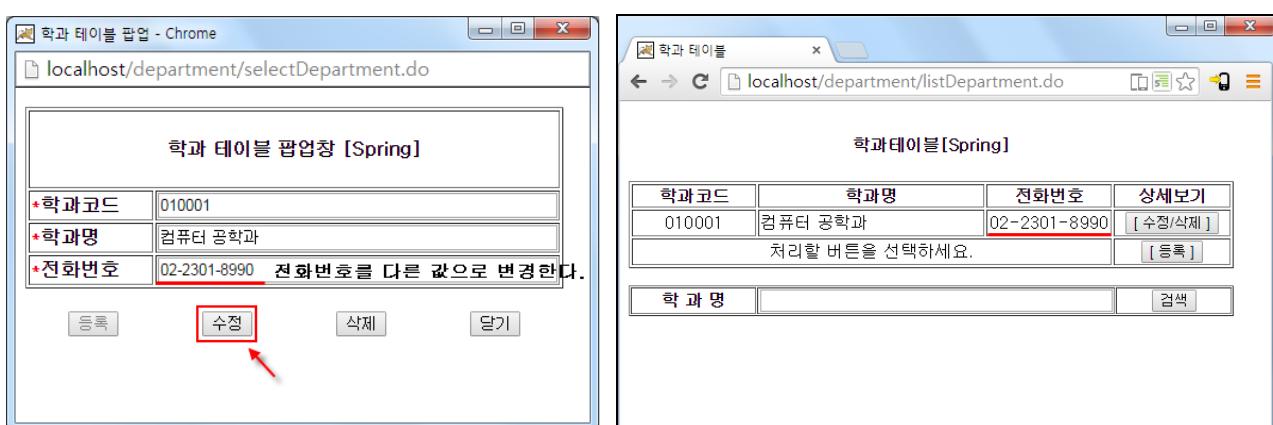
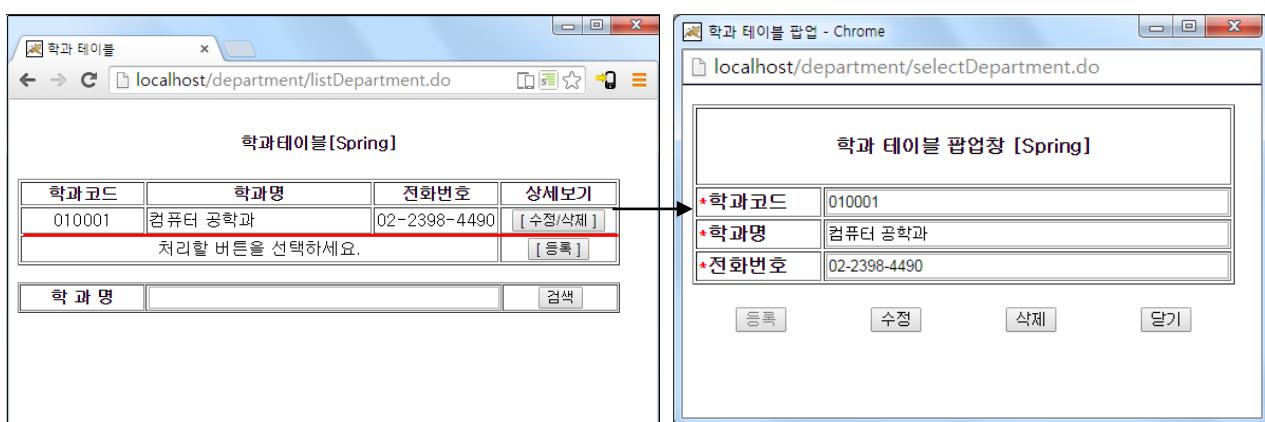
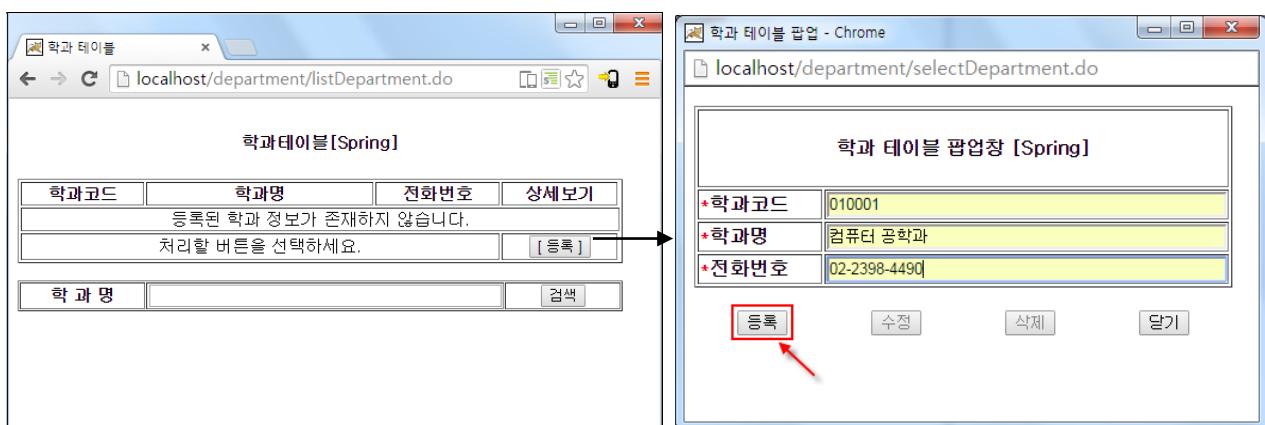
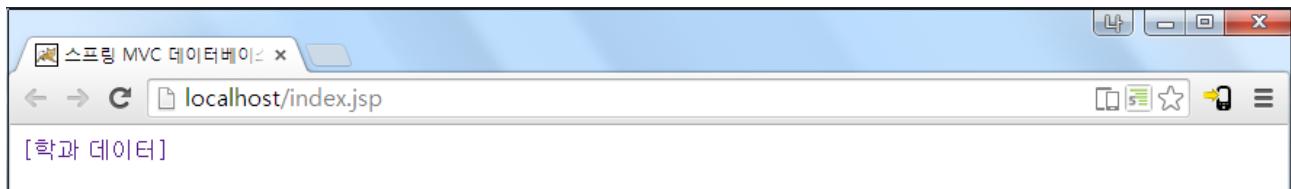
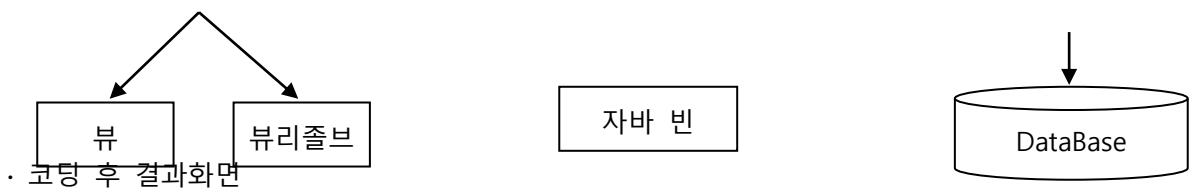
```

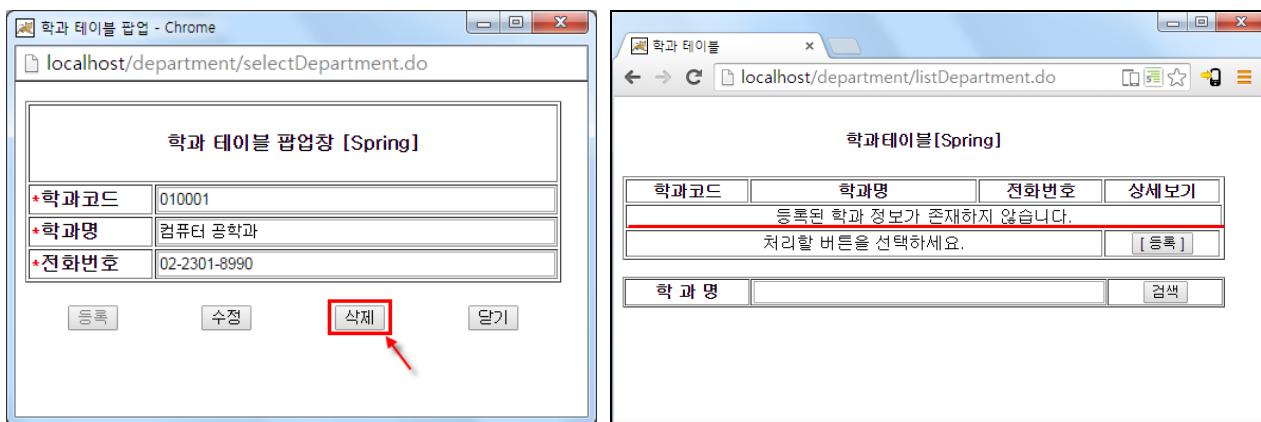
2) 개발 순서

- 테이블을 생성하고, 데이터를 저장한다.
- 프로젝트에 필요한 폴더를 생성한다.
- 모델에 관한 공통의 자바빈을 생성한다.
- 데이터 계층의 SQL 매퍼 XML 파일을 생성한다.
- 데이터 계층의 매퍼 인터페이스와 매퍼 인터페이스를 구현한다.
- 서비스 계층의 로직 인터페이스와 로직 인터페이스를 구현한다.
- 컨트롤러를 작성한다.
- 뷰를 작성한다.
- jdbc.properties 파일을 생성한다.
- 스프링 환경 설정 파일에 dataSource, 매퍼 등을 설정한다.
- 웹 어플리케이션 환경 설정 파일(web.xml)에 디스패처 서블릿을 설정한다.
- 실행에 필요한 웹 페이지를 작성하고 실행한다.
- 실행 결과를 검토하고, 오류에 대하여 디버깅한다.

· 웹 애플리케이션의 계층과 구현 모델







- ① [New] - [Dynamic Web Project] 선택하고 Project name은 “springDBLink” 지정하고 [Next]버튼 클릭, Java 화면에서 Default output folder: “WebContentWEB-INFclasses”로 변경하고 [Next] 버튼 클릭, Web Module 화면에서 “Generate web.xml deployment descriptor” 체크 박스를 클릭하고, [Finish]버튼을 클릭한다.
- ② “springDBLink” Dynamic Web Project에 스프링 프레임워크를 적용시키기 위해서 “springDBLink” 프로젝트를 선택하고, 마우스 오른쪽 버튼을 클릭하고, 팝업창에서 [spring Tools] - [Add Spring Project Nature] 메뉴를 차례로 클릭한다.
- ③ “springDBLink”的 “/WEB-INF/lib” 폴더에 스프링 프레임워크의 jar 파일과 프로그래밍에 관련된 jar 파일들을 붙여넣기 한다.
- ④ [Servers] - [context.xml] 파일에 <Resource>요소가 정의되어 있는지 확인한다.

```
Servers - context.xml
<Context>
...
<Resource auth="Container" driverClassName="oracle.jdbc.driver.OracleDriver" maxActive="100"
maxIdle="30" maxWait="10000" name="jdbc/Oracle11g" password="tiger"
type="javax.sql.DataSource" url="jdbc:oracle:thin:@127.0.0.1:1521:orcl" username="scott"/>
</Context>
```

⑤ 웹 애플리케이션 환경 설정 파일에서 디스패처 서블릿 설정

```
WEB-INF/web.xml 웹 애플리케이션 환경 설정 파일에 추가
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="springDBLink" version="3.0">
<display-name>springDBLink</display-name>
```

```

<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<!-- ===== 스프링 관련 설정 ===== -->
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
<!-- ===== 한글 처리 설정 ===== -->
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>
        org.springframework.web.filter.CharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>euc-kr</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>

```

⑥ 스프링 설정 파일 생성 - 1

WEB-INF/**spring-servlet.xml** 스프링 설정 파일

```

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

    <!-- 어노테이션 설정 -->
    <context:annotation-config />

    <!-- Controller -->
    <context:component-scan base-package="com.spring.**" />

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

    <import resource="servlet-context.xml" />
</beans>

```

⑥ 스프링 설정 파일 생성 - 2

WEB-INF/**servlet-context.xml** 스프링 설정 파일

```

<?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:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

```

```

<!-- DB Connection -->

<!-- DataSource -->
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="java:comp/env/jdbc/Oracle11g" />
</bean>

<!-- Mybatis Setting -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="mapperLocations" value="classpath:mybatis/query/*.xml"/>
</bean>

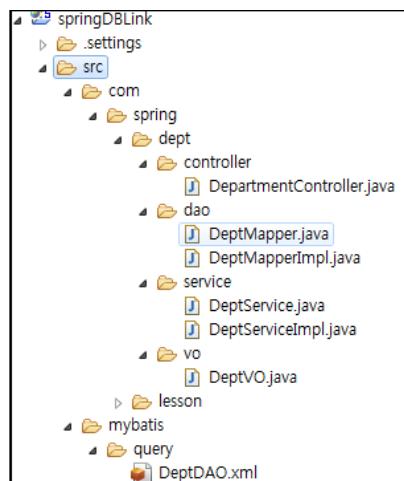
<!-- a PlatformTransactionManager is still required -->
<bean id="transactionManager"
      class= "org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage"><value>com.spring.**.dao</value></property>
</bean>

</beans>

```

⑦ 컨트롤러, 서비스, DAO, 매퍼 XML 문서를 작성하여 보자.



```

package com.spring.dept.vo;
public class DeptVO {
    private String deptid=""; //학과코드
    private String deptname=""; //학과명
    private String depttel=""; //학과전화

    public String getDeptid() {
        return deptid;
    }
    public void setDeptid(String deptid) {
        this.deptid = deptid;
    }
    public String getDeptname() {
        return deptname;
    }
    public void setDeptname(String deptname) {
        this.deptname = deptname;
    }
    public String getDepttel() {
        return depttel;
    }
    public void setDepttel(String depttel) {
        this.depttel = depttel;
    }
}

```

DepartmentController.java

```

package com.spring.dept.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import com.spring.dept.service.DeptService;
import com.spring.dept.vo.DeptVO;

```

```

/* 컨트롤러 */
@Controller
@RequestMapping(value="/department")
public class DepartmentController {
    private static final String CONTEXT_PATH="dept";

    @Autowired
    private DeptService deptService;

    /* 전체 조회 */
    @RequestMapping("/listDepartment")
    public ModelAndView listDepartment(@ModelAttribute DeptVO param) {
        List<DeptVO> list = deptService.listDepartment(param);

        ModelAndView mav = new ModelAndView();
        mav.addObject("departmentList", list);
        mav.setViewName(CONTEXT_PATH + "/department");

        return mav;
    }

    /* 상세 정보 보기 */
    @RequestMapping("/selectDepartment")
    public ModelAndView selectDepartment(@RequestParam("deptid") String deptid) {
        ModelAndView mav = new ModelAndView();

        if(deptid.equals("")) { // [등록] 버튼 클릭시
            mav.addObject("mode", "insert");
        } else { // [수정/삭제] 버튼 클릭시
            mav.addObject("DeptVO", deptService.selectDepartment(deptid));
            mav.addObject("mode", "update");
        }

        mav.setViewName(CONTEXT_PATH + "/department_pop");

        return mav;
    }

    /* 레코드 추가 */
}

```

```

@RequestMapping("/insertDepartment")
public ModelAndView insertDepartment(@ModelAttribute DeptVO param) {
    String resultStr = "";
    int result = deptService.insertDepartment(param);

    if(result > 0) resultStr = "등록 완료이 완료되었습니다.";
    else resultStr = "등록에 문제가 있어 완료하지 못하였습니다.";

    ModelAndView mav = new ModelAndView();
    mav.addObject("result", resultStr);
    mav.setViewName("result");

    return mav;
}

/* 레코드 수정 */
@RequestMapping("/updateDepartment")
public ModelAndView updateDepartment(@ModelAttribute DeptVO param) {
    String resultStr = "";
    int result = deptService.updateDepartment(param);

    if(result > 0) resultStr = "수정 완료이 완료되었습니다.";
    else resultStr = "수정에 문제가 있어 완료하지 못하였습니다.";

    ModelAndView mav = new ModelAndView();
    mav.addObject("result", resultStr);
    mav.setViewName("result");

    return mav;
}

/* 레코드 삭제 */
@RequestMapping("/deleteDepartment")
public ModelAndView deleteDepartment(@ModelAttribute DeptVO param) {
    String resultStr = "";
    int result = deptService.deleteDepartment(param);

    if(result > 0) resultStr = "삭제 완료이 완료되었습니다.";
}

```

```

        else resultStr = "삭제에 문제가 있어 완료하지 못하였습니다.";

        ModelAndView mav = new ModelAndView();
        mav.addObject("result", resultStr);
        mav.setViewName("result");

        return mav;
    }
}

```

DeptService.java

```

package com.spring.dept.service;
import java.util.List;
import com.spring.dept.vo.DeptVO;

public interface DeptService {
    public List<DeptVO> listDepartment(DeptVO param);
    public DeptVO selectDepartment(String deptid);
    public int insertDepartment(DeptVO param);
    public int updateDepartment(DeptVO param);
    public int deleteDepartment(DeptVO param);
}

```

DeptServiceimpl.java

```

package com.spring.dept.service;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.spring.dept.dao.DeptMapper;
import com.spring.dept.vo.DeptVO;

@Service
@Transactional
public class DeptServiceimpl implements DeptService {

    @Autowired

```

```

private DeptMapper deptMapper;

@Override
public List<DeptVO> listDepartment(DeptVO param) {
    List<DeptVO> list = new ArrayList<DeptVO>();
    list = deptMapper.listDepartment(param);
    return list;
}

@Override
public DeptVO selectDepartment(String deptid) {
    return deptMapper.selectDepartment(deptid);
}

@Override
public int insertDepartment(DeptVO param) {
    return deptMapper.insertDepartment(param);
}

@Override
public int updateDepartment(DeptVO param) {
    return deptMapper.updateDepartment(param);
}

@Override
public int deleteDepartment(DeptVO param) {
    return deptMapper.deleteDepartment(param);
}

```

DeptMapper.java

```

package com.spring.dept.dao;
import java.util.List;
import com.spring.dept.vo.DeptVO;

public interface DeptMapper {
    public List<DeptVO> listDepartment(DeptVO param);
    public DeptVO selectDepartment(String deptid);

```

```
    public int insertDepartment(DeptVO param);
    public int updateDepartment(DeptVO param);
    public int deleteDepartment(DeptVO param);
}
```

DeptMapperImpl.java

```
package com.spring.dept.dao;
import java.util.List;
import org.mybatis.spring.support.SqlSessionDaoSupport;
import com.spring.dept.vo.DeptVO;

public class DeptMapperImpl extends SqlSessionDaoSupport implements DeptMapper {
    private final String PACKAGE_PATH = "com.spring.dept.dao.DeptDAO.";

    @Override
    public List<DeptVO> listDepartment(DeptVO param) {
        return getSqlSession().selectList(PACKAGE_PATH+"listDepartment");
    }

    @Override
    public DeptVO selectDepartment(String deptid) {
        return (DeptVO)getSqlSession().selectOne(PACKAGE_PATH+"selectDepartment");
    }

    @Override
    public int insertDepartment(DeptVO param) {
        return (int)getSqlSession().insert(PACKAGE_PATH+"insertDepartment");
    }

    @Override
    public int updateDepartment(DeptVO param) {
        return (int)getSqlSession().update(PACKAGE_PATH+"updateDepartment");
    }

    @Override
    public int deleteDepartment(DeptVO param) {
        return (int)getSqlSession().delete(PACKAGE_PATH+"deleteDepartment");
    }
}
```

```
}
```

DeptDAO.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-
mapper.dtd">

<mapper namespace="com.spring.dept.dao.DeptMapper">

    <resultMap type="com.spring.dept.vo.DeptVO" id="deptVO">
        <result property="deptid" column="Dept_ID" />
        <result property="deptname" column="Dept_Name" />
        <result property="depttel" column="Dept_Tel" />
    </resultMap>

    <select id="listDepartment" parameterType="com.spring.dept.vo.DeptVO" resultMap="deptVO">
        /* Mapper - listDepartment */
        SELECT Dept_ID
            ,Dept_NAME
            ,Dept_Tel
        FROM DEPARTMENT
        <where>
            <if test="deptname != null and deptname != "">
                DEPT_NAME LIKE '%' || #{deptname} || '%'
            </if>
        </where>
    </select>

    <select id="selectDepartment" parameterType="java.lang.String" resultMap="deptVO">
        /* Mapper - selectDepartment */
        SELECT Dept_ID
            ,Dept_NAME
            ,Dept_Tel
        FROM DEPARTMENT
        WHERE DEPT_ID = #{deptid}
    </select>

    <insert id="insertDepartment" parameterType="com.spring.dept.vo.DeptVO">
        /* Mapper - insertDepartment */
        INSERT INTO DEPARTMENT
```

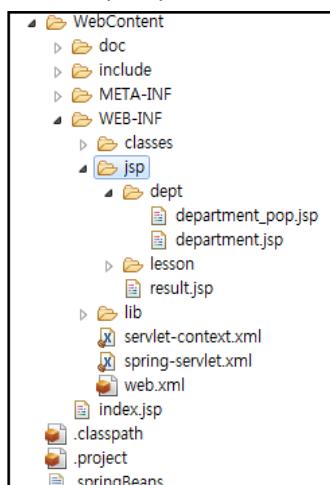
```

        ( DEPT_ID,    DEPT_NAME,   DEPT_TEL)
VALUES
        ( #{deptid}, #{deptname}, #{depttel} )
</insert>

<update id="updateDepartment" parameterType="com.spring.dept.vo.DeptVO">
/* Mapper - updateDepartment */
UPDATE DEPARTMENT
SET DEPT_NAME = #{deptname},
    DEPT_TEL     = #{depttel}
WHERE DEPT_ID = #{deptid}
</update>

<delete id="deleteDepartment" parameterType="com.spring.dept.vo.DeptVO">
/* Mapper - deleteDepartment */
DELETE FROM DEPARTMENT WHERE DEPT_ID = #{deptid}
</delete>
</mapper>
```

- ⑧ “WEB-INF” 폴더에 “jsp” 폴더를 생성한다. 폴더를 선택하고 [New] - [JSP file]을 선택하고, 파일명으로 “department.jsp” 입력한 후, [Finish] 버튼을 클릭한다.



department.jsp (/WEB-INF/jsp/dept/department.jsp)

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>학과 테이블</title>
<script type="text/javascript" src="/include/js/jquery-1.6.4.js"></script>
<script type="text/javascript">
    function insertPopup() {
        $("#deptid").val("");
        window.open("", "pop", "width=480, height=280");
        $("#popupForm").attr("action", "/department/selectDepartment.do");
        $("#popupForm").attr("target", "pop");
        $("#popupForm").submit();
    }

    function updatePopup(deptid) {
        $("#deptid").val(deptid);
        window.open("", "pop", "width=480, height=280");
        $("#popupForm").attr("action", "/department/selectDepartment.do");
        $("#popupForm").attr("target", "pop");
        $("#popupForm").submit();
    }

    function listSearch() {
        $("#searchForm").attr("action", "/department/listDepartment.do");
        $("#searchForm").submit();
    }
</script>
<style type="text/css">
    table{width:570px}
    #searchForm table th{width:114px}
    #searchForm table td{width:104px; text-align:center; }
    #deptname{width:328px}
</style>
</head>
<body>
    <form id="popupForm" name="popupForm" method="post" >
        <input type="hidden" name="deptid" id="deptid"/>
    </form>

```

```

<div>
    <table border="0">
        <tr><th><h4>학과테이블[Spring]</h4></th></tr>
    </table>
</div>
<div>
    <table border="1">
        <colgroup>
            <col width="120px" />
            <col width="220px" />
            <col width="120px" />
            <col width="110px"/>
        </colgroup>
        <thead>
            <tr>
                <th>학과코드</th>
                <th>학과명</th>
                <th>전화번호</th>
                <th>상세보기</th>
            </tr>
        </thead>
        <tbody>
            <c:if test="${empty departmentList}">
                <tr>
                    <td colspan="5" align="center">
                        등록된 학과 정보가 존재하지 않습니다.</td>
                </tr>
            </c:if>
            <c:forEach items="${departmentList}" var="row">
                <tr>
                    <td align="center" >${row.deptid}</td>
                    <td align="left" >${row.deptname}</td>
                    <td align="center" >${row.depttel}</td>
                    <td align="center" >
                        <input type="button" value="[ 수정/삭제 ]"
                               onclick="updatePopup('${row.deptid}')"/>
                    </td>
                </tr>
            </c:forEach>
        </tbody>
    </table>
</div>

```

```

<tr>
    <td colspan="3" align="center">
        처리할 버튼을 선택하세요.</td>
    <td align="center">
        <input type="button" onclick="insertPopup()"
               value="[ 등록 ]"/>
    </td>
</tr>
</tbody>
</table>
</div>
<p></p>
<div><!-- search -->
<form id="searchForm" name="searchForm" method="post">
    <table border="1" cellpadding="1" cellspacing="1">
        <tr>
            <th>학 과 명</th>
            <td><input type="text" id="deptname"
                      name="deptname" /></td>
            <td><input type="button" onclick="listSearch()"
                      value="검색" /></td>
        </tr>
    </table>
</form>
</div>
</body>
</html>

```

[New] - [JSP file]을 선택하고, 파일명으로 "department_pop.jsp" 입력한 후, [Finish] 버튼을 클릭한다.

department_pop.jsp (/WEB-INF/jsp/dept/department_pop.jsp)

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

```

```

<title>학과 테이블 팝업</title>
<style type="text/css">
    .required{color:red;}
    table{width:450px;}
    table th{width:120px; text-align:left;}
    input[type="text"]{width:330px;}
</style>
<script type="text/javascript" src="/include/js/jquery-1.6.4.js"></script>
<script type="text/javascript">
    $(document).ready(function(){
        var mode = "${mode}";
        if(mode == "insert") {
            $('#insertData').attr('disabled',false);
            $('#updateData').attr('disabled',true);
            $('#deleteData').attr('disabled',true);
        } else if(mode == "update") {
            $('#insertData').attr('disabled',true);
            $('#updateData').attr('disabled',false);
            $('#deleteData').attr('disabled',false);
        }
    }

    $("#insertData").click(function(){
        if(!validateForm()) return;
        if(confirm('등록을 진행할까요?')) {
            $("#deptForm").attr("action", "/department/insertDepartment.do");
            $("#deptForm").submit();
        }
    });

    $("#updateData").click(function(){
        if(!validateForm()) return;
        if(confirm('수정을 진행할까요?')) {
            $("#deptForm").attr("action", "/department/updateDepartment.do");
            $("#deptForm").submit();
        }
    });

    $("#deleteData").click(function(){

```

```

        if(confirm('삭제를 진행할까요?')) {
            $("#deptForm").attr("action", "/department/deleteDepartment.do");
            $("#deptForm").submit();
        }
    });

    $("#closeWindow").click(function(){
        window.close();
    });
}

function validateForm() {
    if($("#deptid").val().replace(/\s/g,"") == "") {
        alert('학과코드를 입력해주세요.');
        return false;
    }
    if($("#deptname").val().replace(/\s/g,"") == "") {
        alert('학과명을 입력해주세요.');
        return false;
    }
    return true;
}

```

</script>

</head>

<body>

<p></p>

<div>

<form id="deptForm" name="deptForm" method="post">

<table border="1">

<thead>

<tr>

<td colspan="2" align="center">

<h4>학과 테이블 팝업창 [Spring] </h4></td>

</tr>

</thead>

<tbody>

<tr>

<th>*학과코드</th>

```

<td><input type="text" id="deptid" name="deptid"
           value="${DeptVO.deptid}" /></td>
</tr>
<tr>
    <th><span class="required">*</span>학과명</th>
    <td><input type="text" id="deptname"
               name="deptname" value="${DeptVO.deptname}" />
            </td>
</tr>
<tr>
    <th><span class="required">*</span>전화번호</th>
    <td><input type="text" id="depttel" name="depttel"
               value="${DeptVO.depttel}" /></td>
</tr>
</tbody>
</table>
</form>
</div>
<div>
    <p></p>
    <table border="0">
        <tr align="center">
            <td>
                <input type="button" id="insertData" value="등록" /></td>
            <td>
                <input type="button" id="updateData" value="수정" /></td>
            <td>
                <input type="button" id="deleteData" value="삭제" /></td>
            <td>
                <input type="button" id="closeWindow" value="닫기" /></td>
            </tr>
        </table>
    </div>
</body>
</html>
```

result.jsp (/WEB-INF/jsp/result.jsp)

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
```

```

pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>결과 화면</title>
<script type="text/javascript">
    alert("${result}");
    if("${result}".indexOf("문제") > -1) {
        history.go(-1);
    } else {
        opener.listSearch();
        window.close();
    }
</script>
</head>
<body></body>
</html>

```

index.jsp(/WebContent/index.jsp)

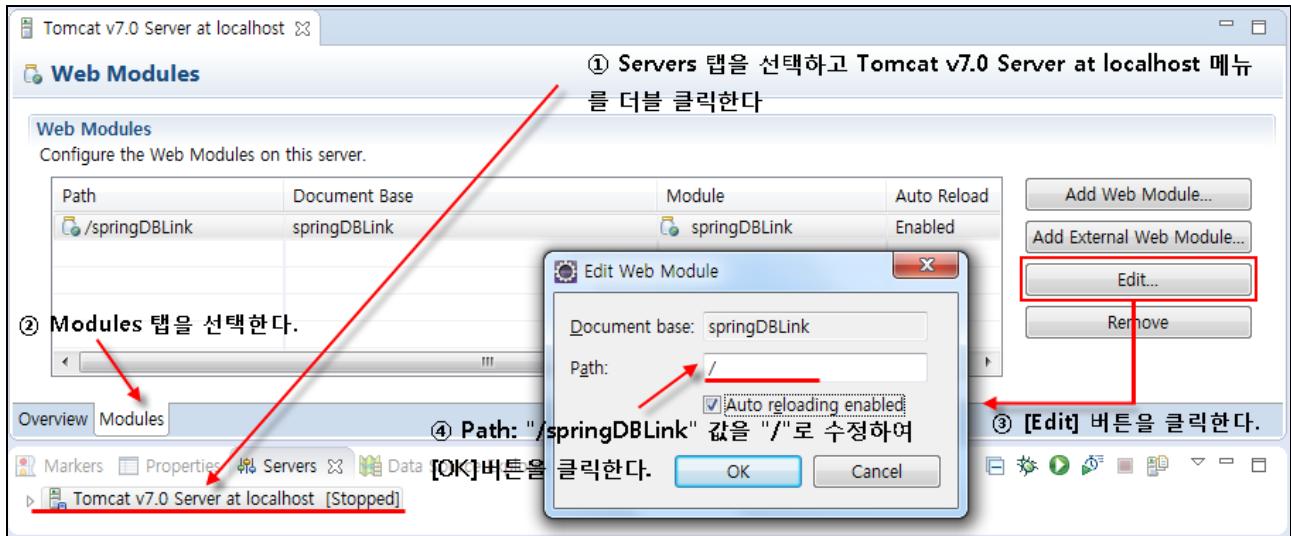
```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>스프링 MVC 데이터베이스 연동</title>
<style type="text/css">
    a{text-decoration:none;}
</style>
</head>
<body>
    <div><a href="department/listDepartment.do">[학과 데이터]</a></div>
</body>
</html>

```

- ⑨ index.jsp 실행하고 난 후 다음과 같이 수정한다. (또는 Add Web Module.. 버튼을 클릭하여 Web

Modules에 path와 Document Base에 /springDBLink 와 springDBLink가 나타나도록 선택한다.)



위 내용을 설정하면 [Servers] - [server.xml] 파일에 다음과 같은 소스가 자동으로 추가된다.

```
<Context docBase="springDBLink" path="/" reloadable="true"  
source="org.eclipse.jst.jee.server:springDBLink"/>
```

3) 두 번째 예제를 위한 테이블 생성

-- 과목테이블 생성

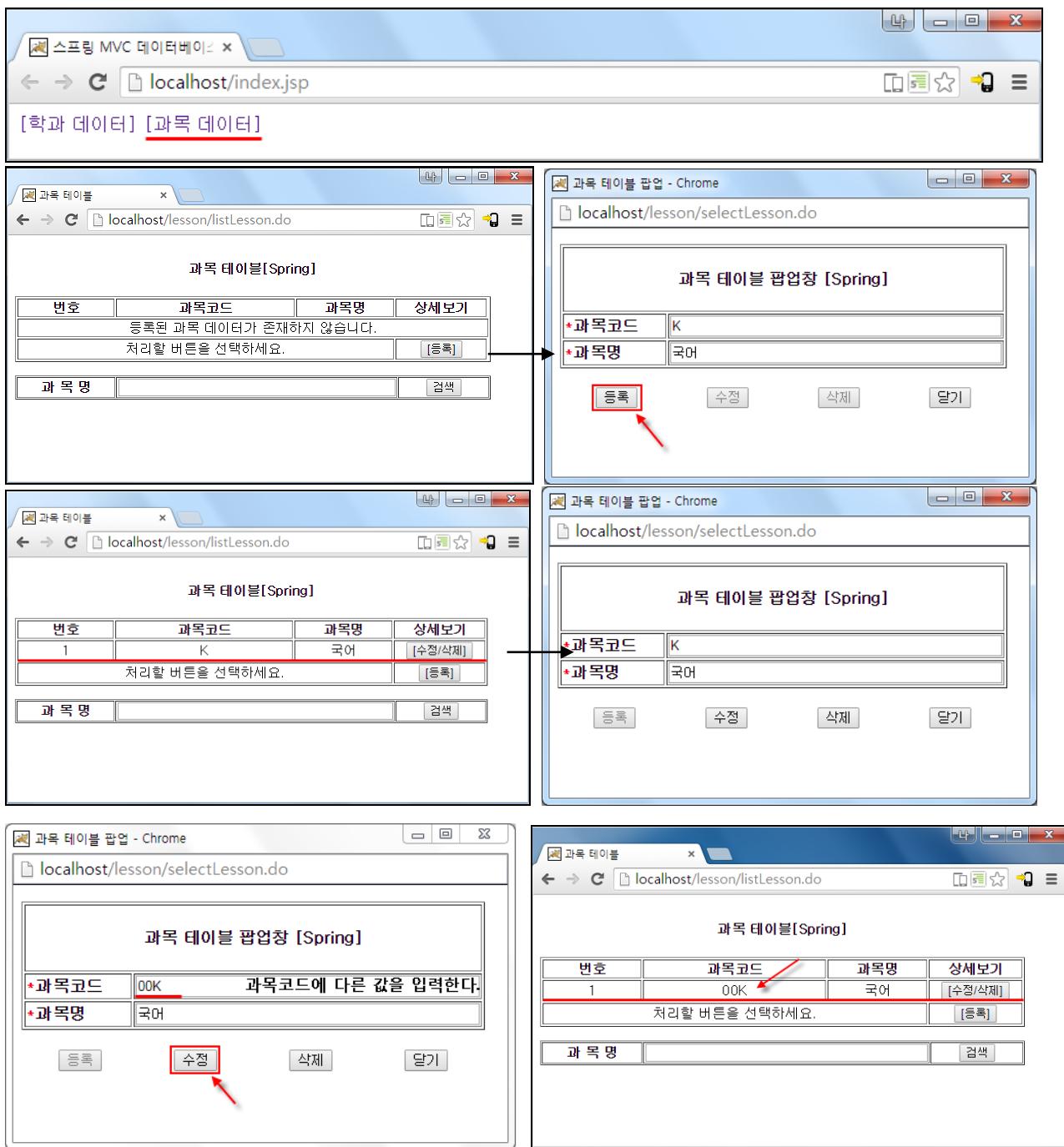
```
create table lesson(  
    no      number      not null,  
    l_num   varchar2(3)  not null,  
    l_name  varchar2(30) not null,  
    constraint lesson_pk primary key(no)  
);
```

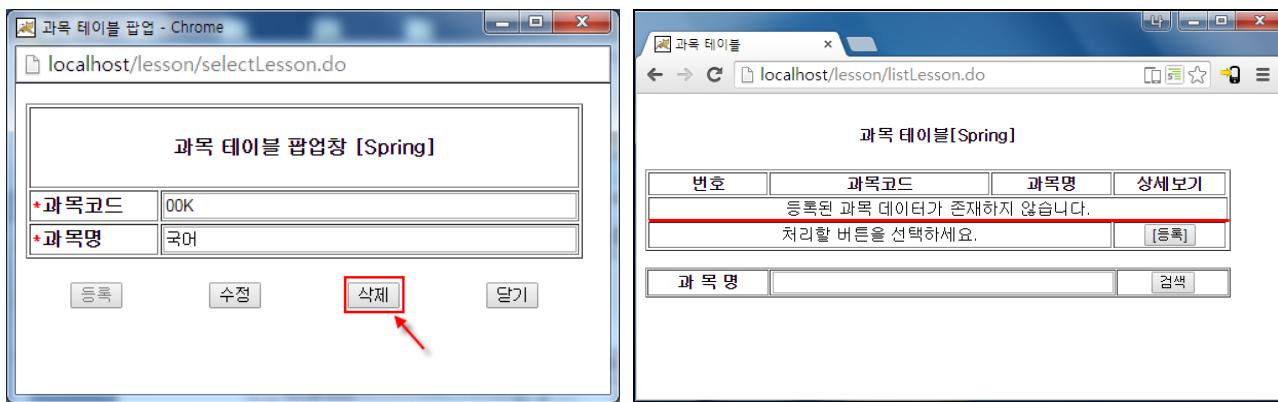
```
comment on table  lesson is '과목테이블 정보';  
comment on column lesson.no is '순번';  
comment on column lesson.l_num is '과목번호';  
comment on column lesson.l_name is '과목명';
```

-- 과목테이블 순번 시퀀스 생성

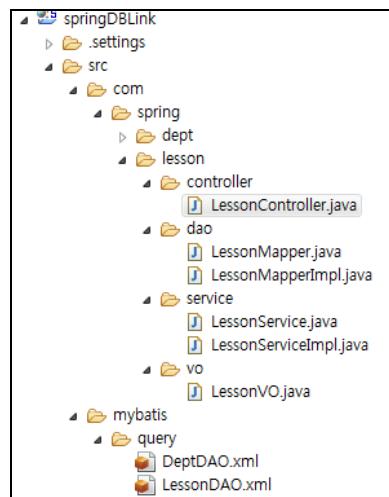
```
create sequence lesson_seq  
start with 1  
increment by 1  
cache 2  
nocycle;
```

· 코딩 후 결과화면





프로그램의 기본 구조는 다음과 같다.



LessonVO.java

```
package com.spring.lesson.vo;

public class LessonVO {
    private int no;
    private String lnum;
    private String lname;

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getLnum() {
        return lnum;
    }
}
```

```

public void setLnum(String lnum) {
    this.lnum = lnum;
}
public String getLname() {
    return lname;
}
public void setLname(String lname) {
    this.lname = lname;
}
}

```

LessonController.java

```

package com.spring.lesson.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import com.spring.lesson.service.LessonService;
import com.spring.lesson.vo.LessonVO;

/* 컨트롤러 */
@Controller
@RequestMapping(value="/lesson")
public class LessonController {
    private static final String CONTEXT_PATH="lesson";

    @Autowired
    private LessonService lessonService;

    /* 전체 조회 */
    @RequestMapping("/listLesson")
    public ModelAndView listLesson(@ModelAttribute LessonVO param) {

        List<LessonVO> list = lessonService.listLesson(param);
    }
}

```

```

        ModelAndView mav = new ModelAndView();
        mav.addObject("lessonList", list);
        mav.addObject("paramName", param.getLname());
        mav.setViewName(CONTEXT_PATH + "/lesson");

        return mav;
    }

/* 상세 정보 보기 */
@RequestMapping("/selectLesson")
public ModelAndView selectLesson(@RequestParam("no") int no) {
    ModelAndView mav = new ModelAndView();
    if(no==0) {
        mav.addObject("mode", "insert");
    } else {
        mav.addObject("lessonVO", lessonService.selectLesson(no));
        mav.addObject("mode", "update");
    }
    mav.setViewName(CONTEXT_PATH + "/lesson_pop");
    return mav;
}

