

- ❑ Spring 프레임워크는 JavaEE 기반의 어플리케이션 개발을 쉽게 해주는 오픈소스 어플리케이션 프레임워크로, 간단한 자바 객체(POJO : Plain Old Java Object) 를 Spring의 경량(Lightweight) 컨테이너를 통해 생성 및 관리하는 빈(Been) 으로 처리해준다.



- 엔터프라이즈 어플리케이션을 쉽게 구성할 수 있도록 각종 빈(Been)의 생성 및 관리를 처리하는 경량(Lightweight) 컨테이너 제공
- Rod Johnson 에 의해 개발된 J2EE 어플리케이션 개발을 위한 오픈소스 어플리케이션 프레임워크

분류 및 성숙도 평가*	설명
라이선스	Apache 2.0
기능성 (Functionality)	✓✓✓✓ (중대형 규모의 기업의 기능적인 요구사항을 충족시킴)
커뮤니티 (Community)	*** (개발, 오류 보고, 수정 등의 활발한 커뮤니티 활동이 있음)
성숙도 (Maturity)	★★★★ (강력하며 높은 품질의 안정적이며 우수한 성능을 충족함)
적용성 (ER-Rating)	◆◆◆ (프레임워크가 성숙하여 기업 환경에 즉시 반영 가능함)
트렌드 (Trend)	↗ (평가 Criteria 전반적으로 발전하고 있으며, 중요도가 커지고 있음)

* Open Source Catalogue 2007, Optaros (Spring 2.0 기준)

□ EJB 명세와 현실의 괴리

- 원격 호출 기반의 EJB는 객체에 대해서 무거운(heavy weight)모델임
- 대부분의 개발자들은 스테이트리스 세션 빈과 비동기 방식이 필요한 경우에 한해서 메시지 드리븐 빈만을 사용함

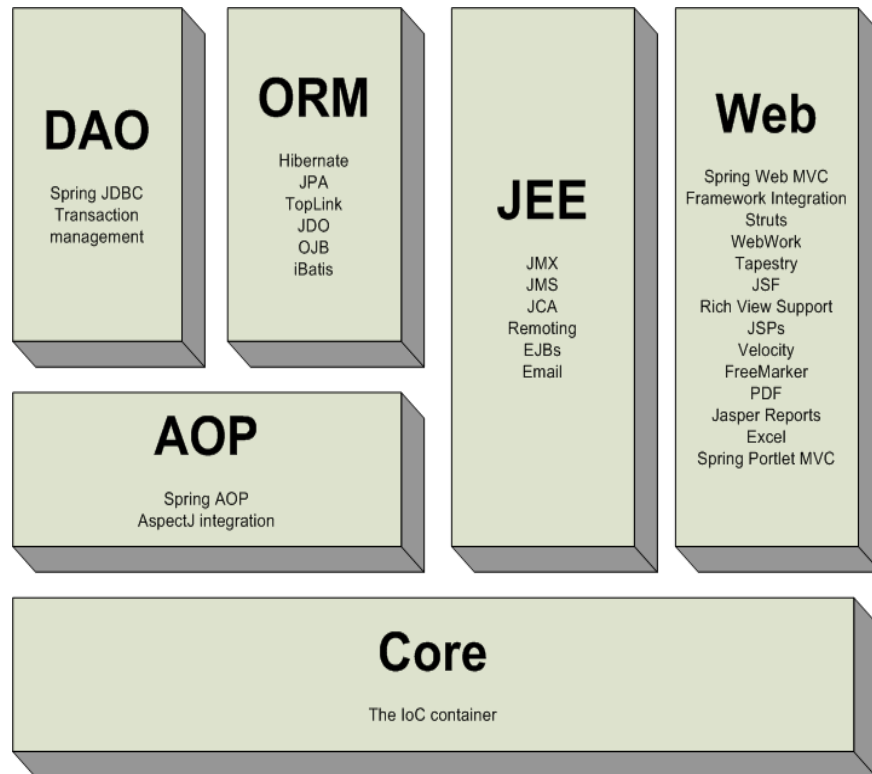
□ EJB 의 실패 부분

- 너무 복잡함
- 저장을 위한 엔티티 빈은 실패작임
- EJB의 이식성은 서블릿과 같은 다른 J2EE 기술보다 떨어짐
- 확장성을 보장한다는 EJB의 약속과 달리, 성능이 떨어지며 확장이 어려움

□ Spring Framework


- Spring이라는 이름의 기원은 전통적인 J2EE를 “겨울”에 빗대어 “겨울” 후의 “봄”으로 새로운 시작을 의미함
- Rod Johnson이 창시한 개발프레임워크
- EJB가 제공했던 대부분의 기능을 일반 POJO(Plain Old Java Object)를 사용하여 개발할 수 있도록 지원함
- 엔터프라이즈 어플리케이션 개발의 복잡성을 줄이기 위한 목적으로 개발됨

- ❑ Spring Framework는 어플리케이션을 구성하는 **Bean** 객체의 생명 주기를 관장하는 **Core**를 기반으로 **DAO, ORM, AOP, JEE, Web**으로 구성됨

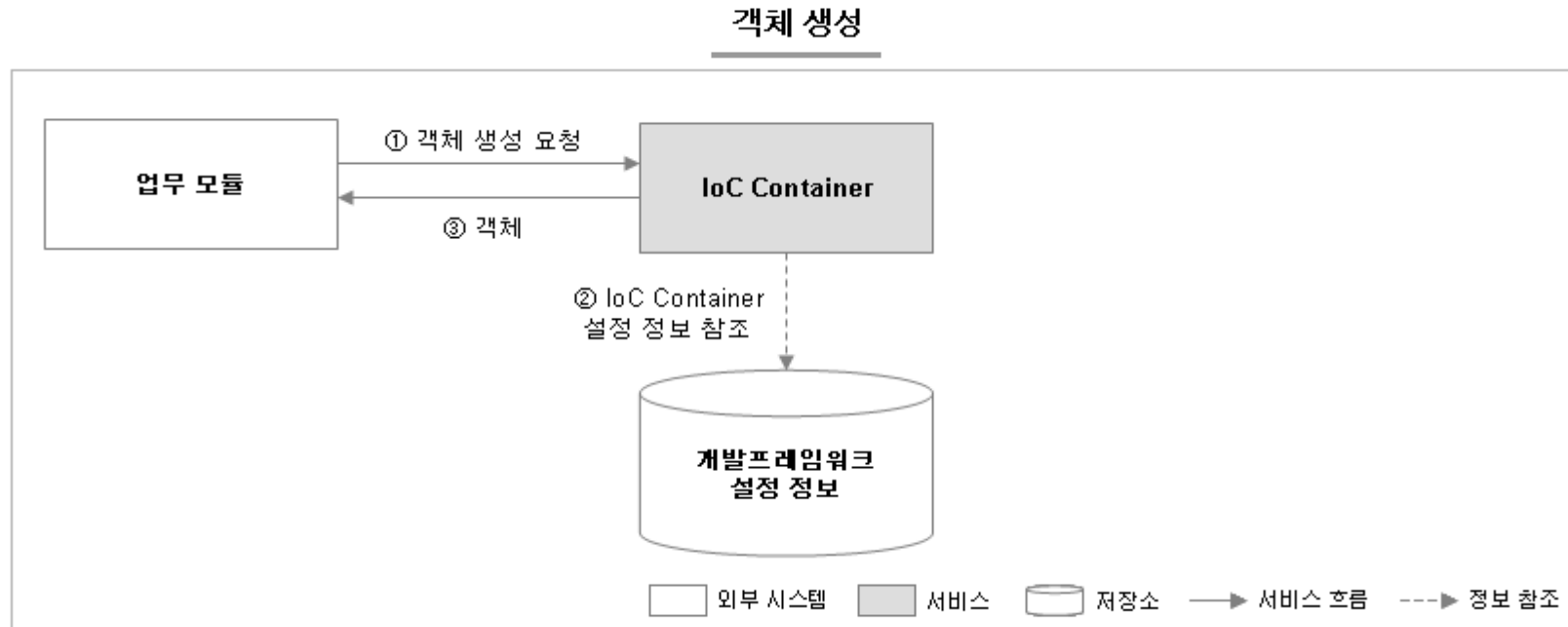


구분	설명
Core	<ul style="list-style-type: none"> Core 패키지는 프레임워크의 가장 기본적인 부분이고 IoC와 의존성 삽입(Dependency Injection-DI)기능을 제공한다.
DAO	<ul style="list-style-type: none"> DAO 패키지는 JDBC 코딩과 특정 데이터베이스 업체의 에러코드 파싱 등과 같은 작업의 필요성을 제거한 JDBC추상화 계층을 제공한다. 또한 프로그램적인 트랜잭션 관리 뿐 아니라 선언적인 트랜잭션 관리 기능을 제공한다.
ORM	<ul style="list-style-type: none"> ORM 패키지는 JAP, JDO, Hibernate, iBatis 등과 같은 객체-관계 맵핑(Object-Relational Mapping)을 위한 통합 계층을 제공한다.
AOP	<ul style="list-style-type: none"> AOP 패키지는 AOP Alliance에서 정의한 Aspect-지향 프로그래밍 방식을 지원한다.
Web	<ul style="list-style-type: none"> Web 패키지는 멀티파트 파일업로드기능, 서블릿 리스너를 사용한 IoC컨테이너의 초기화, 웹-기반애플리케이션 컨텍스트 등과 같은 기본적인 웹-기반 통합 기능들을 제공한다.
JEE	<ul style="list-style-type: none"> Spring 환경에서 다양한 Java EE 기술을 사용할 수 있도록 지원한다.