/* 레코드 추가 */
@RequestMapping("/insertLesson")
public ModelAndView insertLesson(@ModelAttribute LessonVO param) {
    String resultStr = "";
    int result = lessonService.insertLesson(param);

    if(result > 0) resultStr = "등록 완료이 완료되었습니다.";
    else resultStr = "등록에 문제가 있어 완료하지 못하였습니다.";

    ModelAndView mav = new ModelAndView();
    mav.addObject("result", resultStr);
    mav.setViewName("/result");

    return mav;
}

```

```

/* 레코드 수정 */
@RequestMapping("/updateLesson")
public ModelAndView updateLesson(@ModelAttribute LessonVO param) {
    String resultStr = "";
    int result = lessonService.updateLesson(param);

    if(result > 0) resultStr = "수정 완료이 완료되었습니다.";
    else resultStr = "수정에 문제가 있어 완료하지 못하였습니다.";

    ModelAndView mav = new ModelAndView();
    mav.addObject("result", resultStr);
    mav.setViewName("/result");

    return mav;
}

/* 레코드 삭제 */
@RequestMapping("/deleteLesson")
public ModelAndView deleteLesson(@RequestParam("no") int no) {
    String resultStr = "";
    int result = lessonService.deleteLesson(no);

    if(result > 0) resultStr = "삭제 완료이 완료되었습니다.";
    else resultStr = "삭제에 문제가 있어 완료하지 못하였습니다.";

    ModelAndView mav = new ModelAndView();
    mav.addObject("result", resultStr);
    mav.setViewName("/result");

    return mav;
}

```

LessonService.java

```

package com.spring.lesson.service;
import java.util.List;
import com.spring.lesson.vo.LessonVO;

```

```
public interface LessonService {  
    public List<LessonVO> listLesson(LessonVO param);  
    public LessonVO selectLesson(int no);  
    public int insertLesson(LessonVO param);  
    public int updateLesson(LessonVO param);  
    public int deleteLesson(int no);  
}
```

LessonServiceImpl.java

```
package com.spring.lesson.service;  
import java.util.ArrayList;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import com.spring.lesson.dao.LessonMapper;  
import com.spring.lesson.vo.LessonVO;
```

@Service

@Transactional

```
public class LessonServiceImpl implements LessonService {
```

@Autowired

```
private LessonMapper lessonMapper;
```

@Override

```
public List<LessonVO> listLesson(LessonVO param) {  
    List<LessonVO> list = new ArrayList<LessonVO>();  
    list = lessonMapper.listLesson(param);  
    return list;  
}
```

@Override

```
public LessonVO selectLesson(int no) {  
    return lessonMapper.selectLesson(no);  
}
```

@Override

```

public int insertLesson(LessonVO param) {
    return lessonMapper.insertLesson(param);
}

@Override
public int updateLesson(LessonVO param) {
    return lessonMapper.updateLesson(param);
}

@Override
public int deleteLesson(int no) {
    return lessonMapper.deleteLesson(no);
}
}

```

LessonMapper.java

```

package com.spring.lesson.dao;
import java.util.List;
import com.spring.lesson.vo.LessonVO;

public interface LessonMapper {
    public List<LessonVO> listLesson(LessonVO param);
    public LessonVO selectLesson(int no);
    public int insertLesson(LessonVO param);
    public int updateLesson(LessonVO param);
    public int deleteLesson(int no);
}

```

LessonMapperImpl.java

```

package com.spring.lesson.dao;
import java.util.List;
import org.mybatis.spring.support.SqlSessionDaoSupport;
import com.spring.lesson.vo.LessonVO;

public class LessonMapperImpl extends SqlSessionDaoSupport implements LessonMapper {

    private final String PACKAGE_PATH = "com.spring.lesson.dao.LessonDAO.";
}

```

```

@Override
public List<LessonVO> listLesson(LessonVO param) {
    return getSqlSession().selectList(PACKAGE_PATH+"listLesson");
}

@Override
public LessonVO selectLesson(int no) {
    return (LessonVO)getSqlSession().selectOne(PACKAGE_PATH+"selectLesson");
}

@Override
public int insertLesson(LessonVO param) {
    return (int)getSqlSession().insert(PACKAGE_PATH+"insertLesson");
}

@Override
public int updateLesson(LessonVO param) {
    return (int)getSqlSession().update(PACKAGE_PATH+"updateLesson");
}

@Override
public int deleteLesson(int no) {
    return (int)getSqlSession().delete(PACKAGE_PATH+"deleteLesson");
}

```

LessonDAO.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-
mapper.dtd">

<mapper namespace="com.spring.lesson.dao.LessonMapper">
    <resultMap type="com.spring.lesson.vo.LessonVO" id="lessonVO">
        <result property="no" column="NO" />
        <result property="lnum" column="L_NUM" />
        <result property="lname" column="L_NAME" />
    </resultMap>

    <select id="listLesson" parameterType="com.spring.lesson.vo.LessonVO" resultMap="lessonVO">

```

```

/* Mapper - listLesson */
    SELECT NO, L_NUM, L_NAME
    FROM LESSON
    <where>
        <if test="lname != null and lname != "">
            L_NAME LIKE '%' || #{lname} || '%'
        </if>
    </where>
    ORDER BY NO DESC
</select>

<select id="selectLesson" parameterType="int" resultMap="lessonVO">
    /* Mapper - selectLesson */
        SELECT NO, L_NUM, L_NAME
        FROM LESSON
        WHERE NO = #{no}
</select>

<insert id="insertLesson" parameterType="com.spring.lesson.vo.LessonVO">
    /* Mapper - insertLesson */
    <selectKey keyProperty="no" resultType="int" order="BEFORE">
        select lesson_seq.nextval from dual
    </selectKey>
    INSERT INTO LESSON
        (NO, L_NUM, L_NAME )
    VALUES
        (#{no}, #{lnum}, #{lname})
</insert>

<update id="updateLesson" parameterType="com.spring.lesson.vo.LessonVO">
    /* Mapper - updateLesson */
    UPDATE LESSON
    SET L_NUM = #{lnum},
        L_NAME = #{lname}
    WHERE NO = #{no}
</update>

<delete id="deleteLesson" parameterType="int">

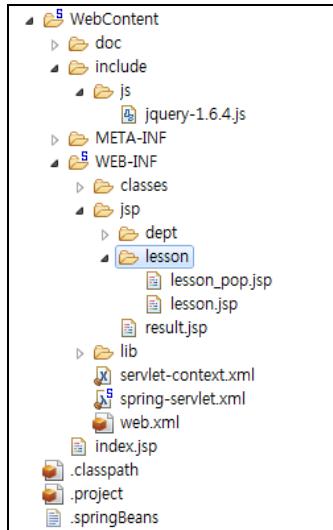
```

```

/* Mapper - deleteLesson */
DELETE FROM LESSON WHERE NO = #{no}
</delete>
</mapper>

```

- ⑧ "WEB-INF" 폴더에 "jsp" 폴더를 생성한다. 폴더를 선택하고 [New] - [JSP file]을 선택하고, 파일명으로 "lesson.jsp" 입력한 후, [Finish] 버튼을 클릭한다.



lesson.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>과목 테이블 </title>
<script type="text/javascript" src="/include/js/jquery-1.6.4.js"></script>
<script type="text/javascript">
    function insertPopup() {
        $("#no").val(0);
        window.open("", "pop", "width=480, height=250");
        $("#popupForm").attr("action", "/lesson/selectLesson.do");
        $("#popupForm").attr("target", "pop");
        $("#popupForm").submit();
    }

```

```

}

function updatePopup(no) {
    $("#no").val(no);
    window.open("", "pop", "width=480, height=250");
    $("#popupForm").attr("action", "/lesson/selectLesson.do");
    $("#popupForm").attr("target", "pop");
    $("#popupForm").submit();
}

function listSearch() {
    $("#searchForm").attr("action", "/lesson/listLesson.do");
    $("#searchForm").submit();
}

```

</script>

```

<style type="text/css">
    table{width:570px}
    #searchForm table th{width:114px}
    #searchForm table td{width:104px; text-align:center; }
    #lname{width:328px}
</style>

```

</head>

<body>

```

    <form id="popupForm" name="popupForm" method="post">
        <input type="hidden" id="no" name="no"/>
    </form>
    <div>
        <table border="0" cellpadding="1" cellspacing="1">
            <tr>
                <th width="650"><h4> 과목 테이블[Spring]</h4></th>
            </tr>
        </table>
    </div>
    <div>
        <table border="1">
            <colgroup>
                <col width="120px" />
                <col width="220px" />

```

```

        <col width="120px" />
        <col width="110px"/>
    </colgroup>
    <thead>
        <tr>
            <th>번호</th>
            <th>과목코드</th>
            <th>과목명</th>
            <th>상세보기</th>
        </tr>
    </thead>
    <tbody>
        <c:if test="${empty lessonList}">
            <tr>
                <td colspan="5" align="center">
                    등록된 과목 데이터가 존재하지 않습니다.</td>
            </tr>
        </c:if>
        <c:forEach items="${lessonList}" var="row">
            <tr align="center">
                <td>${row.no}</td>
                <td>${row.lnum}</td>
                <td>${row.lname}</td>
                <td><input type="button" value="[ 수정/삭제 ]"
                    onclick="updatePopup(${row.no});"/></td>
            </tr>
        </c:forEach>
        <tr>
            <td colspan="3" align="center">
                처리할 버튼을 선택하세요.</td>
            <td align="center">
                <input type="button" value="[ 등록 ]"
                    onclick="insertPopup()"/></td>
            </tr>
        </tbody>
    </table>
</div>
<p></p>

```

```

<div> <!-- search -->
    <form id="searchForm" name="searchForm" method="post">
        <div>
            <table border="1" cellpadding="1" cellspacing="1">
                <tr>
                    <th width="100px">과 목 명</th>
                    <td><input type="text" id="lname"
                        name="lname" /></td>
                    <td><input type="button" value="검색"
                        onclick="listSearch()" /></td>
                </tr>
            </table>
        </div>
    </form>
</div>
</body>
</html>

```

[New] - [JSP file]을 선택하고, 파일명으로 "lesson_pop.jsp" 입력한 후, [Finish] 버튼을 클릭한다.

lesson_pop.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>과목 테이블 팝업</title>
<style type="text/css">
    .required{color:red;}
    table{width:450px;}
    table th{width:120px; text-align:left;}
    input[type="text"]{width:330px;}
</style>
<script type="text/javascript" src="/include/js/jquery-1.6.4.js"></script>
<script type="text/javascript">
    $(document).ready(function(){

```

```

var mode = "${mode}";
if(mode == "insert") {
    $('#insertData').attr('disabled',false);
    $('#updateData').attr('disabled',true);
    $('#deleteData').attr('disabled',true);
} else if(mode == "update") {
    $('#insertData').attr('disabled',true);
    $('#updateData').attr('disabled',false);
    $('#deleteData').attr('disabled',false);
}

$("#insertData").click(function(){
    if(!validateForm()) return;
    if($("#no").val() == "") $("#no").val(0);
    if(confirm('등록을 진행할까요?')) {
        $("#lessonForm").attr("action", "/lesson/insertLesson.do");
        $("#lessonForm").submit();
    }
});

$("#updateData").click(function(){
    if(!validateForm()) return;
    if(confirm('수정을 진행할까요?')) {
        $("#lessonForm").attr("action", "/lesson/updateLesson.do");
        $("#lessonForm").submit();
    }
});

$("#deleteData").click(function(){
    if(confirm('삭제를 진행할까요?')) {
        $("#lessonForm").attr("action", "/lesson/deleteLesson.do");
        $("#lessonForm").submit();
    }
});

$("#closeWindow").click(function(){
    window.close();
});

```

```

    });

function validateForm() {
    if($("#lnum").val().replace(/\s/g,"") == "") {
        alert('과목코드를 입력해주세요.');
        return false;
    }
    if($("#lname").val().replace(/\s/g,"") == "") {
        alert('과목명을 입력해주세요.');
        return false;
    }
    return true;
}

</script>
</head>
<body>
    <p></p>
    <div>
        <form id="lessonForm" name="lessonForm" method="post">
            <input type="hidden" id="no" name="no" value="${lessonVO.no}" />
            <table border="1">
                <thead>
                    <tr>
                        <td colspan="2" align="center">
                            <h4>과목 테이블 팝업창 [Spring] </h4>
                        </td>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <th><span class="required">*</span>과목코드</th>
                        <td><input type="text" id="lnum"
                                name="lnum" value="${lessonVO.lnum}" /></td>
                    </tr>
                    <tr>
                        <th><span class="required">*</span>과목명</th>
                        <td><input type="text" id="lname"
                                name="lname" value="${lessonVO.lname}" /></td>
                    </tr>

```

```

        </tbody>
    </table>
</form>
</div>
<div>
<p></p>
<table border="0">
<tr align="center">
<td>
<input type="button" id="insertData" value="등록" /></td>
<td>
<input type="button" id="updateData" value="수정" /></td>
<td>
<input type="button" id="deleteData" value="삭제" /></td>
<td>
<input type="button" id="closeWindow" value="닫기" /></td>
</tr>
</table>
</div>
</body>
</html>

```

[New] - [JSP file]을 선택하고, 파일명으로 "result.jsp" 입력한 후, [Finish] 버튼을 클릭한다.

result.jsp(/WEB-INF/jsp/result.jsp) - 기존 파일과 동일

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>결과 화면</title>
<script type="text/javascript">
    alert("${result}");
    if("${result}".indexOf("문제") > -1) {
        history.go(-1);
    } else {
        opener.listSearch();
    }
</script>

```

```

        window.close();
    }
</script>
</head>
<body></body>
</html>

```

[New] - [JSP file]을 선택하고, 파일명으로 “index.jsp” 입력한 후, [Finish] 버튼을 클릭한다.

index.jsp (기존 파일에 추가작업)

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>스프링 MVC 데이터베이스 연동</title>
<style type="text/css">
    a{text-decoration:none;}
</style>
</head>
<body>
<div>
    <a href="department/listDepartment.do">[학과 데이터]</a>
    <a href="lesson/listLesson.do">[과목 데이터]</a>
</div>
</body>
</html>

```

Spring-Mybatis 게시판 프로그래밍 예제

1) 테이블을 생성한다.

```
-- 테이블 생성
CREATE TABLE SPRING_BOARD(
    B_NUM NUMBER NOT NULL ,
    B_NAME VARCHAR2(10) NULL ,
    B_TITLE  VARCHAR2(1000) NULL ,
    B_CONTENT CLOB,
    B_FILE VARCHAR2(800),
    B_PWD VARCHAR2(18) NULL,
    B_DATE DATE DEFAULT SYSDATE,
    CONSTRAINT SPRING_BOARD_PK PRIMARY KEY(B_NUM)
);
```

```
COMMENT ON TABLE      SPRING_BOARD IS '게시판 정보';
COMMENT ON COLUMN SPRING_BOARD.B_NUM IS '게시판순번';
COMMENT ON COLUMN SPRING_BOARD.B_NAME IS '게시판작성자';
COMMENT ON COLUMN SPRING_BOARD.B_TITLE IS '게시판제목';
COMMENT ON COLUMN SPRING_BOARD.B_CONTENT IS '게시판내용';
COMMENT ON COLUMN SPRING_BOARD.B_PWD IS '게시판비밀번호';
COMMENT ON COLUMN SPRING_BOARD.B_FILE IS '게시판첨부파일';
COMMENT ON COLUMN SPRING_BOARD.B_DATE IS '게시판등록일';
```

```
CREATE SEQUENCE SPRING_BOARD_SEQ
START WITH 1
INCREMENT BY 1
NOCYCLE;
```

2) 코딩 후 결과화면

The left screenshot shows the '글 목록' (List) page with a table of posts. The right screenshot shows the '글쓰기' (Write) page where a post is being created. A red arrow points from the '내용' (Content) field in the write form to the '내용' (Content) field in the list table.

• 게시물 입력 후 목록 화면으로

A screenshot of the '글 목록' (List) page. The last row in the table represents the new post with ID 5, titled '작게라도 시작해라. 그것은 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카'. The '작성자' (Author) column shows '희성회'.

• 상세 화면

A screenshot of the '글상세 보기' (Detail View) page for the post with ID 5. The post details are displayed, including the title '작게라도 시작해라. 그것은 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카', the author '희성회', and the content '데어났을 때 부터는 모든 것이 첫 경험이다. 그러나 살면서 첫 경험이 점차 없어지지는 않았나 생각해보자. 시작 자체에 의미를 두고 첫 경험에 도전해보자. 편견 없이 선입견없이 자기가 하고 싶은 그 일을 말이다...'. An image of a compass and coins is shown in the '첨부파일 파일명' (Attachment File Name) section.

• 수정버튼 클릭시 화면

A screenshot of the '글상세' (Detail View) page for the post with ID 5. The '수정' (Edit) button is highlighted with a red circle and an arrow. The post details are visible, including the title '작게라도 시작해라. 그것은 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카', the author '희성회', and the content '데어났을 때 부터는 모든 것이 첫 경험이다. 그러나 살면서 첫 경험이 점차 없어지지는 않았나 생각해보자. 시작 자체에 의미를 두고 첫 경험에 도전해보자. 편견 없이 선입견없이 자기가 하고 싶은 그 일을 말이다...'. An image of a compass and coins is shown in the '첨부파일 파일명' (Attachment File Name) section.

• 비밀번호가 일치하지 않을 때 화면

A screenshot of the '글상세' (Detail View) page for the post with ID 5. A red box highlights the error message '비밀번호 : [] 확인 작성시 입력한 비밀번호를 입력해 주세요.' (Password : [] Check Enter the password you entered when creating). The post details are visible, including the title '작게라도 시작해라. 그것은 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카', the author '희성회', and the content '데어났을 때 부터는 모든 것이 첫 경험이다. 그러나 살면서 첫 경험이 점차 없어지지는 않았나 생각해보자. 시작 자체에 의미를 두고 첫 경험에 도전해보자. 편견 없이 선입견없이 자기가 하고 싶은 그 일을 말이다...'. An image of a compass and coins is shown in the '첨부파일 파일명' (Attachment File Name) section.

• 비밀번호 일치 시 수정 화면으로 이동

• 글수정

글수정 화면

작성자: 회성회
글제목: 작게라도 시작해라, 그것을 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카

내용:
태어났을 때 부터는 모든 것이 첫 경험이다. 그러나 살면서 첫 경험이 점차 없어지지는 않았나 생각해보자. 시작 자체에 의미를 두고 첫 경험에 도전해보자. 발견없이 선인경없이 자기가 하고 싶은 그 일을 말이다....
작은 일이라도 새롭게 시작하는 하루가 되길...

첨부파일: 파일선택 [선택된 파일 없음]
비밀번호: 수정할 비밀번호를 입력해 주세요.

[수정] [목록]

글수정 화면

작성자: 회성회
글제목: 작게라도 시작해라, 그것을 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카

내용:
태어났을 때 부터는 모든 것이 첫 경험이다. 그러나 살면서 첫 경험이 점차 없어지지는 않았나 생각해보자. 시작 자체에 의미를 두고 첫 경험에 도전해보자. 발견없이 선인경없이 자기가 하고 싶은 그 일을 말이다....
작은 일이라도 새롭게 시작하는 하루가 되길...

첨부파일: 파일선택 [선택된 파일 없음]
비밀번호: 수정할 비밀번호를 입력해 주세요.

[수정] [목록]

• 상세 화면에서 수정된 내용 확인

글상세 보기

작성자: 회성회 | 작성일: 2015-12-02 07:08:43

제목: 작게라도 시작해라, 그것을 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카

내용:
태어났을 때 부터는 모든 것이 첫 경험이다. 그러나 살면서 첫 경험에 점차 없어지지는 않았나 생각해보자. 시작 자체에 의미를 두고 첫 경험에 도전해보자. 발견없이 선인경없이 자기가 하고 싶은 그 일을 말이다....
작은 일이라도 새롭게 시작하는 하루가 되길...

첨부파일 파일명:

• 글삭제

글상세 보기

작성자: 회성회 | 작성일: 2015-12-02 07:08:43

제목: 작게라도 시작해라, 그것을 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카

내용:
태어났을 때 부터는 모든 것이 첫 경험이다. 그러나 살면서 첫 경험에 점차 없어지지는 않았나 생각해보자. 시작 자체에 의미를 두고 첫 경험에 도전해보자. 발견없이 선인경없이 자기가 하고 싶은 그 일을 말이다....
작은 일이라도 새롭게 시작하는 하루가 되길...

첨부파일 파일명:

비밀번호: [확인] 작성시 입력한 비밀번호를 입력해 주세요.

[수정] [삭제] [목록]

3) 프로젝트 생성 및 설정 작업

- ① [New] - [Dynamic Web Project] 선택하고 Project name은 "springProject" 지정하고 [Next]버튼 클릭, Java 화면에서 Default output folder: "WebContentWEB-INFclasses"로 변경하고 [Next] 버튼 클릭, Web Module 화면에서 "Generate web.xml deployment descriptor" 체크 박스를 클릭하고, [Finish]버튼을 클릭한다.
- ② "springProject" Dynamic Web Project에 스프링 프레임워크를 적용시키기 위해서 "springProject" 프로젝트를 선택하고, 마우스 오른쪽 버튼을 클릭하고, 팝업창에서 [spring Tools] - [Add Spring Project Nature] 메뉴를 차례로 클릭한다.
- ③ "springProject"의 "/WEB-INF/lib" 폴더에 스프링 프레임워크의 jar 파일과 프로그래밍에 관련된 jar 파일을 넣어야 한다.

일들을 붙여넣기한다.

- ④ 커넥션 풀을 이용한 데이터베이스 연동을 위한 설정이 제대로 되어 있는지 확인한다.

[Servers] – [context.xml] 파일에 <Resource> 요소가 정의되어 있는지 확인.

```
<Context>
...
<Resource auth="Container" driverClassName="oracle.jdbc.driver.OracleDriver" maxActive="100"
maxIdle="30" maxWait="10000" name="jdbc/Oracle11g" password="tiger"
type="javax.sql.DataSource" url="jdbc:oracle:thin:@127.0.0.1:1521:orcl" username="scott"/>
</Context>
```

- ⑤ 웹 애플리케이션 환경 설정 파일에서 디스패처 서블릿 선언 및 한글 처리를 위한 코딩을 추가한다.

/WEB-INF/**web.xml** 웹 애플리케이션 환경 설정 파일에 추가

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="boardProject" version="3.0">
    <display-name>boardProject</display-name>
    <welcome-file-list>
        <welcome-file>default.html</welcome-file>
    </welcome-file-list>
    <!-- ===== 스프링 관련 설정 ===== -->
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/framework/*.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    <!-- ===== 한글 처리 ===== -->
    <filter>
        <filter-name>encodingFilter</filter-name>
```

```

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
<init-param>
    <param-name>encoding</param-name>
    <param-value>euc-kr</param-value>
</init-param>
<init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
</init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<!-- 에러 발생 시 보여주고자 하는 파일 설정 (게시판이 다 작성되고 난 후 주석 해제한다)
&lt;error-page&gt;
    &lt;error-code&gt;403&lt;/error-code&gt;
    &lt;location&gt;/common/error/error.html&lt;/location&gt;
&lt;/error-page&gt;
&lt;error-page&gt;
    &lt;error-code&gt;404&lt;/error-code&gt;
    &lt;location&gt;/common/error/error.html&lt;/location&gt;
&lt;/error-page&gt;
&lt;error-page&gt;
    &lt;error-code&gt;405&lt;/error-code&gt;
    &lt;location&gt;/common/error/error.html&lt;/location&gt;
&lt;/error-page&gt;
&lt;error-page&gt;
    &lt;error-code&gt;500&lt;/error-code&gt;
    &lt;location&gt;/common/error/error.html&lt;/location&gt;
&lt;/error-page&gt;
--&gt;
&lt;context-param&gt;
    &lt;param-name&gt;log4jConfigLocation&lt;/param-name&gt;
    &lt;param-value&gt;/WEB-INF/classes/log4j.properties&lt;/param-value&gt;
&lt;/context-param&gt;
&lt;/web-app&gt;
</pre>

```

※ 예전에 작성한 log4j.properties 파일을 src 폴더에 붙여넣기하면 된다. 그리고 진하게 표시된 mybatis 설정 부분만 추가하면 된다.

src/log4j.properties

```
#-----
# Log4j 설정파일
# 두개의 Appender를 사용하여, 하나는 콘솔에, 하나는 파일에 로깅한다.
#-----

# root category의 레벨(priority)를 DEBUG로 설정한다.
log4j.rootCategory=info, stdout, logfile

#log4j를 설정하는 상세 정보 출력여부(true/false)
log4j.debug=false
#-----

# 첫번째 appender: 콘솔에 로깅
#-----


log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.ImmediateFlush=true
log4j.appender.stdout.Target=System.err

#출력 Layout설정 : 호출하는 파일명 출력
log4j.appender.stdout.layout.ConversionPattern=[%d] [%p] (%13F:%L) %3x - %m%n
#-----


# 두번째 appender : 파일에 로깅
#-----


#log4j.appender.logfile=org.apache.log4j.DailyRollingFileAppender
#log4j.appender.logfile.ImmediateFlust=true
#log4j.appender.logfile.File=C:/log/logfile.txt
#log4j.appender.logfile.Append=true

#파일명 패턴
#log4j.appender.logfile.DatePattern = ':yyyy-MM-dd

#출력 Layout설정 : 호출하는 시간, 파일명등 출력
#log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
#log4j.appender.logfile.layout.ConversionPattern=[%d] [%p] (%13F:%L) %3x - %m%n

log4j.logger.java.sql.Connection=INFO
```

```

log4j.logger.java.sql.Statement=INFO
log4j.logger.java.sql.PreparedStatement=INFO
log4j.logger.java.sql.ResultSet=INFO

# mybatis 설정 부분
log4j.logger.com.spring.board.dao=TRACE
#댓글 작성시 아래 주석 해제.
#log4j.logger.com.spring.reply.dao=TRACE

```

⑥ 스프링 설정 파일 생성(1) – MVC 패턴

/WEB-INF/framework/spring-servlet.xml 스프링 설정 파일

```

<?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:c="http://www.springframework.org/schema/c"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd
                           http://www.springframework.org/schema/mvc
                           http://www.springframework.org/schema/mvc-mvc.xsd">

    <!-- 어노테이션 기반 설정 활성화 -->
    <context:annotation-config />

    <mvc:annotation-driven />

    <!-- 어노테이션으로 표시된 클래스를 자동으로 가져오는 기능
         (클래스 검색을 실시할 대상 패키지 지정) -->
    <context:component-scan base-package="com.spring.**" />

    <!-- 논리적인 뷰를 실제 뷰로 변환하는 기능 -->
    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
    
```

```

<property name="prefix" value="/WEB-INF/jsp/" />
<property name="suffix" value=".jsp" />
</bean>

<!-- 첨부파일을 위한 설정-->
<bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="defaultEncoding" value="UTF-8"/>
    <property name="maxUploadSize" value="5242880"/>
</bean>
</beans>

```

⑥ 스프링 설정 파일 생성(2) – 데이터베이스 연동

/WEB-INF/framework/servlet-context.xml 스프링 설정 파일

```

<?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:c="http://www.springframework.org/schema/c"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/mvc
                           http://www.springframework.org/schema/p
                           http://www.springframework.org/schema/c
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context/spring-context.xsd
                           http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- DB Connection -->
    <!-- DataSource -->
    <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName" value="java:comp/env/jdbc/Oracle11g" />
    </bean>

    <!-- Mybatis Setting -->

```

```

<!--
    - SqlSessionFactory는 데이터베이스와의 연결과 SQL의 실행에 대한 모든 것을 가진 가장
        중요한 객체이다.
    - SqlSessionFactory를 생성해 주는 객체를 설정해 주어야 하는데 SqlSessionFactoryBean이
        라는 클래스이다. -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="configLocation" value="classpath:config/mybatis-config.xml"/>
    <property name="mapperLocations" value="classpath:query/*.xml"/>
</bean>

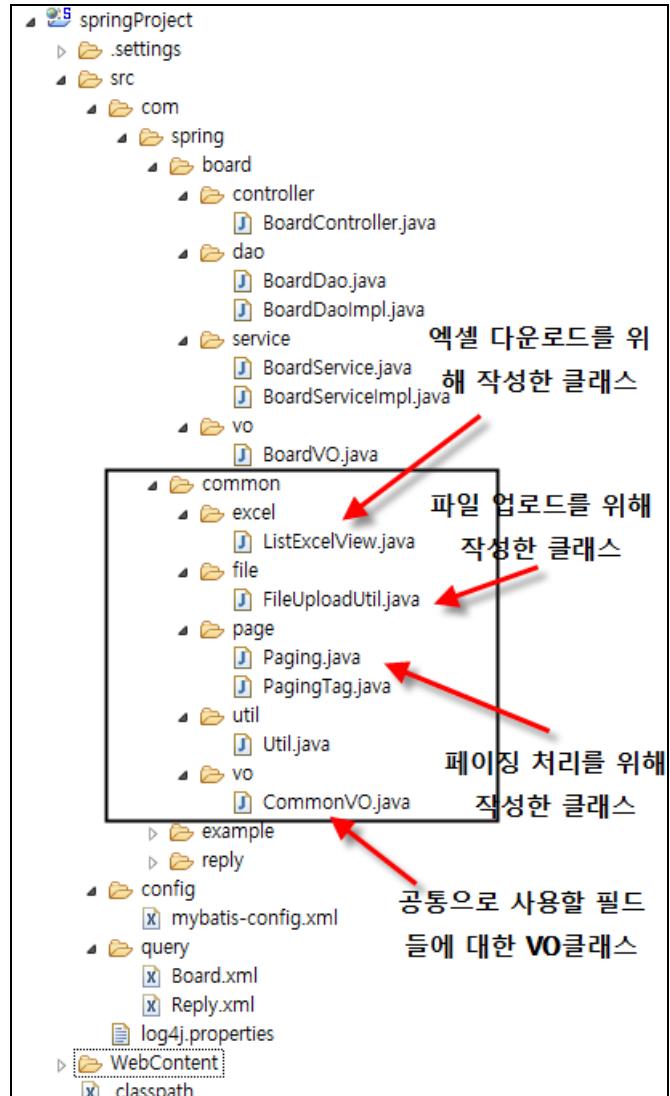
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate"
      destroy-method="clearCache">
    <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>

<!-- a PlatformTransactionManager is still required -->
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" > <value>com.spring.**.dao</value> </property>
</bean>
</beans>

```

- src 이하 구조(java, Mybatis 설정파일, mapper 파일 등)는 다음과 같다.



⑦ VO 클래스, 컨트롤러 클래스, 서비스(모델) 클래스, DAO 클래스, 매퍼 XML 소스파일을 작성하여 보자.

BoardVO.java

```
package com.spring.board.vo;

import org.springframework.web.multipart.MultipartFile;

public class BoardVO{
    private int b_num      =0; // 글번호
    private String b_name   =""; // 작성자
    private String b_title   =""; // 제목
    private String b_content  =""; // 내용
    private String b_date   =""; // 작성일
    private String b_pwd     =""; // 비밀번호

    // 파일 업로드를 위한 속성
}
```

```
private MultipartFile file; //첨부파일
private String b_file = ""; //실제서버에 저장한 파일명

public int getB_num() {
    return b_num;
}

public void setB_num(int b_num) {
    this.b_num = b_num;
}

public String getB_name() {
    return b_name;
}

public void setB_name(String b_name) {
    this.b_name = b_name;
}

public String getB_title() {
    return b_title;
}

public void setB_title(String b_title) {
    this.b_title = b_title;
}

public String getB_content() {
    return b_content;
}

public void setB_content(String b_content) {
    this.b_content = b_content;
}

public String getB_date() {
    return b_date;
}
```

```

public void setB_date(String b_date) {
    this.b_date = b_date;
}

public String getB_pwd() {
    return b_pwd;
}

public void setB_pwd(String b_pwd) {
    this.b_pwd = b_pwd;
}

public MultipartFile getFile() {
    return file;
}

public void setFile(MultipartFile file) {
    this.file = file;
}

public String getB_file() {
    return b_file;
}

public void setB_file(String b_file) {
    this.b_file = b_file;
}
}

```

- 파일 업로드를 위한 클래스 파일을 작성한다.

src/com/spring/common/file/FileUploadUtil.java

```

package com.spring.common.file;

import java.io.File;
import java.io.IOException;
import javax.servlet.http.HttpServletRequest;
import org.apache.log4j.Logger;
import org.springframework.web.multipart.MultipartFile;

```

```

public class FileUploadUtil {
    static Logger logger = Logger.getLogger(FileUploadUtil.class);

    /* 파일 업로드 메서드 */
    public static String fileUpload(MultipartFile file, HttpServletRequest request)
                                                throws IOException{

        logger.info("fileUpload 호출 성공 ");
        String real_name=null;

        // MultipartFile 클래스의 getFile() 메서드로 클라이언트가 업로드한 파일명
        String org_name = file.getOriginalFilename();
        logger.info("org_name :" +org_name);

        // 파일명 변경(중복되지 않게)
        if(org_name != null && (!org_name.equals(""))){
            real_name ="board_"+System.currentTimeMillis() +"_"+ org_name;
            String docRoot
                = request.getSession().getServletContext().getRealPath("/uploadStorage");

            File fileDir =new File(docRoot);
            if(!fileDir.exists()){
                fileDir.mkdir();
            }

            File fileAdd =new File(docRoot+"/"+real_name);
            logger.info("경로 :" + fileAdd);

            file.transferTo(fileAdd);
        }
        return real_name;
    }

    /* 파일 삭제 메서드 */
    public static void fileDelete(String fileName, HttpServletRequest request)
                                                throws IOException{

        logger.info("fileDelete 호출 성공 ");
        boolean result=false;
        String docRoot

```

```

        = request.getSession().getServletContext().getRealPath("/uploadStorage");

        File fileDelete =new File(docRoot+"/"+fileName);
        logger.info(fileDelete);
        if(fileDelete.exists() && fileDelete.isFile()){
            result = fileDelete.delete();
        }
        logger.info(result);
    }
}

```

BoardController.java

```

package com.spring.board.controller;

import java.io.IOException;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotationModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.servlet.ModelAndView;
import com.spring.board.service.BoardService;
import com.spring.board.vo.BoardVO;
import com.spring.common.file.FileUploadUtil;

```

@Controller

```
@RequestMapping(value="/board")
```

```
public class BoardController {
```

```
    Logger logger = Logger.getLogger(BoardController.class);
```

@Autowired

```
private BoardService boardService;
```

```
/*****************************************************************************
```

```

* 글목록 구현하기
*****
@RequestMapping(value="/boardList", method = RequestMethod.GET)
public String boardList(@ModelAttribute BoardVO bvo, Model model) {
    logger.info("boardList 호출 성공");
    List<BoardVO> boardList = boardService.boardList(bvo);

    model.addAttribute("boardList", boardList);
    return "board/boardList";
}

*****
* 글쓰기 폼 출력하기
*****
@RequestMapping(value="/writeForm", method=RequestMethod.GET)
public String writeForm(){
    logger.info("writeForm 호출 성공");

    return "board/writeForm";
}

*****
* 글쓰기 구현하기
*****
@RequestMapping(value="/boardInsert", method=RequestMethod.POST)
public String boardInsert(@ModelAttribute BoardVO bvo,HttpServletRequest request)
        throws IllegalStateException, IOException{
    logger.info("boardInsert 호출 성공");
    logger.info("fileName : " + bvo.getFile().getOriginalFilename());
    logger.info("b_title : " + bvo.getB_title());

    int result = 0;
    String url ="";

    String b_file = FileUploadUtil.fileUpload(bvo.getFile(),request);
    bvo.setB_file(b_file);

    result = boardService.boardInsert(bvo);
}

```

```

        if(result == 1){
            url = "/board/boardList.do";
        }
        return "redirect:"+url;
    }

/*****
 * 글 상세보기 구현
 *****/
@RequestMapping(value="/boardDetail", method=RequestMethod.GET)
public String boardDetail(@ModelAttribute BoardVO pvo, Model model) {
    logger.info("boardDetail 호출 성공");
    logger.info("b_num = " + pvo.getB_num());

    BoardVO detail = new BoardVO();
    detail = boardService.boardDetail(pvo);

    if(detail!=null && (!detail.equals(""))){
        detail.setB_content(detail.getB_content().toString().replaceAll("\n", "<br>"));
    }

    model.addAttribute("detail", detail);
    return "board/boardDetail";
}

/*****
 * 비밀번호 확인
 * @param b_num
 * @param b_pwd
 * @return int
 * 참고 : @ResponseBody는 전달된 뷰를 통해서 출력하는 것이 아니라
 *       HTTP Response Body에 직접 출력하는 방식.
 *****/
@ResponseBody
@RequestMapping(value="/pwdConfirm", method=RequestMethod.POST)
public String pwdConfirm(@ModelAttribute BoardVO bvo) {
    logger.info("pwdConfirm 호출 성공");
}

```

```

//아래 변수에는 입력 성공에 대한 상태값 저장(1 or 0)
int result = 0;
result = boardService.pwdConfirm(bvo);

logger.info("result = " + result);
return result+ "";
}

/*************************************
 * 글수정 폼 출력하기
 * @param : b_num
 * @return : BoardVO
*************************************/

@RequestMapping(value="/updateForm", method=RequestMethod.POST)
public String updateForm(@ModelAttribute BoardVO pvo, Model model) {
    logger.info("updateForm 호출 성공");
    logger.info("b_num = " + pvo.getB_num());

    BoardVO updateData = new BoardVO();
    updateData = boardService.boardDetail(pvo);

    model.addAttribute("updateData", updateData);
    return "board/updateForm";
}

/*************************************
 * 글수정 구현하기
*************************************/

@RequestMapping(value="/boardUpdate", method=RequestMethod.POST)
public String boardUpdate(@ModelAttribute BoardVO bvo, HttpServletRequest request)
        throws IllegalStateException, IOException{
    logger.info("boardUpdate 호출 성공");

    int result=0;
    String url="";
    String b_file="";

    if(!bvo.getFile().isEmpty()){

```

```

        FileUploadUtil.fileDelete(bvo.getB_file(), request);
        b_file = FileUploadUtil.fileUpload(bvo.getFile(), request);
        bvo.setB_file(b_file);
    }else{
        logger.info("첨부파일 없음");
        bvo.setB_file("");
    }
    logger.info("b_file = " +bvo.getB_file());
    result = boardService.boardUpdate(bvo);

    if(result == 1){
        url="/board/boardList.do"; // 수정 후 목록으로 이동
        //아래 url은 수정 후 상세 페이지로 이동
        //url="/board/boardDetail.do?b_num="+bvo.getB_num();
    }
    return "redirect:"+url;
}

/*************************************
 * 글삭제 구현하기
 * @throws IOException
*************************************/

@RequestMapping(value="/boardDelete")
public String boardDelete(@ModelAttribute BoardVO bvo, HttpServletRequest request)
        throws IOException {
    logger.info("boardDelete 호출 성공");

    //아래 변수에는 삭제 성공에 대한 상태값 담습니다.(1 or 0)
    int result = 0;
    String url = "";

    FileUploadUtil.fileDelete(bvo.getB_file(), request);
    result = boardService.boardDelete(bvo.getB_num());
    if(result == 1){
        url="/board/boardList.do";
    }
    return "redirect:"+url;
}

```