□ 개요

- 프레임워크의 기본적인 기능인 **IoC(Inversion of Control) Container** 기능을 제공하는 서비스이다.
- 객체의 생성 시, 객체가 참조하고 있는 타 객체에 대한 의존성을 소스 코드 내부에서 하드 코딩하는 것이 아닌, 소스 코드 외부에서 설정하게 함으로써, 유연성 및 확장성을 향상시킨다.
- 주요 기능 : **Dependency Injection, Bean Lifecycle Management** 
- 오픈소스 : Spring Framework 3.2.의 IoC Container를 수정없이 사용함.
- **의존성 관리의 중요성**

□ 개요



- ① 업무 모듈은 IoC Container 서비스에 객체 생성을 요청한다.
- ② IoC Container는 표준 프레임워크 설정 정보에 객체 생성을 위한 종속성 정보 등과 같은 IoC Container 설정 정보를 참조한다.
- ③ IoC Container는 설정 정보에 따라 객체를 생성하여 업무 모듈에게 돌려준다.

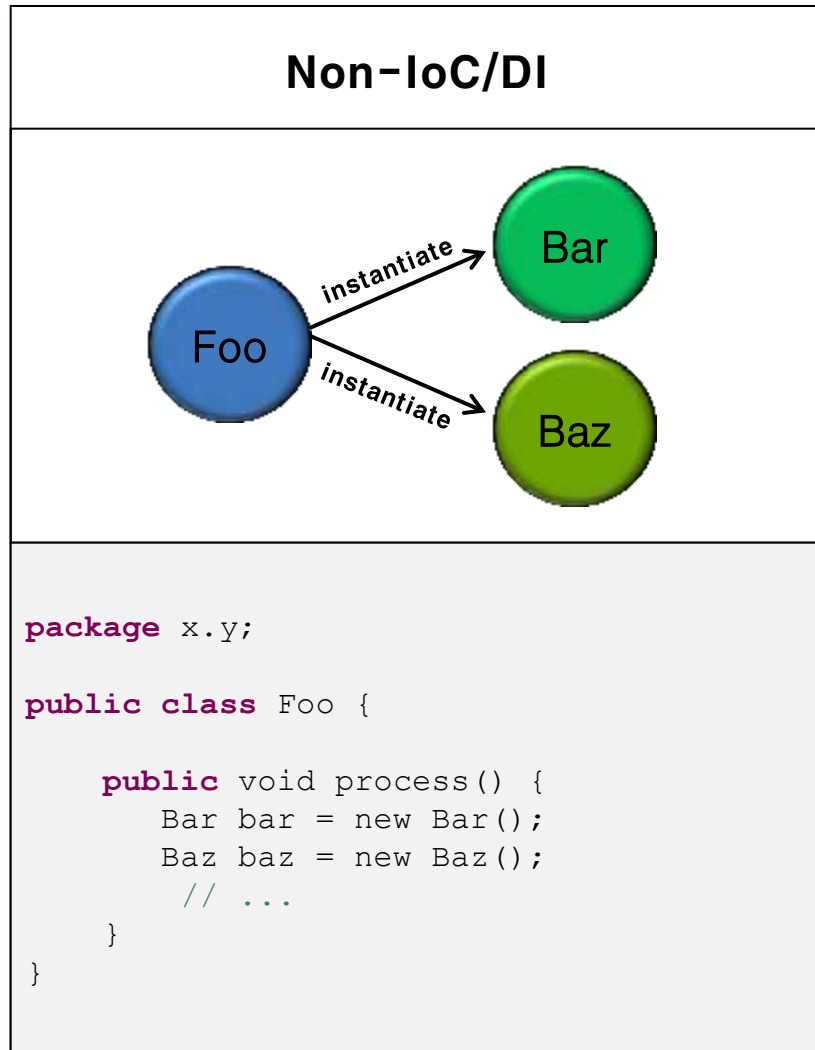
❑ IoC(Inversion of Control)이란?

- IoC는 Inversion of Control의 약자로 한글로 “제어의 역전” 또는 “역제어”라고 부른다. 어떤 모듈이 제어를 가진다는 것은 “어떤 모듈을 사용할 것인지”, “모듈의 함수는 언제 호출할 것인지” 등을 스스로 결정한다는 것을 의미한다. 이러한 제어가 역전되었다는 것은, 어떤 모듈이 사용할 모듈을 스스로 결정하는 것이 아니라 다른 모듈에게 선택권을 넘겨준다는 것을 의미한다.

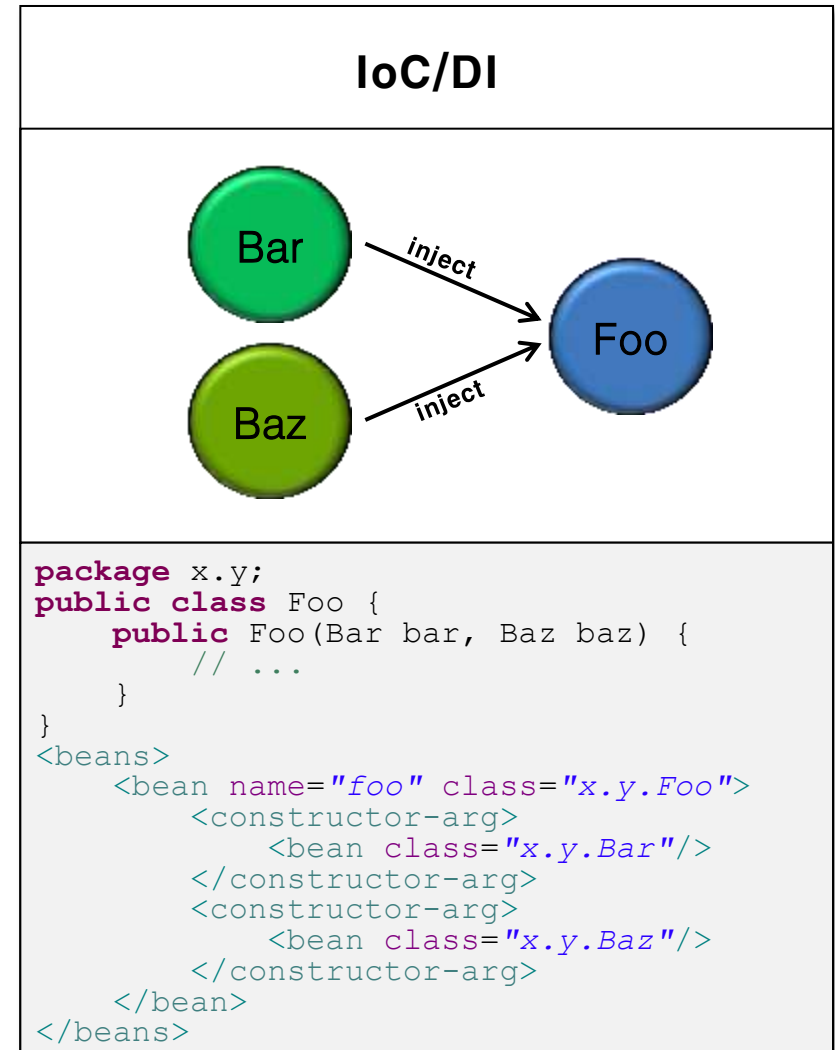
❑ DI(Dependency Injection)이란?

- **Dependency Injection**이란 모듈간의 의존성을 모듈의 외부(컨테이너)에서 주입시켜주는 기능으로 **Inversion of Control**의 한 종류이다.
- 런타임시 사용하게 될 의존대상과의 관계를 Spring Framework 이 총체적으로 결정하고 그 결정된 의존특징을 런타임시 부여한다.

❑ Non-IoC/DI vs IoC/DI



VS



❑ Container

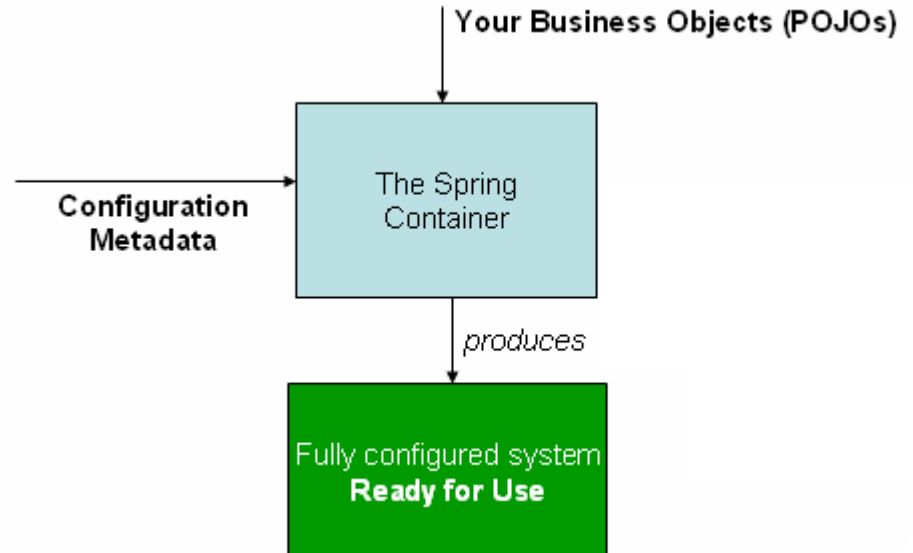
- Spring IoC Container는 객체를 생성하고, 객체 간의 의존성을 이어주는 역할을 한다.

❑ 설정 정보(Configuration Metadata)

- Spring IoC container가 “객체를 생성하고, 객체간의 의존성을 이어줄 수 있도록” 필요한 정보를 제공한다. 설정 정보는 기본적으로 XML 형태로 작성되며, 추가적으로 Java Annotation을 이용하여서도 설정이 가능하다.

❑ Bean

- Spring IoC Container에 의해 생성되고 관리되는 객체를 의미한다.



❑ BeanFactory

- **BeanFactory 인터페이스는 Spring IoC Container의 기능을 정의하고 있는 기본 인터페이스**이다.
- Bean 생성 및 의존성 주입, 생명주기 관리 등의 기능을 제공한다.

❑ ApplicationContext

- BeanFactory 인터페이스를 상속받는 ApplicationContext는 BeanFactory가 제공하는 기능 외에 Spring AOP, 메시지 리소스 처리(국제화에 사용됨), 이벤트 처리 등의 기능을 제공한다.
- 모든 ApplicationContext 구현체는 BeanFactory의 기능을 모두 제공하므로, 특별한 경우를 제외하고는 ApplicationContext를 사용하는 것이 바람직하다.
- Spring Framework는 다수의 ApplicationContext 구현체를 제공한다. 다음은 ClassPathXmlApplicationContext를 생성하는 예제이다.

```
ApplicationContext context = new ClassPathXmlApplicationContext(  
    new String[] { "services.xml", "daos.xml" });  
Foo foo = (Foo) context.getBean("foo");  
  
// an Application is also a BeanFactory (via inheritance)  
BeanFactory factory = context;
```

※ Spring Container = Bean Factory = ApplicationContext = DI Container = IoC Container

□ XML 설정 파일(1/2)

- XML 설정 파일은 <beans/> element를 root로 갖는다. 아래는 기본적인 XML 설정 파일의 모습이다.

```
<?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-3.2.xsd">

    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <!-- more bean definitions go here -->

</beans>
```

□ XML 설정 파일(2/2)

- XML 설정은 여러 개의 파일로 구성될 수 있으며, <import/> element를 사용하여 다른 XML 설정 파일을 import 할 수 있다.

```
<beans>

    <import resource="services.xml"/>
    <import resource="resources/messageSource.xml"/>
    <import resource="/resources/themeSource.xml"/>

    <bean id="bean1" class="..."/>
    <bean id="bean2" class="..."/>

</beans>
```

□ Bean 정의

- Bean 정의는 Bean을 객체화하고 의존성을 주입하는 등의 관리를 위한 정보를 담고 있다. XML 설정에서는 `<bean/>` element가 Bean 정의를 나타낸다. Bean 정의는 아래와 같은 속성을 가진다.



□ Bean 이름

- 모든 Bean은 하나의 id를 가지며, 하나 이상의 name을 가질 수 있다. id는 container 안에서 고유해야 한다.

```
<bean id="exampleBean" class="example.ExampleBean"/>

<bean name="anotherExample" class="examples.ExampleBeanTwo"/>
```

- Bean은 <alias/> element를 이용하여 추가적인 name을 가질 수 있다.

```
<alias name="fromName" alias="toName"/>
```

□ Bean Class

- 모든 Bean은 객체화를 위한 Java class가 필요하다. (예외적으로 상속의 경우 class가 없어도 된다.)

```
<bean id="exampleBean" class="example.ExampleBean"/>

<bean name="anotherExample" class="examples.ExampleBeanTwo"/>
```

□ Bean 객체화(1/2)

- 일반적으로 Bean 객체화는 Java 언어의 'new' 연산자를 사용한다. 이 경우 별도의 설정은 필요없다.
- 'new' 연산자가 아닌 static factory 메소드를 사용하여 Bean을 객체화할 수 있다. 이 경우 Constructor Injection 방식의 의존성 주입 설정을 따른다.

```
<bean id="exampleBean"
      class="examples.ExampleBean"
      factory-method="createInstance"/>
```

- 자신의 static factory 메소드가 아닌 별도의 Factory 클래스의 static 메소드를 사용하여 Bean을 객체화할 수 있다. 이 경우 역시 Constructor Injection 방식의 의존성 주입 설정을 따른다.

```
<!-- the factory bean, which contains a method called createInstance() -->
<bean id="serviceLocator" class="com.foo.DefaultServiceLocator">
  <!-- inject any dependencies required by this locator bean -->
</bean>

<!-- the bean to be created via the factory bean -->
<bean id="exampleBean"
      factory-bean="serviceLocator"
      factory-method="createInstance"/>
```

```
ExampleBean exampleBean = new ExampleBean();
ExampleBean exampleBean = ExampleBean.createInstance();
ExampleBean exampleBean = DefaultServiceLocator.createInstance();
```

□ Bean 객체화(2/2)

- <bean/> element의 'lazy-init' attribute를 사용하여 Bean 객체화 시기를 설정할 수 있다.
 - 일반적으로 Bean 객체화는 BeanFactory가 객체화되는 시점에 수행된다. 만약, 'lazy-init' attribute 값이 'true'인 경우, 설정된 Bean의 객체가 실제로 필요하다고 요청한 시점에 객체화가 수행된다.
 - 'lazy-init' attribute가 설정되어 있지 않으면 기본값을 사용한다. Spring Framework의 기본값은 'false'이다.

```
<bean id="lazy" class="com.foo.ExpensiveToCreateBean" lazy-init="true"/>

<bean name="not.lazy" class="com.foo.AnotherBean"/>
```

- <beans/> element의 'default-lazy-init' attribute를 사용하여 XML 설정 파일 내의 모든 Bean 정의에 대한 lazy-init attribute의 기본값을 설정할 수 있다.

```
<beans default-lazy-init="true">
    <!-- no beans will be pre-instantiated... -->
</beans>
```

□ 의존성 주입(1/16)

- 의존성 주입에는 **Constructor Injection**과 **Setter Injection**의 두가지 방식이 있다.
- Constructor Injection(1/3)
 - Constructor Injection은 **argument**를 갖는 생성자를 사용하여 의존성을 주입하는 방식이다. **<constructor-arg/>** element를 사용한다. 생성자의 argument와 **<constructor-arg/>** element는 class가 같은 것끼리 매핑한다.

```
package x.y;

public class Foo {

    public Foo(Bar bar, Baz baz) {
        // ...
    }
}
```

```
<beans>
    <bean name="foo" class="x.y.Foo">
        <constructor-arg>
            <bean class="x.y.Bar"/>
        </constructor-arg>
        <constructor-arg>
            <bean class="x.y.Baz"/>
        </constructor-arg>
    </bean>
</beans>
```


□ 의존성 주입(2/16)

– Constructor Injection(2/3)

- 만약 생성자가 같은 class의 argument를 가졌거나 primitive type인 경우 argument와 <constructor-arg/> element간의 매핑이 불가능하다. 이 경우, **Type**을 지정하거나 순서를 지정할 수 있다.

```
package examples;

public class ExampleBean {

    // No. of years to the calculate the Ultimate Answer
    private int years;

    // The Answer to Life, the Universe, and Everything
    private String ultimateAnswer;

    public ExampleBean(int years, String ultimateAnswer) {
        this.years = years;
        this.ultimateAnswer = ultimateAnswer;
    }
}
```

□ 의존성 주입(3/16)

– Constructor Injection(3/3)

- Type 지정

```
<bean id="exampleBean" class="examples.ExampleBean">  
    <constructor-arg type="int" value="7500000"/>  
    <constructor-arg type="java.lang.String" value="42"/>  
</bean>
```

- 순서 지정

```
<bean id="exampleBean" class="examples.ExampleBean">  
    <constructor-arg index="0" value="7500000"/>  
    <constructor-arg index="1" value="42"/>  
</bean>
```