```
}
```

BoardService.java

```
package com.spring.board.service;  
  
import java.util.List;  
  
import com.spring.board.vo.BoardVO;  
  
public interface BoardService {  
    public List<BoardVO> boardList(BoardVO pvo);  
    public int boardInsert(BoardVO pvo);  
    public BoardVO boardDetail(BoardVO pvo);  
    public int pwdConfirm(BoardVO pvo);  
    public int boardUpdate(BoardVO pvo);  
    public int boardDelete(int b_num);  
}
```

BoardServiceImpl.java

```
package com.spring.board.service;  
  
import java.util.List;  
  
import org.apache.log4j.Logger;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import com.spring.board.dao.BoardDao;  
import com.spring.board.vo.BoardVO;  
  
@Service  
@Transactional  
public class BoardServiceImpl implements BoardService {  
    Logger logger = Logger.getLogger(BoardServiceImpl.class);  
  
    @Autowired  
    private BoardDao boardDao;  
  
    // 글목록 구현  
    @Override  
    public List<BoardVO> boardList(BoardVO pvo) {  
        List<BoardVO> myList = null;
```

```
myList = boardDao.boardList(pvo);
return myList;
}

// 글입력 구현
@Override
public int boardInsert(BoardVO pvo) {
    int result = 0;
    result = boardDao.boardInsert(pvo);
    return result;
}

// 글상세 구현
@Override
public BoardVO boardDetail(BoardVO pvo) {
    BoardVO detail = null;
    detail = boardDao.boardDetail(pvo);
    return detail;
}

// 글수정 구현
@Override
public int boardUpdate(BoardVO pvo) {
    int result = 0;
    result = boardDao.boardUpdate(pvo);
    return result;
}

// 비밀번호 확인 구현
@Override
public int pwdConfirm(BoardVO pvo) {
    int result = 0;
    result = boardDao.pwdConfirm(pvo);
    return result;
}

// 글삭제 구현
@Override
```

```

public int boardDelete(int b_num) {
    int result = 0;
    result = boardDao.boardDelete(b_num);
    return result;
}
}

```

BoardDao.java

```

package com.spring.board.dao;
import java.util.List;
import com.spring.board.vo.BoardVO;

public interface BoardDao {
    public List<BoardVO> boardList(BoardVO pvo);
    public int boardInsert(BoardVO pvo);
    public BoardVO boardDetail(BoardVO pvo);
    public int pwdConfirm(BoardVO pvo);
    public int boardUpdate(BoardVO pvo);
    public int boardDelete(int b_num);
}

```

BoardDaolmpl.java

```

package com.spring.board.dao;
import java.util.List;
import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import com.spring.board.vo.BoardVO;

```

@Repository

```

public class BoardDaolmpl implements BoardDao {
    @Autowired
    private SqlSession session;

    // 글목록 구현
    @Override
    public List<BoardVO> boardList(BoardVO pvo) {
        return session.selectList("boardList");
    }
}

```

```

}

// 글상세 구현
@Override
public BoardVO boardDetail(BoardVO pvo){
    return (BoardVO)session.selectOne("boardDetail");
}

// 글입력 구현
@Override
public int boardInsert(BoardVO pvo){
    return session.insert("boardInsert");
}

// 비밀번호 확인 구현
@Override
public int pwdConfirm(BoardVO pvo) {
    return (Integer)session.selectOne("pwdConfirm");
}

// 글수정 구현
@Override
public int boardUpdate(BoardVO pvo) {
    return session.update("boardUpdate");
}

// 글삭제 구현
@Override
public int boardDelete(int b_num) {
    return session.delete("boardDelete",b_num);
}
}

```

- mybatis 설정 파일

src/config/mybatis-config.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration

```

```

PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
        <typeAlias type="com.spring.board.vo.BoardVO" alias="board"/>
        <!-- 댓글 작성 시 ReplyVO 클래스 생성 후 주석 해제 -->
        <!-- <typeAlias type="com.spring.reply.vo.ReplyVO" alias="reply"/> -->
    </typeAliases>
</configuration>

```

src/query/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="com.spring.board.dao.BoardDao">
    <!-- 게시물 전체 조회 -->
    <select id="boardList" parameterType="board" resultType="board">
        select b_num, b_name, b_title,
               to_char(b_date,'YYYY-MM-DD') as b_date
        from spring_board
    </select>

    <!-- 상세 페이지를 위한 게시물 조회 -->
    <select id="boardDetail" parameterType="int" resultType="board">
        /* Board - boardDetail */
        SELECT
            b_num, b_name, b_title, b_content, b_file,
            TO_CHAR(b_date,'YYYY-MM-DD HH24:MI:SS') AS b_date
        FROM spring_board
        WHERE b_num = #{b_num}
    </select>

    <!-- 게시물 등록 -->
    <insert id="boardInsert" parameterType="board" >
        /* Board - insertBoard */
        <selectKey keyProperty="b_num" resultType="int" order="BEFORE">
            select spring_board_seq.nextval from dual
        </selectKey>
        INSERT INTO spring_board(

```

```

        b_num,
        b_name,
        b_title,
        b_content,
        b_file,
        b_pwd
    ) VALUES(
        #{b_num}
        ,#{b_name}
        ,#{b_title}
        ,#{b_content}
        ,#{b_file}
        ,#{b_pwd}

    )
</insert>

<!-- 해당 번호의 비밀번호 확인 -->
<select id="pwdConfirm" parameterType="board" resultType="int">
    /* Board - pwdConfirm */
    SELECT NVL(
        SELECT 1 FROM spring_board
        WHERE b_num = #{b_num} AND b_pwd = #{b_pwd}
        ), 0) as result
    FROM dual
</select>

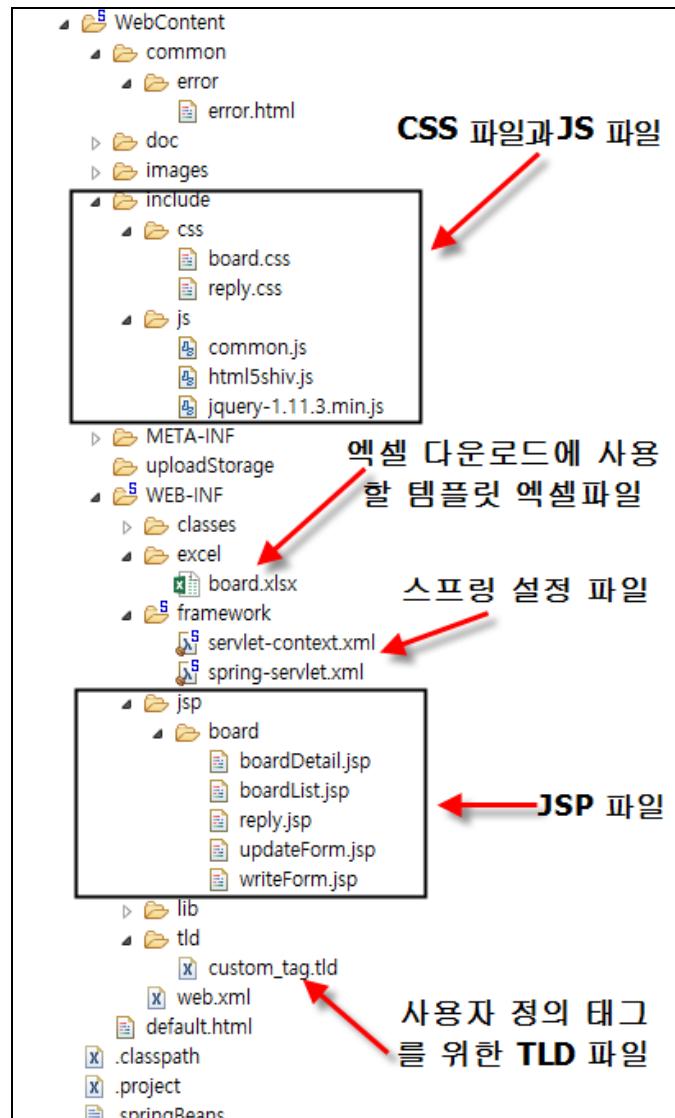
<!-- 게시물 수정 -->
<update id="boardUpdate" parameterType="board" >
    /* Board - boardUpdate */
    UPDATE spring_board SET
        b_title = #{b_title}
        , b_name = #{b_name}
        , b_content = #{b_content}
        , b_date = sysdate
    <if test="b_file != ''">
        , b_file = #{b_file}
    </if>

```

```
<if test="b_pwd != null and b_pwd != "">
    , b_pwd = #{b_pwd}
</if>
WHERE b_num = #{b_num}
</update>

<!-- 게시물 삭제 -->
<delete id="boardDelete" parameterType="board" >
    /* Board - boardDelete */
    DELETE FROM spring_board WHERE b_num = #{b_num}
</delete>
</mapper>
```

- ⑧ WebContent 이하 구조는 다음과 같다.



※ jsp 파일을 생성하기 전에 jsp 파일에서 사용할 css 파일과 js 파일을 생성한다. jquery-xxx.js 파일(이 파일 내용은 소스에서 제외)은 각자 가지고 있는 파일로 사용하면 된다.

/include/css/board.css

```
@CHARSET "EUC-KR";
/* 공통 부분*/
*{margin:0px; padding:0px;}
body, td, input, select{font-family:'돋움'; font-size:12px; }
a {text-decoration:none; color:#000000;}
caption{display:none;}
img{border:0px;}
```

/* 게시판 리스트 화면 */

```
/* 제목 부분 */
```

```

#boardContainer {width:800px;}
#boardTit{font-size: large; border-left: 10px solid #000066;
           border-bottom: 1px solid #000066;
           padding: 10px; width:760px;}
/* 검색 부분 */
#boardSearch table{width:780px; text-align:right; margin:10px; border-collapse:collapse;}
#boardSearch #btd1{width:390px; text-align:left;}
#boardSearch #btd2{width:390px; text-align:right;}

/* 리스트 부분 */
#boardList{margin:10px;}
#boardList table{width:780px; padding:0px; border:1px solid #000066;
                 border-bottom:0px; border-collapse:collapse;}
#boardList tr{padding:0px;}
#boardList th{border-bottom:1px solid #000066; border-right:1px solid #000066;
              padding:5px; background-color:#e7e7e7;}
#boardList td{padding:5px; border-bottom:1px solid #000066;}
#boardList .borcle{border-right:0px; }

/* 버튼 부분 */
#boardContainer #boardBut{width:780px; margin:10px; text-align:right; }

/* 페이지 출력 div 부분 */
#boardContainer #boardPage{width:780px; margin:10px; }

/* 페이지 버튼 부분 */
.paginate{width:100%; height:20px; text-align:center; vertical-align:top; font-size:14px;}
.paginate span .textAn{vertical-align:top;}
.paginate a.on{color:#FF3300; }

/* 게시판 입력 화면 */
#boardWrite{width:780px; padding:0px; border:1px solid #000066; border-bottom:0px;
            margin:10px; border-collapse:collapse;}
#boardWrite td{padding:5px; border-bottom:1px solid #000066;}
input[type="text"],input[type="password"]{border:1px solid #000066; width:150px}
#b_title{width:480px}
#b_content{border:1px solid #000066; width:150px; width:480px; height:200px; }
#boardBut{width:800px; text-align:right; margin-top:15px; }

```

```

#tel_no1,#tel_no2,#tel_no3{width:50px;}
#mop_no1,#mop_no2,#mop_no3{width:50px;}
#idCheckMsg{font-weight:bold; }

/* 게시판 상세 화면 */
#boardPwdBut{width:780px; /* margin:10px; */ margin-left:8px;}
#boardPwdBut #btd1{width:580px; text-align:left; vertical-align: bottom;}
#boardPwdBut #btd2{width:180px; text-align:right; vertical-align: bottom;}
#boardPwdBut input{margin-bottom:0px; margin-left:5px;}
#msg{padding-left:4px; vertical-align:middle; font-weight:bold;}
#pwdChk #l_pwd{vertical-align:bottom;}
#pwdChk #b_pwd{border:1px dotted #000066; }

#boardDetail table{width:770px; padding:0px; border:1px solid #000066;
border-bottom:0px; border-left:0px; margin:10px; border-collapse:collapse;}
#boardDetail td{padding:5px; border-bottom:1px solid #000066;
border-left:1px solid #000066; }

#boardDetail .vm{vertical-align:middle;}
#boardDetail .ctr{height:150px; vertical-align:top}
.ac{text-align:center; background-color: #ebebeb; font-weight:bold;}

```

/include/js/common.js

```

/* chkSubmit(유효성 검사 대상, 메시지 내용) */
function chkSubmit(v_item,v_msg) {
    if( v_item.val().replace(/\s/g,"")=="") {
        alert(v_msg+" 확인해 주세요.");
        v_item.val("");
        v_item.focus();
        return false;
    } else {
        return true;
    }
}

```

- ⑨ "WEB-INF" 폴더에 "jsp" 폴더를 생성한다. 폴더를 선택하고 [New] - [JSP file]을 선택하고, 파일명으로 "boardList.jsp" 입력한 후, [Finish] 버튼을 클릭한다.

/WEB-INF/jsp/board/boardList.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="EUC-KR">
    <title>글 목록</title>
    <link rel="stylesheet" type="text/css" href="/include/css/board.css" />
    <script type="text/javascript" src="/include/js/jquery-1.11.3.min.js"></script>
    <script type="text/javascript">
        $(function(){
            /* 글쓰기 버튼 클릭 시 처리 이벤트 */
            $("#writeForm").click(function(){
                location.href = "/board/writeForm.do";
            });

            /* 제목 클릭시 상세 페이지 이동을 위한 처리 이벤트 */
            $(".goDetail").click(function(){
                var b_num = $(this).parents("tr").attr("data-num");
                $("#b_num").val(b_num);
                // 상세 페이지로 이동하기 위해 form 추가 (id : detailForm)
                $("#detailForm").attr({
                    "method":"get",
                    "action":"/board/boardDetail.do"
                });
                $("#detailForm").submit();
            });
        });
    </script>
</head>
<body>
    <div id="boardContainer">
        <div id="boardTit"><h3>글목록</h3></div>
        <!-- 상세 페이지 이동을 위한 form -->
        <form name="detailForm" id="detailForm">
```

```

<input type="hidden" name="b_num" id="b_num">
</form>

<%-- ===== 리스트 시작 ===== --%>
<div id="boardList">
<table summary="게시판 리스트">
    <colgroup>
        <col width="10%" />
        <col width="62%" />
        <col width="15%" />
        <col width="13%" />
    </colgroup>
    <thead>
        <tr>
            <th>글번호</th>
            <th>글제목</th>
            <th>작성일</th>
            <th class="borcle">작성자</th>
        </tr>
    </thead>
    <tbody>
        <!-- 데이터 출력 -->
        <c:choose>
            <c:when test="${not empty boardList}">
                <c:forEach var="board" items="${boardList}" varStatus="status">
                    <tr align="center" data-num="${board.b_num}">
                        <td>${board.b_num}</td>
                        <td align="left">
                            <span class="goDetail">${board.b_title}</span>
                        </td>
                        <td>${board.b_date}</td>
                        <td>${board.b_name}</td>
                    </tr>
                </c:forEach>
            </c:when>
            <c:otherwise>
                <tr>
                    <td colspan="4" align="center">등록된 게시물이
                </tr>
            </c:otherwise>
        </c:choose>
    </tbody>
</table>

```

data-num 속성을 이용하여 현재 게시물의 글번호를 저장한다.

```

존재하지 않습니다.</td>
</tr>
</c:otherwise>
</c:choose>
</tbody>
</table>
</div>
<%-- ===== 리스트 종료 ===== --%>

<%-- ===== 글쓰기 버튼 출력 시작 ===== --%>
<div id="boardBut">
    <input type="button" value="글쓰기" id="writeForm">
</div>
<%-- ===== 글쓰기 버튼 출력 종료 ===== --%>
</div>
</body>
</html>

```

⑩ [New] - [JSP file]을 선택하고, 파일명으로 "writeForm.jsp" 입력한 후, [Finish] 버튼을 클릭한다.

/WEB-INF/jsp/board/writeForm.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="EUC-KR">
    <title>글쓰기 화면 </title>
    <link rel="stylesheet" type="text/css" href="/include/css/board.css" />
    <script type="text/javascript" src="/include/js/jquery-1.11.3.min.js"></script>
<script type="text/javascript" src="/include/js/common.js"></script>
<script type="text/javascript">
    $(function(){
        /* 저장 버튼 클릭 시 처리 이벤트 */
        $("#boardInsert").click(function(){
            //입력값 체크
            if (!chkSubmit($("#b_name"),"이름을")) return;

```

```

else if (!chkSubmit($("#b_title"),"제목을")) return;
else if (!chkSubmit($("#b_content"),"작성할 내용을")) return;
else if (!chkSubmit($("#file"),"첨부파일을")) return;
else if (!chkSubmit($("#b_pwd"),"비밀번호를")) return;
else {
    /* 배열내의 값을 찾아서 인덱스를 반환(요소가 없을 경우-1반환)
    jQuery.inArray(찾을 값, 검색 대상의 배열)*/
    var ext = $('#file').val().split('.').pop().toLowerCase();
    if(jQuery.inArray(ext, ['gif','png','jpg','jpeg']) == -1) {
        alert('gif,png,jpg,jpeg 파일만 업로드 할수 있습니다.');
        return;
    }
    $("#f_writeForm").attr({
        "method":"POST",
        "action":"/board/boardInsert.do"
    });
    $("#f_writeForm").submit();
}

/* 목록 버튼 클릭 시 처리 이벤트 */
$("#boardList").click(function(){
    location.href="/board/boardList";
});

});

```

첨부파일은 이미지 파일
만 가능하도록 확장자에
대해 유효성 체크

```

</script>
</head>
<body>
    <div id="boardTit"><h3>글쓰기</h3></div>
    <form id="f_writeForm" name="f_writeForm" enctype="multipart/form-data">
        <table id="boardWrite">
            <tr>
                <td>작성자</td>
                <td><input type="text" name="b_name" id="b_name"></td>
            </tr>
            <tr>
                <td>글제목</td>
                <td><input type="text" name="b_title" id="b_title"></td>
            
```

```

        </tr>
        <tr>
            <td>내용</td>
            <td height="200"><textarea name="b_content" id="b_content"
                rows="10" cols="70"></textarea></td>
        </tr>
        <tr>
            <td>첨부파일</td>
            <td><input type="file" name="file" id="file"></td>
        </tr>
        <tr>
            <td>비밀번호</td>
            <td><input type="password" name="b_pwd" id="b_pwd"></td>
        </tr>
    </table>
</form>
<div id="boardBut">
    <input type="button" value="저장" class="but" id="boardInsert">
    <input type="button" value="목록" class="but" id="boardList">
</div>
</body>
</html>

```

⑪ [New] - [JSP file]을 선택하고, 파일명으로 "boardDetail.jsp" 입력한 후, [Finish] 버튼을 클릭한다.

/WEB-INF/jsp/board/boardDetail.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="EUC-KR">
<title>글상세 보기</title>
    <link rel="stylesheet" type="text/css" href="/include/css/board.css" />
    <script type="text/javascript" src="/include/js/jquery-1.11.3.min.js"></script>
    <script type="text/javascript" src="/include/js/common.js"></script>
    <script type="text/javascript">

```

```

var butChk = 0; // 수정버튼과 삭제버튼을 구별하기 위한 변수
$(function(){
    $("#pwdChk").hide();

    /* 첨부파일 이미지 보여주기 위한 속성 추가 */
    var file = "<c:out value='${detail.b_file}' />";
    if(file!=""){
        $("#fileImage").attr({
            src:"/uploadStorage/${detail.b_file}",
            width:"450px",
            height:"200px"
        });
    }

    /* 수정 버튼 클릭 시 처리 이벤트 */
    $("#updateForm").click(function(){
        $("#pwdChk").show();
        $("#msg").text("작성시 입력한 비밀번호를 입력해 주세요.")
            .css("color","#000099");
        butChk = 1;
    });

    /* 삭제 버튼 클릭 시 처리 이벤트 */
    $("#boardDelete").click(function(){
        $("#pwdChk").show();
        $("#msg").text("작성시 입력한 비밀번호를 입력해 주세요.")
            .css("color","#000099");
        butChk = 2;
    });

    /* 비밀번호 확인 버튼 클릭 시 처리 이벤트 */
    $("#pwdBut").click(function(){
        pwdConfirm();
    });

    /* 목록 버튼 클릭 시 처리 이벤트 */
    $("#boardList").click(function(){
        location.href="/board/boardList.do";
    });
})

```

```

        });
    });

/* 비밀번호 확인 버튼 클릭시 실질적인 처리 함수 */
function pwdConfirm(){
    if (!chkSubmit($("#b_pwd"),"비밀번호를")) return;
    else {
        $.ajax({
            url : "/board/pwdConfirm.do", //전송 url
            type : "POST",           // 전송 시 method 방식
            data : $("#f_pwd").serialize(), //폼전체 데이터 전송
            error : function(){          //실행시 오류가 발생하였을 경우
                alert('시스템 오류 입니다. 관리자에게 문의 하세요');
            },                         //정상적으로 실행이 되었을 경우
            success : function(resultData){
                var goUrl="";           // 이동할 경로를 저장할 변수
                if(resultData==0){      // 일치하지 않는 경우
                    $("#msg").text("작성시 입력한 비밀번호가 일치
                        하지 않습니다.").css("color","red");
                    $("#b_pwd").select();
                }else if(resultData==1){ // 일치할 경우
                    $("#msg").text("");
                    if(butChk==1){
                        goUrl = "/board/updateForm.do";
                    }else if(butChk==2){
                        goUrl = "/board/boardDelete.do";
                    }
                    $("#f_data").attr("action",goUrl);
                    $("#f_data").submit();
                }
            }
        });
    }
}

</script>
</head>
<body>
    <div id="boardTit"><h3>글상세</h3></div>

```

```

<form name="f_data" id="f_data" method="POST">
    <input type="hidden" name="b_num" value="${detail.b_num}"/>
    <input type="hidden" name="b_file" id="b_file" value="${detail.b_file}" />
</form>
<%-- ===== 비밀번호 확인 버튼 및 버튼 추가 시작 ===== --%>
<table id="boardPwdBut">
    <tr>
        <td id="btd1">
            <div id="pwdChk">
                <form name="f_pwd" id="f_pwd">
                    <input type="hidden" name="b_num"
                           id="b_num" value="${detail.b_num}"/>
                    <label for="b_pwd" id="l_pwd">비밀번호 : </label>
                    <input type="password" name="b_pwd" id="b_pwd" />
                    <input type="button" id="pwdBut" value="확인" />
                    <span id="msg"></span>
                </form>
            </div>
        </td>
        <td id="btd2">
            <input type="button" value="수정" id="updateForm">
            <input type="button" value="삭제" id="boardDelete">
            <input type="button" value="목록" id="boardList">
        </td>
    </tr>
</table>
<%-- ===== 비밀번호 확인 버튼 및 버튼 추가 종료 ===== --%>

<%-- ===== 상세 정보 보여주기 시작 ===== --%>
<div id="boardDetail">
    <table>
        <colgroup>
            <col width="25%" />
            <col width="25%" />
            <col width="25%" />
            <col width="25%" />
        </colgroup>
        <tbody>

```

```

<tr>
    <td class="ac">작성자</td>
    <td>${detail.b_name}</td>
    <td class="ac">작성일</td>
    <td>${detail.b_date}</td>
</tr>
<tr>
    <td class="ac">제목</td>
    <td colspan="3">${detail.b_title}</td>
</tr>
<tr class="ctr">
    <td class="ac">내용</td>
    <td colspan="3">${detail.b_content}</td>
</tr>
<tr class="ctr">
    <td class="ac">첨부파일 파일</td>
    <td colspan="3"><img id="fileImage" /></td>
</tr>
</tbody>
</table>
</div>
<%-- ===== 상세 정보 보여주기 종료 ===== --%>
</body>
</html>

```

⑫ [New] - [JSP file]을 선택하고, 파일명으로 "updateForm.jsp" 입력한 후, [Finish] 버튼을 클릭한다.
수정시 이미지파일과 비밀번호를 입력하지 않으면 입력시 저장한 값 그대로 유지된다.

/WEB-INF/jsp/board/updateForm.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="EUC-KR">
    <title>글수정 화면</title>
    <link rel="stylesheet" type="text/css" href="/include/css/board.css" />

```

```

<script type="text/javascript" src="/include/js/jquery-1.11.3.min.js"></script>
<script type="text/javascript" src="/include/js/common.js"></script>
<script type="text/javascript">

$(function(){
    /* 수정 버튼 클릭 시 처리 이벤트 */
    $("#boardUpdate").click(function(){
        //입력값 체크
        if (!chkSubmit($("#b_name"),"이름을")) return;
        else if (!chkSubmit($("#b_title"),"제목을")) return;
        else if (!chkSubmit($("#b_content"),"작성할 내용을")) return;
        else{
            if ($("#file").val().indexOf(".") > -1){
                var ext = $("#file").val().split('.').pop().toLowerCase();
                if(jQuery.inArray(ext, ['gif','png','jpg','jpeg']) == -1) {
                    alert('gif,png,jpg,jpeg 파일만 업로드 할 수 있습니다.');
                    return;
                }
            }
            //console.log("기본 파일명 : "+$("#b_file").val());
            $("#f_writeForm").attr({
                "method":"POST",
                "action":"/board/boardUpdate.do"
            });
            $("#f_writeForm").submit();
        }
    });

    /* 목록 버튼 클릭 시 처리 이벤트 */
    $("#boardList").click(function(){
        location.href="/board/boardList.do";
    });
});

</script>
</head>
<body>
    <div id="boardTit"><h3>글수정</h3></div>
    <form id="f_writeForm" name="f_writeForm" enctype="multipart/form-data">
        <input type="hidden" id="b_num" name="b_num" value="${updateData.b_num}" />
        <input type="hidden" name="b_file" id="b_file" value="${updateData.b_file}" />

```

```

<table id="boardWrite">
    <tr>
        <td>작성자</td>
        <td><input type="text" name="b_name" id="b_name"
            value="${updateData.b_name}" /></td>
    </tr>
    <tr>
        <td>글제목</td>
        <td><input type="text" name="b_title" id="b_title"
            value="${updateData.b_title}" /></td>
    </tr>
    <tr>
        <td>내용</td>
        <td height="200"><textarea name="b_content" id="b_content"
            rows="10" cols="70">${updateData.b_content}</textarea></td>
    </tr>
    <tr>
        <td>첨부파일</td>
        <td><input type="file" name="file" id="file"></td>
    </tr>
    <tr>
        <td>비밀번호</td>
        <td><input type="password" name="b_pwd" id="b_pwd" />
            <label>수정할 비밀번호를 입력해 주세요.</label></td>
    </tr>
</table>
</form>
<div id="boardBut">
    <input type="button" value="수정" class="but" id="boardUpdate" />
    <input type="button" value="목록" class="but" id="boardList" />
</div>
</body>
</html>

```

⑬ 처음 시작할 기본 페이지인 default.html파일을 생성한다.

```

/WebContent/default.html
<!DOCTYPE html>
<html><head><meta charset="EUC-KR"><title>default 페이지</title>

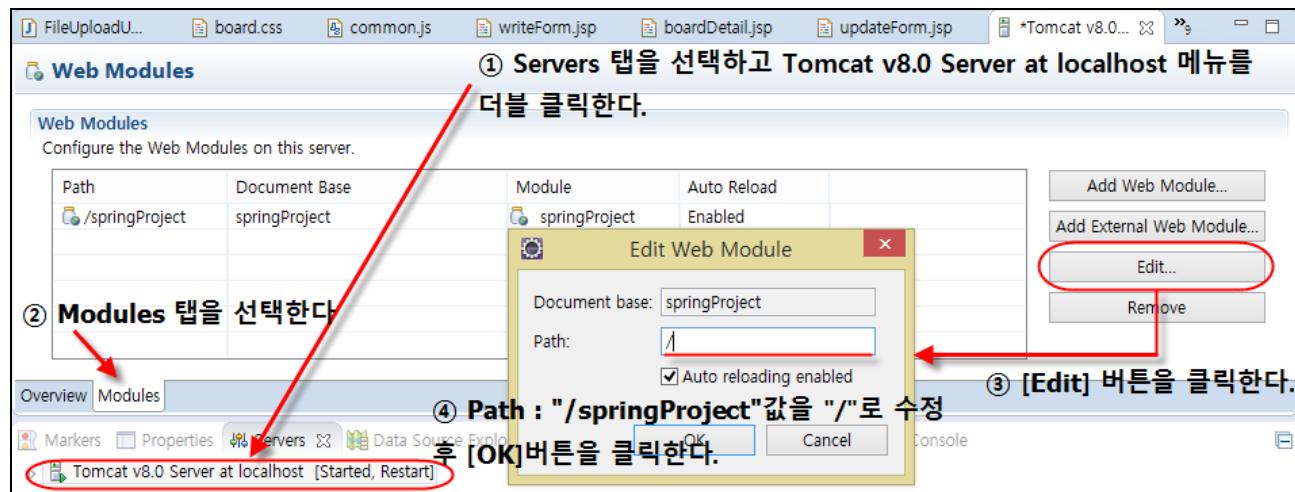
```

```

<!-- <style type="text/css">a{text-decoration:none}</style> -->
<script type="text/javascript">
    window.location.href="/board/boardList.do";
</script>
</head><body>
<!-- 추후 메뉴 내용이 추가될 것을 고려하여 아래 코드 추가한다. -->
<!-- <a href="/board/boardList.do">[게시판 리스트]</a> &nbsp;&nbsp; -->
</body></html>

```

- ⑯ default.html 파일을 실행하기 전에 다음과 같이 수정한다. ①, ② 실행 후 Path/Document Base 영역에 아무 데이터도 보이지 않으면 Add Web Module.. 버튼을 클릭하여 Web Modules에 /springProject를 선택하면 된다. 그리고 ③, ④을 진행하면 된다.



위 내용을 설정하면 [Servers] - [server.xml] 파일에 다음과 같은 소스가 추가된 것을 확인할 수 있다.

```

<Context docBase="boardProject" path="/" reloadable="true"
source="org.eclipse.jst.jee.server:boardProject"/>

```

4) 게시판에 검색과 정렬이 가능하도록 추가

글번호	글제목	작성일	작성자
5	작게라도 시작하라. 그것을 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카	2015-12-02 07:08:43	최성희
4	열정이 있으면 꼭 만난다. 열정(desire)이 운명과 만남을 주선한다. -김영세	2015-12-02 06:35:19	도희라
3	우리들은 과거에의 집착보다 미래의 희망으로 살고 있다. -G. 무어	2015-12-02 06:33:08	김희건
2	노력에 관한 명언	2015-12-01 18:28:56	김철수
1	마음을 움직이는 명언	2015-12-01 18:28:04	홍길동

글번호	글제목	작성일	작성자
5	작게라도 시작해라. 그것은 전혀 시작하지 않는 것보다 낫다 - 프레드 라우카	2015-12-02 07:08:43	최성희
4	불정이 있으면 꼭 만난다. 울망(desire)이 운명과 만남을 주선한다. -김경세	2015-12-02 06:35:19	도희라
3	우리들은 과거에의 질착보다 미래의 희망으로 살고 있다. -G. 무어	2015-12-02 06:23:08	김희진
2	노력에 관한 명언	2015-12-01 18:28:56	김철수
1	마음을 깨끗이는 명언	2015-12-01 18:28:04	홍길동

글번호	글제목	작성일	작성자
5	마음은 울망이라는 명언	2015-12-01 18:28:04	홍길동
4	노력에 관한 명언	2015-12-01 18:28:56	김철수
3	우리들은 과거에의 질착보다 미래의 희망으로 살고 있다. -G. 무어	2015-12-02 06:33:08	김희진
2	불정이 있으면 꼭 만난다. 울망(desire)이 운명과 만남을 주선한다. -김경세	2015-12-02 06:35:19	도희라
1	작게라도 시작해라. 그것은 전혀 시작하지 않는 것보다 낫다 - 프레드 라우카	2015-12-02 07:08:43	최성희

① 검색 및 정렬, 페이징 처리를 위해 사용할 필드를 CommonVO로 정의하자.

src/com/spring/common/vo/CommonVO.java

```
package com.spring.common.vo;
```

```
public class CommonVO {
    private String page;          //페이지 번호
    private String pageSize; //페이지에 보여주는 줄수
    private String start_row; //시작 레코드 번호
    private String end_row; //종료 레코드 번호

    //조건검색시 사용할 속성
    private String search = "";
    private String keyword = "";

    //제목 클릭시 정렬을 위한 속성
    private String order_by ;
    private String order_sc ;

    public String getPage() {
        return page;
    }
    public void setPage(String page) {
        this.page = page;
    }
    public String getPageSize() {
        return pageSize;
    }
    public void setPageSize(String pageSize) {
        this.pageSize = pageSize;
    }
}
```

```
}

public String getStart_row() {
    return start_row;
}

public void setStart_row(String start_row) {
    this.start_row = start_row;
}

public String getEnd_row() {
    return end_row;
}

public void setEnd_row(String end_row) {
    this.end_row = end_row;
}

public String getSearch() {
    return search;
}

public void setSearch(String search) {
    this.search = search;
}

public String getKeyword() {
    return keyword;
}

public void setKeyword(String keyword) {
    this.keyword = keyword;
}

public String getOrder_by() {
    return order_by;
}

public void setOrder_by(String order_by) {
    this.order_by = order_by;
}

public String getOrder_sc() {
    return order_sc;
}

public void setOrder_sc(String order_sc) {
    this.order_sc = order_sc;
}
```

```
    }  
}
```

- CommonVO를 상속해서 사용하도록 BoardVO 클래스를 수정하자.

BoardVO.java

```
package com.spring.board.vo;  
  
import com.spring.common.vo.CommonVO;  
  
public class BoardVO extends CommonVO{  
... 중략  
}
```

- ② 검색과 정렬이 가능하도록 "boardList.jsp" 파일에 다음과 같이 소스를 추가 및 수정해 보자.

/WEB-INF/jsp/board/boardList.jsp 파일에 추가 및 수정

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"  
pageEncoding="EUC-KR"%>  
... 중략  
    <link rel="stylesheet" type="text/css" href="/include/css/board.css" />  
    <script type="text/javascript" src="/include/js/jquery-1.11.3.min.js"></script>  
    <script type="text/javascript" src="/include/js/common.js"></script>  
    <script type="text/javascript">  
        $(function(){  
            /* 검색 후 검색 대상과 검색 단어 출력 */  
            if("<c:out value='${data.keyword}' />"!=""){  
                $("#keyword").val("<c:out value='${data.keyword}' />");  
                $("#search").val("<c:out value='${data.search}' />");  
            }  
  
            /* 검색 대상이 변경될 때마다 처리 이벤트 */  
            $("#search").change(function() {  
                if($("#search").val() == "all"){  
                    $("#keyword").val("전체 데이터 조회합니다.");  
                }else if($("#search").val() != "all"){  
                    $("#keyword").val("");  
                    $("#keyword").focus();  
                }  
            });  
        });  
    
```

```

/* 검색 버튼 클릭 시 처리 이벤트 */
$("#searchData").click(function(){
    if($("#search").val()!="all"){
        if(!chkSubmit($("#keyword"),"검색어를")) return;
    }
    goPage(1);
});

/* 글쓰기 버튼 클릭 시 처리 이벤트 */
$("#writeForm").click(function(){
    location.href = "/board/writeForm.do";
});

/* 제목 클릭시 상세 페이지 이동을 위한 처리 이벤트 */
$(".goDetail").click(function(){
    var b_num = $(this).parents("tr").attr("data-num");
    $("#b_num").val(b_num);
    // 상세 페이지로 이동하기 위해 form 추가 (id : detailForm)
    $("#detailForm").attr({
        "method":"get",
        "action":"/board/boardDetail.do"
    });
    $("#detailForm").submit();
});

/* 정렬 버튼 클릭 시 처리 함수 */
function setOrder(order_by){
    $("#order_by").val(order_by);
    if($("#order_sc").val()=='DESC'){
        $("#order_sc").val('ASC');
    } else{
        $("#order_sc").val('DESC');
    }
    goPage(1);
}

/* 검색과 한 페이지에 보여줄 레코드 수 처리 및 페이지를 위한 실질적인 처리 함수 */

```

```

function goPage(page){
    if($("#search").val() == "all"){
        $("#keyword").val("");
    }
    $("#page").val(page);
    $("#f_search").attr({
        "method": "get",
        "action": "/board/boardList.do"
    });
    $("#f_search").submit();
}

```

</script>

</head>

<body>

<div id="boardContainer">

<div id="boardTit"><h3>글목록</h3></div>

<!-- 상세 페이지 이동을 위한 form -->

<form name="detailForm" id="detailForm">

<input type="hidden" name="b_num" id="b_num">

</form>

<%-- ===== 검색기능 시작(이 부분 전체 추가) ===== --%>

<div id="boardSearch">

<form id="f_search" name="f_search">

<input type="hidden" id="page" name="page" value="1"/>

<input type="hidden" id="order_by" name="order_by" value="\${data.order_by}"/>

<input type="hidden" id="order_sc" name="order_sc" value="\${data.order_sc}"/>

<table summary="검색">

<colgroup>

<col width="70%"></col>

<col width="30%"></col>

</colgroup>

<tr>

<td id="btd1">

<label>검색조건</label>

<select id="search" name="search">

<option value="all">전체</option>

```

        <option value="b_title">제목</option>
        <option value="b_content">내용</option>
        <option value="b_name">작성자</option>
    </select>
    <input type="text" name="keyword"
           id="keyword" value="검색어를 입력하세요" />
    <input type="button" value="검색"
           id="searchData" />
</td>
</tr>
</table>
</form>
</div>
<%-- ===== 검색기능 종료 ===== --%>
<%-- ===== 리스트 시작 ===== --%>
<div id="boardList">
<table summary="게시판 리스트">
<colgroup>
    <col width="10%" />
    <col width="62%" />
    <col width="15%" />
    <col width="13%" />
</colgroup>
<thead>
<tr>


"<th>글번호</th>" 부분을 다음과 같이 수정한다.



"<th>작성일</th>" 부분을 다음과 같이 수정한다.


```

```

<c:when test="${data.order_by=='b_date'
    and data.order_sc=='DESC'}">▼</c:when>
<c:otherwise>▲</c:otherwise>
</c:choose></a></th>
<th class="borcle">작성자</th>
</tr>
</thead>
<tbody><!-- 데이터 출력 -->
<c:choose>
    <c:when test="${not empty boardList}" >
        <c:forEach var="board" items="${boardList}" varStatus="status">
            <tr align="center" data-num="${board.b_num}">
                <td>${board.b_num}</td>
                <td align="left">
                    <span class="goDetail">${board.b_title}</span>
                </td>
                <td>${board.b_date}</td>
                <td>${board.b_name}</td>
            </tr>
        </c:forEach>
    </c:when>
    <c:otherwise>
        <tr>
            <td colspan="4" align="center">등록된 게시물이
                존재하지 않습니다.</td>
        </tr>
    </c:otherwise>
</c:choose>
</tbody>
</table>
</div>
<%-- ===== 리스트 종료 ===== --%>

<%-- ===== 글쓰기 버튼 출력 시작 ===== --%>
<div id="boardBut">
    <input type="button" value="글쓰기" id="writeForm">
</div>
<%-- ===== 글쓰기 버튼 출력 종료 ===== --%>

```

```
</div>
</body>
</html>
```

③ "BoardController.java" 파일에 boardList() 메서드에 검색과 정렬이 가능하도록 소스가 변경되어야 한다.