□ 의존성 주입(4/16)

– Setter Injection(1/2)

- Setter Injection은 **argument가 없는 기본 생성자를 사용하여 객체를 생성한 후, setter 메소드를 사용하여 의존성을 주입하는 방식으로, <property/> element를 사용한다.**
- Class에 attribute(또는 setter 메소드 명)과 <property/> element의 'name' attribute를 사용하여 매핑한다.

```
public class ExampleBean {  
  
    private AnotherBean beanOne;  
    private YetAnotherBean beanTwo;  
    private int i;  
  
    public void setBeanOne(AnotherBean beanOne) {  
        this.beanOne = beanOne;  
    }  
  
    public void setBeanTwo(YetAnotherBean beanTwo) {  
        this.beanTwo = beanTwo;  
    }  
  
    public void setIntegerProperty(int i) {  
        this.i = i;  
    }  
}
```

□ 의존성 주입(5/16)

– Setter Injection(2/2)

```
<bean id="exampleBean" class="examples.ExampleBean">
  <!-- setter injection using the nested <ref/> element -->
  <property name="beanOne"><ref bean="anotherExampleBean"/></property>

  <!-- setter injection using the neater 'ref' attribute -->
  <property name="beanTwo" ref="yetAnotherBean"/>
  <property name="integerProperty" value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
```

- 매핑 규칙은 <property/> element의 'name' attribute의 첫문자를 알파벳 대문자로 변경하고 그 앞에 'set'을 붙인 setter 메소드를 호출한다.

```
<bean id="exampleBean"
      class="examples.ExampleBean">
  <property name="beanOne">
    <ref bean="anotherExampleBean"/>
  </property>
</bean>
```

```
public class ExampleBean {
  public void setBeanOne(
    AnotherBean beanOne)
  {
    this.beanOne = beanOne;
  }
}
```

□ 의존성 주입(6/16)

– 의존성 상세 설정(1/11)

- <constructor-arg/> element 과 <property/> element는 ‘명확한 값’, ‘다른 Bean에 대한 참조’, ‘Inner Bean’, ‘Collection’, ‘Null’ 등의 값을 가질 수 있다.
- 명확한 값

Java Primitive Type, String 등의 명확한 값을 나타낸다. 사람이 인식 가능한 문자열 형태를 값으로 갖는 <value/> element를 사용한다. Spring IoC container가 String 값을 해당하는 type으로 변환하여 주입해준다.

```
<bean id="myDataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close">
    <!-- results in a setDriverClassName(String) call -->
    <property name="driverClassName">
        <value>com.mysql.jdbc.Driver</value>
    </property>
    <property name="url">
        <value>jdbc:mysql://localhost:3306/mydb</value>
    </property>
    <property name="username">
        <value>root</value>
    </property>
    <property name="password">
        <value>masterkaoli</value>
    </property>
</bean>
```

□ 의존성 주입(7/16)

– 의존성 상세 설정(2/11)

- 다른 Bean에 대한 참조

<ref/> element를 사용하여 다른 Bean 객체를 참조할 수 있다. 참조할 객체를 지정하는 방식은 'container', 'local', 'parent' 등이 있다.

1. container : 가장 일반적인 방식으로 같은 container 또는 부모 container에서 객체를 찾는다.

```
<ref bean="someBean"/>
```

2. local : 같은 XML 설정 파일 내에 정의된 Bean 객체를 찾는다.

```
<ref local="someBean"/>
```

3. parent : 부모 XML 설정 파일 내에 정의된 Bean 객체를 찾는다.

```
<!-- in the parent context -->
<bean id="accountService" class="com.foo.SimpleAccountService">
  <!-- insert dependencies as required as here -->
</bean>
```

```
<!-- in the child (descendant) context -->
<bean id="accountService" class="org.springframework.aop.framework.ProxyFactoryBean">
  <property name="target">
    <ref parent="accountService"/>
  </property>
</bean>
```

□ 의존성 주입(8/16)

– 의존성 상세 설정(3/11)

- Inner Bean

<property/> 또는 <constructor-arg/> element 안에 있는 <bean/> element를 inner bean이라고 한다. Inner bean의 'scope' flag 와 'id', 'name'은 무시된다. Inner bean의 scope은 항상 prototype이다. 따라서 inner bean을 다른 bean에 주입하는 것은 불가능하다.

```
<bean id="outer" class="...">
  <!-- instead of using a reference to a target bean, simply define the target bean
        inline -->
  <property name="target">
    <bean class="com.example.Person"> <!-- this is the inner bean -->
      <property name="name" value="Fiona Apple"/>
      <property name="age" value="25"/>
    </bean>
  </property>
</bean>
```

□ 의존성 주입(9/16)

– 의존성 상세 설정(4/11)

• Collection(1/2)

Java Collection 타입인 List, Set, Map, Properties를 표현하기 위해 <list/>, <set/>, <map/>, <props/> element가 사용된다. map의 key와 value, set의 value의 값은 아래 element 중 하나가 될 수 있다.

bean ref idref list set map props value null
--

```
<bean id="moreComplexObject" class="example.ComplexObject">
  <!-- results in a setAdminEmails(java.util.Properties) call -->
  <property name="adminEmails">
    <props>
      <prop key="administrator">administrator@example.org</prop>
      <prop key="support">support@example.org</prop>
      <prop key="development">development@example.org</prop>
    </props>
  </property>
  <!-- results in a setSomeList(java.util.List) call -->
  <property name="someList">
    <list>
      <value>a list element followed by a reference</value>
      <ref bean="myDataSource" />
    </list>
  </property>
  ...
</bean>
```


□ 의존성 주입(10/16)

- 의존성 상세 설정(5/11)
 - Collection(2/2)

```
...

<!-- results in a setSomeMap(java.util.Map) call -->
<property name="someMap">
  <map>
    <entry>
      <key><value>an entry</value></key>
      <value>just some string</value>
    </entry>
    <entry>
      <key><value>a ref</value></key>
      <ref bean="myDataSource" />
    </entry>
  </map>
</property>
<!-- results in a setSomeSet(java.util.Set) call -->
<property name="someSet">
  <set>
    <value>just some string</value>
    <ref bean="myDataSource" />
  </set>
</property>
</bean>
```

□ 의존성 주입(11/16)

– 의존성 상세 설정(6/11)

- Null

Java의 null 값을 사용하기 위해서 `<null/>` element를 사용한다. Spring IoC container는 value 값이 설정되어 있지 않은 경우 빈문자열("")로 인식한다.

```
<bean class="ExampleBean">
  <property name="email"><value/></property>
</bean>
```

위 ExampleBean의 email 값은 ""이다. 아래는 email의 값이 null인 예제이다.

```
<bean class="ExampleBean">
  <property name="email"><null/></property>
</bean>
```

□ 의존성 주입(12/16)

– 의존성 상세 설정(7/11)

- 간편한 표기 1

<property/>, <constructor-arg/>, <entry/> element의 <value/> element는 'value' attribute로 대체될 수 있다.

```
<property name="myProperty">
  <value>hello</value>
</property>
```

=

```
<property name="myProperty" value="hello"/>
```

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

=

```
<constructor-arg value="hello"/>
```

```
<entry key="myKey">
  <value>hello</value>
</entry>
```

=

```
<entry key="myKey" value="hello"/>
```

□ 의존성 주입(13/16)

– 의존성 상세 설정(8/11)

- 간편한 표기 2

<property/>, <constructor-arg/> element의 <ref/> element는 'ref' attribute로 대체될 수 있다.

```
<property name="myProperty">
  <ref bean="myBean">
</property>
```

=

```
<property name="myProperty" ref="myBean"/>
```

```
<constructor-arg>
  <ref bean="myBean">
</constructor-arg>
```

=

```
<constructor-arg ref="myBean"/>
```

- 간편한 표기 3

<entry/> element의 'key', 'ref' element 는 'key-ref', 'value-ref' attribute로 대체될 수 있다.

```
<entry>
  <key>
    <ref bean="myKeyBean" />
  </key>
  <ref bean="myValueBean" />
</entry>
```

=

```
<entry key-ref="myKeyBean"
  value-ref="myValueBean"/>
```

□ 의존성 주입(14/16)

– 의존성 상세 설정(9/11)

- p-namespace(1/2)

<property/> element 대신 'p-namespace'를 사용하여 XML 설정을 작성할 수 있다. 아래 classic bean과 p-namespace bean은 동일한 Bean 설정이다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">

  <bean name="classic" class="com.example.ExampleBean">
    <property name="email" value="foo@bar.com"/>
  </bean>

  <bean name="p-namespace" class="com.example.ExampleBean"
    p:email="foo@bar.com"/>
</beans>
```

□ 의존성 주입(15/16)

– 의존성 상세 설정(10/11)

- p-namespace(2/2)

Attribute 이름 끝에 '-ref'를 붙이면 참조로 인식한다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:p="http://www.springframework.org/schema/p"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">

    <bean name="john-classic" class="com.example.Person">
        <property name="name" value="John Doe"/>
        <property name="spouse" ref="jane"/>
    </bean>

    <bean name="john-modern"
        class="com.example.Person"
        p:name="John Doe"
        p:spouse-ref="jane"/>

    <bean name="jane" class="com.example.Person">
        <property name="name" value="Jane Doe"/>
    </bean>
</beans>
```

□ 의존성 주입(16/16)

– 의존성 상세 설정(11/11)

- Compound property name

복합 형식의 property 이름도 사용할 수 있다.

```
<bean id="foo" class="foo.Bar">  
    <property name="fred.bob.sammy" value="123" />  
</bean>
```

□ Autowiring

- Spring IoC container는 서로 관련된 Bean 객체를 자동으로 엮어줄 수 있다.
자동엮기(**autowiring**)는 각각의 **bean** 단위로 설정되며, 자동엮기 기능을 사용하면 **<property/>**나 **<constructor-arg/>**를 지정할 필요가 없으므로, 타이핑일 줄일 수 있다.
- 자동엮기에는 5가지 모드가 있으며, XML 기반 설정에서는 **<bean/>** element의 'autowire' attribute를 사용하여 설정할 수 있다.