BoardController.java

```
package com.spring.board.controller;

import java.util.List;
...중략

@Controller
@RequestMapping(value="/board")
public class BoardController {

    Logger logger = Logger.getLogger(BoardController.class);

    @Autowired
    private BoardService boardService;

    /**
     * 글목록 구현하기
     */
    @RequestMapping(value="/boardList", method=RequestMethod.GET)
    public String boardList(BoardVO bvo, Model model) {
        logger.info("boardList 호출 성공");

        // 정렬에 대한 기본값 설정
        if(bvo.getOrder_by()==null) bvo.setOrder_by("b_num");
        if(bvo.getOrder_sc()==null) bvo.setOrder_sc("DESC");

        // 정렬에 대한 데이터 확인
        logger.info("order_by = "+bvo.getOrder_by());
        logger.info("order_sc = "+bvo.getOrder_sc());

        // 검색에 대한 데이터 확인
        logger.info("search = "+bvo.getSearch());
        logger.info("keyword = "+bvo.getKeyword());
        List<BoardVO> boardList = boardService.boardList(bvo);

        model.addAttribute("boardList", boardList);
    }
}
```

```

        model.addAttribute("data", bvo);

        return "board/boardList";
    }

    중략...
}

```

- ④ "Board.xml" 검색과 정렬이 가능하도록 쿼리문이 변경되어야 한다.

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="com.spring.board.dao.BoardDao">

    <sql id="boardCommon">
        SELECT b_num, b_name, b_title, b_date
        FROM spring_board
        <trim prefix="WHERE" prefixOverrides="AND">
            <if test="search=='b_title'">
                <![CDATA[ b_title LIKE '%'|| #{keyword} ||'%' ]]>
            </if>
            <if test="search=='b_content'">
                <![CDATA[ AND b_content LIKE '%'|| #{keyword} ||'%' ]]>
            </if>
            <if test="search=='b_name'">
                <![CDATA[ AND b_name LIKE '%'|| #{keyword} ||'%' ]]>
            </if>
        </trim>
    </sql>
    <!-- 게시물 기본 조회(현재 사용하고 있는 쿼리문을 주석 처리 한다) -->
    <select id="boardList" parameterType="board" resultType="board">
        select b_num, b_name, b_title,
        to_char(b_date,'YYYY-MM-DD') as b_date
        from spring_board
    </select> -->
    <!-- 게시물 목록 조회 -->
    <select id="boardList" parameterType="board" resultType="board">
        /* Board - boardList */
        SELECT b_num, b_name, b_title,

```

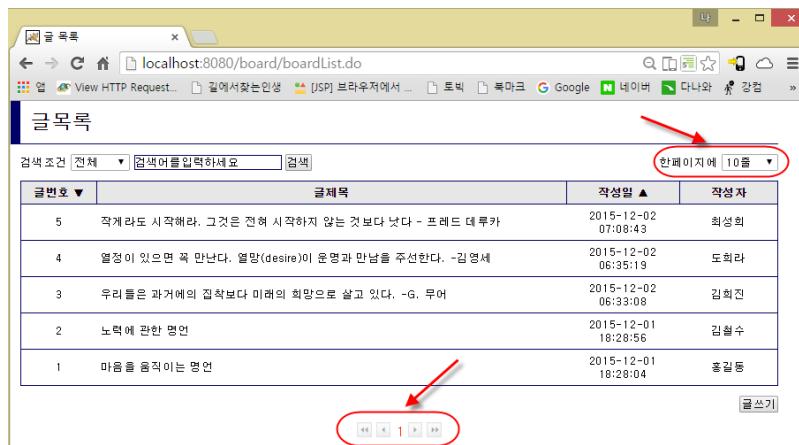
```

TO_CHAR(b_date,'YYYY-MM-DD HH24:MI:SS') AS b_date
FROM
(
    SELECT list.*, ROWNUM AS RNUM
    FROM (
        <include refid="boardCommon"></include>
        <choose>
            <when test="order_by=='b_num'">
                ORDER BY b_num
            </when>
            <when test="order_by=='b_date'">
                ORDER BY b_date
            </when>
        </choose>
        <choose>
            <when test="order_sc=='ASC'">
                ASC
            </when>
            <when test="order_sc=='DESC'">
                DESC
            </when>
        </choose>
    ) list
)
</select>
중략...
</mapper>

```

5) 페이징 처리

페이징 처리를 하기 위해서는 기존 파일이 변경되거나 새로운 소스파일을 생성해야 한다.



① 공통으로 사용될 작업에 대해 메소드로 정의하여 사용하도록 한다.

src/com/spring/common/util/Util.java

```
package com.spring.common.util;
```

```
public class Util {
```

```
    public static int nvl(String text){  
        return nvl(text, 0);  
    }
```

```
    /***************************************************************************/
```

```
    * nvl() 메서드는 문자열을 숫자로 변환하는 메서드.
```

```
    * @param (숫자로 변환할 문자열, 초기값으로 사용할 값(대체값))
```

```
    참고 : 예외 처리는 체크예외와 비체크예외로 구분.
```

```
        체크 예외는 파일입출력 / 네트워크 입출력 / 데이터베이스 입출력.
```

```
        나머지는 비체크 예외로 인식.
```

```
    * @return int
```

```
    /**************************************************************************/
```

```
    public static int nvl(String text, int def){
```

```
        int ret = def;
```

```
        try{
```

```
            ret = Integer.parseInt(text);
```

```
        }catch(Exception e){
```

```
            ret = def;
```

```
        }
```

```
        return ret;
```

```
}
```

```
    public static String nvl(Object text, String def){
```

```
        if(text==null || "".equals(text.toString().trim())){
```

```
            return def;
```

```
        }else{
```

```
            return text.toString();
```

```
        }
```

```
}
```

```
}
```

② 페이징 처리를 위해 사용할 필드에 대해 초기값을 설정한다.

Paging.java

```
package com.spring.common.page;  
import com.spring.common.util.Util;  
import com.spring.common.vo.CommonVO;  
  
public class Paging {  
    /**  
     * 페이지를 위한 설정 작업  
     * @param cvo  
     */  
    public static void setPage(CommonVO cvo){  
        int page = Util.nvl(cvo.getPage(), 1);  
        int pageSize = Util.nvl(cvo.getPageSize(), 10);  
  
        if( cvo.getPage()==null) cvo.setPage(page+"");  
        if( cvo.getPageSize()==null) cvo.setPageSize(pageSize+"");  
  
        int start_row = (page-1)*pageSize +1;  
        int end_row = (page-1)*pageSize+pageSize;  
  
        cvo.setStart_row(start_row+"");  
        cvo.setEnd_row(end_row+"");  
    }  
}
```

③ 컨트롤러 클래스(BoardController) 내 존재하는 boardList() 메서드를 다음과 같이 수정한다.

BoardController.java

```
package com.spring.board.controller;  
  
중략...  
import com.spring.common.page.Paging;  
import com.spring.common.util.Util;  
  
@Controller  
@RequestMapping(value="/board")  
public class BoardController {  
    Logger logger = Logger.getLogger(BoardController.class);
```

```

@Autowired
private BoardService boardService;

/*************************************
 * 글목록 구현하기
 *****/
@RequestMapping(value="/boardList", method=RequestMethod.GET)
public String boardList(@ModelAttribute BoardVO bvo, Model model) {
    logger.info("boardList 호출 성공");

    // 정렬에 대한 기본값 설정
    if(bvo.getOrder_by()==null) bvo.setOrder_by("b_num");
    if(bvo.getOrder_sc()==null) bvo.setOrder_sc("DESC");

    // 정렬에 대한 데이터 확인
    logger.info("order_by = "+bvo.getOrder_by());
    logger.info("order_sc = "+bvo.getOrder_sc());

    // 검색에 대한 데이터 확인
    logger.info("search = "+bvo.getSearch());
    logger.info("keyword = "+bvo.getKeyword());

    //페이지 세팅
    Paging setPage(bvo);

    // 전체 레코드수 구현
    int total = boardService.boardListCnt(bvo);
    logger.info("total = "+total);

    // 글번호 재설정
    int count = total - (Util.nvl(bvo.getPage())-1) * Util.nvl(bvo.getPageSize());
    logger.info("count = "+count);

    List<BoardVO> boardList = boardService.boardList(bvo);
    model.addAttribute("boardList", boardList);
    model.addAttribute("count", count);
    model.addAttribute("total", total);
    model.addAttribute("data", bvo);
}

```

```

        return "board/boardList";
    }
    중략...
}

```

※ 위 소스에서 전체 레코드 수를 구하기 위해서는 **BoardService.java**, **BoardServiceImpl.java**, **BoardDao.java**, **BoardDaoImpl.java** 파일에 **boardListCnt()** 메서드에 대해 추가 코딩해야 한다.

BoardService.java 파일 수정

```

package com.spring.board.service;
import java.util.List;
import com.spring.board.vo.BoardVO;
public interface BoardService {
    public List<BoardVO> boardList(BoardVO pvo);
    public int boardListCnt(BoardVO pvo);
    중략...
}

```

BoardServiceImpl.java 파일 수정

```

package com.spring.board.service;
중략...

@Service
@Transactional
public class BoardServiceImpl implements BoardService {
    Logger logger = Logger.getLogger(BoardServiceImpl.class);
    @Autowired
    private BoardDao boardDao;

    // 글목록 구현
    @Override
    public List<BoardVO> boardList(BoardVO pvo) {
        List<BoardVO> myList = null;
        myList = boardDao.boardList(pvo);
        return myList;
    }

    // 전체 레코드 수 구현

```

```

@Override
public int boardListCnt(BoardVO pvo) {
    return boardDao.boardListCnt(pvo);
}

중략...
}

```

BoardDao.java

```

package com.spring.board.dao;
import java.util.List;
import com.spring.board.vo.BoardVO;
public interface BoardDao {
    public List<BoardVO> boardList(BoardVO pvo);
    public int boardListCnt(BoardVO pvo);
    중략...
}

```

BoardDaolmpl.java

```

package com.spring.board.dao;
..중략
@Repository
public class BoardDaolmpl implements BoardDao {
    @Autowired
    private SqlSession session;

    // 글목록 구현
    @Override
    public List<BoardVO> boardList(BoardVO pvo) {
        return session.selectList("boardList");
    }

    // 전체 레코드 건수 구현
    @Override
    public int boardListCnt(BoardVO pvo) {
        return (Integer)session.selectOne("boardListCnt");
    }
    중략...
}

```

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="com.spring.board.dao.BoardDao">
    <!-- 게시물 목록 조회 -->
    <select id="boardList" parameterType="board" resultType="board">
        /* Board - boardList */
        SELECT b_num, b_name, b_title,
               TO_CHAR(b_date,'YYYY-MM-DD HH24:MI:SS') AS b_date
        FROM
            (
                SELECT list.*, ROWNUM AS RNUM
                FROM (
                    ..중략
                ) list
            )
        <where>
            <if test="start_row != null and start_row != "">
                <if test="end_row != null and end_row != "">
                    RNUM BETWEEN #{start_row} AND #{end_row}
                </if>
            </if>
        </where>
    </select>
    <select id="boardListCnt" parameterType="board" resultType="int">
        /* Board - boardListCnt */
        SELECT NVL(count(1), 0) FROM
            (
                SELECT list.*, ROWNUM AS RNUM
                FROM (
                    <include refid="boardCommon"></include>
                    ORDER BY b_num desc
                ) list
            )
    </select>
..중략
</mapper>
```

id="boardList" 요소 제일 마지막에 아래 소스를 추가한다.

이 부분을 새롭게 코딩되어야 할 부분이다.

④ 페이징 처리를 사용자 정의 태그로 표현하기 위해서 TagSupport 클래스를 상속받아서 정의한다.

PagingTag.java

```
package com.spring.common.page;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;

public class PagingTag extends TagSupport {
    private static final long serialVersionUID = 1L;

    /**
     * @param page      현재 페이지 번호
     * @param total     전체 조회된 Row 수
     * @param list_size 페이지에 보여주는 레코드수
     * @param page_size 페이지 네비게이터에 표시되는 페이지 버튼의 갯수
     */

    private int page = 1;
    private int total = 1;
    private int list_size = 10;
    private int page_size = 10;

    @Override
    public int doStartTag() throws JspException {
        try {
            pageContext.getOut().println(getPaging());
        } catch (IOException e) {
            e.printStackTrace();
        }
        return super.doStartTag();
    }

    public void setPage(int page) {
        this.page = page;
    }

    public void setTotal(int total) {
        this.total = total;
    }
}
```

```

}

public void setList_size(int list_size) {
    this.list_size = list_size;
}

public void setPage_size(int page_size) {
    this.page_size = page_size;
}

public String getPaging(){
    String ret = "";

    if(page < 1)
        page = 1;
    if(total < 1)
        return "";

    /* page가 1페이지이고 page_size가 20이면 */
    /* currentFirst는 1 */
    int currentFirst = ((page-1)/page_size) * page_size + 1;

    /* currentlast는 2 */
    int currentlast = ((page-1)/page_size) * page_size + page_size;

    /* nextFirst는 3 */
    int nextFirst = (((page-1)/page_size)+1) * page_size + 1;

    /* prevLast는 0 */
    int prevLast = (((page-1)/page_size)-1) * page_size + 1;

    int lastPage = 1;
    lastPage = total / list_size;
    /* lastPage(총 페이지수)는 total이 11이고 list_size가 10이면 1을 가진다.
     * 그래서 총 페이지 수가 나누어 떨어지지 않으면 나머지 레코드를 출력할
     * 페이지가 필요하다는 의미. */
    if ( total%list_size != 0 ) lastPage = lastPage + 1;
}

```

```

/* currentlast가 lastPage(총 페이지수)보다 크면 마지막 페이지로 수정*/
currentlast = (currentlast>lastPage)?lastPage:currentlast;

ret += " <div class='paginate'> ";

if ( page>1 ) {
    ret += " <a href="#" javascript:goPage('1')><span><img
src='..../images/common/btn_paginate_first.gif' alt='처음' /></span></a>";
}
else{
    ret += " <span><img src='..../images/common/btn_paginate_first.gif'
alt='처음' /></span> ";
}

if ( prevLast > 0 ) {
    ret += " <a href="#" javascript:goPage("'" +prevLast+"');"><span><img
src='..../images/common/btn_paginate_prev.gif' alt='이전' /></span></a> ";
}
else{
    ret += " <span><img src='..../images/common/btn_paginate_prev.gif'
alt='이전' /></span> ";
}

for (int j=currentFirst; j<currentFirst+page_size && j<=lastPage; j++) {
    if ( j <= currentlast ) {
        if ( j == page ) {
            ret += " <a href='#' class='on textAn'>" +j+ "</a> ";
        } else {
            ret += " <a href="#" javascript:goPage("'" +j+"');&
class='textAn'>" +j+ "</a> ";
        }
    }
}

if ( nextFirst <= lastPage ) {
    ret += " <a href="#" javascript:goPage("'" +nextFirst+"')><span><img
src='..../images/common/btn_paginate_next.gif' alt='다음' /></span></a> ";
}

```

```

        else{
            ret += " <span><img src='..../images/common/btn_paginate_next.gif'
            alt='다음' /></span> ";
        }

        if ( page<lastPage ) {
            ret += " <a href="#"javascript:goPage('"+lastPage+"')><span><img
            src='..../images/common/btn_paginate_last.gif' alt='마지막' /></span></a> ";
        }
        else{
            ret += " <span><img src='..../images/common/btn_paginate_last.gif'
            alt='마지막' /></span> ";
        }
        ret += " </div> ";
    }
    return ret;
}
}

```

⑤ 사용자 정의 태그를 사용하기 위해 TLD 파일을 생성한다.

WebContent/WEB-INF/tld/custom_tag.tld

```

<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd"
         version="2.1">

    <description>paging 처리를 위한 사용자 정의 태그</description>
    <display-name>customTag</display-name>
    <tlib-version>1.2</tlib-version>
    <short-name>customTag</short-name>

    <tag>
        <name>paging</name>
        <tag-class>com.spring.common.page.PagingTag</tag-class>
        <body-content>JSP</body-content>
        <attribute>
            <name>page</name>
            <required>false</required>
        
```

```

<rtextrvalue>true</rtextrvalue>
<type>int</type>
</attribute>
<attribute>
    <name>total</name>
    <required>false</required>
    <rtextrvalue>true</rtextrvalue>
    <type>int</type>
</attribute>
<attribute>
    <name>list_size</name>
    <required>false</required>
    <rtextrvalue>true</rtextrvalue>
    <type>int</type>
</attribute>
<attribute>
    <name>page_size</name>
    <required>false</required>
    <rtextrvalue>true</rtextrvalue>
    <type>int</type>
</attribute>
</tag>
</taglib>

```

⑥ 사용자 정의 태그를 사용하여 페이지 처리 결과를 출력하도록 "boardList.jsp"파일을 수정한다.

/WEB-INF/jsp/board/boardList.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="tag" uri="/WEB-INF/tld/custom_tag.tld" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="EUC-KR">
    <title>글 목록</title>
    <link rel="stylesheet" type="text/css" href="/include/css/board.css" />
    <script type="text/javascript" src="/include/js/jquery-1.11.3.min.js"></script>

```

```

<script type="text/javascript" src="/include/js/common.js"></script>
<script type="text/javascript">
$(function(){
    /* 검색 후 검색 대상과 검색 단어 출력 */
    if("<c:out value='${data.keyword}' />"!=""){
        $("#keyword").val("<c:out value='${data.keyword}' />");
        $("#search").val("<c:out value='${data.search}' />");
    }

    /* 한 페이지에 보여줄 레코드 수 조회 후 선택한 값 그대로
       유지하기 위한 설정*/
    if("<c:out value='${data.pageSize}' />"!=""){
        $("#pageSize").val("<c:out value='${data.pageSize}' />");
    }

    /* 검색 대상이 변경될 때마다 처리 이벤트 */
    $("#search").change(function() {
        if($("#search").val() == "all"){
            $("#keyword").val("전체 데이터 조회합니다.");
        }else if($("#search").val() != "all"){
            $("#keyword").val("");
            $("#keyword").focus();
        }
    });
    /* 검색 버튼 클릭 시 처리 이벤트 */
    $("#searchData").click(function(){
        if($("#search").val() != "all"){
            if(!chkSubmit($("#keyword"),"검색어를")) return;
        }
        goPage(1);
    });

    /* 한 페이지에 보여줄 레코드 수 변경될 때마다 처리 이벤트 */
    $("#pageSize").change(function() {
        goPage(1);
    });

    /* 글쓰기 버튼 클릭 시 처리 이벤트 */
}

```

```

        $("#writeForm").click(function(){
            location.href = "/board/writeForm.do";
        });

        /* 제목 클릭시 상세 페이지 이동을 위한 처리 이벤트 */
        $(".goDetail").click(function(){
            var b_num = $(this).parents("tr").attr("data-num");
            $("#b_num").val(b_num);
            // 상세 페이지로 이동하기 위해 form 추가 (id : detailForm)
            $("#detailForm").attr({
                "method":"get",
                "action":"/board/boardDetail.do"
            });
            $("#detailForm").submit();
        });
    });

    ..중략

</script>
</head>
<body>
    <div id="boardContainer">
        <div id="boardTit"><h3>글목록</h3></div>
        <!-- 상세 페이지 이동을 위한 form -->
        <form name="detailForm" id="detailForm">
            <input type="hidden" name="b_num" id="b_num">
            <b><input type="hidden" name="page" value="${data.page}"></b>
            <b><input type="hidden" name="pageSize" value="${data.pageSize}"></b>
        </form>
        <%-- ===== 검색기능 시작 ===== --%>
        <div id="boardSearch">
            <form id="f_search" name="f_search">
                <b><input type="hidden" id="page" name="page" value="${data.page}"></b>
                <input type="hidden" id="order_by" name="order_by"
                    value="${data.order_by}">
                <input type="hidden" id="order_sc" name="order_sc"
                    value="${data.order_sc}">
            <table summary="검색">
                <colgroup>

```

상세 페이지 이동 시
페이지 번호와 한페
이지에 보여줄 레코
드 수를 가지고 이동
할 수 있도록 소스
수정한다.

```

<col width="70%"></col>
<col width="30%"></col>
</colgroup>
<tr>
    <td id="btd1">
        <label>검색조건</label>
        <select id="search" name="search">
            <option value="all">전체</option>
            <option value="b_title">제목</option>
            <option value="b_content">내용</option>
            <option value="b_name">작성자</option>
        </select>
        <input type="text" name="keyword"
               id="keyword" value="검색어를 입력하세요" />
        <input type="button" value="검색"
               id="searchData" />
    </td>
    <td id="btd2">한페이지에
        <select id="pageSize" name="pageSize">
            <option value="10">10줄</option>
            <option value="20">20줄</option>
            <option value="30">30줄</option>
            <option value="50">50줄</option>
            <option value="70">70줄</option>
            <option value="100">100줄</option>
        </select>
    </td>
</tr>
</table>
</form>
</div>
<%-- ===== 검색기능 종료 ===== --%>

<%-- ===== 리스트 시작 ===== --%>
<div id="boardList">
    <table summary="게시판 리스트">
        <colgroup>
            <col width="10%" />

```

```

<col width="62%" />
<col width="15%" />
<col width="13%" />
</colgroup>
<thead>
<tr>
    <th><a href="javascript:setOrder('b_num');">글 번호
    <c:choose>
        <c:when test="${data.order_by=='b_num' and data.order_sc=='ASC'}">▲ </c:when>
        <c:when test="${data.order_by=='b_num' and data.order_sc=='DESC'}">▼ </c:when>
        <c:otherwise>▲</c:otherwise>
    </c:choose></a></th>
    <th>글 제목 </th>
    <th><a href="javascript:setOrder('b_date');">작성 일
    <c:choose>
        <c:when test="${data.order_by=='b_date' and data.order_sc=='ASC'}">▲ </c:when>
        <c:when test="${data.order_by=='b_date' and data.order_sc=='DESC'}">▼ </c:when>
        <c:otherwise>▲</c:otherwise>
    </c:choose></a></th>
    <th class="borcle">작성자</th>
</tr>
</thead>
<tbody><!-- 데이터 출력 -->
<c:choose>
    <c:when test="${not empty boardList}">
        <c:forEach var="board" items="${boardList}" varStatus="status">
            <tr align="center" data-num="${board.b_num}">
                <td>${count - (status.count-1)}</td>
                <td align="left">
                    <span class="goDetail">${board.b_title}</span>
                </td>
                <td>${board.b_date}</td>
                <td>${board.b_name}</td>
            </tr>
        </c:forEach>
    </c:when>
    <c:otherwise>
        <tr>
            <td colspan="4" style="text-align: center;">등록된 글이 없습니다.
        </tr>
    </c:otherwise>
</c:choose>
</tbody>

```

```

        </c:forEach>
        </c:when>
        <c:otherwise>
            <tr>
                <td colspan="4" align="center">등록된 게시물이
                    존재하지 않습니다.</td>
            </tr>
        </c:otherwise>
    </c:choose>
</tbody>
</table>
</div>
<%-- ===== 리스트 종료 ===== --%>

<%-- ===== 글쓰기 버튼 출력 시작 ===== --%>
<div id="boardBut">
    <input type="button" value="글쓰기" id="writeForm">
</div>
<%-- ===== 글쓰기 버튼 출력 종료 ===== --%>

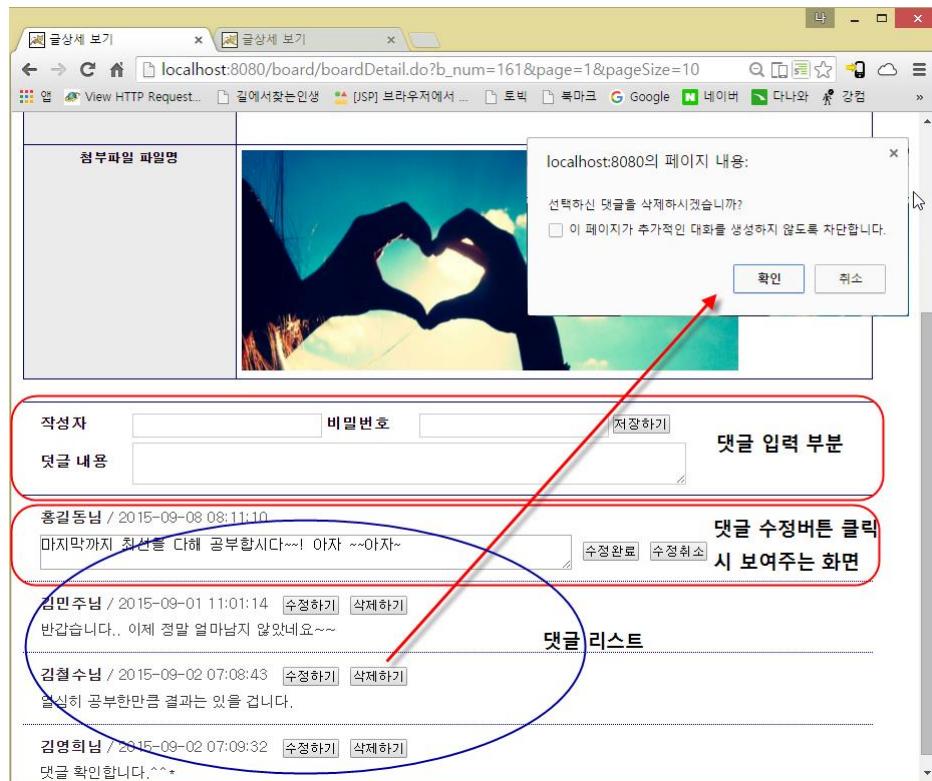
<%-- ===== 페이지 네비게이션 시작 ===== --%>
<div id="boardPage">
<tag:paging page="${param.page}" total="${total}" list_size="${data.pageSize}"/>
</div>
<%-- ===== 페이지 네비게이션 종료 ===== --%>
</div>
</body>
</html>

```

6) 댓글 작성하기

REST(Representational State Transfer)의 약어로 하나의 URI는 하나의 고유한 리소스를 대표하도록 설계된다는 개념이다. 최근에는 서버에 접근하는 기기의 종류가 다양해지면서 다양한 기기에서 공통으로 데이터를 처리할 수 있는 규칙을 만들려고 하는데 이러한 시도가 REST방식이다.

REST방식은 특정 URL는 반드시 그에 상응하는 데이터 자체라는 것을 의미하는 방식이다. 예를 들어 'board/125'은 게시물 중에서 125번이라는 고유한 의미를 가지도록 설계하고, 이에 대한 처리는 GET, POST 방식과 같이 추가적인 정보를 통해서 결정한다.



① 댓글을 저장하기 위해 테이블을 생성한다.

-- 테이블 생성

```
CREATE TABLE SPRING_REPLY(
    R_NUM NUMBER NOT NULL ,
    B_NUM NUMBER NOT NULL ,
    R_NAME VARCHAR2(10) NULL ,
    R_CONTENT VARCHAR2(2000) NULL ,
    R_PWD VARCHAR2(18) NULL ,
    R_DATE DATE DEFAULT SYSDATE,
    CONSTRAINT SPRING_REPLY_PK PRIMARY KEY(R_NUM),
    CONSTRAINT SPRING_REPLY_FK FOREIGN KEY(B_NUM) REFERENCES SPRING_BOARD(B_NUM)
);

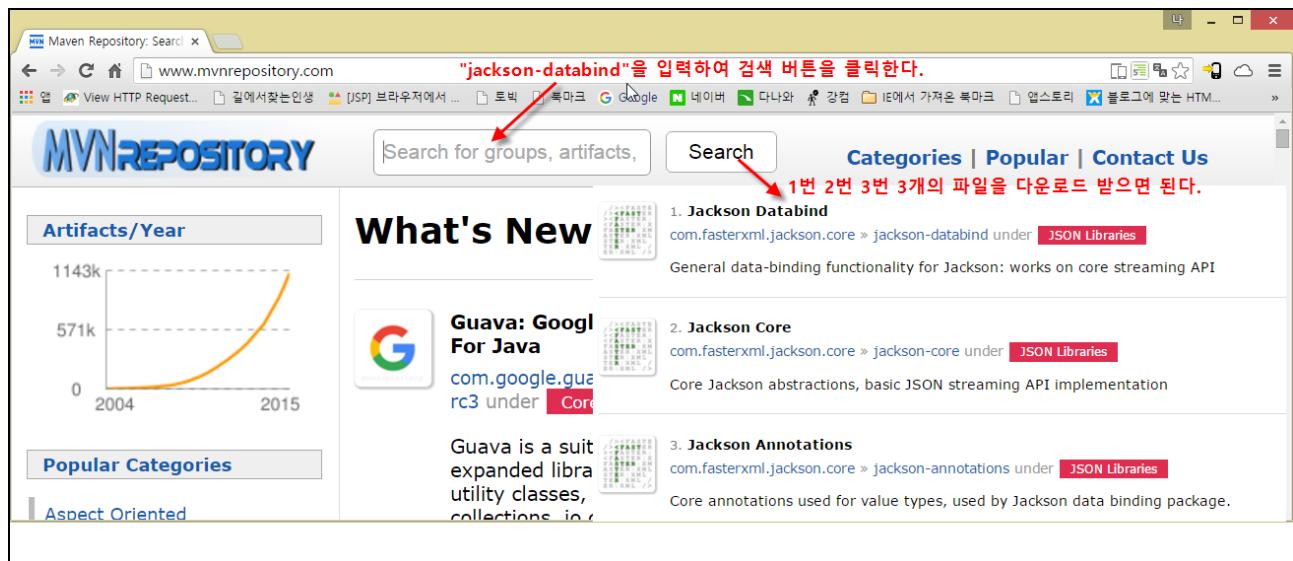
COMMENT ON TABLE SPRING_REPLY IS '댓글 정보';
COMMENT ON COLUMN SPRING_REPLY.R_NUM IS '댓글 번호';
COMMENT ON COLUMN SPRING_REPLY.B_NUM IS '게시판 글번호';
COMMENT ON COLUMN SPRING_REPLY.R_NAME IS '댓글 작성자';
COMMENT ON COLUMN SPRING_REPLY.R_CONTENT IS '댓글 내용';
COMMENT ON COLUMN SPRING_REPLY.R_PWD IS '댓글 비밀번호';
COMMENT ON COLUMN SPRING_REPLY.R_DATE IS '댓글 등록일';
```

```

CREATE SEQUENCE SPRING_REPLY_SEQ
START WITH 1
INCREMENT BY 1
NOCYCLE;

```

- ② 댓글 데이터를 교환은 JSON으로 하기 위해 jackson-databind-2.5.4.jar, jackson-core-2.5.4.jar, jackson-annotations-2.5.4.jar 3개의 jar 파일을 다운로드 받아 "WEB-INF\lib" 폴더에 추가한다.



그리고 REST 방식에서 UPDATE 작업은 PUT, PATCH 방식을 이용해서 처리하고 DELETE 작업은 DELETE 방식으로 처리하게 된다. 이때 브라우저에 따라서 PUT, PATCH, DELETE 방식을 지원하지 않는 경우가 발생 한다. 그래서 스프링은 이를 위해서 hiddenHttpMethodFilter라는 것을 제공한다.

/WEB-INF/web.xml에 추가 작업

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
    <display-name>springProject</display-name>
    <welcome-file-list>
        <welcome-file>default.html</welcome-file>
    </welcome-file-list>

    <!-- ===== 메서드 처리(댓글에서 사용한 메서드(method) 구분) ===== -->
    <filter>
        <filter-name>hiddenHttpMethodFilter</filter-name>
        <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
    
```

```

</filter>
<filter-mapping>
    <filter-name>hiddenHttpMethodFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
</web-app>

```

③VO 클래스, 컨트롤러 클래스, 서비스(모델) 클래스, DAO 클래스, 매퍼 XML 소스파일을 작성하여 보자.

ReplyVO.java

```

package com.spring.reply.vo;

public class ReplyVO {
    private int r_num      =0; // 댓글번호
    private int b_num      =0; // 게시판 글번호
    private String r_name   =""; // 댓글 작성자
    private String r_content =""; // 댓글 내용
    private String r_date   =""; // 댓글 작성일
    private String r_pwd     =""; // 댓글 비밀번호

    public int getR_num() {
        return r_num;
    }

    public void setR_num(int r_num) {
        this.r_num = r_num;
    }

    public int getB_num() {
        return b_num;
    }

    public void setB_num(int b_num) {
        this.b_num = b_num;
    }

    public String getR_name() {
        return r_name;
    }
}

```

```

public void setR_name(String r_name) {
    this.r_name = r_name;
}

public String getR_content() {
    return r_content;
}

public void setR_content(String r_content) {
    this.r_content = r_content;
}

public String getR_date() {
    return r_date;
}

public void setR_date(String r_date) {
    this.r_date = r_date;
}

public String getR_pwd() {
    return r_pwd;
}

public void setR_pwd(String r_pwd) {
    this.r_pwd = r_pwd;
}

```

ReplyController.java

```

package com.spring.reply.controller;

import java.util.List;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

```

```

import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.spring.reply.service.ReplyService;
import com.spring.reply.vo.ReplyVO;

/*************************************
 * 참고 : @RestController (@Controller + @ResponesBody)
 * 기존의 특정한 JSP와 같은 뷰를 만들어 내는 것이 목적이 아닌 REST 방식의 데이터 처리를
 * 위해서 사용하는(데이터 자체를 반환) 어노테이션이다.
************************************/

```

@RestController

```
@RequestMapping(value = "/replies")
```

```
public class ReplyController {
```

```
    Logger logger = Logger.getLogger(ReplyController.class);
```

@Autowired

```
private ReplyService replyService;
```

```
/*****
```

- * 댓글 글목록 구현하기

- * @return List<ReplyVO>

- * 참고: **@PathVariable**은 URI의 경로에서 원하는 데이터를 추출하는 용도로 사용.

- * **ResponseEntity** 타입은 개발자가 직접 결과 데이터와 HTTP의 상태 코드를

- * 직접 제어할 수 있는 클래스이다. 404나 500 같은 상태코드를 전송하고 싶은

- * 데이터와 함께 전송할 수 있기 때문에 좀더 세밀한 제어가 가능.

```
*****/
```

```
@RequestMapping(value = "/all/{b_num}.do", method = RequestMethod.GET)
```

```
public ResponseEntity<List<ReplyVO>> list(@PathVariable("b_num") Integer b_num) {
```

```
    ResponseEntity<List<ReplyVO>> entity = null;
```

```
    try {
```

```
        entity = new ResponseEntity<>(replyService.replyList(b_num), HttpStatus.OK);
```

```
    } catch (Exception e) {
```

```
        e.printStackTrace();
```

```

        entity = new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
    return entity;
}

//*********************************************************************
* 댓글 글쓰기 구현하기
* @return String
* 참고: @RequestBody
//********************************************************************

@RequestMapping(value="/replyInsert")
public ResponseEntity<String> replyInsert(@RequestBody ReplyVO rvo) {
    logger.info("replyInsert 호출 성공");
    ResponseEntity<String> entity = null;
    int result;
    try{
        result = replyService.replyInsert(rvo);
        if(result==1){
            entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);
        }
    }catch(Exception e){
        e.printStackTrace();
        entity = new ResponseEntity<String>(e.getMessage(), HttpStatus.BAD_REQUEST);
    }
    return entity;
}

//*********************************************************************
* 댓글 수정 구현하기
* @return
* 참고: REST 방식에서 UPDATE 작업은 PUT, PATCH 방식을 이용해서 처리.
* 전체 데이터를 수정하는 경우에는 PUT을 이용하고,
* 일부의 데이터를 수정하는 경우에는 PATCH를 이용.
//********************************************************************

@RequestMapping(value = "/{r_num}.do",
        method = {RequestMethod.PUT, RequestMethod.PATCH})
public ResponseEntity<String> replyUpdate(@PathVariable("r_num") Integer r_num,
                                           @RequestBody ReplyVO rvo) {

```

```

        logger.info("replyUpdate 호출 성공");
        ResponseEntity<String> entity = null;
        try {
            rvo.setR_num(r_num);
            replyService.replyUpdate(rvo);
            entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
            entity = new ResponseEntity<String>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
        return entity;
    }

    /**
     * 댓글 삭제 구현하기
     * @return
     * 참고: REST 방식에서 DELETE 작업은 DELETE방식을 이용해서 처리.
     */
    @RequestMapping(value = "/{r_num}.do", method = RequestMethod.DELETE)
    public ResponseEntity<String> replyDelete(@PathVariable("r_num") Integer r_num) {
        logger.info("replyDelete 호출 성공");
        ResponseEntity<String> entity = null;
        try {
            replyService.replyDelete(r_num);
            entity = new ResponseEntity<String>("SUCCESS", HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
            entity = new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
        }
        return entity;
    }
}

```

ReplyService.java

```

package com.spring.reply.service;

import java.util.List;
import com.spring.reply.vo.ReplyVO;

```

```
public interface ReplyService {  
    public List<ReplyVO> replyList(Integer b_num);  
    public int replyInsert(ReplyVO rvo);  
    public int replyUpdate(ReplyVO rvo);  
    public int replyDelete(int r_num);  
}
```

ReplyServiceImpl.java

```
package com.spring.reply.service;  
import java.util.List;  
import org.apache.log4j.Logger;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import com.spring.reply.dao.ReplyDao;  
import com.spring.reply.vo.ReplyVO;  
  
@Service  
@Transactional  
public class ReplyServiceImpl implements ReplyService {  
    Logger logger = Logger.getLogger(ReplyServiceImpl.class);  
  
    @Autowired  
    private ReplyDao replyDao;  
  
    // 글목록 구현  
    @Override  
    public List<ReplyVO> replyList(Integer b_num){  
        List<ReplyVO> myList = null;  
        myList = replyDao.replyList(b_num);  
        return myList;  
    }  
  
    // 글입력 구현  
    @Override  
    public int replyInsert(ReplyVO rvo) {  
        int result = 0;
```

```

        result = replyDao.replyInsert(rvo);
        return result;
    }

    // 글수정 구현
    @Override
    public int replyUpdate(ReplyVO rvo){
        int result = 0;
        result = replyDao.replyUpdate(rvo);
        return result;
    }

    // 글삭제 구현
    @Override
    public int replyDelete(int r_num){
        int result = 0;
        result = replyDao.replyDelete(r_num);
        return result;
    }
}

```

ReplyDao.java

```

package com.spring.reply.dao;
import java.util.List;
import com.spring.reply.vo.ReplyVO;

public interface ReplyDao {
    public List<ReplyVO> replyList(Integer b_num);
    public int replyInsert(ReplyVO rvo);
    public int replyUpdate(ReplyVO rvo);
    public int replyDelete(int r_num);
}

```

ReplyDaoImpl.java

```

package com.spring.reply.dao;
import java.util.List;
import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;

```

```

import org.springframework.stereotype.Repository;
import com.spring.reply.vo.ReplyVO;

@Repository
public class ReplyDaoImpl implements ReplyDao {

    @Autowired
    private SqlSession session;

    // 글목록 구현
    @Override
    public List<ReplyVO> replyList(Integer b_num) {
        return session.selectList("replyList", b_num);

    }

    // 글입력 구현
    @Override
    public int replyInsert(ReplyVO rvo){
        return session.insert("replyInsert");
    }

    // 글수정 구현
    @Override
    public int replyUpdate(ReplyVO rvo) {
        return session.update("replyUpdate");
    }

    // 글삭제 구현
    @Override
    public int replyDelete(int r_num) {
        return session.delete("replyDelete",r_num);
    }
}

```

src/query/Reply.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-

```

```

mapper.dtd">

<mapper namespace="com.spring.reply.dao.ReplyDao">
    <!-- 게시물 기본 조회 -->
    <select id="replyList" parameterType="reply" resultType="reply">
        select r_num, b_num, r_name, r_content, r_pwd,
               to_char(r_date,'YYYY-MM-DD HH24:MI:SS') as r_date
        from spring_reply
        where b_num = #{b_num}
        order by r_num desc
    </select>

    <!-- 게시물 등록 -->
    <insert id="replyInsert" parameterType="reply">
        /* Reply - replyInsert */
        <selectKey keyProperty="r_num" resultType="int" order="BEFORE">
            select spring_reply_seq.nextval from dual
        </selectKey>
        insert into spring_reply(
            r_num, b_num, r_name, r_content, r_pwd
        )values(
            #{r_num}, #{b_num}, #{r_name}, #{r_content}, #{r_pwd}
        )
    </insert>
    <update id="replyUpdate" parameterType="reply">
        /* Reply - replyUpdate */
        update spring_reply
        set r_content = #{r_content},
            r_date = sysdate
        where r_num = #{r_num}
    </update>

    <delete id="replyDelete" parameterType="reply">
        /* Reply - replyDelete */
        delete from spring_reply where r_num = #{r_num}
    </delete>
</mapper>

```

/WEB-INF/jsp/boardDetail.jsp 파일 마지막 부분에 아래와 같이 추가한다.

```
..종료
```

```
</div>  
<%-- ===== 상세 정보 보여주기 종료 ===== --%>  
  
<jsp:include page="reply.jsp"></jsp:include>  
</body>
```

/include/css/reply.css

```
@CHARSET "EUC-KR";  
ul {list-style: none;}  
#replyContainer {width:770px; height:300px; margin:10px;  
}  
#replyContainer h1 {  
    font-size: 10px; width: auto;  
    border-bottom: 1px solid #000066; padding-top: 10px;  
}  
#replyContainer > #comment_write {  
    padding: 10px 15px; border-bottom: 1px solid #000066;  
}  
#replyContainer > #comment_write label {  
    display: inline-block; width: 80px;  
    font-size: 14px; font-weight: bold; margin-bottom: 10px;  
}  
#replyContainer > #comment_write input[type='text'],  
#replyContainer > #comment_write input[type='password'],  
#replyContainer > #comment_write textarea {  
    border: 1px solid #ccc; vertical-align: middle;  
    padding: 3px 10px; font-size: 12px; line-height: 120%;  
}  
textarea {width: 480px; height: 30px; vertical-align: middle; }  
.comment_item {  
    font-size: 13px; color: #333; line-height: 150%;  
    padding: 10px 15px; border-bottom: 1px dotted #000066;  
}  
.comment_item .writer {  
    color: #555; line-height: 150%; margin-bottom: 5px;  
}  
.comment_item .writer input {
```

```

    vertical-align: middle;
}

.comment_item .writer .name {
    color: #222; font-weight: bold; font-size: 14px;
}

.update_form, .delete_btn, .update_btn, .reset_btn{
    margin-left:10px;
}

```

/WEB-INF/jsp/board/reply.jsp

```

<%@ page language="java" contentType="text/html; charset=utf-8"
pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0, user-scalable=no"/>
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
    <title>comment</title>
    <link rel="stylesheet" type="text/css" href="/include/css/reply.css" />
    <script type="text/javascript" src="/include/js/jquery-1.11.3.min.js"></script>
    <script type="text/javascript" src="/include/js/common.js"></script>
    <script type="text/javascript">
$(function() {
    /** 기본 댓글 목록 불러오기 */
    var b_num = "<c:out value='${detail.b_num}' />";
    listAll(b_num)

    /** 댓글 내용 저장 이벤트 */
    $("#replyInsert").click(function() {
        // 작성자 이름에 대한 입력여부 검사
        if (!chkSubmit($("#r_name"), "이름을")) return;
        else if (!chkSubmit($("#r_content"), "내용을")) return;
        else{
            var insertUrl = "/replies/replyInsert.do";
            /** 글 저장을 위한 Post 방식의 Ajax 연동 처리 */
        }
    });
});

```

on() 함수 : 동적으로 추가할 요소에 대한 이벤트를 미리 정의해 놓는 함수이다. 새로 추가될 요소에 이벤트 이기 때문에 이 이벤트가 정의되는 시점에서는 대상이 존재하지 않을 것이기 때문에 이 이벤트는 document 객체에 설정해야 한다.

```

$(document).on("이벤트명", "셀렉터"
" function(){ ...
}
});
```

```

$.ajax({
    url : insertUrl, //전송 url
    type : "post", // 전송 시 method 방식
    headers : {
        "Content-Type": "application/json",
        "X-HTTP-Method-Override": "POST"
    },
    dataType: "text",
    data : JSON.stringify({
        b_num:b_num,
        r_name:$("#r_name").val(),
        r_pwd:$("#r_pwd").val(),
        r_content:$("#r_content").val()
    }),
    error : function(){ //실행시 오류가 발생하였을 경우
        alert('시스템 오류입니다. 관리자에게 문의 하세요');
    },
    success : function(resultData){
        if(resultData=="SUCCESS"){
            alert("댓글 등록이 완료되었습니다.");
            dataReset();
            listAll(b_num);
        }
    }
});
```

/* 수정버튼 클릭시 수정폼 출력 */

```

$(document).on("click", ".update_form", function() {
    $(".reset_btn").click();
    var conText = $(this).parents("li").children().eq(1).html();
    console.log("conText: " + conText);
    $(this).parents("li").find("input[type='button']").hide();
    $(this).parents("li").children().eq(0).html();
    var conArea = $(this).parents("li").children().eq(1);
```

conArea.html("");

```

        var data = "<textarea name='content' id='content'>" + conText + "</textarea>";
        data += "<input type='button' class='update_btn' value='수정완료'>";
        data += "<input type='button' class='reset_btn' value='수정취소'>";
        conArea.html(data);

    });

    /** 초기화 버튼 */
    $(document).on("click", ".reset_btn", function() {
        var conText = $(this).parents("li").find("textarea").html();
        $(this).parents("li").find("input[type='button']").show();
        var conArea = $(this).parents("li").children().eq(1);
        conArea.html(conText);
    });

    /** 글 수정을 위한 Ajax 연동 처리 */
    $(document).on("click", ".update_btn", function() {
        var r_num = $(this).parents("li").attr("data-num");
        var r_content = $("#content").val();
        if (!chkSubmit($("#content"), "댓글 내용을")) return;
        else {
            $.ajax({
                url: '/replies/' + r_num + ".do",
                type: 'put',
                headers: {
                    "Content-Type": "application/json",
                    "X-HTTP-Method-Override": "PUT"
                },
                data: JSON.stringify({
                    r_content: r_content}),
                dataType: 'text',
                success: function(result) {
                    console.log("result: " + result);
                    if (result == 'SUCCESS') {
                        alert("수정 되었습니다.");
                        listAll(b_num);
                    }
                }
            });
        }
    });
}

```

```

        }
    });

    /* 글 삭제를 위한 Ajax 연동 처리 */
    $(document).on("click", ".delete_btn", function() {
        var r_num = $(this).parents("li").attr("data-num");
        console.log("r_num: " + r_num);

        if (confirm("선택하신 댓글을 삭제하시겠습니까?")) {
            $.ajax({
                type : 'delete',
                url : '/replies/' + r_num + ".do",
                headers : {
                    "Content-Type" : "application/json",
                    "X-HTTP-Method-Override" : "DELETE"
                },
                dataType : 'text',
                success : function(result) {
                    console.log("result: " + result);
                    if (result == 'SUCCESS') {
                        alert("삭제 되었습니다.");
                        listAll(b_num);
                    }
                }
            });
        }
    });

    // 리스트 요청 함수
    function listAll(b_num){
        $("#comment_list").html("");
        var url = "/replies/all/" + b_num + ".do";
        $.getJSON(url, function(data) {
            console.log(data.length);

            $(data).each(function() {
                var r_num = this.r_num;

```

```

        var r_name = this.r_name;
        var r_content = this.r_content;
        var r_date = this.r_date;
        addNewItem(r_num, r_name, r_content, r_date);
    });
}).fail(function() {
    alert("덧글 목록을 불러오는데 실패하였습니다. 잠시후에 다시 시도해 주세요.");
});
}

/* 새로운 글을 화면에 추가하기 위한 함수*/
function addNewItem(r_num, r_name, r_content, r_date) {
    // 새로운 글이 추가될 li태그 객체
    var new_li = $("<li>");
    new_li.attr("data-num", r_num);
    new_li.addClass("comment_item");

    // 작성자 정보가 지정될 <p>태그
    var writer_p = $("<p>");
    writer_p.addClass("writer");

    // 작성자 정보의 이름
    var name_span = $("<span>");
    name_span.addClass("name");
    name_span.html(r_name + "님");

    // 작성일시
    var date_span = $("<span>");
    date_span.html(" / " + r_date + " ");

    // 수정하기 버튼
    var up_input = $("<input>");
    up_input.attr({"type" : "button", "value" : "수정하기"});
    up_input.addClass("update_form");

    // 삭제하기 버튼
    var del_input = $("<input>");
    del_input.attr({"type" : "button", "value" : "삭제하기"});
}

```

```

    del_input.addClass("delete_btn");

    // 내용
    var content_p = $("<p>");
    content_p.addClass("con");
    content_p.html(r_content);

    // 조립하기
    writer_p.append(name_span).append(date_span).append(up_input).append(del_input)
    new_li.append(writer_p).append(content_p);
    $("#comment_list").append(new_li);

}

/** INPUT 태그들에 대한 초기화 함수*/
function dataReset() {
    $("#r_name").val("");
    $("#r_pwd").val("");
    $("#r_content").val("");
}

</script>

</head>
<body>

<div id="replyContainer">
    <h1></h1>
    <div id="comment_write">
        <form id="comment_form">
            <div>
                <label for="r_name">작성자</label>
                <input type="text" name="r_name" id="r_name" />
                <label for="r_name">비밀번호</label>
                <input type="password" name="r_pwd" id="r_pwd" />
                <input type="button" id="replyInsert" value="저장하기" />
            </div>
            <div>
                <label for="r_content">덧글 내용</label>
                <textarea name="r_content" id="r_content"></textarea>
            </div>
        </form>
    </div>
</div>

```

```

<ul id="comment_list">
    <!-- 여기에 동적 생성 요소가 들어가게 됩니다. -->
</ul>
</div>
</body>
</html>

```

7) 엑셀 파일로 다운로드 가능하도록 소스 추가

글번호	글제목	작성일	작성자
5	작게라도 시작해라. 그것은 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카	2015-12-02 07:08:43	최성희
4	열정이 있으면 꼭 만난다. 열망(desire)이 운명과 만남을 주선한다. -김영세	2015-12-02 06:35:19	도희라
3	우리들은 과거에의 집착보다 미래의 희망으로 살고 있다. -G. 무어	2015-12-02 06:33:08	김희진
2	노력에 관한 명언	2015-12-01 18:28:56	김철수
1	마음을 움직이는 명언	2015-12-01 18:28:04	홍길동

A	B	C	D
번호	제목	작성일	작성자
1	작게라도 시작해라. 그것은 전혀 시작하지 않는 것보다 낫다 - 프레드 데루카	2015-12-02 07:08:43	최성희
2	열정이 있으면 꼭 만난다. 열망(desire)이 운명과 만남을 주선한다. -김영세	2015-12-02 06:35:19	도희라
3	우리들은 과거에의 집착보다 미래의 희망으로 살고 있다. -G. 무어	2015-12-02 06:33:08	김희진
4	노력에 관한 명언	2015-12-01 18:28:56	김철수
5	마음을 움직이는 명언	2015-12-01 18:28:04	홍길동

- ① BoardController 클래스에 엑셀 다운로드 가능하도록 boardExcel() 메서드를 추가한다.

BoardController.java 파일 수정

```

package com.spring.board.controller;

import java.util.List;
중략...
@Controller
@RequestMapping(value="/board")

```

```

public class BoardController {
    Logger logger = Logger.getLogger(BoardController.class);

    @Autowired
    private BoardService boardService;
}

중략...
*****
* 엑셀 다운로드 구현하기
* 참고 : ListExcelView 클래스에서 맵타입으로 Model에 접근하게 된다.
*****
```

소스 제일 마지막 부분에 아래 소스를 추가한다.

```

@RequestMapping(value="/boardExcel", method=RequestMethod.GET)
public ModelAndView boardExcel(@ModelAttribute BoardVO bvo) {
    logger.info("boardExcel 호출 성공");
    // 정렬에 대한 기본값 설정
    if(bvo.getOrder_by()==null) bvo.setOrder_by("b_num");
    if(bvo.getOrder_sc()==null) bvo.setOrder_sc("DESC");

    //페이지 세팅
    Paging setPage(bvo);

    List<BoardVO> boardList = boardService.boardList(bvo);
    ModelAndView mv = new ModelAndView(new ListExcelView());
    mv.addObject("list", boardList);
    mv.addObject("template", "board.xlsx");
    mv.addObject("file_name", "board");

    return mv;
}
}
```

아직 ListExcelView 클래스를 정의 하지 않았기 때문에 에러 표시가 나타나게 된다. 그래서 바로 다음 페이지에 있는 클래스를 정의하게 되면 에러 표시가 사라지게 될 것이다.

② AbstractExcelView 추상 클래스의 buildExcelDocument 메소드를 재정의함으로써 구현하게 된다.

ListExcelView.java 파일 생성

```

package com.spring.common.excel;
import java.io.BufferedInputStream;
```

```

import java.io.FileInputStream;
import java.io.InputStream;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import net.sf.jxls.transformer.XLSTransformer;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.Workbook;
import org.springframework.web.servlet.view.document.AbstractExcelView;

public class ListExcelView extends AbstractExcelView {

    /**
     * 참고 : "Content-disposition: attachment"은 브라우저 인식 파일 확장자를 포함하여
     * 모든 확장자의 파일들에 대해, 다운로드시 무조건 "파일 다운로드" 대화상자를
     * 보여주도록 헤더속성이라 할 수 있다.
     */
    @Override
    protected void buildExcelDocument(Map<String, Object> model, HSSFWorkbook arg1,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {
        response.setHeader("Content-Disposition", "attachment;fileName=" + model.get("file_name")
            + "_" + new SimpleDateFormat("yyyyMMdd").format(Calendar.getInstance().getTime()) + ".xlsx");
        response.setContentType("application/x-msexcel; charset=EUC-KR");

        /**
         * 참고 : jXLS은 엑셀파일 포맷의 템플릿을 이용하여 엑셀 파일을 손쉽게
         * 생성하기 위한 패키지 (jXLS은 템플릿을 기반으로 최종 엑셀파일을 생성)
         */
        XLSTransformer trans = new XLSTransformer();
        String docRoot = request.getSession().getServletContext().getRealPath("");
        InputStream is = new BufferedInputStream(
            new FileInputStream(docRoot + "/WEB-INF/excel/" + model.get("template")));
        Workbook workbook = trans.transformXLS(is, model);
        is.close();
    }
}

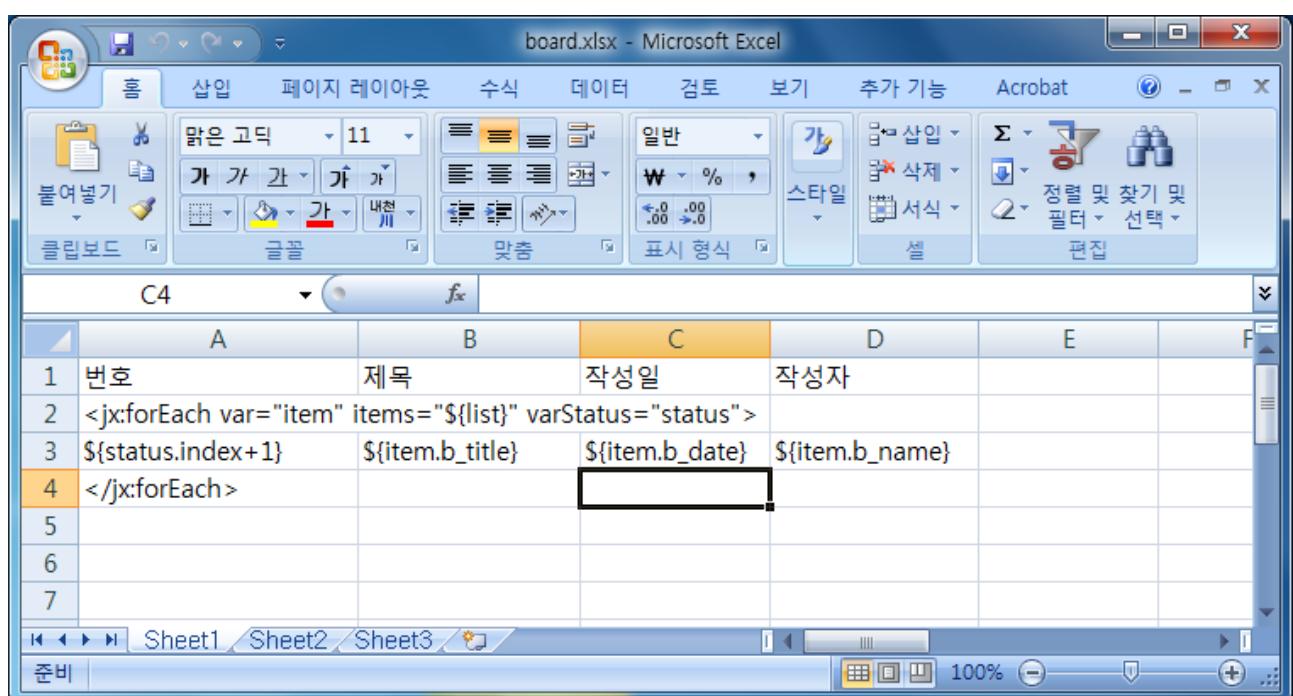
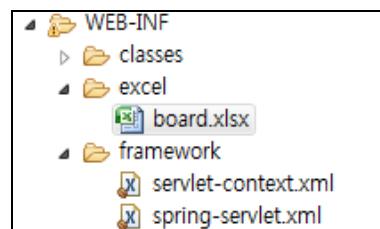
```

```

        workbook.write(response.getOutputStream());
    }
}

```

- ③ 템플릿 파일인 board.xlsx 엑셀 파일을 생성하여 /WEB-INF/excel/ 폴더에 붙여넣기 한다.



- ④ 엑셀 파일로 다운로드가 가능하도록 "boardList.jsp" 파일에 버튼을 추가한다. 위치는 "검색" 버튼 다음에 생성하면 된다.

```
/WEB-INF/jsp/board/boardList.jsp
```

중략...

```

<script type="text/javascript">
$(document).ready(function(){
    ...
    $("#excelDown").click(function(){
        $("#f_search").attr("method", "get");
        $("#f_search").attr('action', '/board/boardExcel.do');
        $("#f_search").submit();
    });
});

```

```

    });
    /* 글쓰기 버튼 클릭 시 처리 이벤트 */
});
</script>

```

중략...

```

<%-- ===== 검색기능 추가 시작 ===== --%>
<div id="boardSearch">
    <form id="f_search" name="f_search">
        <input type="hidden" id="page" name="page" value="${data.page}" />
        <input type="hidden" id="order_by" name="order_by" value="${data.order_by}" />
        <input type="hidden" id="order_sc" name="order_sc" value="${data.order_sc}" />
        <table border="0" summary="검색">
            <colgroup>
                <col width="70%"></col>
                <col width="30%"></col>
            </colgroup>
            <tr>
                <td id="btd1">
                    <label>검색조건 </label>
                    <select id="search" name="search">
                        <option value="all">전체 </option>
                        <option value="b_title">제목 </option>
                        <option value="b_content">내용 </option>
                        <option value="b_name">작성자 </option>
                    </select>
                    <input type="text" name="keyword" id="keyword"
                           value="검색어를 입력하세요" />
                    <input type="button" value="검색" id="searchData" />
                    <input type="button" value="엑셀 다운로드"
                           id="excelDown" />
                </td>
                <td id="btd2">한페이지에

```

중략...

Part 1. Spring Tool Suite 설정

1. 스프링 프로젝트

1.1 Spring Tool Suite(STS)을 이용한 스프링 프로젝트 개발 환경 설정

<http://spring.io/tools/sts/all>

The screenshot shows the 'Spring Tool Suite™ Downloads' section of the official Spring website. It features a large image of the STS logo (a shield with a gear and a wrench) and the text 'STS 3.7.1.RELEASE New & Noteworthy'. Below this are download links for Windows, Mac, and Linux, all based on Eclipse 4.5.1. The Windows section includes links for 32-bit and 64-bit versions (both 428MB). The Mac and Linux sections also link to 428MB zip files.

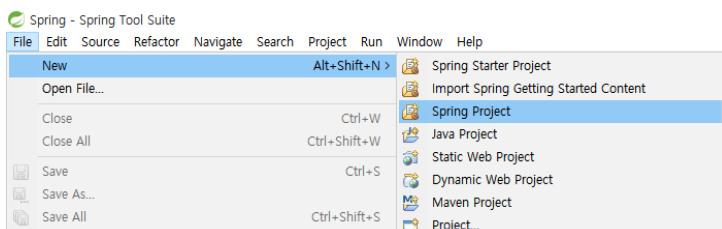
Use one of the links below to download an all-in-one distribution for your platform.
Or check the list of [previous Spring Tool Suite™ versions](#).

Update Site Archives
You can download archived versions of the update sites, if you want to install STS into an existing Eclipse installation or if you want to update an existing STS installation without accessing the hosted version of the update repository.

이클립스 설치와 동일하다.

1.2 스프링 프로젝트를 시작하는 두 가지 방법

- 1) Spring Boot를 이용하는 프로젝트의 생성(Spring Starter Project)
- 2) 스프링의 템플릿 프로젝트를 이용하는 프로젝트 생성(Spring Project)



'Spring Starter Project'의 경우 2014년에 개발된 Spring Boot라는 스프링의 하위 프로젝트를 이용해서 작성하는 방법이다. Spring Boot를 이용하면 개발자가 복잡한 설정 없이 모든 개발 환경이 준비되기 때문에 전혀 경험이 없는 개발자라도 쉽게 웹 프로젝트를 생성할 수 있다.

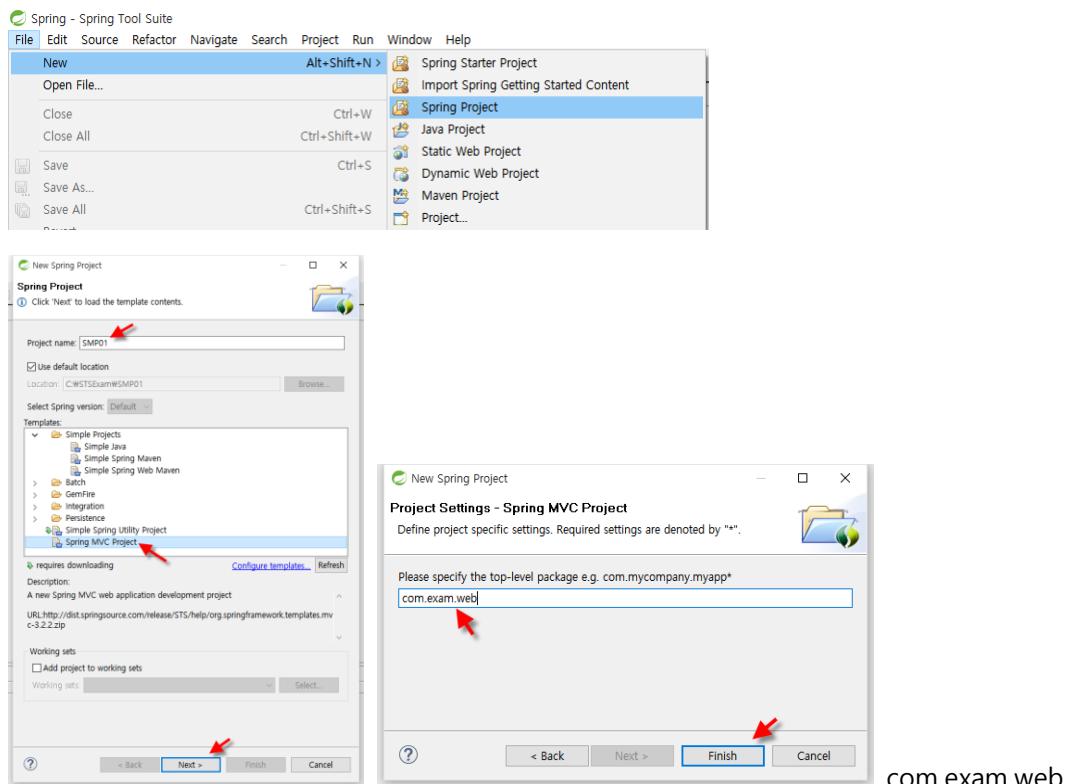
Spring Boot는 별도의 WAS 설정이 없어도 실행이 가능하다.

'Spring Project'의 경우는 조금 더 복잡하지만 실무에서 많이 적용되는 개발 방식이다.

	Spring Starter Project(Spring Boot)	Spring Project
장점	별도의 설정이 필요 없다. WAS 없이 실행이 가능하다. 로딩 시간이 짧아서 테스트가 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.	초기 테스트 환경 구성 등이 어렵다. WAS와 연동하는 경우 결과 확인에 많은 리소스를 소모한다.

'Spring Project' 을 이용한 프로젝트로 진행한다.

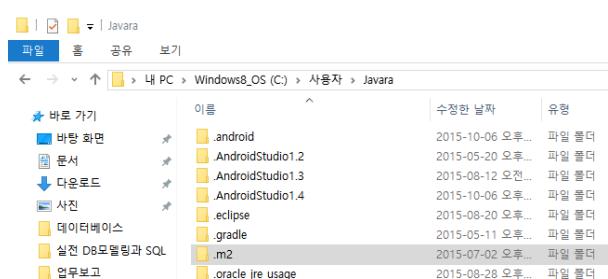
1.3 Spring Project 생성



Maven이 필요한 라이브러리를 다운로드 받기 위해 시간이 소요된다.

STS는 기본적으로 maven을 내장하고 있어서 프로젝트에 필요한 의존적 라이브러리를 자동으로 관리해 준다.

Maven이 다운로드 경로는 '.m2' 이다. PC 사용자 이름은 영문이어야 한다.



1.4 프로젝트 실행 점검

WAS(Tomcat 8)를 연결한 후에 실행한다.

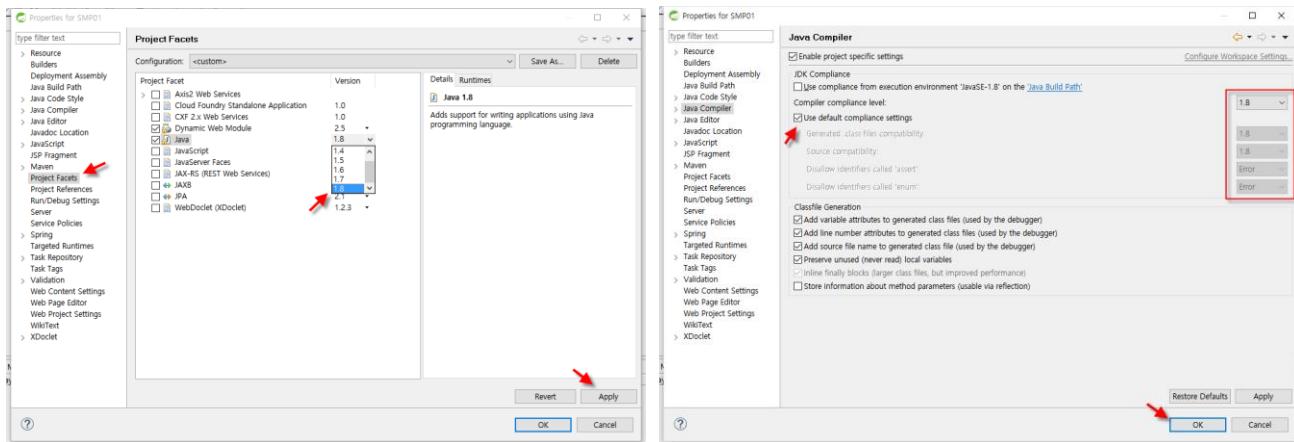


실행 결과 한글이 제대로 출력되지 않는다. 뒤에서 설명한다.

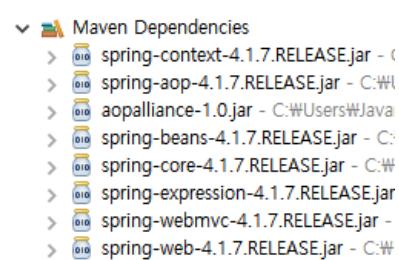
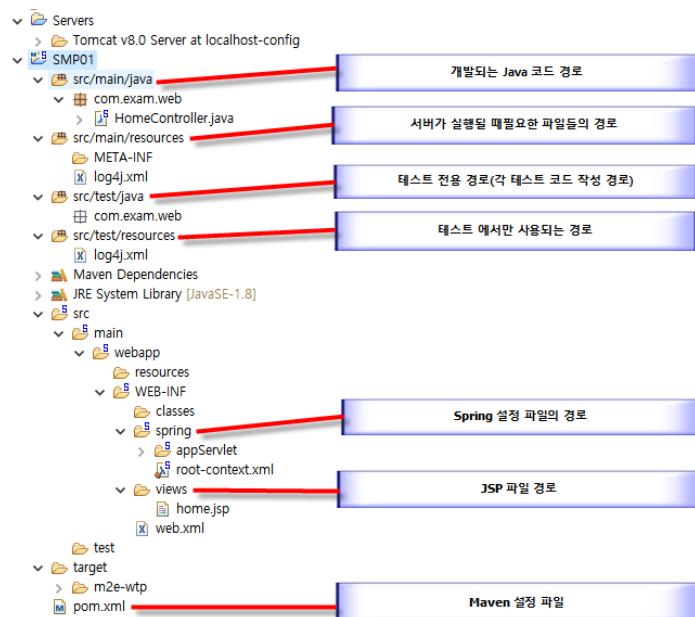
1.4.1 JDK 버전 처리

스프링 MVC 프로젝트를 생성하면 기본값으로 JDK 1.6으로 생성된다. 버전을 변경하려면 'Project Facets'로 변경한다.

프로젝트명 마우스 오른쪽 클릭 Properties 선택



1.4.2 스프링 MVC 프로젝트 구조



1.4.3 Spring 프레임워크 버전 설정(4.x)

Spring 4.x 이상의 버전을 사용하기 위해 버전 설정을 변경한다.

pom.xml 파일 수정한 후 저장

```
<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>
</properties>
```

2. Oracle Maven Repo(Repository : 저장소) 설정

Maven Site : <http://maven.apache.org>

The screenshot shows the Apache Maven Project website at <http://maven.apache.org>. The main navigation bar includes links for Apache, Maven, and Download Apache Maven. The left sidebar has a 'Download' menu item highlighted with a red arrow. The central content area is titled 'Downloading Apache Maven 3.3.3'. It contains system requirements and a table of available files. A second red arrow points to the 'apache-maven-3.3.3-bin.zip' link in the 'Files' section.

Link	Checksum	Signature
Binary tar.gz archive apache-maven-3.3.3-bin.tar.gz	apache-maven-3.3.3-bin.tar.gz.md5	apache-maven-3.3.3-bin.tar.gz.asc
Binary zip archive apache-maven-3.3.3-bin.zip	apache-maven-3.3.3-bin.zip.md5	apache-maven-3.3.3-bin.zip.asc
Source tar.gz archive apache-maven-3.3.3-src.tar.gz	apache-maven-3.3.3-src.tar.gz.md5	apache-maven-3.3.3-src.tar.gz.asc
Source zip archive apache-maven-3.3.3-src.zip	apache-maven-3.3.3-src.zip.md5	apache-maven-3.3.3-src.zip.asc

2.1 Oracle 연동

1) zip파일을 C:\에 풀고 bin 디렉토리까지 path를 잡아준다. cmd에서 mvn을 쳐서 확인한다.

2) maven install을 이용하여 .m2에 oracle의 local repo(지역 저장소)를 작성한다.

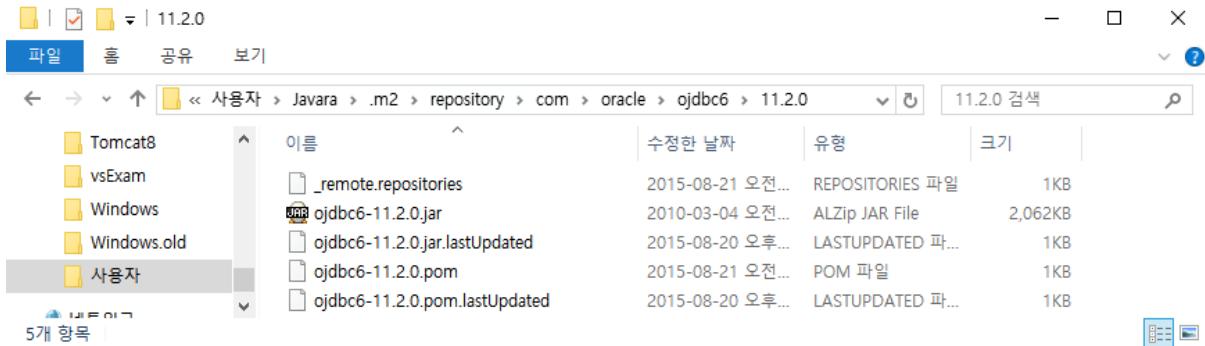
```
mvn install:install-file -Dfile=C:\app\Javara\product\11.2.0\dbhome_1\jdbc\lib\ojdbc6.jar  
DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.2.0 -Dpackaging=jar
```

```

선택 영역 표시판
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Javara>mvn install:install-file -Dfile=C:\app\Javara\product\11.2.0\dbhome_1\lib\ojdbc6.jar -DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.2.0 -Dpackaging=jar
[INFO] Scanning for projects...
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/maven-install-plugin-2.4.pom (7 KB at 4.8 KB/sec)
[INFO] -----
[INFO] Building Maven Stub Project (No POM)
[INFO] -----
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
Downloaded: https://repo.maven.apache.org/maven2/org/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.pom (3 KB at 8.2 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.pom (3 KB at 8.2 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.pom (2 KB at 3.5 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.pom (2 KB at 3.5 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.7/plexus-components-1.1.7.pom (5 KB at 16.1 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.7/plexus-components-1.1.7.pom (5 KB at 16.1 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/plexus/plexus/plexus-1.0.8/plexus-1.0.8.pom (8 KB at 23.9 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-8/plexus-container-default-1.0-alpha-8.pom (8 KB at 23.3 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.jar (12 KB at 19.0 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar (226 KB at 227.1 KB/sec)
[INFO] Installing C:\app\Javara\product\11.2.0\dbhome_1\lib\ojdbc6.jar to C:\Users\Javara\.m2\repository\com\oracle\ojdbc6\11.2.0\ojdbc6-11.2.0.jar
[INFO] Installing C:\Users\Javara\AppData\Local\Temp\mvninstall17897912266389833942.pom to C:\Users\Javara\.m2\repository\com\oracle\ojdbc6\11.2.0\ojdbc6-11.2.0.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.810 s
[INFO] Finished at: 2015-08-21T11:07:04+09:00
[INFO] Final Memory: 8M/188M
[INFO] -----
```

3) 아래와 같이 ojdbc6.jar의 이름이 ojdbc6-11.2.0.jar로 변경되어 repo에 복사된다.



4) 프로젝트의 pom.xml에 아래와 같은 repo 정보를 추가한다.