Scope	설명
no	자동엮기를 사용하지 않는다. Bean에 대한 참조는 <ref/> element를 사용하여 지정해야만 한다. 이 모드가 기본(default)이다.
byName	Property의 이름으로 자동엮기를 수행한다. Property의 이름과 같은 이름을 가진 bean을 찾아서 엮어준다.
byType	Property의 타입으로 자동엮기를 수행한다. Property의 타입과 같은 타입을 가진 bean을 찾아서 엮어준다. 만약 같은 타입을 가진 bean이 container에 둘 이상 존재할 경우 exception이 발생한다. 만약 같은 타입을 가진 bean이 존재하지 않는 경우, 아무 일도 발생하지 않는다. 즉, property에는 설정되지 않는다.
constructor	<i>byType</i> 과 유사하지만, 생성자 argument에만 적용된다. 만약 같은 타입의 bean이 존재하지 않거나 둘 이상 존재할 경우, exception이 발생한다.
autodetect	Bean class의 성질에 따라 <i>constructor</i> 와 <i>byType</i> 모드 중 하나를 선택한다. 만약 default 생성자가 존재하면, <i>byType</i> 모드가 적용된다.

- **<bean/>** element의 'autowire-candidate' attribute 값을 'false'로 설정함으로써, 대상 bean이 다른 bean과 자동으로 엮이는 것을 방지한다.

□ Bean Scope(1/4)

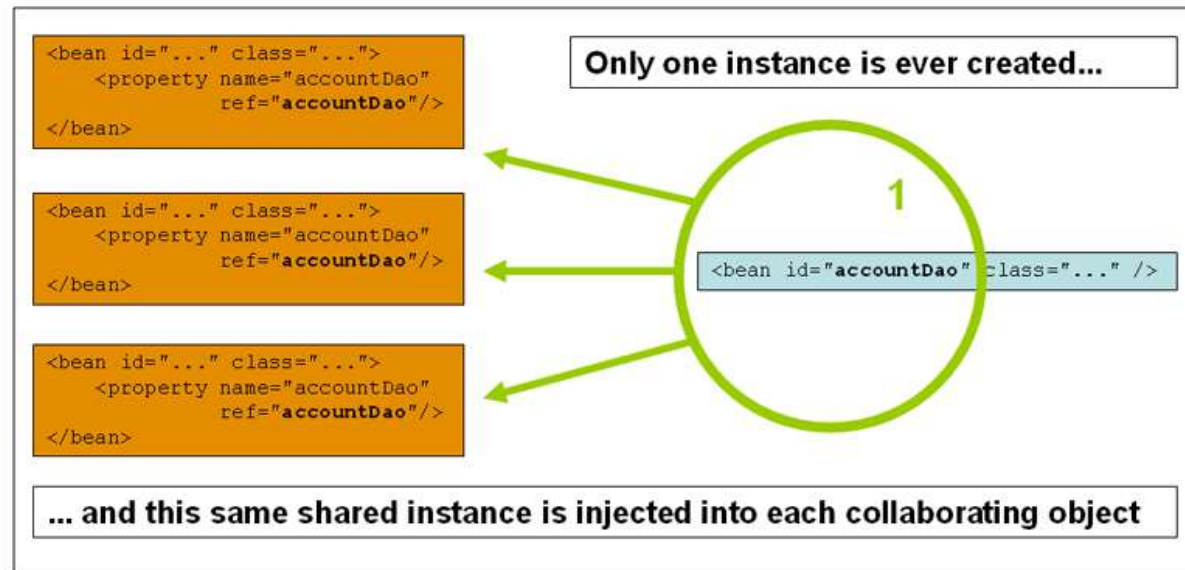
- Bean Scope은 객체가 유효한 범위로 아래 5가지의 scope이 있다.
- Spring Bean의 기본 Scope는 **singleton** 이다.

Scope	설 명
singleton	하나의 Bean 정의에 대해서 Spring IoC Container 내에 단 하나의 객체만 존재한다.
prototype	하나의 Bean 정의에 대해서 다수의 객체가 존재할 수 있다.
request	하나의 Bean 정의에 대해서 하나의 HTTP request의 생명주기 안에 단 하나의 객체만 존재한다; 즉, 각각의 HTTP request는 자신만의 객체를 가진다. Web-aware Spring ApplicationContext 안에서만 유효하다.
session	하나의 Bean 정의에 대해서 하나의 HTTP Session의 생명주기 안에 단 하나의 객체만 존재한다. Web-aware Spring ApplicationContext 안에서만 유효하다.
global session	하나의 Bean 정의에 대해서 하나의 global HTTP Session의 생명주기 안에 단 하나의 객체만 존재한다. 일반적으로 portlet context 안에서 유효하다. Web-aware Spring ApplicationContext 안에서만 유효하다.

□ Bean Scope(2/4)

– Singleton Scope

- Bean이 singleton인 경우, 단 하나의 객체만 공유된다.



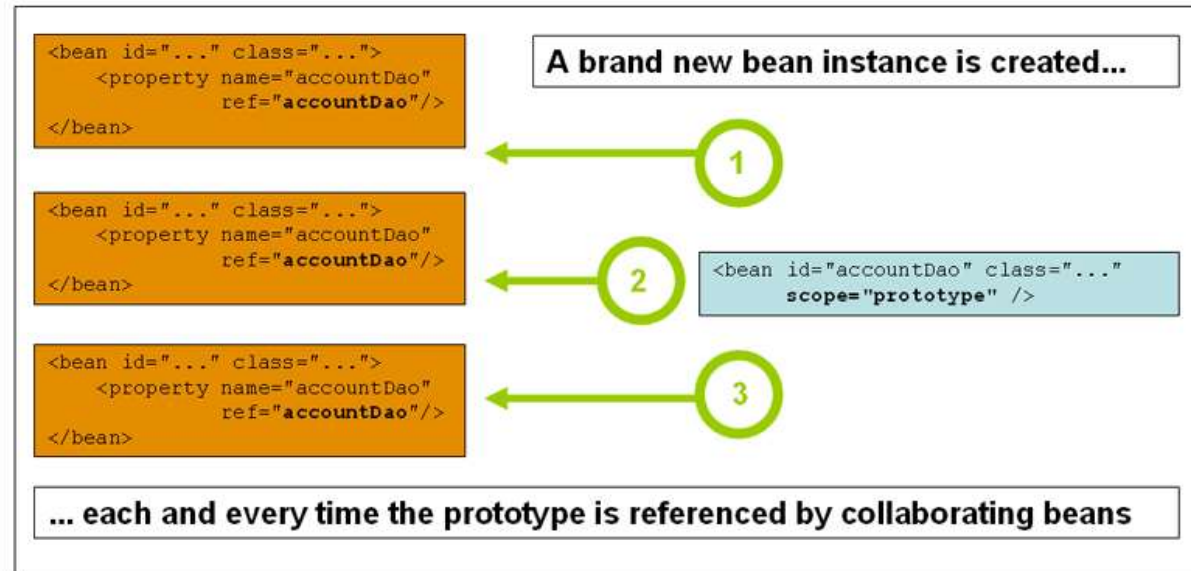
- Spring의 기본 scope은 'singleton'이다. 설정하는 방법은 아래와 같다.

```
<bean id="accountService" class="com.foo.DefaultAccountService"/>  
  
<bean id="accountService" class="com.foo.DefaultAccountService" scope="singleton"/>  
  
<bean id="accountService" class="com.foo.DefaultAccountService" singleton="true"/>
```

□ Bean Scope(3/4)

– Prototype Scope

- Singleton이 아닌 prototype scope의 형태로 정의된 **bean**은 필요한 매 순간 새로운 **bean** 객체가 생성된다.