```

<!-- oracle 11g -->
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0</version>
</dependency>
```

OracleConnectionTest.java

```

package com.exam.web;
import java.sql.Connection;
import java.sql.DriverManager;
import org.junit.Test;
```

```

public class OracleConnectionTest {

    private static final String DRIVER = "oracle.jdbc.OracleDriver";
    private static final String URL = "jdbc:oracle:thin:@localhost:1521:orcl";
    private static final String USER = "scott";
    private static final String PW = "tiger";

    @Test
    public void testConnection() throws Exception{
        Class.forName(DRIVER);
        try( Connection con = DriverManager.getConnection(URL, USER,PW)){
            System.out.println(con);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

클래스명을 오른쪽 클릭하여 Run As -> JUnit Test 실행

Console Markers Progress

<terminated> OracleConnectionTest (1) [JUnit] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe

oracle.jdbc.driver.T4CConnection@2ff5659e

Connection 객체 생성 확인

2.2 MySQL 연동

MySQL Connector/J을 '<http://dev.mysql.com/downloads/connector/j/>'에서 다운로드 받는다.

pom.xml

```

<!-- MySQL -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.36</version>
</dependency>

```

MySQLConnectionTest.java

```

package com.exam.web;
import java.sql.Connection;
import java.sql.DriverManager;
import org.junit.Test;

```

```

public class MySQLConnectionTest {

    private static final String DRIVER = "com.mysql.jdbc.Driver";
    private static final String URL = "jdbc:mysql://127.0.0.1:3306/books";
    private static final String USER = "root";
    private static final String PW = "1234";

    @Test
    public void testConnection() throws Exception{
        Class.forName(DRIVER);
        try(Connection con = DriverManager.getConnection(URL, USER,PW)){
            System.out.println(con);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

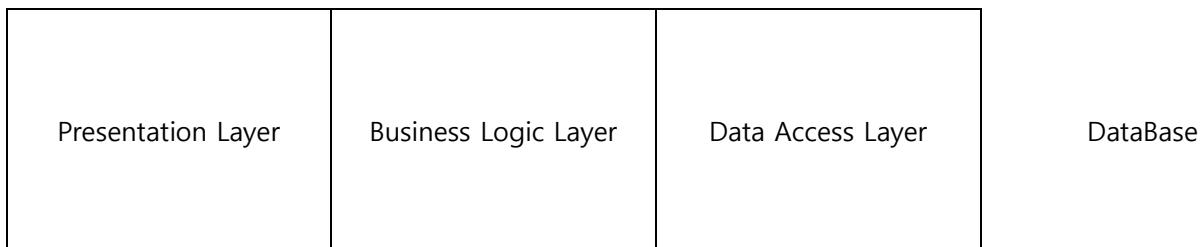
2.3 Spring, MyBatis, Oracle의 연결 설정

아래와 같은 작업으로 진행한다.

- 1) Spring과 MyBatis를 연동하기 위한 라이브러리 설정
 - 2) 데이터베이스와의 연결을 담당하는 DataSource 객체 설정
 - 3) MyBatis의 핵심인 SqlSessionFactory 객체 설정 및 테스트
- 개발 초기에 한 번만 설정하면 된다.

2.4 스프링 웹 프로젝트의 구성

일반적인 웹 프로젝트는 3개의 레이어(혹은 tier)로 구성된다.

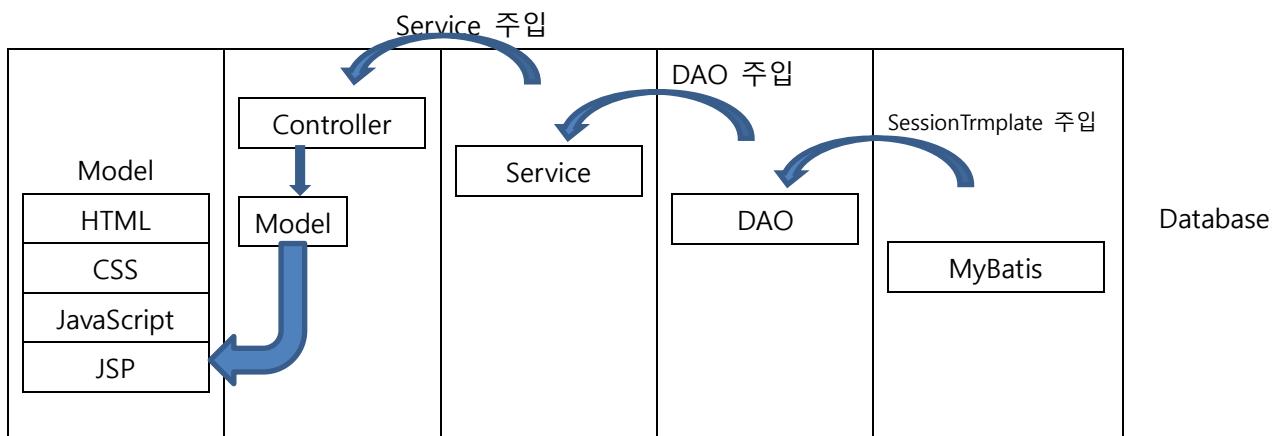


Presentation Layer – UI를 담당하는 구성요소들이 들어간다.

Business Logic Layer – 서비스 계층이라고 하며, 고객의 요구사항을 반영하는 계층이다. 사용자의 환경이 아닌 기능적인 요구사항을 구현한 곳이다. 비즈니스 계층은 어떤 형태의 데이터가 필요하고 반환될 것인지를 결정한다.

Data Access Layer – 흔히 Persistence Layer라고 하기도 한다. 데이터 처리를 전문으로 담당한다.

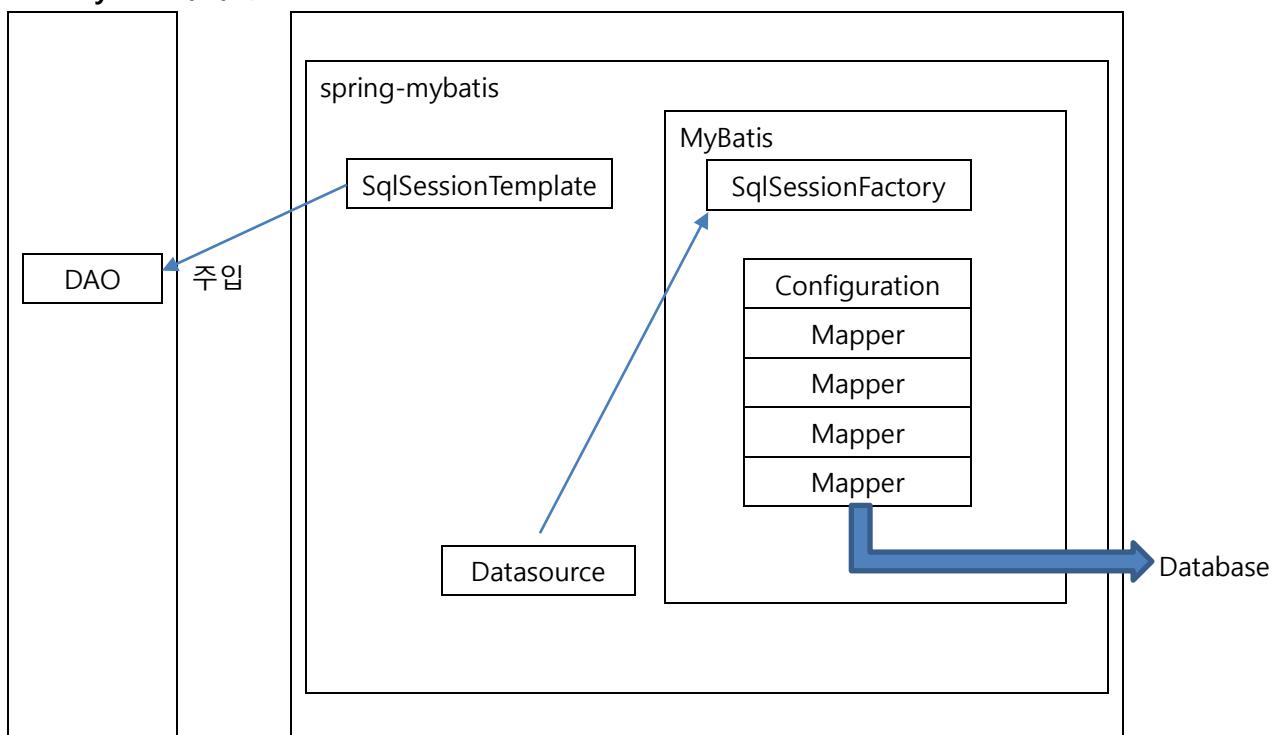
위의 구성을 좀 더 세분화 시켜서 다음과 같은 형태로 사용된다.



Presentation Layer의 경우 JSP와 같은 View를 구성하는 부분과 Controller 부분으로 분리하여 작성한다.

Data Access Layer의 경우 MyBatis가 추가되고 DAO의 경우 MyBatis를 호출하고 사용하는 구조로 작성된다.

2.5 MyBatis의 구성



2.6 Spring과 MyBatis 연동

spring-jdbc, spring-test, MyBatis, mybatis-spring 추가

pom.xml을 이용해서 필요한 jar 파일들을 다운로드 하는 것과 스프링 설정 파일을 수정하는 작업을 한다. Spring과 MyBatis 사이에는 두 프레임워크의 연결 역할을 하는 MyBatis-Spring 모듈이 필요하다. 이를 위해 pom.xml에 다음과 같은 프레임워크 라이브러리를 추가해야 한다.

이름	pom.xml
MyBatis	<dependency>

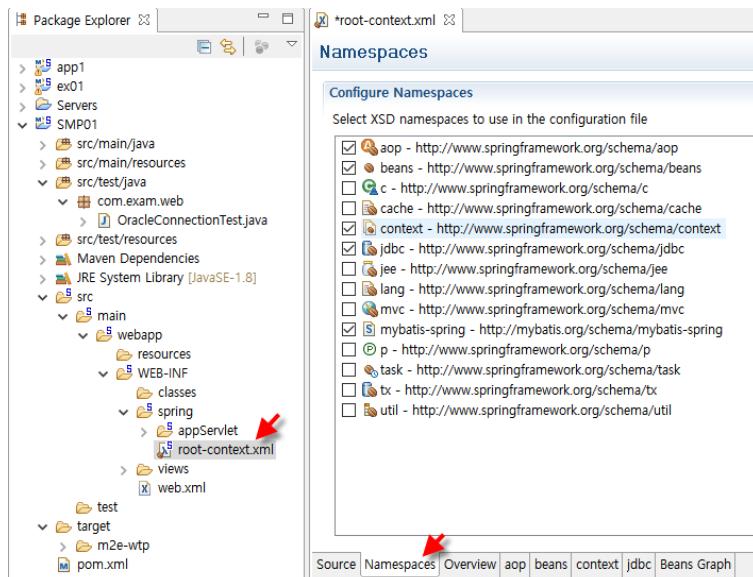
	<pre> <groupId>org.mybatis</groupId> <artifactId>mybatis</artifactId> <version>3.2.8</version> </dependency></pre>
MyBatis-Spring	<pre> <dependency> <groupId>org.mybatis</groupId> <artifactId>mybatis-spring</artifactId> <version>1.2.2</version> </dependency></pre>
spring-jdbc	<pre> <dependency> <groupId>org.springframework</groupId> <artifactId>spring-jdbc</artifactId> <version>\${org.springframework-version}</version> </dependency></pre>
spring-test	<pre> <dependency> <groupId>org.springframework</groupId> <artifactId>spring-test</artifactId> <version>\${org.springframework-version}</version> </dependency></pre>

spring-test는 Spring과 MyBatis가 정상적으로 연동되었는지를 확인하는 용도로만 사용한다.

2.6.1 Spring Project에서 root-context.xml 파일 설정

SMP01의 src/main/webapp/WEB-INF/spring/root-context.xml 파일은 STS가 스프링 프로젝트를 생성할 때 만들어 주는 파일에서 중요한 파일이다.

스프링과 관련된 설정을 할 때, 웹 자원과 관련되지 않은 모든 자원(resources)의 설정을 위해 존재한다.(웹과 관련된 설정은 appServlet 폴더 내에 있는 servlet-context.xml 파일에서 한다.)



옵션을 선택한 후 Source 탭을 선택하여 확인한다.

```
*root-context.xml [x]
<?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:mybatis="http://mybatis.org/schema/mybatis-spring"
       xsi:schemaLocation="http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc-4.1.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-4.1.xsd
                           http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.1.xsd">

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

</beans>
```

2.7 Oracle과의 연결을 담당하는 DataSource 설정

spring-jdbc 모듈의 클래스를 이용해서 root-context.xml에 다음과 같이 DataSource을 추가한다.

```
<!-- Oracle과 연결을 담당하는 DataSource 설정 -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl" />
    <property name="username" value="scott" />
    <property name="password" value="tiger" />
</bean>
```

driverClass가 아닌 driverClassName임에 유의한다.

JUnit 테스트

DataSourceTest.java

```
package com.exam.web;
import java.sql.Connection;
import javax.sql.DataSource;
import javax.inject.Inject;
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
    private DataSource ds;

    @Test
}
```

```

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

```

@RunWith, @ContextConfiguration 애노테이션은 테스트 코드를 실행할 때 스프링이 로딩되도록 하는 부분이다. @ContextConfiguration의 location 속성 경로는 xml 파일을 이용해서 스프링이 로딩된다. 인스턴스 변수의 @Inject 애노테이션 처리된 DataSource는 스프링이 생성해서 주입되므로 개발자가 객체 생성 혹은 다른 작업을 하지 않아도 된다. Junit 버전은 4.1.1 이상을 사용한다.

```

Console > Markers > Progress
<terminated> DataSourceTest (3) [JUnit] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015.10.19. 오후 3:35:02)
INFO : org.springframework.test.context.support.DefaultTestContextBootstrapper - Loaded default TestExecutionListener class names from location [null]
INFO : org.springframework.test.context.support.DefaultTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.test.context.web.ServletTestExecutionListener@1d1f33c]
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from file [C:\STSEXAM\SMP01\src\main\webapp\WEB-INF\applicationContext.xml]
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from file [C:\STSEXAM\SMP01\src\main\webapp\WEB-INF\datasource.xml]
INFO : org.springframework.context.support.GenericApplicationContext - Refreshing org.springframework.context.support.GenericApplicationContext@1d1f33c
INFO : org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor - JSR-330 'javax.inject.Inject' annotation found on field of class org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter - Mapped "[/,methods=[GET]]" onto public java.lang.Object org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleRequest(org.springframework.web.HttpRequest)
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter - Looking for @ControllerAdvice: org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter
INFO : org.springframework.web.servlet.handler.SimpleUrlHandlerMapping - Mapped URL path [/resources/**] onto handler 'org.springframework.web.servlet.resource.ResourceHttpRequestHandler'
INFO : org.springframework.jdbc.datasource.DriverManagerDataSource - Loaded JDBC driver: oracle.jdbc.OracleDriver
oracle.jdbc.driver.T4CConnection@3e08ff24
INFO : org.springframework.context.support.GenericApplicationContext - Closing org.springframework.context.support.GenericApplicationContext@1d1f33c

```

2.8 MyBatis 연결

DataSource의 연결은 MyBatis의 설정과는 관계 없으므로 먼저 설정하고 테스트해야 한다. DataSource가 정상적으로 설정된 이후에는 MyBatis와 Oracle을 연동시키는 작업을 한다.

2.8.1 SqlSessionFactory 객체 설정

Oracle과 스프링 연동의 핵심은 Connection을 생성하고 처리하는 SqlSessionFactory 객체이다.

SqlSessionFactory는 데이터베이스와의 연결과 SQL의 실행에 대한 모든 것을 가진 중요한 객체이다.

스프링을 이용할 때는 SqlSessionFactory를 생성해 주는 특별한 객체를 설정해주는 SqlSessionFactoryBean이라는 클래스를 사용한다. SqlSessionFactoryBean은 'root-context.xml'에 등록한다.

```

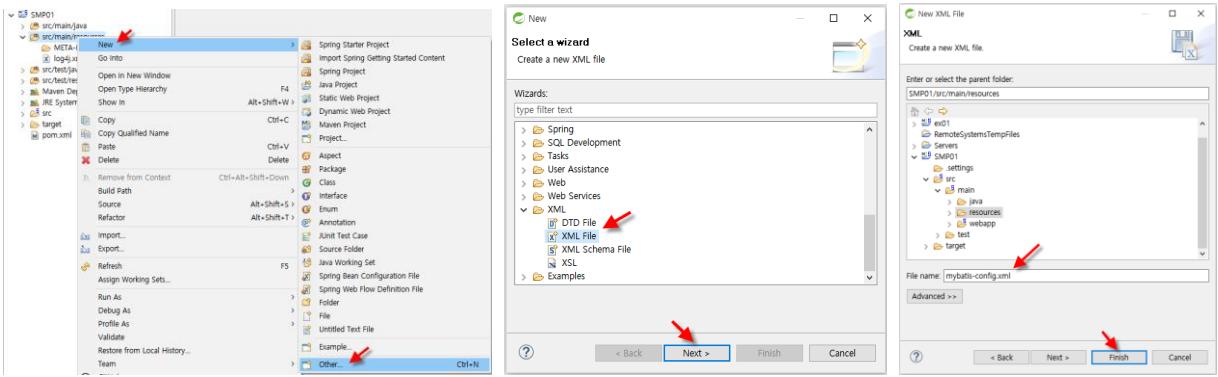
<!-- DataBase와의 연결과 SQL의 실행을 위한 SqlSessionFactory 객체 설정 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
</bean>

```

2.8.2 mybatis-config.xml 파일 추가

MyBatis는 SQL Mapper 프레임워크로 별도의 설정 파일을 가질 수 있다. 이를 이용하면 스프링 설정과 별도로 사용하는 모든 MyBatis의 설정 기능을 활용할 수 있다.

'/src/main/resources'에 mybatis-config.xml 파일을 생성한다.



mybatis-config.xml 설정

```
<?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>
```

수정한 mybatis-config.xml 파일이 스프링이 동작할 때 같이 동작하도록 설정한다.

root-context.xml 수정

```
<!-- DataBase와의 연결과 SQL의 실행을 위한 SqlSessionFactory 객체 설정 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:/mybatis-config.xml" />
</bean>
```

2.8.3 Mybatis 연결 테스트

MyBatis와 Oracle 연결이 제대로 되었는지를 확인한다.

MyBatisTest.java

```
package com.exam.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)
@ContextConfiguration(locations={"file:src/main/webapp/WEB-INF/spring/**/*.xml"})
```

```

public class MyBatisTest {

    @Inject
    private SqlSessionFactory sqlFactory;

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

    @Test
    public void testSession() throws Exception{
        try(SqlSession session = sqlFactory.openSession()){
            System.out.println(session);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

```

Console [ ] Markers [ ] Progress [ ]
<terminated> MyBatisTest (2) [JUnit] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015. 10. 19. 오후 3:46:30)
INFO : org.springframework.test.context.support.DefaultTestContextBootstrapper - Loaded default TestExecutionListener class names from location [META-INF\org\springframework\test\context\support\DefaultTestExecutionListener.class]
INFO : org.springframework.test.context.support.DefaultTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.test.context.web.ServletTestExecutionListener@37313c65]
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from file [C:\STSEXAM\SPM01\src\main\webapp\WEB-INF\beans.xml]
INFO : org.springframework.beans.factory.xml.XmlBeanDefinitionReader - Loading XML bean definitions from file [C:\STSEXAM\SPM01\src\main\webapp\WEB-INF\context.xml]
INFO : org.springframework.context.support.GenericApplicationContext - Refreshing org.springframework.context.support.GenericApplicationContext@6767c1fc
INFO : org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor - JSR-330 'javax.inject.Inject' annotation found and supported
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - Mapped "[[/],methods=[GET]]" onto public java.lang.String com.example.controller.HelloController.hello()
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter - Looking for @ControllerAdvice: org.springframework.context.annotation.ConfigurationClassPostProcessor$Processor@2f67a4d3
INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter - Looking for @ControllerAdvice: org.springframework.context.annotation.ConfigurationClassPostProcessor$Processor@2f67a4d3
INFO : org.springframework.web.servlet.handler.SimpleUrlHandlerMapping - Mapped URL path [/resources/**] onto handler 'org.springframework.web.servlet.resource.ResourceHttpRequestHandler@37313c65'
INFO : org.springframework.jdbc.datasource.DriverManagerDataSource - Loaded JDBC driver: oracle.jdbc.OracleDriver
org.apache.ibatis.session.defaults.DefaultSqlSessionFactory@37313c65
org.apache.ibatis.session.defaults.DefaultSqlSession@2f67a4d3
INFO : org.springframework.context.support.GenericApplicationContext - Closing org.springframework.context.support.GenericApplicationContext@6767c1fc: stopping

```

org.apache.ibatis.session.defaults.DefaultSqlSessionFactory@...

org.apache.ibatis.session.defaults.DefaultSqlSession@...

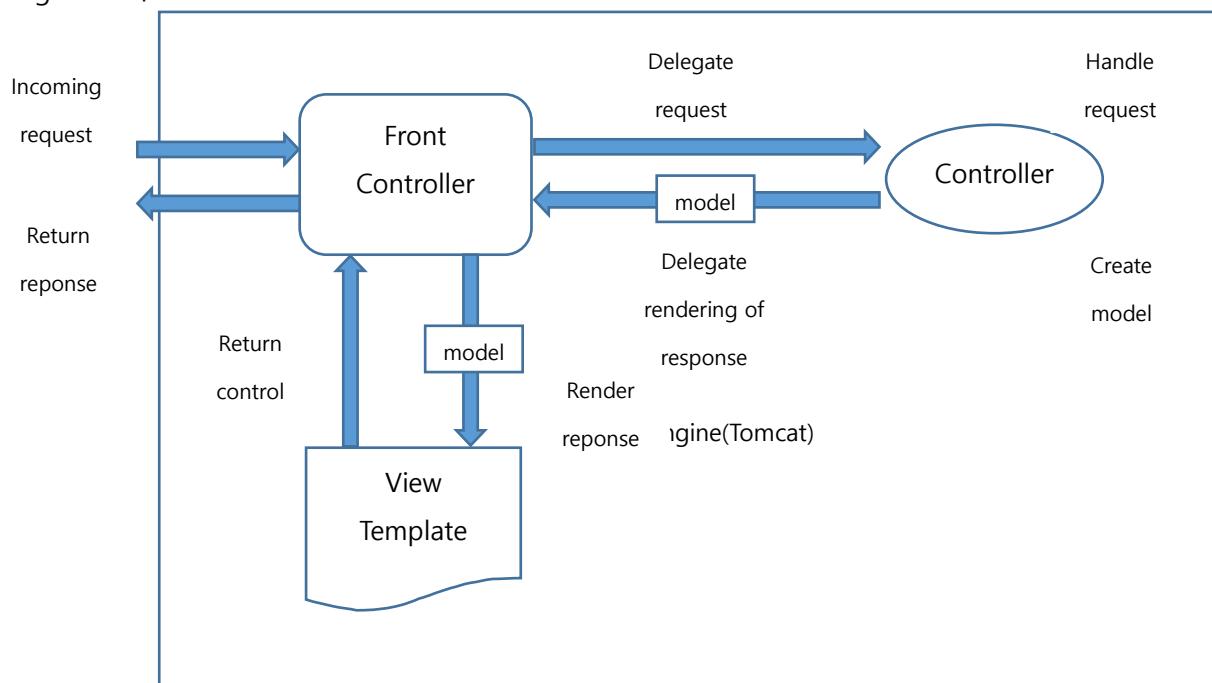
위와 같은 로그가 보이면 정상적으로 동작하는 것이다.

3. 모델 2 방식의 Spring MVC

3.1 모델 2에서 Front Controller 패턴

모델 2 방식이 개발자와 웹 퍼블리셔 간의 분업에 좋은 방식이지만 각 컨트롤러 사이의 중복적인 코드의 문제와 개발자의 개발 패턴의 차이 등의 문제로 모델 2 방식은 좀 더 강제적인 형태인 Front Controller 방식을 적용한다.

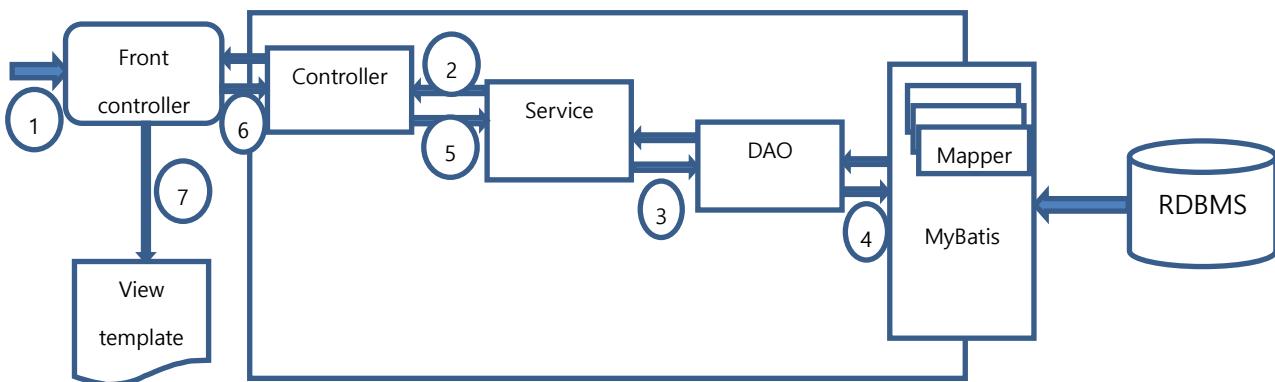
Spring MVC 구조



Front Controller 패턴의 가장 중요한 변화는 전체 로직의 일부분만을 컨트롤러가 처리하도록 한다. 흔히 '위임(Delegation)'이라 하는데 전체 로직의 일부를 컨트롤러에 위임하고 모든 흐름의 제어는 Front Controller가 담당한다.

위와 같은 구조를 사용하면 개발자가 작성하는 컨트롤러는 전체 로직의 일부분만을 처리하는 형태가 되기 때문에 개발자가 작성해야 하는 전체 코드는 줄어들게 된다. 모든 컨트롤러는 Front Controller의 일부분을 구현하는 형태이므로 좀 더 규격화된 코드를 작성하게 된다.

Spring MVC 구조의 흐름



Spring MVC가 처리해 주는 작업	개발자가 직접 해야 하는 작업
URI를 분석해서 적절한 컨트롤러를 찾는 작업	특정 URI에 동작하는 컨트롤러를 설계하는 작업
컨트롤러에 필요한 메소드를 호출하는 작업	서비스 객체의 생성
컨트롤러의 결과 데이터를 뷰로 전달하는 작업	DAO 객체의 생성
적절한 뷰를 찾는 작업	컨트롤러 내에서 원하는 결과를 얻는 메소드 설계 뷰에서 전달받은 데이터의 출력

Spring MVC를 사용하면 개발의 전체 흐름은 개발자가 제어하지 않고 개발자는 필요한 부품을 끼워 넣는 형태의 작업을하게 된다. Spring MVC의 경우에는 이 부품이 컨트롤러가 된다.

3.2 Spring MVC의 컨트롤러

- 파라미터 수집 : 웹에서 가장 많이 하는 작업은 사용자의 request에 필요한 데이터를 추출하고, 이를 VO(Value Object) 혹은 DTO(Data Transfer Object)로 변환하는 파라미터의 수집 작업이다. Spring MVC의 컨트롤러는 이러한 처리를 자동으로 해주기 때문에 개발 시간을 크게 단축할 수 있다.
- 애노테이션을 통한 간편 설정 : Spring MVC의 설정은 크게 XML과 애노테이션을 사용할 수 있지만 애노테이션을 사용하는 경우가 더 많습니다. 애노테이션을 사용하기 때문에 개발자는 클래스나 메소드의 선언에 필요한 애노테이션을 추가하는 작업을 통해서 request나 response에 필요한 모든 처리를 완료할 수 있다.
- 로직의 집중 : 기존 모델 2는 특정한 URI마다 컨트롤러를 개발하는 경우가 많았지만 Spring MVC 컨트롤러의 경우 각 메소드마다 필요한 애노테이션을 설정할 수 있기 때문에 여러 메소드를 하나의 컨트롤러에 집중해서 작성할 수 있다.
- 테스트의 편리함 : 스프링은 테스트 모듈을 사용해서 Spring MVC로 작성된 코드를 WAS의 실행 없이도 테스트 할 수 있는 편리한 방법을 제공한다.

Spring MVC 컨트롤러는 기존의 다른 프레임워크와 비교하면 구성 방식에서 특이한 점이 많다. 컨트롤러는 다음과 같은 점이 기존 Java 코드에 비해 특이하다.

- Spring MVC 컨트롤러는 상속이나 인터페이스를 구현하지 않아도 된다. 기존의 프레임워크들과 달리 Spring MVC에서는 컨트롤러 작성 시 아무런 제약이 없다. 그대신 해야 하는 작업은 '@Controller'라는 애노테이션을 추가해야 한다.
- 메소드의 파라미터와 리턴 타입에 대한 제약이 없습니다. 클래스가 특정 부모 클래스나 인터페이스가 없으니 메소드에 대한 제약도 존재하지 않는다. 파라미터 타입과 리턴 타입에 대한 제약이 없는 관계로 기존보다 훨씬 자유로운 코드를 만들 수 있다.
- Spring MVC가 제공하는 유용한 클래스들이 존재한다. Spring MVC의 경우 다양한 클래스를 이용해서 필요한 작업을 수월하게 진행할 수 있다. 예를 들어 파일 업로드 처리나 유효성 검사 등을 제공하고 있기 때문에 개발자는 필요한 클래스를 이용해서 빠른 시간 내에 개발할 수 있다.

3.2.1 Spring Project의 servlet-context.xml

가장 먼저 Spring MVC 컨트롤러가 어떤 설정을 통해서 동작하는지에 대한 이해이다. appServlet 폴더에 있는 servlet-context.xml은 Spring MVC 관련 설정만을 분리하기 위해서 만들어진 파일이다.

servlet-context.xml

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

    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

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

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

    <context:component-scan base-package="com.exam.web" />
</beans:beans>
```

설정	설명
<annotation-driven>	클래스 선언에 애노테이션을 이용해서 컨트롤러를 작성할 수 있다는 선언. InternalResourceViewResolver 부분은 뷰를 어떻게 처리하는지 설정. /WEB-INF/ 경로는 절대로 브라우저에서 직접 접근할 수 없는 경로이기 때문에 컨트롤러의 호출이 우선되는 모델 2 방식에 맞는 구조가 된다.
<resources>	웹에서 이미지나 CSS, JavaScript 파일과 같이 고정된 자원들의 위치.

<context:component-scan>	base-package 속성에 해당되는 패키지 내부의 클래스들을 조사한다는 의미. 이는 <annotation-driven>과 같이 결합해서 해당 패키지에 애노테이션 처리가 된 컨트롤러를 작성만 해주면 자동으로 인식되게 한다.
--------------------------	---

3.2.2 Spring MVC에서 주로 사용하는 애노테이션

애노테이션	설명	사용
@Controller	Spring MVC의 컨트롤러 객체임을 명시	클래스
@RequestMapping	특정 URI에 매칭되는 클래스나 메소드임을 명시	클래스, 메소드
@RequestParam	Request에서 특정한 파라미터의 값을 찾아낼 때 사용	파라미터
@RequestHeader	Request에서 특정 HTTP 헤더 정보를 추출할 때 사용	파라미터
@PathVariable	현재의 URI에서 원하는 정보를 추출할 때 사용.	파라미터
@CookieValue	현재 사용자의 쿠키가 존재할 때 쿠키의 이름을 이용해서 쿠키의 값을 추출.	파라미터
@ModelAttribute	자동으로 해당 객체를 뷰까지 전달.	메소드, 파라미터
@SessionAttribute	세션상에서 모델의 정보를 유지하고 싶은 경우 사용.	클래스
@InitBinder	파라미터를 수집해서 객체로 만들 경우에 커스터아미징	메소드
@ResponseBody	리턴 타입이 HTTP의 응답 메시지로 전송	메소드, 리턴타입
@RequestBody	Request 문자열이 그대로 파라미터에 전달	파라미터
@Repository	DAO 객체	클래스
@Service	서비스 객체	클래스

3.2.3 기초적인 컨트롤러 생성

1) void 리턴 타입의 경우

SampleController.java

```
package com.exam.web;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class SampleController {

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

    @RequestMapping("callA")
    public void callA(){
        logger.info("A 호출");
    }
}
```

```

    }

    @RequestMapping("callB")
    public void callB(){
        logger.info("B 호출");
    }
}

```

@RequestMapping을 이용해서 특정한 URI 경로로 메소드가 실행된다.

'Run on Server'로 실행하면 스프링이 다음과 같은 로그를 남긴다.

```

INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - 
Mapped           "{[/],methods=[GET]}"      onto      public      java.lang.String
com.exam.web.HomeController.home(java.util.Locale,org.springframework.ui.Model)

INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - 
Mapped "{/callA}" onto public void com.exam.web.SampleController.callA()

INFO : org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping - 
Mapped "{/callB}" onto public void com.exam.web.SampleController.callB()

```

<http://localhost/web/callA>로 실행하면



Servers의 server.xml

```

125     <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs" pattern="%h %l %u %t &quot;
126
127     <Context docBase="SMP01" path="/web" reloadable="true" source="org.eclipse.jst.jee.server:SMP01"/></Host>
128   </Engine>
129 </Service>
130 </Server>
<

```

Design Source

Console Markers Progress

Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_60\bin\javaw.exe (2015-10-20 오후 3:23:39)
INFO : com.exam.web.HomeController - Welcome home! The client locale is ko_KR.
INFO : com.exam.web.SampleController - A 호출

현재 메소드의 리턴 타입이 void인 경우에는 Spring MVC는 현재 경로에 해당하는 jsp 파일을 실행한다.

2) String이 리턴 타입인 경우

ReturnStringController.java

```

package com.exam.web;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;

```

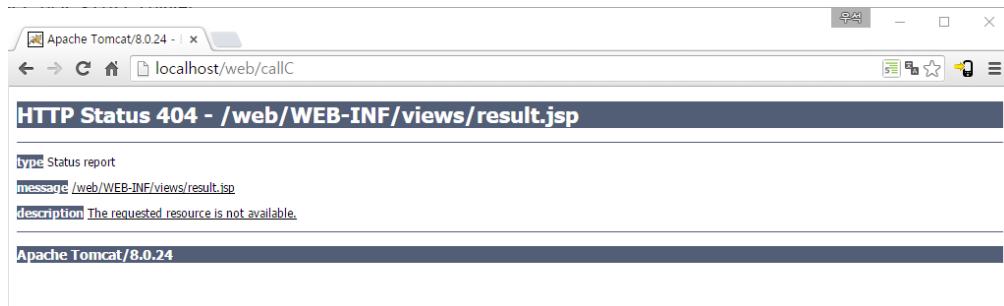
```

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

@Controller
public class ReturnStringController {
    private static final Logger logger = LoggerFactory.getLogger(SampleController2.class);
    @RequestMapping("callC")
    public String callC(@ModelAttribute("msg") String msg){
        logger.info("C 호출");
        return "result";
    }
}

```

callC() 메소드 내의 파라미터인 @ModelAttribute("msg")는 request 시 'msg' 이름의 파라미터를 문자열로 처리해 주고 뷰에 전달한다.



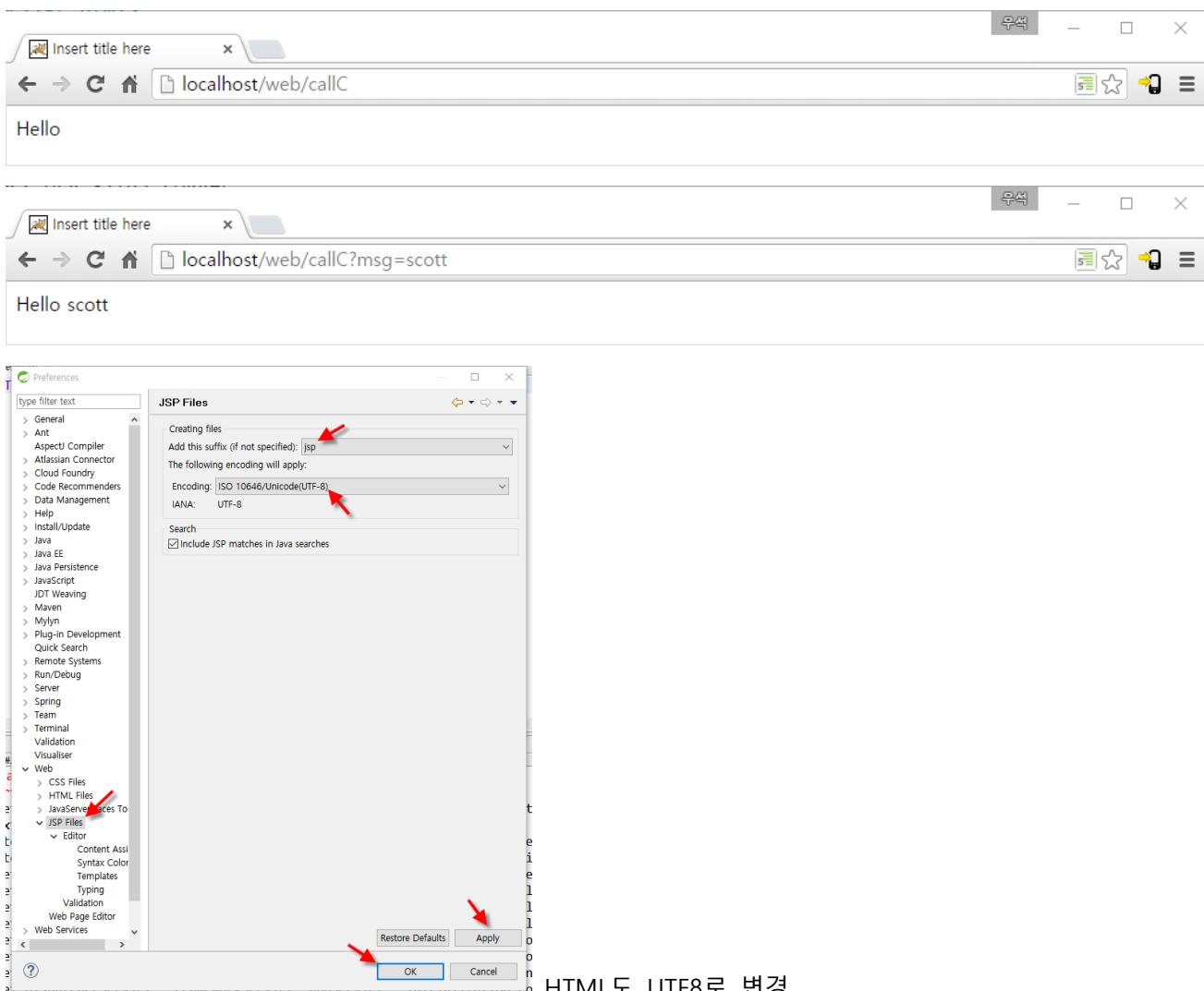
WEB-INF/views/result.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>메시지 출력 </title>
</head>
<body>
    <span>Hello ${msg}</span>
</body>
</html>

```

EL을 사용한 \${msg} 출력부분이다. Spring MVC의 @ModelAttribute("msg")는 자동으로 해당 객체를 뷰까지 전달한다.



3) 만들어진 결과 데이터를 전달해야 하는 경우

컨트롤러를 제작하면서 가장 많이 하는 작업은 다른 객체의 도움을 받아 만들어진 데이터를 뷰로 전달하는 일을 하는 것이다. 이때는 Spring MVC의 Model 객체를 사용해서 간단하게 처리할 수 있다.

MemberVO.java

```
package com.exam.domain;

public class MemberVOTest {

    private String name;
    private int age;

    public MemberVOTest(String name, int age) {
        super();
        this.name = name;
    }
}
```

```

        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    @Override
    public String toString() {
        return "MemberVO [name=" + name + ", age=" + age + "]";
    }
}

```

ResultDataController.java

```

package com.exam.web;
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 com.exam.domain.MemberVOTest;

@Controller
public class ResultDataController {

    private static final Logger logger = LoggerFactory.getLogger(SampleController3.class);

    @RequestMapping("member")
    public String member(Model model{

        MemberVOTest member = new MemberVOTest("홍길동", 20);
        logger.info("member 호출");
        model.addAttribute(member);

        return "memberDetail";
    }
}

```

Model 클래스는 뷰에 원하는 데이터를 전달하는 일종의 컨테이너 역할을 한다.

MemberVOTest 클래스의 객체를 Model이라는 객체를 이용해서 필요한 데이터를 담은 후 뷰(jsp)로 전달한다.

MemberVOTest 클래스의 객체 생성 이후에 addAttribute() 메소드를 이용해서 MemberVO 객체를 보관한다. addAttribute() 메소드는 크게 두 가지 형태로 사용한다.

- addAttribute("이름", 객체) : 객체의 특별한 이름을 부여해 뷰에서 이름값을 이용하여 객체를 처리.
- **addAttribute(객체)** : 이름을 지정하지 않은 경우에는 자동으로 저장되는 객체의 클래스명 첫 글자를 소문자로 처리한 클래스명을 이름으로 간주한다.

WEB-INF/views/memberDetail.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>회원 정보</title>
</head>
<body>
    <span>${memberVOTest.name}</span>
    <span>${memberVOTest.age}</span>
</body>
</html>
```



4) 리다이렉트를 해야 하는 경우

특정한 컨트롤러의 로직을 처리할 때 다른 경로를 호출해야 하는 경우 Spring MVC의 특별한 문자열인 '**redirect**'를 이용하는데 ':'을 이용하는 것을 주의한다.

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

RedirectController.java

```
package com.exam.web;
import org.slf4j.Logger;
```

```

import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

@Controller
public class RedirectController {

    private static final Logger logger = LoggerFactory.getLogger(SampleController4.class);

    @RequestMapping("redirectA")
    public String redirectA(RedirectAttributes retrr){
        logger.info("redirect 호출");

        retrr.addFlashAttribute("msg", "리다이렉트 호출");

        return "redirect:/callID"; //forward: 도 지원한다.
    }

    @RequestMapping("callID")
    public void callID(){
        logger.info("D 호출");
    }
}

```

redirectA() 메소드의 파라미터로 RedirectAttributes를 사용한다. 전달하고자 하는 데이터 'msg'를 추가하고 addFlashAttribute() 메소드는 임시 데이터를 전달한다. 리턴값은 'redirect:/...'로 시작하는 문자열을 반환하기 때문에 Spring MVC는 브라우저에서 callID를 호출한다.

redirectA.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>리다이렉트 A</title>
</head>
<body>
    <span>리다이렉트 : ${msg}</span>
</body>

```

```
</html>
```

callD.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>리다이렉트 Call D</title>
</head>
<body>
<span>리다이렉트 : ${msg}</span>
</body>
</html>
```

http://localhost/web/redirectA로 연결



5) JSON(JavaScript Object Notation) 데이터를 생성하는 경우

pom.xml에 Jackson-databind 라이브러리를 추가해야 한다.

pom.xml에 추가

```
<!-- JSON -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.5.4</version>
</dependency>
```

Spring MVC의 컨트롤러에서 JSON 데이터를 생성하기 위해서 개발자는 적절한 객체를 반환해주고 @ResponseBody 애노테이션을 추가해 주어야 한다.

JSONController.java

```
package com.exam.web;
```

```

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
import com.exam.domain.MemberVOTest;

@Controller
public class JSONController {

    @RequestMapping("/json")
    public @ResponseBody MemberVOTest json(){

        MemberVOTest mvo = new MemberVOTest("이순신", 21);

        return mvo;
    }
}

```

메소드의 실행 결과로 일반 객체를 리턴한다. 이것으로 JSON 처리가 끝난다.



3.3 WAS 없이 컨트롤러 테스트

Spring MVC에서는 spring-test 모듈을 통해서 별도의 WAS 구동 없이도 컨트롤러를 테스트 할 수 있다. 스프링 3.2부터는 jUnit만을 사용해서 Spring MVC에서 작성된 컨트롤러를 테스트할 수 있다. 테스트를 하기 위해서는 pom.xml의 javax.servlet 라이브러리의 버전을 변경해야 한다.

pom.xml 변경

```

<!-- Servlet -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>

```

src/test/java/

WasNoControllerTest.java

```

package com.exam.web;
import javax.inject.Inject;

```

```

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;

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

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

    @Inject
    private WebApplicationContext wac;

    private MockMvc mockMvc;

    @Before
    public void setup(){
        this.mockMvc = MockMvcBuilders.webAppContextSetup(this.wac).build();
        logger.info("Mock 설정");
    }

    @Test
    public void testMock() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/callA"));
    }
}

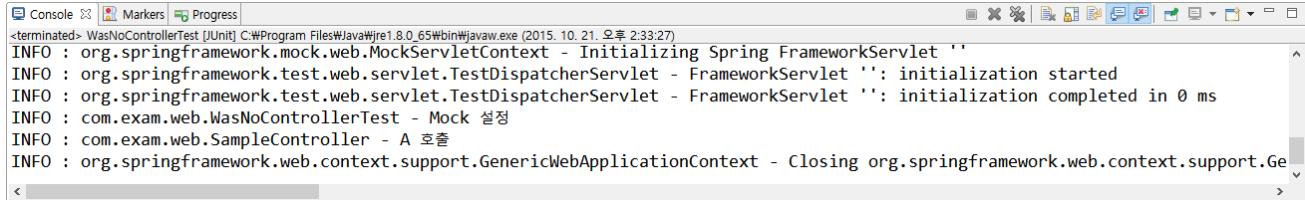
```

선언부에 `@WebAppConfiguration`을 사용하는 것이 기존 Spring와 Spring MVC를 테스트하는데 가장 큰 차이이다.

MockMvc는 브라우저에서 요청과 응답을 의미하는 객체로 간주하면 된다. 매번 테스트를 진행할 때마다 가상의 요청과 응답을 처리하기 위해서 setup() 메소드에서는 @Before 애노테이션으로 처리되어 매번 테스트 메소드의 실행 전에 MockMvc 객체를 만들어 낸다.

testMock()에서 MockMvc를 사용해서 perform()이라는 메소드를 실행하게 되는데 이때 get(), post() 등을 이용해서 GET 방식이나 POST 방식의 호출을 사용한다.

아래와 같은 로그가 출력되면 테스트 코드가 정상이다.