- 설정하는 방법은 아래와 같다.

```
<bean id="accountService" class="com.foo.DefaultAccountService" scope="prototype"/>

<bean id="accountService" class="com.foo.DefaultAccountService" singleton="false"/>
```

□ Bean Scope(4/4)

– 기타 Scope

- request, session, global session은 Web 기반 어플리케이션에서만 유효한 scope이다.

- Request Scope

```
<bean id="loginAction" class="com.foo.LoginAction" scope="request"/>
```

위 정의에 따라, Spring container는 모든 HTTP request에 대해서 'loginAction' bean 정의에 대한 LoginAction 객체를 생성할 것이다. 즉, 'loginAction' bean은 HTTP request 수준에 한정된다(scoped). 요청에 대한 처리가 완료되었을 때, 한정된(scoped) bean도 폐기된다.

- Session Scope

```
<bean id="userPreferences" class="com.foo.UserPreferences" scope="session"/>
```

위 정의에 따라, Spring container는 하나의 HTTP Session 일생동안 'userPreferences' bean 정의에 대한 UserPreferences 객체를 생성할 것이다. 즉, 'userPreferences' bean은 HTTP Session 수준에 한정된다(scoped). HTTP Session이 폐기될 때, 한정된(scoped) bean도 폐기된다.

- Global Session Scope

```
<bean id="userPreferences" class="com.foo.UserPreferences" scope="globalSession"/>
```

global session scope은 HTTP Session scope과 비슷하지만 단지 portlet-based web 어플리케이션에서만 사용할 수 있다.

Portlet 명세(specifications)는 global Session을 하나의 portlet web 어플리케이션을 구성하는 여러 portlet들 모두가 공유하는 것으로 정의하고 있다. global session scope으로 설정된 bean은 global portlet Session의 일생에 한정된다.



□ Bean 성질 변화(1/4)

– Lifecycle Callback

- Spring IoC Container는 **Bean의 각 생명주기에 호출되도록 설정된 메소드를 호출해준다.**
- Initialization callback

org.springframework.beans.factory.InitializingBean interface를 구현하면 bean에 필요한 모든 property를 설정한 후, 초기화 작업을 수행한다. InitializingBean interface는 다음 메소드를 명시하고 있다.

```
void afterPropertiesSet() throws Exception;
```

일반적으로, InitializingBean interface의 사용을 권장하지 않는다. 왜냐하면 code가 불필요하게 Spring과 결합되기(couple) 때문이다. 대안으로, bean 정의는 초기화 메소드를 지정할 수 있다. XML 기반 설정의 경우, 'init-method' attribute를 사용한다.

```
public class ExampleBean {  
  
    public void init() {  
        // do some initialization work  
    }  
}
```

```
<bean id="exampleInitBean" class="examples.ExampleBean" init-method="init"/>
```

□ Bean 성질 변화(2/4)

– Lifecycle Callback

- Destruction callback

org.springframework.beans.factory.DisposableBean interface를 구현하면, container가 파괴될 때 bean이 callback을 받을 수 있다. DisposableBean interface는 다음 메소드를 명시하고 있다.

```
void destroy() throws Exception;
```

일반적으로, DisposableBean interface의 사용을 권장하지 않는다. 왜냐하면 code가 불필요하게 Spring과 결합되기(couple) 때문이다. 대안으로, bean 정의는 초기화 메소드를 지정할 수 있다. XML 기반 설정의 경우, 'destroy-method' attribute를 사용한다.

```
public class ExampleBean {  
  
    public void cleanup() {  
        // do some destruction work (like releasing pooled connections)  
    }  
}
```

```
<bean id="exampleInitBean" class="examples.ExampleBean" destroy-method="cleanup"/>
```

□ Bean 성질 변화(3/4)

– Lifecycle Callback

- 기본 Instantiation & Destruction callback

Spring IoC container 레벨에서 기본 Instantiation & Destruction callback 메소드를 지정할 수 있다. <beans/> element의 'default-init-method', 'default-destroy-method' attribute를 사용한다.

```
<beans default-init-method="init" default-destroy-method="destroy">

    <bean id="blogService" class="com.foo.DefaultBlogService">
        <property name="blogDao" ref="blogDao" />
    </bean>

</beans>
```

□ Bean 성질 변화(4/4)

– Knowing who you are

- BeanFactoryAware

org.springframework.beans.factory.BeanFactoryAware interface를 구현한 class는 자신을 생성한 BeanFactory를 참조할 수 있다.

```
public interface BeanFactoryAware {  
  
    void setBeanFactory(BeanFactory beanFactory) throws BeansException;  
  
}
```

BeanFactoryAware interface를 사용하면, 자신을 생성한 BeanFactory를 알 수 있고, 프로그램적으로 다른 bean을 검색할 수 있다. 하지만 이 방법은 Spring과의 결합을 발생시키고, Inversion of Control 스타일에도 맞지 않으므로 피하는 것이 좋다. 대안으로는 org.springframework.beans.factory.config.ObjectFactoryCreatingFactoryBean을 사용할 수 있다.

❑ Bean Profile (표준프레임워크 3.0부터 추가)

– Bean의 선택적 사용

- Bean profile

Profile을 이용하여 bean을 중복으로 선언하고 실제 사용시 선택적으로 사용하도록 설정할 수 있다. Bean Profile설정은 <beans profile/> element를 이용한다.

```
<beans profile="dev">
  <jdbc:embedded-database id="dataSource"> ... 생략
</jdbc:embedded-database>
</beans>

<beans profile="production">
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
    destroy-method="close"> ... 생략
</bean>
</beans>
```

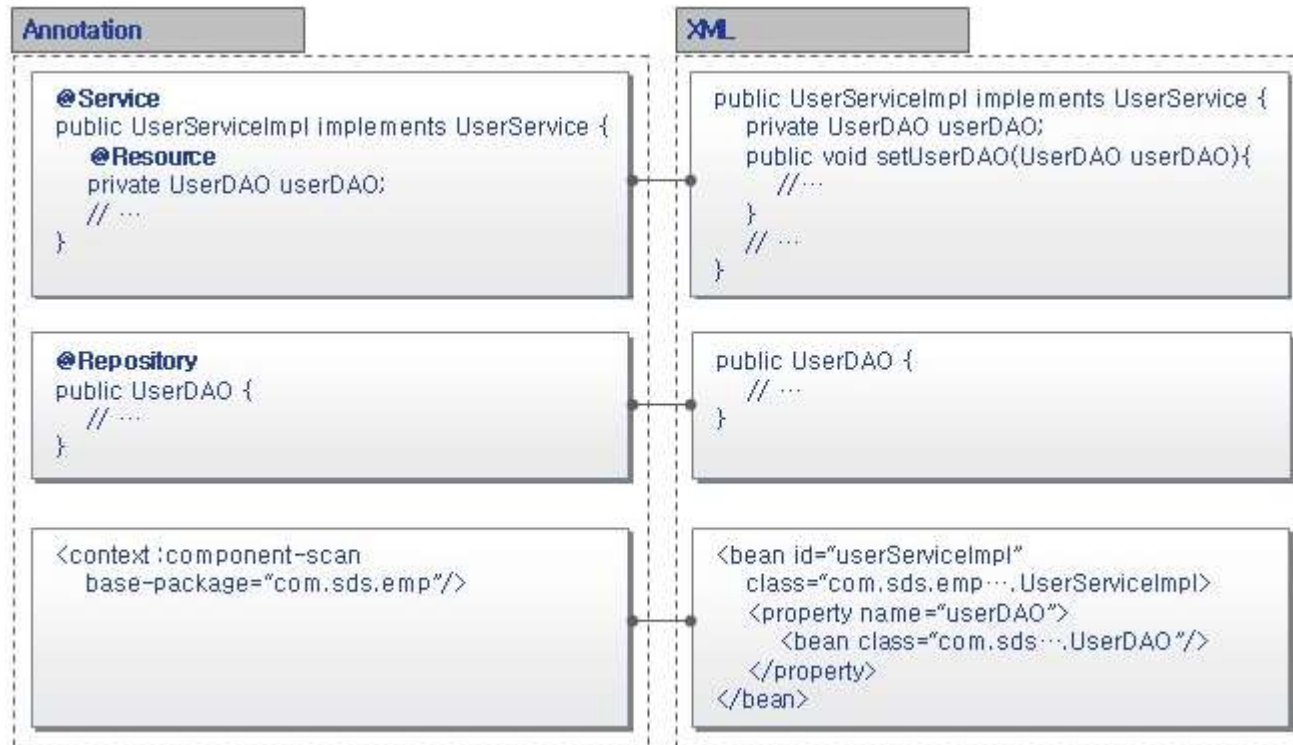
- Profile별 bean설정을 적용하기 위해서는 web.xml의 context-param (spring.profiles.active의 param-value)을 설정해주어야 한다.

```
<context-param>
  <param-name>spring.profiles.active</param-name>
  <param-value>production</param-value>
</context-param>
```

LAB 201-IoC 실습(1)

□ Annotaion

- XML 설정 파일을 사용하는 대신 자바 어노테이션을 사용할 수 있음 (자바 5 이상)
- annotation의 사용으로 설정파일을 간결화하고, View 페이지와 객체 또는 메소드의 매핑을 명확하게 할 수 있다



□ Annotation 기반 설정(1/12)

- Spring은 Java Annotation을 사용하여 Bean 정의를 설정할 수 있다.