```
<terminated> WasNoControllerTest [JUnit] C:\#Program Files\Java\jre1.8.0_65\bin\javaw.exe (2015.10.21. 오후 2:33:27)
INFO : org.springframework.mock.web.MockServletContext - Initializing Spring FrameworkServlet ''
INFO : org.springframework.test.web.servlet.TestDispatcherServlet - FrameworkServlet '':: initialization started
INFO : org.springframework.test.web.servlet.TestDispatcherServlet - FrameworkServlet '':: initialization completed in 0 ms
INFO : com.exam.web.WasNoControllerTest - Mock 설정
INFO : com.exam.web.SampleController - A 호출
INFO : org.springframework.web.context.support.GenericWebApplicationContext - Closing org.springframework.web.context.support.Ge
```

테스트를 사용할 때 불편함 보다는 좋은 면도 있다.

- 웹 페이지를 테스트 하려면 매번 입력 항목을 입력해서 제대로 동작하는지를 확인하는데 이때 여러 번 웹 페이지에서 입력하는 것보다 테스트 코드를 통해서 처리하는 것이 훨씬 개발 시간을 단축시킬 수 있다.
- JSP 등에서 발생하는 에러를 해결하는 과정에 매번 WAS에 만들어진 컨트롤러 코드를 수정해서 배포하는 작업은 많은 시간을 소모하게 된다.
- 컨트롤러에서 결과 데이터만을 확인할 수 있기 때문에 문제 발생 시 원인을 파악할 수 있는 시간이 절약된다.

4. Spring과 MyBatis

MyBatis는 JDBC에서 개발자가 직접 처리하는 PreparedStatement의 '?'에 대한 설정이나 ResultSet을 이용한 처리가 이루어지기 때문에 기존 방식에 비해 개발의 생산성이 좋아진다.

MyBatis는 애노테이션을 지원하고 인터페이스 애노테이션을 통해서 SQL 문을 설정하고 처리할 수 있다.

MyBatis를 이용할 때 SQL 문을 사용하는 방식은 크게 다음과 같다.

- XML만을 이용해 SQL 문을 설정하고 DAO에서 XML을 찾아서 실행하는 코드를 작성하는 방식

장점 : SQL문은 별도의 XML로 작성되기 때문에 SQL문의 수정이나 유지보수가 쉽다.

단점 : 개발 시 코드의 양이 많아지고 복잡성이 증가 한다.

- 애노테이션과 인터페이스만을 이용해서 SQL문을 설정한다.

장점 : 별도의 DAO 없이도 개발이 가능하기 때문에 생산성이 크게 증가한다.

단점 : SQL문을 애노테이션으로 작성하므로 매번 수정이 일어나는 경우에 다시 컴파일한다.

- 인터페이스와 XML로 작성된 SQL문의 활용

장점 : 간단한 SQL문은 애노테이션으로 하고 복잡한 SQL문은 XML로 처리하므로 상황에 따라서 유연하게 처리한다.

단점 : 개발자에 따라 개발 방식의 차이가 있을 수 있기 때문에 유지보수가 중요한 프로젝트의 경우 부적

합하다.

국내의 대부분 프로젝트는 XML만을 이용해서 SQL문을 작성하고 별도의 DAO를 만드는 방식으로 개발한다. 이 방식의 최대 장점은 SQL문을 완전히 분리해서 처리하기 때문에 추후에 SQL문의 변경이 일어나도 대체가 수월하다.

MyBatis를 XML을 이용해서 작성하는 경우 코딩의 순서는 다음과 같다.

- 테이블의 생성 및 개발 준비
- 테이블의 생성 및 기타 데이터베이스 관련 설정
- 도메인 객체의 설계 및 클래스 작성
- DAO 인터페이스 작성
- 실행해야 하는 기능을 인터페이스로 정의
- XML Mapper의 생성과 SQL문의 작성
- XML 작성 및 SQL문 작성
- MyBatis에서 작성된 XML Mapper를 인식하도록 설정
- DAO 구현
- DAO 인터페이스를 구현한 클래스 작성
- 스프링상에 DAP 등록 및 테스트

4.1 테이블 생성 및 개발 준비

4.1.1 데이터베이스의 테이블 생성

tbl_member 테이블

```
create table TBL_MEMBER (
    userid VARCHAR2(30) not null,
    userpw VARCHAR2(30) not null,
    username VARCHAR2(50) not null,
    email VARCHAR2(100),
    regdate DATE default sysdate,
    updateday DATE default sysdate,
    PRIMARY KEY (userid)
);
```

4.1.2 도메인 객체를 위한 클래스 설계

MemberVO.java

```
package com.exam.domain;
import java.util.Date;

public class MemberVO {
```

```
private String userid;
private String userpw;
private String username;
private String email;
private Date regdate;
private Date updateday;

public MemberVO() {
    super();
}

public MemberVO(String userid, String userpw, String username, String email, Date regdate,
Date updateday) {
    super();
    this.userid = userid;
    this.userpw = userpw;
    this.username = username;
    this.email = email;
    this.regdate = regdate;
    this.updateday = updateday;
}

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 Date getRegdate() {
    return regdate;
}
public void setRegdate(Date regdate) {
    this.regdate = regdate;
}
public Date getUpdateday() {
    return updateday;
}
public void setUpdateday(Date updateday) {
    this.updateday = updateday;
}
@Override
public String toString() {
    return "MemberVO [userid=" + userid + ", userpw=" + userpw + ", username=" +
username + ", email=" + email + ", regdate=" + regdate + ", updateday=" + updateday + "]";
}
}

```

4.2 DAO 인터페이스 작성

DAO로 작성하는 일은 추후에 데이터베이스 관련 기술이 변경되더라도 DAO만을 변경해서 처리할 수 있기 때문이다.

예제를 위해 현재 날짜를 체크하는 기능과 tbl_member 테이블에 데이터를 추가하는 기능으로 구성한다.

MemberDAO.java

```

package com.exam.dao;
import com.exam.domain.MemberVO;

public interface MemberDAO {
    public String getDate();
    public void insertMember(MemberVO mvo);
    public MemberVO selectMember();
}

```

4.3 XML Mapper 작성

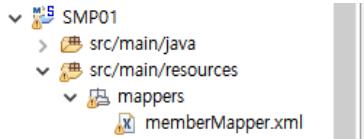
DAO 인터페이스를 작성한 후에 이를 사용하는 SQL문을 작성한다. MyBatis에서는 SQL문을 저장하기 위해 Mapper를 사용한다. Mapper는 XML과 인터페이스를 이용할 수 있는데 XML을 사용하는 경우에는 다음과 같은 순서로 작업한다.

- XML로 저장된 Mapper의 위치(저장 경로) 결정
- XML Mapper 파일을 작성하고 필요한 DTD 추가
- SQL 작성

1) Mapper 파일의 저장 경로와 XML Mapper 작성

XML로 작성되는 Mapper 파일의 경우 Java로 작성된 클래스와 경로를 분리해 주는 것이 유지보수에 있어서 필수적이다.

Java 파일이 아닌 경우 'resources' 폴더에 'mappers'를 생성하고 'memberMapper.xml' 파일을 생성한다.



memberMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.exam.mapper.MemberMapper">

    <select id="getDate" resultType="string">
        select sysdate from DUAL
    </select>

    <insert id="insertMember">
        insert into TBL_MEMBER (userid, userpw, username, email)
        values (#{userid}, #{userpw}, #{username}, #{email})
    </insert>

    <select id="selectMember" resultType="com.exam.domain.MemberVO">
        select *
        from TBL_MEMBER
    </select>
</mapper>
```

'namespace' 속성은 클래스의 패키지와 유사한 용도로 사용되며 MyBatis 내에서 원하는 SQL 문을 찾아서 실행할 때 동작한다.

2) mybatis-spring에서 XML Mapper 인식

MyBatis가 동작하면 XML Mapper를 인식해야만 정상적인 동작이 가능하므로 root-context.xml에 아래와 같이 수정한다.

root-context.xml 수정

```
<!-- DataBase와의 연결과 SQL의 실행을 위한 SqlSessionFactory 객체 설정 -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:/mybatis-config.xml"></property>
    <b><property name="mapperLocations" value="classpath:mappers/**/*Mapper.xml"></property></b>
</bean>
```

4.4 DAO 인터페이스 구현

MyBatis에서 DAO를 이용하는 경우는 SqlSessionTemplate이라는 것을 이용해서 DAO를 구현하므로 우선적으로 SqlSessionTemplate을 설정하는 작업부터 한다.

1) SqlSessionTemplate 설정

DAO의 작업에서 가장 번거로운 작업은 데이터베이스와 연결하고 작업이 완료된 후에 연결을 close()하는 작업이다. mybatis-spring 라이브러리에는 이것을 처리할 수 있는 SqlSessionTemplate라는 클래스가 있어 이를 사용하면 개발자들이 직접 연결하고 종료하는 작업을 줄일 수 있다.

SqlSessionTemplate은 SqlSession 인터페이스를 구현한 클래스로 기본적인 트랜잭션의 관리나 쓰레드 처리의 안정성 등을 보장해 주고 데이터베이스의 연결과 종료를 책임진다.

SqlSessionTemplate은 SqlSessionFactory를 생성자로 주입해서 설정한다.

root-context.xml에 추가

```
<!-- DAO 인터페이스 구현을 위한 SqlSessionTemplate 설정 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate" destroy-method="clearCache">
    <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

2) 구현 클래스 작성

MemberDAOImpl.java

```
package com.exam.dao;
import javax.inject.Inject;
import org.apache.ibatis.session.SqlSession;
import org.springframework.stereotype.Repository;
import com.exam.domain.MemberVO;

@Repository
public class MemberDAOImpl implements MemberDAO {
```

```

@Inject
private SqlSession sqlSession;

private static final String namespace = "com.exam.mapper.MemberMapper";

@Override
public String getDate() {
    // TODO Auto-generated method stub
    return sqlSession.selectOne(namespace + ".getDate");
}

@Override
public void insertMember(MemberVO mvo) {
    // TODO Auto-generated method stub
    sqlSession.insert(namespace + ".inserMember", mvo);
}

@Override
public MemberVO selectMember() {
    // TODO Auto-generated method stub
    return (MemberVO)sqlSession.selectOne(namespace + ".selectMember");
}
}

```

선언부의 @Repository 애노테이션은 DAO를 스프링에 인식시키기 위해 사용한다. (Repository:저장소)

4.5 Spring에 Bean으로 등록

MemberDAOImpl이 @Repository 애노테이션이 설정되더라도 Spring에서 해당 패키지를 스캔하지 않으면 제대로 Bean이 등록되지 못한다.

root-context.xml에 추가

```

<!-- Spring의 Bean으로 등록 -->
<context:component-scan base-package="com.exam.dao"></context:component-scan>

```

4.6 테스트 코드 작성

src/test/java

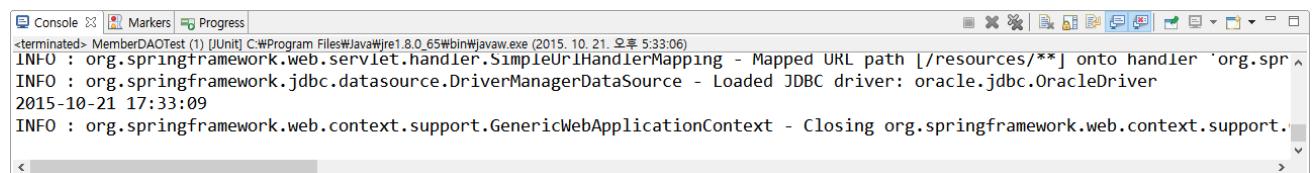
MemberDAOTest.java

```
package com.exam.web;
```

```
import javax.inject.Inject;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;
import com.exam.dao.MemberDAO;
import com.exam.domain.MemberVO;

@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(locations={"file:src/main/webapp/WEB-INF/spring/**/*.xml"})
public class MemberDAOTest {
    @Inject
    private MemberDAO mDao;
    @Test
    public void testDay() throws Exception {
        System.out.println(mDao.getDate());
    }
    @Test
    public void testInsertMember() throws Exception {
        MemberVO mvo = new MemberVO();
        mvo.setUserid("testuser1");
        mvo.setUserpw("test1234");
        mvo.setUsername("홍길동");
        mvo.setEmail("testuser@abc1234.com");
        mDao.insertMember(mvo);
    }
}
```

jUnit 테스트 : testDay() 메소드가 실행되어 날짜와 시간이 출력된다.



Insert 문장의 실행 내용을 출력되지 않는다.

4.7 MyBatis의 로그 log4jdbc-log4j2

자세한 로그를 보기 위해서는 log4jdbc-log4j2 라이브러리가 필요하다.

pom.xml에 추가

```
<!-- mybatis의 로그 log4jdbc-log4j2 -->
<dependency>
    <groupId>org.bgee.log4jdbc-log4j2</groupId>
    <artifactId>log4jdbc-log4j2-jdbc4</artifactId>
    <version>1.16</version>
</dependency>
```

라이브러리가 추가된 후에 데이터베이스와 연결되는 드라이버 클래스와 연결 URL을 아래와 같이 수정한다.

root-context.xml 설정

```
<!-- mybatis의 로그 log4jdbc-log4j2 -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
    <property name="url" value="jdbc:log4jdbc:oracle:thin:@localhost:1521:orcl"></property>
    <property name="username" value="scott"></property>
    <property name="password" value="tiger"></property>
</bean>
```

log4jdbc-log4j2가 제대로 동작하려면 별도의 로그 관련 설정이 필요하다.

'/src/main/resources' 폴더에 'log4jdbc.log4j2.properties' 파일과 'logback.xml' 파일을 추가한다.

log4jdbc.log4j2.properties

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.Slf4j.Slf4jSpyLogDelegator
```

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include resource="org/springframework/boot/logging/logback/base.xml"/>
    <!-- log4jdbc-log4j2 -->
    <logger name="jdbc.sqlonly" level="DEBUG"/>
    <logger name="jdbc.sqltiming" level="INFO"/>
    <logger name="jdbc.audit" level="WARN"/>
    <logger name="jdbc.resultset" level="ERROR"/>
    <logger name="jdbc.resultsettable" level="ERROR"/>
    <logger name="jdbc.connection" level="INFO"/>
</configuration>
```

위 두 파일을 '/src/test/resources'에 복사한다.

MemberDAO.java에 추가

```
public MemberVO selectMember();
```

memberMapper.xml에 추가

```
<select id="selectMember" resultType="com.exam.domain.MemberVO">
    select *
    from TBL_MEMBER
</select>
```

MemberDAOImpl.java에 재정의

```
@Override
public MemberVO selectMember() {
    // TODO Auto-generated method stub
    return (MemberVO)sqlSession.selectOne(namespace + ".selectMember");
}
```

MemberDAOTest.java 파일 jUnit 테스트

MemberDAOTest 클래스에서 기존 테스트 메소드는 주석 처리하고 아래의 테스트 메소드를 추가한다.

```
@Test
public void testInsertMember() throws Exception {
    MemberVO mvo = new MemberVO();
    mvo.setUserid("testuser2");
    mvo.setUserpw("test1234");
    mvo.setUsername("이순신");
    mvo.setEmail("testuser2@abc1234.com");
    mDao.insertMember(mvo);
}

@Test
public void testSelectMemeber() throws Exception {
    MemberVO mvo = new MemberVO();
    mvo = mDao.selectMember();
    System.out.println(mvo);
}
```

jUnit 실행하면 테이블에 있는 데이터를 가져오고 insert 정보를 출력한다.

```

Console [ ] Markers [ ] Progress [ ]
<terminated> MemberDAOtest (1) [Junit] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (2015.10.22 오후 1:42:40)
INFO : org.springframework.jdbc.datasource.DriverManagerDataSource - Loaded JDBC driver: net.sf.log4jdbc.sql.jdbcapi.DriverSpy
INFO : jdbc.connection - 1. Connection opened
INFO : jdbc.audit - 1. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(select *
   from TBL_MEMBER) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@24b6b8f6
INFO : jdbc.sqlonly - select * from TBL_MEMBER

INFO : jdbc.sqltiming - select * from TBL_MEMBER
(executed in 16 msec)
INFO : jdbc.audit - 1. PreparedStatement.execute() returned true
INFO : jdbc.resultset - 1. ResultSet.new ResultSet returned
INFO : jdbc.audit - 1. PreparedStatement.executeQuery() returned net.sf.log4jdbc.sql.jdbcapi.ResultSetSpy@37b70343
INFO : jdbc.resultset - 1. ResultSet.new MetaData returned oracle.jdbc.driver.OracleResultSetMetaData@1080b026
INFO : jdbc.resultset - 1. ResultSet.getType() returned 1003
INFO : jdbc.resultset - 1. ResultSet.next() returned true
INFO : jdbc.resultset - 1. ResultSet.getString(USERID) returned testuser1
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getString(USERPW) returned test1234
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getString(USERNAME) returned 흥길동
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getString(EMAIL) returned testuser@abc1234.com
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getTimestamp(REGDATE) returned 2015-10-22 13:23:00.0
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getTimestamp(UPDATEDAY) returned 2015-10-22 13:23:00.0
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|
|userid |userpw |username |regdate |updateday |
|-----|-----|-----|-----|-----|
|testuser1 |test1234 |흥길동 |2015-10-22 13:23:00.0 |2015-10-22 13:23:00.0 |
|-----|-----|-----|-----|-----|

```

```

Console [ ] Markers [ ] Progress [ ]
<terminated> MemberDAOtest (1) [Junit] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (2015.10.22 오후 1:42:40)
|testuser1 |test1234 |흥길동 |testuser@abc1234.com |2015-10-22 13:23:00.0 |2015-10-22 13:23:00.0 |
|-----|-----|-----|-----|-----|-----|
INFO : jdbc.resultset - 1. ResultSet.next() returned false
INFO : jdbc.resultset - 1. ResultSet.close() returned void
INFO : jdbc.audit - 1. PreparedStatement.getConnection() returned net.sf.log4jdbc.sql.jdbcapi.ConnectionSpy@435871cb
INFO : jdbc.audit - 1. Connection.getMetaData() returned oracle.jdbc.driver.OracleDatabaseMetaData@609640d5
INFO : jdbc.audit - 1. PreparedStatement.close() returned
INFO : jdbc.connection - 1. Connection closed
INFO : jdbc.audit - 1. Connection.close() returned
MemberVO [userid=testuser1, userpw=test1234, username=흥길동, email=testuser@abc1234.com, regdate=Thu Oct 22 13:23:00 KST 2015,
INFO : jdbc.connection - 2. Connection opened
INFO : jdbc.audit - 2. Connection.new Connection returned
INFO : jdbc.audit - 2. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 2. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 2. Connection.prepareStatement(insert into TBL_MEMBER (userid, userpw, username, email)
   values (?, ?, ?, ?)) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@749c877b
INFO : jdbc.audit - 2. PreparedStatement.setString(1, "testuser2") returned
INFO : jdbc.audit - 2. PreparedStatement.setString(2, "test1234") returned
INFO : jdbc.audit - 2. PreparedStatement.setString(3, "이순신") returned
INFO : jdbc.audit - 2. PreparedStatement.setString(4, "testuser2@abc1234.com") returned
INFO : jdbc.sqlonly - insert into TBL_MEMBER (userid, userpw, username, email) values ('testuser2', 'test1234', '이순신',
'testuser2@abc1234.com')
(INFO : jdbc.sqltiming - insert into TBL_MEMBER (userid, userpw, username, email) values ('testuser2', 'test1234', '이순신',
'testuser2@abc1234.com')
(executed in 0 msec)
INFO : jdbc.audit - 2. PreparedStatement.execute() returned false
INFO : jdbc.audit - 2. PreparedStatement.executeUpdate() returned 1
INFO : jdbc.audit - 2. PreparedStatement.close() returned
INFO : jdbc.connection - 2. Connection closed
INFO : jdbc.audit - 2. Connection.close() returned
INFO : org.springframework.web.context.support.GenericWebApplicationContext - Closing org.springframework.web.context.support.

```

4.8 MyBatis의 #{} 문법

새로운 사용자의 등록과 조회 처리를 한다.

XML Mapper를 사용하면 DAO 인터페이스 작업부터 한다.

MemberDAO.java 내용 추가

```

package com.exam.dao;
import com.exam.domain.MemberVO;

public interface MemberDAO {
    public String getDate();
}

```

```

public void insertMember(MemberVO mvo);
public MemberVO selectMember();

public MemberVO readMemberId(String userid) throws Exception;
public MemberVO readWithPW(String userid, String userpw) throws Exception;
}

```

main/resources/mappers

memberMapper.xml 내용 추가

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.exam.mapper.MemberMapper">
    <select id="getDate" resultType="string">
        select sysdate from DUAL
    </select>

    <insert id="insertMember">
        insert into TBL_MEMBER (userid, userpw, username, email)
        values (#{userid}, #{userpw}, #{username}, #{email})
    </insert>

    <select id="selectMember" resultType="com.exam.domain.MemberVO">
        select *
        from TBL_MEMBER
    </select>

    <select id="selectMember" resultType="com.exam.domain.MemberVO">
        select *
        from TBL_MEMBER
        where userid = #{userid}
    </select>

    <select id="readWithPW" resultType="com.exam.domain.MemberVO">
        select *
        from TBL_MEMBER
        where userid = #{userid} and userpw = #{userpw}

```

```
</select>  
</mapper>
```

작성된 XML Mapper는 parameterType은 생략하고 작성한다.

MyBatis의 경우 기본적으로 PreparedStatement를 이용해서 처리한다. 개발자가 PreparedStatement에 들어가는 파라미터를 사용할 때는 '#{ }' 기호를 이용해서 처리한다.

'#{ }'는 다음과 같은 규칙이 적용된다.

- 파라미터가 여러 속성을 가진 객체인 경우 '#{num}'은 getNum() 혹은 setNum()을 의미한다.
- 파라미터가 하나이고 기본 자료형이나 문자열인 경우 값이 그대로 전달된다.
- 파라미터가 Map타입인 경우 '#{num}'은 Map 객체의 키 값이 'num'인 값을 찾는다.

Mapper 인터페이스를 구현한 클래스를 작성한다.

MemberDAOImpl.java 수정

```
package com.exam.dao;  
import java.util.HashMap;  
import java.util.Map;  
import javax.inject.Inject;  
import org.apache.ibatis.session.SqlSession;  
import org.springframework.stereotype.Repository;  
import com.exam.domain.MemberVO;  
  
@Repository  
public class MemberDAOImpl implements MemberDAO {  
  
    @Inject  
    private SqlSession sqlSession;  
  
    private static final String namespace = "com.exam.mapper.MemberMapper";  
  
    @Override  
    public String getDate() {  
        // TODO Auto-generated method stub  
        return sqlSession.selectOne(namespace + ".getDate");  
    }  
  
    @Override  
    public void insertMember(MemberVO mvo) {
```

```

        // TODO Auto-generated method stub
        sqlSession.insert(namespace+".inserMember", mvo);
    }

    @Override
    public MemberVO selectMember() {
        // TODO Auto-generated method stub
        return (MemberVO)sqlSession.selectOne(namespace + ".selectMember");
    }

    @Override
    public MemberVO readMemberId(String userid) throws Exception {
        // TODO Auto-generated method stub
        return (MemberVO)sqlSession.selectOne(namespace + ".selectMember", userid);
    }

    @Override
    public MemberVO readWithPW(String userid, String userpw) throws Exception {
        // TODO Auto-generated method stub
        Map<String, Object> paramMap = new HashMap<String, Object>();

        paramMap.put("userid", userid);
        paramMap.put("userpw", userpw);

        return sqlSession.selectOne(namespace + ".readWithPW", paramMap);
    }
}

```

readWithPW()의 경우 파라미터가 2개 이상 전달되면 Map 타입의 객체를 구성해서 파라미터로 사용한다.

MemberDAOTest.java에 추가하고 다른 테스트 메소드는 주석 처리한다.

```

@Test
public void testSelectMemeberId() throws Exception {
    MemberVO mvo = new MemberVO();
    mvo = mDao.readMemberId("testuser2");
    System.out.println(mvo);
}

@Test
public void testReadWithPW() throws Exception {

```

```

MemberVO mvo = new MemberVO();
mvo = mDao.readWithPW("testuser1","test1234");
System.out.println(mvo);
}

```

Console | Markers | Progress |

```

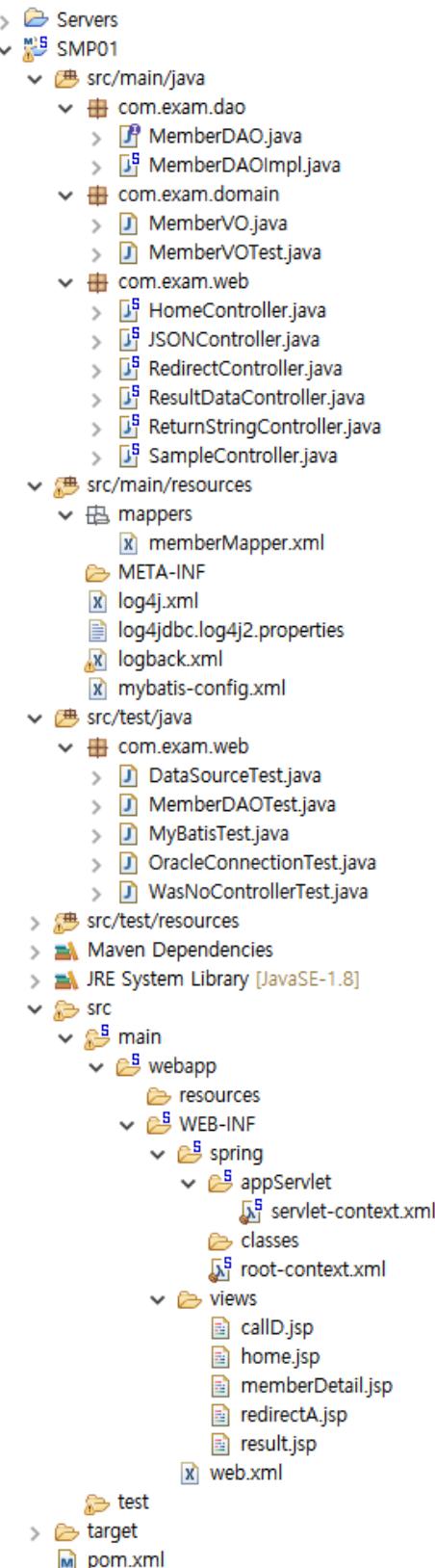
<terminated> MemberDAOTest (1) [JUnit] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (2015-10-22 오전 2:15:06)
INFO : jdbc.connection - 1. Connection opened
INFO : jdbc.audit - 1. Connection.new Connection returned
INFO : jdbc.audit - 1. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 1. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 1. Connection.prepareStatement(select *
    from TBL_MEMBER
    where userid = ? and userpw = ?) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@24b6b8f6
INFO : jdbc.audit - 1. PreparedStatement.setString(1, "testuser1") returned
INFO : jdbc.audit - 1. PreparedStatement.setString(2, "test1234") returned
INFO : jdbc.sqlonly - select * from TBL_MEMBER where userid = 'testuser1' and userpw = 'test1234'

INFO : jdbc.sqltiming - select * from TBL_MEMBER where userid = 'testuser1' and userpw = 'test1234'
{executed in 16 msec}
INFO : jdbc.audit - 1. PreparedStatement.execute() returned true
INFO : jdbc.resultset - 1. ResultSet.new ResultSet returned
INFO : jdbc.audit - 1. PreparedStatement.getResultSet() returned net.sf.log4jdbc.sql.jdbcapi.ResultSetSpy@37b70343
INFO : jdbc.resultset - 1. ResultSet.getMetaData() returned oracle.jdbc.driver.OracleResultSetMetaData@1080b026
INFO : jdbc.resultset - 1. ResultSet.getType() returned 1003
INFO : jdbc.resultset - 1. ResultSet.next() returned true
INFO : jdbc.resultset - 1. ResultSet.getString(USERID) returned testuser1
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getString(USERPW) returned test1234
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getString(USERNAME) returned 풍길동
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getString(EMAIL) returned testuser@abc1234.com
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getTimestamp(REGDATE) returned 2015-10-22 13:23:00.0
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 1. ResultSet.getTimestamp(UPDATEDAY) returned 2015-10-22 13:23:00.0
INFO : jdbc.resultset - 1. ResultSet.wasNull() returned false
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|
|userid |userpw |username |email |regdate |updateday |
|-----|-----|-----|-----|-----|
|testuser1 |test1234 |풍길동 |testuser@abc1234.com |2015-10-22 13:23:00.0 |2015-10-22 13:23:00.0 |
|-----|-----|-----|-----|-----|
INFO : jdbc.resultset - 1. ResultSet.next() returned false
INFO : jdbc.resultset - 1. ResultSet.close() returned void
<

INFO : jdbc.connection - 2. Connection opened
INFO : jdbc.audit - 2. Connection.new Connection returned
INFO : jdbc.audit - 2. Connection.getAutoCommit() returned true
INFO : jdbc.audit - 2. PreparedStatement.new PreparedStatement returned
INFO : jdbc.audit - 2. Connection.prepareStatement(select *
    from TBL_MEMBER
    where userid = ?) returned net.sf.log4jdbc.sql.jdbcapi.PreparedStatementSpy@749c877b
INFO : jdbc.audit - 2. PreparedStatement.setString(1, "testuser2") returned
INFO : jdbc.sqlonly - select * from TBL_MEMBER where userid = 'testuser2'

INFO : jdbc.sqltiming - select * from TBL_MEMBER where userid = 'testuser2'
{executed in 0 msec}
INFO : jdbc.audit - 2. PreparedStatement.execute() returned true
INFO : jdbc.resultset - 2. ResultSet.new ResultSet returned
INFO : jdbc.audit - 2. PreparedStatement.getResultSet() returned net.sf.log4jdbc.sql.jdbcapi.ResultSetSpy@efde75f
INFO : jdbc.resultset - 2. ResultSet.getMetaData() returned oracle.jdbc.driver.OracleResultSetMetaData@16ecee1
INFO : jdbc.resultset - 2. ResultSet.getType() returned 1003
INFO : jdbc.resultset - 2. ResultSet.next() returned true
INFO : jdbc.resultset - 2. ResultSet.getString(USERID) returned testuser2
INFO : jdbc.resultset - 2. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 2. ResultSet.getString(USERPW) returned test1234
INFO : jdbc.resultset - 2. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 2. ResultSet.getString(USERNAME) returned 이순신
INFO : jdbc.resultset - 2. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 2. ResultSet.getString(EMAIL) returned testuser2@abc1234.com
INFO : jdbc.resultset - 2. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 2. ResultSet.getTimestamp(REGDATE) returned 2015-10-22 13:42:43.0
INFO : jdbc.resultset - 2. ResultSet.wasNull() returned false
INFO : jdbc.resultset - 2. ResultSet.getTimestamp(UPDATEDAY) returned 2015-10-22 13:42:43.0
INFO : jdbc.resultset - 2. ResultSet.wasNull() returned false
INFO : jdbc.resultsettable -
|-----|-----|-----|-----|-----|
|userid |userpw |username |email |regdate |updateday |
|-----|-----|-----|-----|-----|
|testuser2 |test1234 |이순신 |testuser2@abc1234.com |2015-10-22 13:42:43.0 |2015-10-22 13:42:43.0 |
|-----|-----|-----|-----|-----|
INFO : jdbc.resultset - 2. ResultSet.next() returned false
INFO : jdbc.resultset - 2. ResultSet.close() returned void
INFO : jdbc.audit - 2. PreparedStatement.getConnection() returned net.sf.log4jdbc.sql.jdbcapi.ConnectionSpy@3b220bcb
<
```

프로젝트 전체 구조



Part 2. Interceptor를 활용한 로그인 처리

세션 트래킹이라 불리는 로그인 처리는 웹 페이지에서 필수적으로 적용된다.

- HttpSession을 이용한 로그인
- Cookie와 HttpSession을 활용한 자동 로그인

1. Spring MVC의 Interceptor

스프링을 이용해서 사용자의 단순 로그인을 처리하는 가장 간단한 방법은 Servlet의 Filter와 유사한 Interceptor를 활용하는 방법이다.

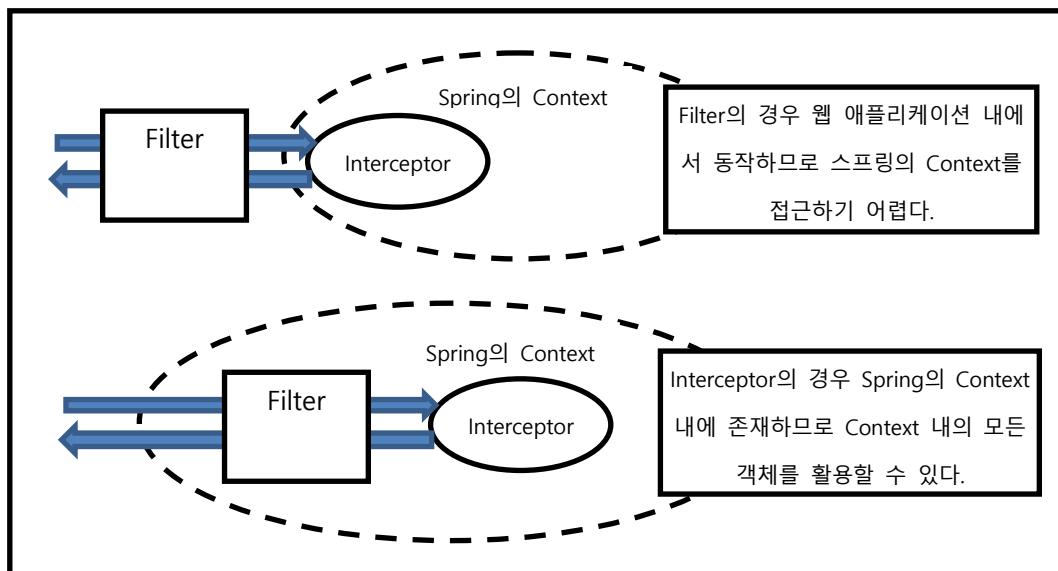
Spring MVC에서 Interceptor는 웹 애플리케이션 내에서 특정한 URI 호출을 가로채는 역할을 한다. 이를 활용하면 로그인한 사용자만이 사용할 수 있는 기능을 제어할 수 있기 때문에 기존 컨트롤러의 로직을 수정하지 않고도 사전이나 사후에 제어가 가능하다.

1.1 Filter와 Interceptor의 공통점과 차이점

Servlet 기술의 Filter와 Spring MVC의 HandlerInterceptor는 특정 URI에 접근할 때 제어하는 용도로 사용 된다는 공통점을 가지고 있다. 설정 역시 web.xml에 사용하는 필터의 설정과 유사한 부분이 많다.

차이점은 실행 시점에 속하는 영역(Context)에 있다. Filter는 동일한 웹 애플리케이션의 영역 내에서 필요한 자원들을 활용한다.

Interceptor의 경우 스프링에서 관리되기 때문에 스프링 내의 모든 객체에 접근이 가능하다는 차이가 있다.



HandlerInterceptor의 경우 스프링의 빈으로 등록된 컨트롤러나 서비스 객체들을 주입 받아서 사용할 수 있기 때문에 기존의 구조를 그대로 활용하면서 추가적인 작업이 가능하다.

1.2 Spring AOP 기능과 HandlerInterceptor의 차이

일반적인 경우라면 컨트롤러를 이용할 때는 AOP의 BeforeAdvice 등을 활용하기보다는 HandlerInterceptor

인터페이스 혹은 HandlerInterceptorAdaptor 클래스를 활용하는 경우가 더 많다.

AOP의 Advice와 HandlerInterceptor의 가장 큰 차이는 파라미터의 차이이다. Advice의 경우 JoinPoint나 ProceedingJoinPoint 등을 활용해서 호출 대상이 되는 메소드의 파라미터 등을 처리하는 방식이다.

반면에 HandlerInterceptor는 Filter와 유사하게 HttpServletRequest, HttpServletResponse를 파라미터로 받는 구조이다.

일반적인 경우라면 Controller에서는 DTO(Data Transfer Object)나 VO(Value Object) 타입을 주로 파라미터로 활용하고 Servlet API에 해당하는 HttpServletRequest나 HttpServletResponse를 활용하는 경우는 많지 않다.

HandlerInterceptor는 기존의 컨트롤러에서는 순수하게 필요한 파라미터와 결과 데이터를 만들어 내고 Interceptor를 이용해서 웹과 관련된 처리를 도와주는 역할을 한다.

HandlerInterceptor의 메소드

- preHandle(request, response, handler) – 지정된 컨트롤러의 동작 이전에 가로채는 역할로 사용한다.
- postHandle(request, response, handler, modelAndView) – 지정된 컨트롤러의 동작 이후에 처리한다. Spring MVC의 Front Controller인 DispatcherServlet이 화면을 처리하기 전에 동작한다.
- afterCompletion(request, response, handler, exception) – DispatcherServlet의 화면 처리가 완료된 상태에서 처리한다. 대부분은 preHandle()을 이용해서 로그인에 대한 처리를 한다.

1.2.1 HandlerInterceptorAdapter 클래스

HandlerInterceptor는 인터페이스로 정의되어 있지만 HandlerInterceptorAdapter는 인터페이스를 구현한 추상 클래스로 설계되어 있다. 일반적으로 디자인 패턴에서 Adapter라는 용어가 붙으면 특정 인터페이스를 미리 구현해서 사용하기 쉽게 한다. HandlerInterceptorAdapter 역시 HandlerInterceptor를 쉽게 사용하기 위해서 인터페이스의 메소드를 미리 구현한 클래스이다.

1.3 로그인 예제 프로젝트 생성

Spring Project -> Spring MVC Project

프로젝트명 : LoginExam, 패키지명 : com.inter.controller

SampleInterceptor.java

```
package com.inter.interceptor;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
public class SampleInterceptor extends HandlerInterceptorAdapter {
}
```

1.3.1 servlet-context.xml의 Interceptor 설정

SampleInterceptor를 스프링에서 인식하려면 servlet-context.xml에 설정을 추가해야 한다. Interceptor의 경우 URI 설정이 추가되므로 설정 방식이 다르다.

servlet-context.xml에 추가

```
<!-- Interceptor 설정 -->
<beans:bean id="sampleInterceptor" class="com.inter.interceptor.SampleInterceptor"></beans:bean>
    <interceptors>
        <interceptor>
            <mapping path="/interA" />
            <mapping path="/interB" />
            <beans:ref bean="sampleInterceptor" />
        </interceptor>
    </interceptors>
```

Interceptor의 설정은 <interceptors>태그를 이용해서 설정한다. 이 태그를 이용하기 위해서는 XML 네임스페이스에 spring mvc 관련 설정이 추가되어 있어야 한다.

각 Interceptor의 <mapping>에 원하는 URI를 지정한다. 이 설정은 web.xml의 필터나 Servlet의 설정과 동일하므로 필요한 경로를 직접 지정하거나 '**', '*'와 같은 패턴을 적용해 줄 수 있다.

위 설정은 현재 프로젝트의 '/interA' 경로와 '/interB' 경로를 호출할 때 SampleInterceptor가 작동한다.

1.3.2 HomeController의 설정

간단한 테스트를 위해 HomeController에 간단한 메소드를 작성하고 @RequestMapping을 '/interA' 경로와 '/interB'로 작성한다.

HomeController.java에 메소드 추가

```
@RequestMapping(value = "/interA", method = RequestMethod.GET)
public String interA(Locale locale, Model model) {
    System.out.println("interA 호출");

    return "home";
}

@RequestMapping(value = "/interB", method = RequestMethod.GET)
public String interB(Locale locale, Model model) {
    System.out.println("interB 호출");
    model.addAttribute("result", "inter B RESULT");

    return "home";
}
```

1.3.3 SampleInterceptor 처리

SampleInterceptor에서 preHandle()와 postHandler()를 이용해서 출력하는 코드를 작성한다.

SampleInterceptor.java 수정

```

package com.inter.interceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class SampleInterceptor extends HandlerInterceptorAdapter {

    @Override
    public void postHandle(HttpServletRequest request,
                           HttpServletResponse response, Object handler,
                           ModelAndView modelAndView) throws Exception {
        System.out.println("post handle 호출");
    }

    @Override
    public boolean preHandle(HttpServletRequest request,
                           HttpServletResponse response, Object handler) throws Exception {
        System.out.println("pre handle 호출");

        return true;
    }
}

```

preHandle()의 리턴 타입은 boolean이다. 이는 Interceptor나 대상 컨트롤러를 호출하도록 할 것인지를 결정한다.

The screenshot shows the Eclipse IDE interface. At the top, there is a 'Tomcat v8.0 Server at localhost' window titled 'Web Modules'. It lists a single module: 'Path: /' with 'Document Base: interceptorTest', 'Module: InterceptorTest', and 'Auto Reload: Enabled'. Below this are buttons for 'Add Web Module...', 'Edit...', and 'Remove'. At the bottom of this window are tabs for 'Overview' and 'Modules'.

Below the server configuration are two browser windows. The left browser window has a title bar 'localhost/interA' and displays the content 'Hello world!'. Below it, a message says 'The time on the server is .'. The right browser window has a title bar 'localhost/interB' and also displays 'Hello world!' with the same message below it.

```

Console [Markers] Progress
Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (2015.10.26 오후 4:38:24)
INFO : com.inter.controller.HomeController - Welcome home! The client locale is ko_KR.
pre handle 호출
interA 호출
post handle 호출
pre handle 호출
interB 호출
post handle 호출

```

실행되는 로그를 확인하면 preHandle()가 먼저 호출되고 메소드가 호출된 후 postHandle()가 호출되는 것을 확인할 수 있다.

1.4 Interceptor의 request, response 활용

1.4.1 preHandle()의 Object 파라미터

preHandle()의 경우 세 개의 파라미터는 HttpServletRequest, HttpServletResponse, Object로 구성된다. 마지막 Object는 실행하려는 메소드 자체를 의미하며 이를 활용하면 현재 실행되는 컨트롤러를 파악하거나 추가적인 메소드를 실행하는 등의 작업이 가능하다.

SampleInterceptor의 preHandle()을 이용해서 현재 실행되는 컨트롤러와 메소드의 정보를 파악한다.

SampleInterceptor.java 의 preHandle() 수정

```

@Override
public boolean preHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler) throws Exception {

    System.out.println("pre handle 호출");

    HandlerMethod method = (HandlerMethod) handler;
    Method methodObj = method.getMethod();

    System.out.println("Bean: " + method.getBean());
    System.out.println("Method: " + methodObj);

    return true;
}

```

<http://localhost/interB>

```

Console [Markers] Progress
Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (2015.10.26 오후 5:07:02)
INFO : com.inter.controller.HomeController - Welcome home! The client locale is ko_KR.
pre handle 호출
Bean: com.inter.controller.HomeController@6821f652
Method: public java.lang.String com.inter.controller.HomeController.interB(java.util.Locale,org.springframework.ui.Model)
interB호출
post handle 호출

```

1.4.2 postHandle()을 이용한 추가작업

postHandle()의 경우 컨트롤러 aphhem의 실행이 끝나고 화면 처리는 안 된 상태이므로 실행 이후에 추가 작업이 가능하다.

특정한 메소드의 실행 결과를 HttpSession 객체에 같이 담아야 하는 경우에 컨트롤러에서는 Model 객체에 결과 데이터를 저장하고 Interceptor의 postHandle()에서 이를 이용해 HttpSession에 결과를 담는다면 컨트롤러에서 HttpSession을 처리할 필요가 없어진다.

컨트롤러에서 'result'라는 변수가 저장되었다면 HttpSession 객체이 이를 보관한다.

SampleInterceptor.java 의 postHandle() 수정

```
@Override  
public void postHandle(HttpServletRequest request, HttpServletResponse response, Object  
handler, ModelAndView modelAndView) throws Exception {  
  
    System.out.println("post handle 호출");  
  
    Object result = modelAndView.getModel().get("result");  
  
    if (result != null) {  
        System.out.println("result가 존재한다.");  
        request.getSession().setAttribute("result", result);  
        response.sendRedirect("/interA");  
    }  
}
```

컨트롤러에서 '/interB'를 호출하면 'result'라는 이름으로 하나의 문자열이 보관된다. 이후에 postHandle()에서 'result'라는 변수가 ModelAndView에 존재하면 이를 추출해서 HttpSession에 추가한다.

HomeController의 '/interB'를 먼저 호출하면 HttpSession에 'result'라는 이름을 보관한 후 '/interA'로 sendRedirect를 수행한다.

/WEB-INF/views/home.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  
<html>  
<head>  
    <title>Home</title>  
</head>  
<body>  
<h1>  
    Hello world!  
</h1>  
<h2>${result}</h2>
```

```
</body>  
</html>
```

<%@ page session="false" %> 제거해야 HttpSession을 이용할 수 있다.

http://localhost/interB로 호출하면 http://localhost/interA로 이동하면서 'result'의 문자열이 출력된다.

A screenshot of the Eclipse IDE's Console view. The log output shows the startup of the Tomcat server and the execution of interceptors. It includes messages like "Hello world!", "inter B RESULT", and logs for both "interB" and "interA" interceptors, including their handle methods and the presence of a "result" attribute.

위의 예제는 로그인 처리에 유용하게 사용할 수 있다. 컨트롤러에서 로그인 처리 후에 결과를 반환하고 Interceptor를 이용해서 HttpSession에 로그인에 필요한 객체를 보관하는 형태로 작성하면 컨트롤러에서 직접 HttpSession 등의 API를 사용하지 않아도 된다.

2. HttpSession을 이용한 로그인 처리

웹에서 로그인의 가장 기본적인 방식은 HttpSession 객체를 이용해서 사용자의 정보를 보관하고 필요한 경우 사용하거나 수정한다.

HttpSession의 동작은 session cookie를 통해서 이루어지는데 서버는 필요한 경우 접속한 브라우저에게 고유한 세션 쿠키를 전달하고 매번 브라우저에게 서버를 호출할 때 세션 쿠키를 같이 가지고 있기 때문에 이를 마치 열쇠처럼 사용해서 필요한 데이터를 보관한다.

세션 쿠키가 열쇠(key)라면 HttpSession은 열쇠가 필요한 잠금 장치가 되어있는 장소이다. 이러한 장소들이 모여있는 공간을 '세션 저장소(Session Repository)'라고 한다. 너무나 많은 세션이 존재하면 서버의 성능에 영향을 미치기 때문에 서버에는 일정 시간 이상 사용되지 않는 것을 정리하는 기능이 있다.(web.xml에는 HttpSession의 timeout을 지정할 수 있다.)

세션을 이용하는 방식의 핵심은 HttpSession을 이용해서 원하는 객체를 보관할 수 있다는 점이다. 사용자는 항상 열쇠에 해당하는 세션 쿠키를 가지고 접근하고, 서버의 내부에 필요한 객체를 보관하기 때문에 안전하다는 장점이 있다.

세션에 보관된 객체는 JSP의 EL을 이용해서 자동으로 추적하는 방식을 사용한다.

예로 '\${name}'은 page -> request -> session -> application의 순서대로 원하는 데이터를 검색한다. 이와 같은 방식으로 동작하기 때문에 JSP를 개발하는 개발자는 사용하는 변수가 request에 존재하는지, session에 존재하는 것인지에 대해 고민하지 않아도 된다.

2.1 로그인 준비 작업

로그인을 하는 경우 사용자의 정보는 DTO(Data Transfer Object) 또는 VO(Value Object)를 이용해서 보관되고, 세션에 보관하게 된다.

2.1.1 테이블 생성 및 객체 처리

```
CREATE TABLE tbl_user(
    userid VARCHAR2(50) NOT NULL,
    userpw VARCHAR2(50) NOT NULL,
    username VARCHAR2(100) NOT NULL,
    userpoint NUMBER default 0,
    PRIMARY KEY(userid)
);

insert into TBL_USER(userid, userpw, username) values('user01', '1234', '홍길동');
insert into TBL_USER(userid, userpw, username) values('user02', '1234', '이순신');
insert into TBL_USER(userid, userpw, username) values('user03', '1234', '강감찬');
insert into TBL_USER(userid, userpw, username) values('user04', '1234', '유성룡');
insert into TBL_USER(userid, userpw, username) values('user05', '1234', '정도전');
commit;
```

UserVO.java

```
package com.login.domain;  
public class UserVO {  
    private String userid;  
    private String userpw;  
    private String username;  
    private int userpoint;  
  
    public UserVO() {  
        super();  
    }  
    public UserVO(String userid, String userpw, String username, int userpoint) {  
        super();  
        this.userid = userid;  
        this.userpw = userpw;  
        this.username = username;  
        this.userpoint = userpoint;  
    }  
    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 int getUserpoint() {  
        return userpoint;  
    }
```

```

    }

    public void setUserpoint(int userpoint) {
        this.userpoint = userpoint;
    }

    @Override
    public String toString() {
        return "UserVO [userid=" + userid + ", userpw=" + userpw + ", username=" +
username + ", userpoint=" + userpoint + "]";
    }
}

```

화면에 전달되는 데이터를 수집하는 용도

LoginDTO.java

```

package com.login.dto;

public class LoginDTO {
    private String userid;
    private String userpw;
    private boolean useCookie;

    public LoginDTO() {
        super();
    }

    public LoginDTO(String userid, String userpw, boolean useCookie) {
        super();
        this.userid = userid;
        this.userpw = userpw;
        this.useCookie = useCookie;
    }

    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 boolean isUseCookie() {
    return useCookie;
}
public void setUseCookie(boolean useCookie) {
    this.useCookie = useCookie;
}

@Override
public String toString() {
    return "LoginDTO [userid=" + userid + ", userpw=" + userpw + ", useCookie=" +
useCookie + "]";
}
}

```

※ VO(Value Object)와 DTO(Data Transfer Object)

일반적으로 컨트롤러에 전달되는 데이터를 수집하는 용도로 VO를 사용하는 경우도 있고, DTO를 사용하는 경우도 있다.

VO와 DTO의 용도는 데이터의 수집과 전달에 사용할 수 있다는 공통점이 있다. 모두 파라미터나 리턴 타입으로 사용하는 것이 가능하다. VO는 데이터베이스의 테이블의 구조를 이용해서 작성한다. DTO는 화면에 전달되는 데이터를 수집하는 용도로 사용하는 경우가 많다.

Spring MVC를 이용하는 경우 DTO는 검증을 위한 처리가 들어간다. 스프링은 Controller에 전달되는 데이터에 대해 검증하는 기능을 추가할 수 있는데 별도의 DTO를 구성하여 사용한다.

2.1.2 UserDAO의 생성 및 SQL 처리

로그인할 때 사용자의 아이디와 패스워드를 이용해서 사용자의 정보를 조회하는 SQL문을 처리한다.

UserDAO.java

```

package com.login.dao;
import com.login.domain.UserVO;
import com.login.dto.LoginDTO;

public interface UserDAO {
    public UserVO login(LoginDTO dto) throws Exception;
}

```

/main/resources/mappers/userMapper.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.login.mapper.UserMapper">
    <select id="login" resultType="UserVO">
        select userid, userpw, username from tbl_user where userid = #{userid} and userpw = #{userpw}
    </select>
</mapper>
```

pom.xml 에 추가

```
<!-- oracle 11g -->
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0</version>
</dependency>

<!-- mybatis -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.2.8</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.2.2</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>

<!-- mybatis의 로그 log4jdbc-log4j2 -->
<dependency>
    <groupId>org.bgee.log4jdbc-log4j2</groupId>
    <artifactId>log4jdbc-log4j2-jdbc4</artifactId>
    <version>1.16</version>
</dependency>

<!-- Servlet -->
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>

```

UserDAOImpl.java

```

package com.login.dao;
import javax.inject.Inject;
import org.apache.ibatis.session.SqlSession;
import org.springframework.stereotype.Repository;
import com.login.domain.UserVO;
import com.login.dto.LoginDTO;

@Repository
public class UserDAOImpl implements UserDAO {

    @Inject
    private SqlSession session;

    private static String namespace = "com.login.mapper.UserMapper";

    @Override
    public UserVO login(LoginDTO dto) throws Exception {
        // TODO Auto-generated method stub

        return session.selectOne(namespace + ".login", dto);
    }
}

```

```
    }  
}
```

2.1.3 UserService와 UserServiceImpl

UserService.java

```
package com.login.service;  
import com.login.domain.UserVO;  
import com.login.dto.LoginDTO;  
  
public interface UserService {  
    public UserVO login(LoginDTO dto) throws Exception;  
}
```

UserServiceImpl.java

```
package com.login.service;  
import javax.inject.Inject;  
import org.springframework.stereotype.Service;  
import com.login.dao.UserDAO;  
import com.login.domain.UserVO;  
import com.login.dto.LoginDTO;  
  
@Service  
public class UserServiceImpl implements UserService {  
  
    @Inject  
    private UserDAO dao;  
  
    @Override  
    public UserVO login(LoginDTO dto) throws Exception {  
        // TODO Auto-generated method stub  
        return dao.login(dto);  
    }  
}
```

2.2 컨트롤러의 처리

서비스 계층까지의 구현이 완료되면 컨트롤러와 인터셉터를 적용하는 작업을 한다. 이때 중요한 결정은 '컨트롤러에서 HttpSession 객체를 처리할 것인가' 또는 '인터셉터에서 HttpSession 객체를 처리할 것인가'이다.

Spring MVC는 컨트롤러에서 필요한 모든 자원을 파라미터에서 수집해서 처리하기 때문에 HttpSession과 HttpServletRequest와 같은 자원들 역시 파라미터로 처리해도 문제없다.
인터셉터를 이용해서 HttpSession을 처리한다.

2.2.1 UserController의 작성

스프링의 인터셉터는 웹 관련 API를 처리하는 용도로 사용되기 때문에 가능하면 컨트롤러 역시 기본 방식과 동일하게 작성한다.

UserController.java

```
package com.login.controller;

import javax.inject.Inject;

import javax.servlet.http.HttpSession;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.ModelAttribute;

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

import org.springframework.web.bind.annotation.RequestMethod;

import com.login.domain.UserVO;

import com.login.dto.LoginDTO;

import com.login.service.UserService;

@Controller

@RequestMapping("/user")

public class UserController {

    @Inject

    private UserService service;

    @RequestMapping(value = "/login", method = RequestMethod.GET)

    public void loginGET(@ModelAttribute("dto") LoginDTO dto) {

    }

    @RequestMapping(value = "/loginPost", method = RequestMethod.POST)

    public void loginPOST(LoginDTO dto, HttpSession session, Model model) throws Exception {

        UserVO vo = service.login(dto);

        if (vo == null) {

            return;
        }
    }
}
```

```

        }
        model.addAttribute("userVO", vo);
    }
}

```

POST 방식으로 파라미터를 이용해서 Model에 UserVO 객체를 추가한다.

기본경로는 '/user'이고 로그인 화면은 '/user/login' 결과 처리는 '/user/loginPost'로 한다. 로그인 처리가 이루어지는 LoginPost()에서 Model 객체에 사용자가 존재하는 경우 'userVO'라는 이름으로 저장된다.

2.3 LoginInterceptor의 작성과 설정

UserController에서 HttpSession과 관련된 아무런 작업도 처리된 적이 없기 때문에 HttpSession에 관련된 모든 설정은 인터셉터에서 처리된다.

LoginInterceptor.java

```

package com.login.interceptor;

import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.ui.ModelMap;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class LoginInterceptor extends HandlerInterceptorAdapter {

    private static final String LOGIN = "login";
    private static final Logger logger = LoggerFactory.getLogger(LoginInterceptor.class);

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse response, Object
handler, ModelAndView modelAndView) throws Exception {

        HttpSession session = request.getSession();

        ModelMap modelMap = modelAndView.getModelMap();
        Object userVO = modelMap.get("userVO");

        if (userVO != null) {

```

```

        logger.info("new login success");
        session.setAttribute(LOGIN, userVO);
        response.sendRedirect("/");
    }

}

@Override
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object
handler) throws Exception {

    HttpSession session = request.getSession();

    if (session.getAttribute(LOGIN) != null) {
        logger.info("clear login data before");
        session.removeAttribute(LOGIN);
    }
    return true;
}
}

```

LoginInterceptor는 '/loginPost'로 접근하도록 설정한다. preHandle()에서는 기존 HttpSession에 남아있는 정보가 있는 경우에 삭제한다.

postHandle()에서는 UserController에서 'userVO'라는 이름으로 객체에 담아둔 상태이므로, 이 상태를 체크해서 HttpSession에 저장한다.

2.3.1 LoginInterceptor의 설정

LoginInterceptordml 설정은 '/LoginPost'의 동작으로 이루어 져야하므로 servlet-context.xml의 설정은 다음과 같다.

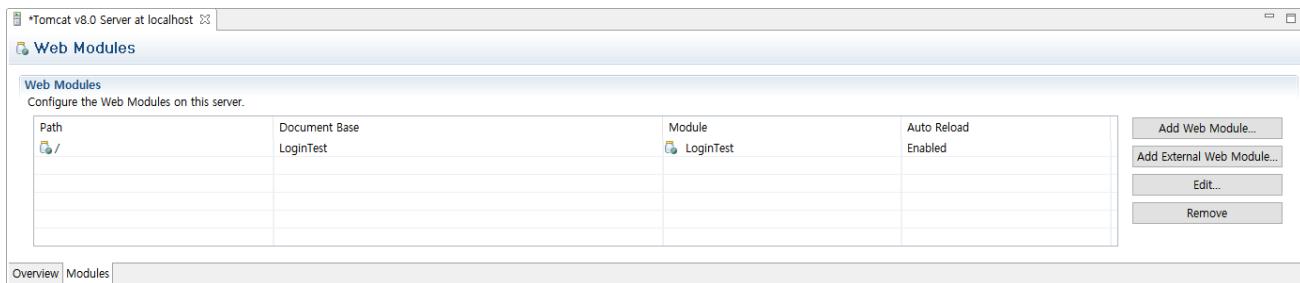
servlet-context.xml에 추가

```

<context:component-scan base-package="com.login.controller" />
<beans:bean id="loginInterceptor" class="com.login.interceptor.LoginInterceptor"></beans:bean>
<interceptors>
    <interceptor>
        <mapping path="/user/loginPost" />
        <beans:ref bean="loginInterceptor" />
    </interceptor>
</interceptors>

```

인터셉터의 설정에는 프로젝트의 경로를 포함하지 않는다. '/'를 경로로 설정한다.



2.3.2 그 밖의 설정

HomeController.java 수정

```
package com.login.controller;

import java.util.Locale;
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.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("로그인 {}", locale);
        model.addAttribute("loginSuccess", "Login Success" );
        return "home";
    }
}
```

/main/resources/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>
<typeAliases>
<package name="com.login.domain"/>
</typeAliases>
```

```
</configuration>
```

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:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jdbc="http://www.springframework.org/schema/jdbc"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc-4.1.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-4.1.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.1.xsd">

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

    <!-- Oracle과 연결을 담당하는 DataSource 설정 -->
    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl" />
        <property name="username" value="scott" />
        <property name="password" value="tiger" />
    </bean>

    <!-- DataBase와의 연결과 SQL의 실행을 위한 SqlSessionFactory 객체 설정 -->
    <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation" value="classpath:/mybatis-config.xml" />
        <property name="mapperLocations" value="classpath:mappers/**/*Mapper.xml" />
    </bean>
```

```

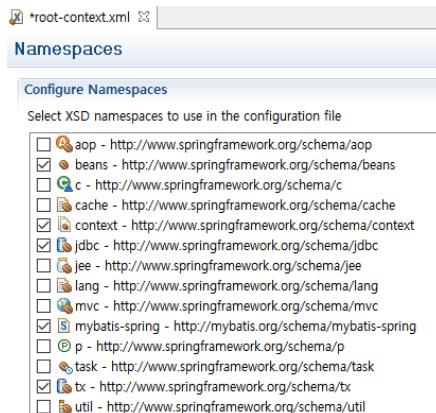
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"></property>
</bean>

<tx:annotation-driven />

<!-- DAO 인터페이스 구현을 위한 SqlSessionTemplate 설정 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate"
      destroy-method="clearCache">
    <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory"></constructor-arg>
</bean>

<!-- Spring의 Bean으로 등록 -->
<context:component-scan base-package="com.login.dao"></context:component-scan>
<context:component-scan base-package="com.login.service"></context:component-scan>
</beans>

```



2.3.3 로그인 화면 처리

login.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>로그인</title>
</head>

```

```

<body>
    <div>
        <p>로그인 SIGN IN</p>
        <form action="/user/loginPost" method="post">
            <p><input type="text" name="userid" placeholder="USER ID" /> </p>
            <p><input type="password" name="userpw" placeholder="Password" /> </p>
        /></p>
        <p><label>
            <input type="checkbox" name="useCookie"> 아이디 기억
        </label></p>
        <p><button type="submit">로그인 </button></p>
    </form>
    </div>
</body>

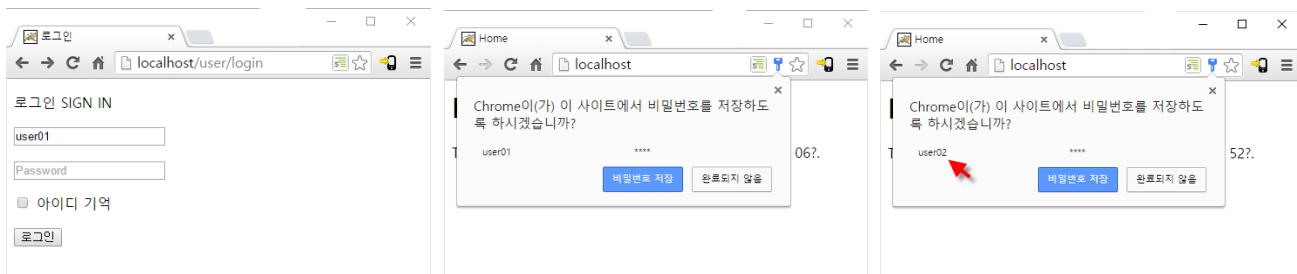
```

loginPost.jsp

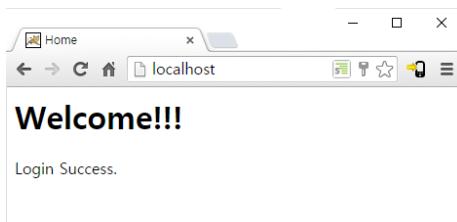
```

<%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>SIGN IN</title>
</head>
<body>
    <div>
        <p>아이디나 패스워가 틀림</p>
        <script type="text/javascript">
            self.location = "/user/login";
        </script>
    </div>
</body>
</html>

```



크롬에서 아이디와 패스워드를 다르게 해서 로그인 해본다.



4. 자동 로그인과 쿠키

자동 로그인은 브라우저가 서버에 접속할 때 특정한 쿠키를 같이 전송하고, 이를 이용해서 로그인을 처리하는 방식이다. HttpSession에서 사용되는 세션과 달리 개발자가 만드는 쿠키의 경우는 '만료기한'을 지정할 수 있기 때문에 보관이 가능하다.

일반적으로 웹에서 로그인은 세션을 사용한다. 가장 큰 이유는 보안 때문이다. 세션을 이용하는 경우 로그인한 사용자의 정보는 서버의 내부에서만 사용되기 때문에 보안상 유리하다.

반면 쿠키는 매번 브라우저와 서버 사이에서 주고받는 방식으로 동작하기 때문에 쿠키에 특정한 값이 기록된 경우 보안에 취약하다는 단점이 있다. 세션과 달리 쿠키에는 길이의 제한, 문자열만 보관할 수 있기 때문에 과거에는 사용을 자제하였다.

쿠키가 자동 로그인에 활용된 계기는 모바일에서 매번 로그인하기가 번거롭다는 문제의 대안으로 사용되면서부터이다. 그리고 로그인한 정보를 오랜 시간 유지할 수 있는 쿠키를 선호한다.

4.1 쿠키를 이용하는 자동 로그인

자동 로그인을 처리하기 전에 사용자가 로그인한 후 쿠키를 만들어서 브라우저로 전송하고, 다시 서버에 접속할 때 쿠키가 전달되는지를 확인한다.

4.1.1 LoginInterceptor에서 쿠키 생성

LoginInterceptor의 경우 postHandle()을 이용해서 HttpSession에 UserVO 타입의 객체를 보관한다. 이를 수정해서 중간에 쿠키를 생성하고 HttpServletResponse에 같이 담아서 전송하도록 코드를 수정한다.

LoginInterceptor.java 수정

```

@Override
public void postHandle(HttpServletRequest request, HttpServletResponse response, Object
handler, ModelAndView modelAndView) throws Exception {

    HttpSession session = request.getSession();
}

```

```

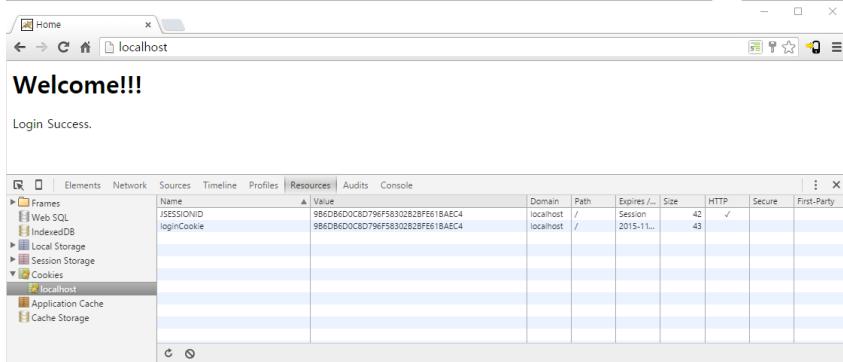
ModelMap modelMap = modelAndView.getModelMap();
Object userVO = modelMap.get("userVO");

if (userVO != null) {
    logger.info("로그인 성공");
    session.setAttribute(LOGIN, userVO);
    if (request.getParameter("useCookie") != null) {
        logger.info("자동 로그인");
        Cookie loginCookie = new Cookie("loginCookie", session.getId());
        loginCookie.setPath("/");
        loginCookie.setMaxAge(60 * 60 * 24 * 7);
        response.addCookie(loginCookie);
    }
    Object dest = session.getAttribute("dest");
    response.sendRedirect(dest != null ? (String) dest : "/");
}
}

```

서용자가 자동 로그인을 선택한 경우 쿠키를 생성하고 생성된 쿠키의 이름은 loginCookie로 저장한다. 생성된 loginCookie에 값(value)으로 현재 세션의 아이디 값을 보관한다. 세션 아이디는 쿠키의 값을 의미한다.

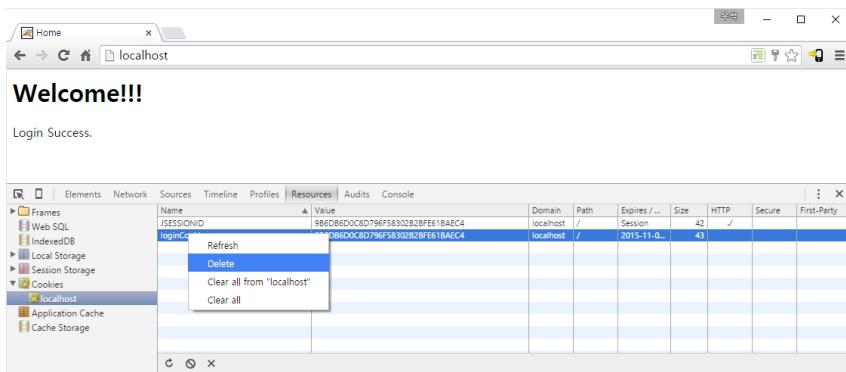
쿠키의 경우 브라우저를 종료하면 사라지지만, loginCookie의 경우 오랜 시간 보관하기 위해서 setMaxAge()를 이용한다. setMaxAge()는 초 단위의 시간 동안 유효하다. $60 * 60 * 24 * 7$ 계산은 일주일간 브라우저에 보관된다. 만들어진 쿠키는 반드시 HttpServletResponse에 담아서 전송한다.



개발자 도구에서 쿠키 정보를 확인하면 JSESSIONID는 Tomcat에서 생성한 세션 쿠키의 이름이고 loginCookie는 인터셉터에 의해서 만들어진 쿠키이다. 두 개의 value는 동일하다.

서버의 세션시간이 지나면 'JSESSIONID' 세션 쿠키는 존재하지 않지만 7일간 유효기간이 지정된 loginCookie는 값을 유지한다.

쿠키를 삭제하고 다시 서버에 접속하면 쿠키가 없는 상태로 전송된다.



4.1.3 자동 로그인 이해

자동 로그인 처리는 '세션 + 쿠키'를 이용한다. 보통은 HttpSession만을 이용하거나 쿠키만을 이용하지만, 이 두 가지를 섞어서 사용한다.

쿠키와 세션으로 발생하는 상황은 아래의 네 가지 경우가 있을 수 있다.

- HttpSession에 login 이름으로 보관된 객체가 없고 loginCookie가 없는 경우
로그인과 관련된 아무런 정보가 없으므로 사용자는 로그인이 필요하다.
- HttpSession에 login 이름으로 보관된 객체가 있고 loginCookie가 없는 경우
현재 사용자가 로그인한 상황이다.
- HttpSession에 login 이름으로 보관된 객체가 없고 loginCookie가 있는 경우
사용자가 이전에 로그인을 한 적이 있다.
- HttpSession에 login 이름으로 보관된 객체가 있고 loginCookie가 있는 경우
사용자가 현재 접속 중이다.

위 경우 중에 자동 로그인은 HttpSession에는 login 이름으로 보관된 객체가 없지만, loginCookie가 존재하는 경우이다. 이 경우 사용자는 이전에 로그인을 한 적이 있으므로 과거 로그인 시점에 기록된 정보를 이용해서 다시 HttpSession에 login 이름으로 UserVO 객체를 보관해 줘야 한다.

4.2 자동 로그인 구현

사용자가 loginCookie를 가지고 있다면, 그 값은 과거에 로그인한 시점의 세션 아이디이다.

즉, loginCookie에 있는 값을 이용해서 데이터베이스에서 UserVO의 정보를 읽어오고, 읽어온 UserVO 객체를 현재의 HttpSession에 보관하면 로그인이 된다는 것이다. 다음과 같은 순서로 이루어진다.

- 사용자가 로그인하면 데이터베이스에 현재 세션의 ID값과 유효기간(7일)을 기록한다.
- 사용자가 로그인하지 않은 상태에서 쿠키를 가지고 접속하면 쿠키의 내용을 추출한다.
- 쿠키의 내용으로 데이터베이스를 조회해서 유효기간이 맞는지 확인한다.
- 확인된 사용자는 세션에 로그인한 정보를 기록해서 자동으로 로그인이 되도록 한다.

4.2.1 데이터베이스 변경

```
ALTER TABLE tbl_user
ADD(sessionkey varchar2(50) default 'none' not null);
```

```
ALTER TABLE tbl_user  
ADD(sessionlimit date);
```

세션 아이디를 보관하는 sessionkey 칼럼과 유효시간을 기록하는 sessionlimit 칼럼 추가

4.2.2 코드 변경

4.2.2.1 UserDAO 변경

UserDAO에는 로그인한 사용자의 sessionKey와 sessionLimit를 업데이트하는 기능과 loginCookie에 기록된 값으로 사용자의 정보를 조회하는 기능을 추가한다.

UserDAO.java 수정

```
package com.login.dao;  
import java.util.Date;  
import com.login.domain.UserVO;  
import com.login.dto.LoginDTO;  
  
public interface UserDAO {  
    public UserVO login(LoginDTO dto) throws Exception;  
    public void keepLogin(String uid, String sessionId, Date next);  
    public UserVO checkUserWithSessionKey(String value);  
}
```

userMapper.xml에 추가

```
<update id="keepLogin">  
    update tbl_user set sessionKey = #{sessionId}, sessionLimit = #{next} where  
    userid = #{userid}  
</update>  
  
<select id="checkUserWithSessionKey" resultType="UserVO">  
    select * from tbl_user where sessionKey = #{value} and sessionlimit > now()  
</select>
```

UserDAOImpl.java 수정

```
@Override  
public void keepLogin(String userid, String sessionId, Date next) {  
    // TODO Auto-generated method stub  
    Map<String, Object> paramMap = new HashMap<String, Object>();  
    paramMap.put("userid", userid);  
    paramMap.put("sessionId", sessionId);  
    paramMap.put("next", next);
```

```

        session.update(namespace + ".keepLogin", paramMap);
    }

    @Override
    public UserVO checkUserWithSessionKey(String value) {
        // TODO Auto-generated method stub
        return session.selectOne(namespace + ".checkUserWithSessionKey", value);
    }
}

```

4.2.2.2 UserService의 변경

UserService에는 로그인 정보를 유지하는 keepLogin과 과거에 접속한 사용자인지를 확인하는 기능을 추가한다.

UserService.java 수정

```

package com.login.service;

import java.util.Date;
import com.login.domain.UserVO;
import com.login.dto.LoginDTO;

public interface UserService {
    public UserVO login(LoginDTO dto) throws Exception;
    public void keepLogin(String uid, String sessionId, Date next) throws Exception;
    public UserVO checkLoginBefore(String value);
}

```

UserServiceImpl.java 수정

```

@Override
public void keepLogin(String uid, String sessionId, Date next) throws Exception {
    // TODO Auto-generated method stub
    dao.keepLogin(uid, sessionId, next);
}

@Override
public UserVO checkLoginBefore(String value) {
    // TODO Auto-generated method stub
    return dao.checkUserWithSessionKey(value);
}

```

```
}
```

4.2.3 UserController 변경

UserController는 사용자가 '자동 로그인'을 선택한 경우 필요한 기능을 추가한다.

UserController.java 수정

```
@RequestMapping(value = "/loginPost", method = RequestMethod.POST)
public void loginPOST(LoginDTO dto, HttpSession session, Model model) throws Exception {

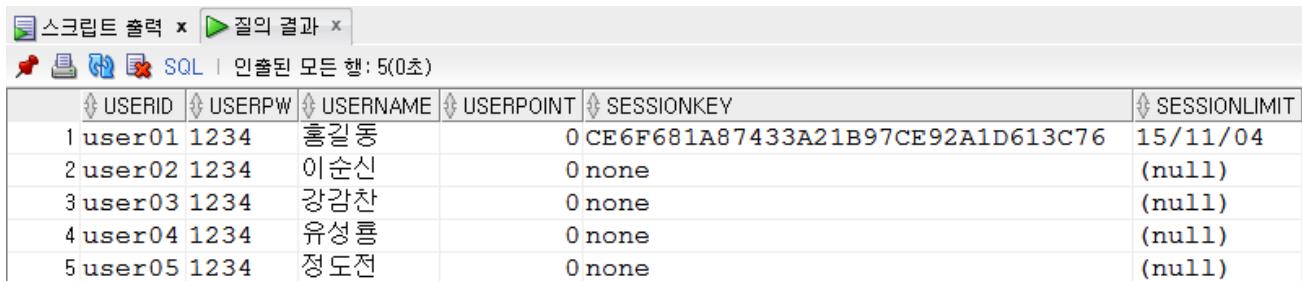
    UserVO vo = service.login(dto);

    if (vo == null) {
        return;
    }

    model.addAttribute("userVO", vo);
    if (dto.isUseCookie()) {
        int amount = 60 * 60 * 24 * 7;
        Date sessionLimit = new Date(System.currentTimeMillis() + (1000 * amount));
        service.keepLogin(vo.getUserid(), session.getId(), sessionLimit);
    }
}
```

4.2.4 자동 로그인 테스트

자동 로그인을 체크한 후 user01로 로그인하고 데이터베이스에서 확인한다.



USERID	USERPW	USERNAME	USERPOINT	SESSIONKEY	SESSIONLIMIT
1 user01	1234	홀길동	0	CE6F681A87433A21B97CE92A1D613C76	15/11/04
2 user02	1234	이순신	0	none	(null)
3 user03	1234	강감찬	0	none	(null)
4 user04	1234	유성룡	0	none	(null)
5 user05	1234	정도전	0	none	(null)

자동 로그인을 체크하지 않고 user02로 로그인하고 데이터베이스에서 확인한다.

4.3 로그아웃 처리

로그아웃 처리는 HttpSession인 경우 login과 같이 저장된 정보를 삭제하고 invalidate()를 주는 작업과 쿠키의 유효시간을 변경하는 방법이 있다.

자동 로그인에 데이터베이스를 이용하면 데이터베이스 갱신도 같이 진행되어야 한다.

UserController에서 로그아웃의 처리를 위해서 직접 파라미터로 HttpServletRequest등을 받는 방식이 가장 간단하고 별도의 인터셉터를 이용하는 방식도 있다.

UserController.java에 추가

```
@RequestMapping(value = "/logout", method = RequestMethod.GET)
public void logout(HttpServletRequest request, HttpServletResponse response, HttpSession session)
throws Exception {
    Object obj = session.getAttribute("login");
    if (obj != null) {
        UserVO vo = (UserVO) obj;

        session.removeAttribute("login");
        session.invalidate();

        Cookie loginCookie = WebUtils.getCookie(request, "loginCookie");
        if (loginCookie != null) {
            loginCookie.setPath("/");
            loginCookie.setMaxAge(0);
            response.addCookie(loginCookie);
            service.keepLogin(vo.getUserid(), session.getId(), new Date());
        }
    }
}
```

logout.jsp는 별다른 내용없이 '/'과 같은 경로로 이동하는 코드만 작성하면 된다.