@Required, @Autowired, @Qualifier, @Resource, @PostConstruct, @PreDestroy 기능을 사용하기 위해서는 다음 namespace와 element를 추가해야 한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-3.2.xsd">

    <context:annotation-config/>

</beans>
```

□ Annotation 기반 설정(2/12)

– @Required

- @Required annotation은 setter 메소드에 적용된다. @Required annotation이 설정된 property는 <property/>, <constructor-arg/> element를 통해서 명시적으로 값이 설정되거나, autowiring에 의해서 값이 설정되어야 한다.

```
public class SimpleMovieLister {  
  
    private MovieFinder movieFinder;  
  
    @Required  
    public void setMovieFinder(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
  
    // ...  
}
```

□ Annotation 기반 설정(3/12)

– @Autowired(1/3)

- @Autowired annotation은 자동으로 역을 property를 지정하기 위해 사용한다. setter 메소드, 일반적인 메소드, 생성자, field 등에 적용된다.
- Setter 메소드

```
public class SimpleMovieLister {  
  
    private MovieFinder movieFinder;  
  
    @Autowired  
    public void setMovieFinder(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
  
    // ...  
}
```

□ Annotation 기반 설정(4/12)

– @Autowired(2/3)

- 일반적인 메소드

```
public class MovieRecommender {  
  
    private MovieCatalog movieCatalog;  
  
    private CustomerPreferenceDao customerPreferenceDao;  
  
    @Autowired  
    public void prepare(MovieCatalog movieCatalog, CustomerPreferenceDao  
customerPreferenceDao) {  
        this.movieCatalog = movieCatalog;  
        this.customerPreferenceDao = customerPreferenceDao;  
    }  
  
    // ...  
}
```

□ Annotation 기반 설정(5/12)

– @Autowired(3/3)

- 생성자 및 field

```
public class MovieRecommender {  
  
    @Autowired  
    private MovieCatalog movieCatalog;  
  
    private CustomerPreferenceDao customerPreferenceDao;  
  
    @Autowired  
    public MovieRecommender(CustomerPreferenceDao customerPreferenceDao) {  
        this.customerPreferenceDao = customerPreferenceDao;  
    }  
  
    // ...  
}
```


□ Annotation 기반 설정(6/12)

– @Qualifier(1/3)

- @Autowired annotation만을 사용하는 경우, 같은 Type의 Bean이 둘 이상 존재할 때 문제가 발생한다. 이를 방지하기 위해서 @Qualifier annotation을 사용하여 찾을 Bean의 대상 집합을 좁힐 수 있다. @Qualifier annotation은 field 뿐 아니라 생성자 또는 메소드의 parameter에도 사용할 수 있다.
- Field

```
public class MovieRecommender {  
  
    @Autowired  
    @Qualifier("main")  
    private MovieCatalog movieCatalog;  
  
    // ...  
}
```

□ Annotation 기반 설정(7/12)

- @Qualifier(2/3)
 - 메소드 Parameter

```
public class MovieRecommender {  
  
    private MovieCatalog movieCatalog;  
  
    private CustomerPreferenceDao customerPreferenceDao;  
  
    @Autowired  
    public void prepare(@Qualifier("main") MovieCatalog movieCatalog,  
                        CustomerPreferenceDao customerPreferenceDao) {  
        this.movieCatalog = movieCatalog;  
        this.customerPreferenceDao = customerPreferenceDao;  
    }  
  
    // ...  
}
```

□ Annotation 기반 설정(7/12)

– @Qualifier(3/3)

- @Qualifier annotation의 값으로 사용되는 qualifier는 <bean/> element의 <qualifier/> element로 설정한다.

```
<?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-2.5.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-2.5.xsd">

    <context:annotation-config/>

    <bean class="example.SimpleMovieCatalog">
        <qualifier value="main"/>
        <!-- inject any dependencies required by this bean -->
    </bean>

    <bean class="example.SimpleMovieCatalog">
        <qualifier value="action"/>
        <!-- inject any dependencies required by this bean -->
    </bean>

    <bean id="movieRecommender" class="example.MovieRecommender"/>
</beans>
```

□ Annotation 기반 설정(8/12)

– @Resource

- @Resource annotation의 name 값으로 대상 bean을 찾을 수 있다. @Resource annotation은 field 또는 메소드에 사용할 수 있다.

```
public class SimpleMovieLister {  
  
    private MovieFinder movieFinder;  
  
    @Resource(name="myMovieFinder")  
    public void setMovieFinder(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
}
```

- @Resource annotation에 name 값이 없을 경우, field 명 또는 메소드 명을 이용하여 대상 bean을 찾는다.

```
public class SimpleMovieLister {  
  
    private MovieFinder movieFinder;  
  
    @Resource  
    public void setMovieFinder(MovieFinder movieFinder) {  
        this.movieFinder = movieFinder;  
    }  
}
```

□ Annotation 기반 설정(9/12)

– @PostConstruct & @PreDestroy

- @PostConstruct와 @PreDestroy는 각각 Instantiation callback, Destruction callback 메소드를 지정하기 위해 사용한다.

```
public class CachingMovieLister {  
  
    @PostConstruct  
    public void populateMovieCache() {  
        // populates the movie cache upon initialization...  
    }  
  
    @PreDestroy  
    public void clearMovieCache() {  
        // clears the movie cache upon destruction...  
    }  
}
```

□ Annotation 기반 설정(10/12)

– Auto-detecting components(1/3)

- Spring은 @Repository, @Service, @Controller annotation을 사용하여, 각각 Persistence, Service, Presentation 레이어의 컴포넌트로 지정하여 특별한 관리 기능을 제공하고 있다. @Repository, @Service, @Controller는 @Component annotation을 상속받고 있다. Spring IoC Container는 @Component annotation(또는 자손)으로 지정된 class를 XML Bean 정의 없이 자동으로 찾을 수 있다.

```
<?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-2.5.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context-2.5.xsd">

    <context:component-scan base-package="org.example"/>

</beans>
```

<context:component-scan/> element의 'base-package' attribute는 컴포넌트를 찾을 기본 package이다. ','를 사용하여 다수의 기본 package를 지정할 수 있다.

- <context:component-scan/>을 쓰는 경우에는 <context:annotation-config/>를 별도로 쓰지 않아도 된다.

□ Annotation 기반 설정(11/12)

– Auto-detecting components(2/3)

- 이름 설정

@Component, @Repository, @Service, @Controller annotation의 name 값으로 bean의 이름을 지정할 수 있다. 아래 예제의 Bean 이름은 “myMovieLister”이다.

```
@Service("myMovieLister")
public class SimpleMovieLister {
    // ...
}
```

만약 name 값을 지정하지 않으면, class 이름의 첫문자를 소문자로 변환하여 Bean 이름을 자동으로 생성한다. 아래 예제의 Bean 이름은 “movieFinderImpl”이다.

```
@Repository
public class MovieFinderImpl implements MovieFinder {
    // ...
}
```

□ Annotation 기반 설정(12/12)

– Auto-detecting components(3/3)

- Scope 설정

@Scope annotation을 사용하여, 자동으로 찾은 Bean의 scope를 설정할 수 있다.

```
@Scope("prototype")
@Repository
public class MovieFinderImpl implements MovieFinder {
    // ...
}
```

- Qualifier 설정

@Qualifier annotation을 사용하여, 자동으로 찾은 Bean의 qualifier를 설정할 수 있다.

```
@Component
@Qualifier("Action")
public class ActionMovieCatalog implements MovieCatalog {
    // ...
}
```


❑ ApplicationContext for web application(1/2)

- Spring은 Web Application에서 ApplicationContext를 쉽게 사용할 수 있도록 각종 class들을 제공하고 있다.
- Servlet 2.4 이상
 - **Servlet 2.4 specification**부터 **Listener**를 사용할 수 있다. **Listener**를 사용하여 **ApplicationContext**를 설정하기 위해서 **web.xml** 파일에 다음 설정을 추가한다.

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/daoContext.xml /WEB-INF/applicationContext.xml</param-value>
</context-param>

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

contextParam의 '**contextConfigLocation**' 값은 **Spring XML** 설정 파일의 위치를 나타낸다.

❑ ApplicationContext for web application(2/2)

– Servlet 2.3 이하

- Servlet 2.3이하에서는 Listener를 사용할 수 없으므로, Servlet를 사용한다. web.xml 파일에 다음 설정을 추가한다.

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/daoContext.xml /WEB-INF/applicationContext.xml</param-value>
</context-param>

<servlet>
  <servlet-name>context</servlet-name>
  <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

LAB 201-IoC 실습(2)

❑ The Spring Framework - Reference Documentation / Chapter 5. The IoC container

- <http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html>
- 이전 버전 참조
 - <http://static.springframework.org/spring/docs/2.5.x/reference/beans.html>
 - <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html>

❑ Inversion of Control

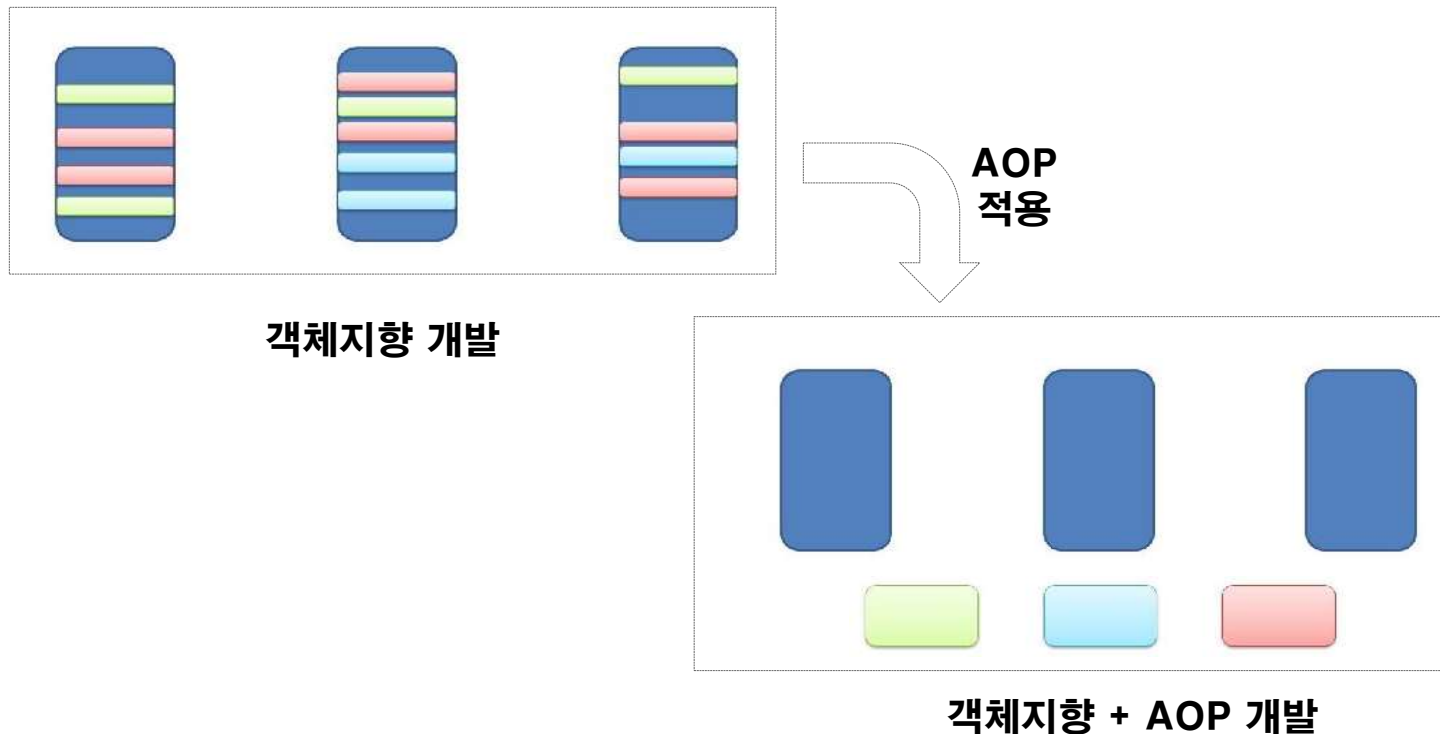
- <http://martinfowler.com/bliki/InversionOfControl.html>

❑ Inversion of Control Containers and the Dependency Injection pattern

- <http://martinfowler.com/articles/injection.html>

□ 서비스 개요

- 객체지향 프로그래밍(Object Oriented Programming)을 보완하는 개념으로 어플리케이션을 객체지향적으로 모듈화 하여 작성하더라도 다수의 객체들에 분산되어 중복적으로 존재하는 공통 관심사가 여전히 존재한다. AOP는 이를 횡단관심으로 분리하여 핵심관심과 엮어서 처리할 수 있는 방법을 제공한다.
- 로깅, 보안, 트랜잭션 등의 공통적인 기능의 활용을 기존의 비즈니스 로직에 영향을 주지 않고 모듈화 처리를 지원하는 프로그래밍 기법



□ 주요 개념

– Join Point

- 횡단 관심(Crosscutting Concerns) 모듈이 삽입되어 동작할 수 있는 실행 가능한 특정 위치를 말함
- 메소드 호출, 메소드 실행 자체, 클래스 초기화, 객체 생성 시점 등

– Pointcut

- Pointcut은 어떤 클래스의 어느 JoinPoint를 사용할 것인지를 결정하는 선택 기능을 말함
- 가장 일반적인 Pointcut은 '특정 클래스에 있는 모든 메소드 호출'로 구성된다.

– 애스펙트(Aspect)

- Advice와 Pointcut의 조합
- 어플리케이션이 가지고 있어야 할 로직과 그것을 실행해야 하는 지점을 정의한 것

– Advice

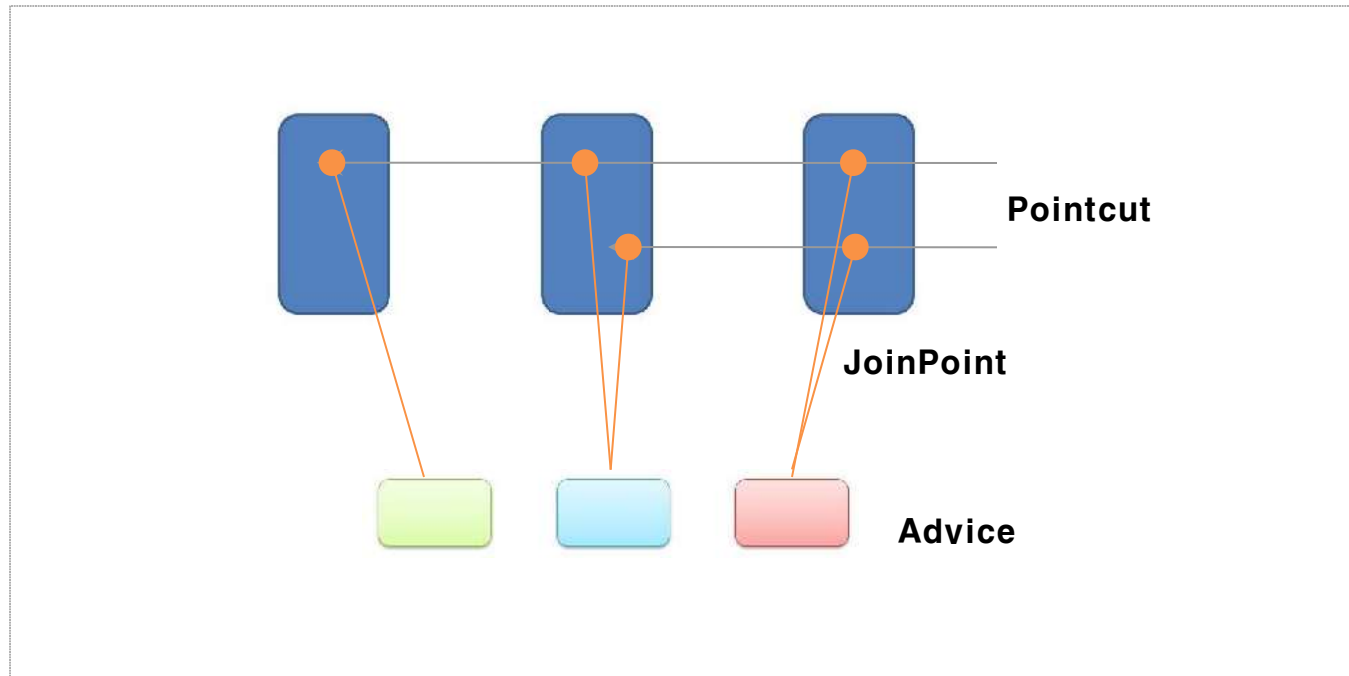
- Advice는 관점(Aspect)의 실제 구현체로 결합점에 삽입되어 동작할 수 있는 코드이다
- Advice 는 결합점(JoinPoint)과 결합하여 동작하는 시점에 따라 before advice, after advice, around advice 타입으로 구분된다
- 특정 Join point에 실행하는 코드

– Weaving

- Pointcut에 의해서 결정된 JoinPoint에 지정된 Advice를 삽입하는 과정
- Weaving은 AOP가 기존의 Core Concerns 모듈의 코드에 전혀 영향을 주지 않으면서 필요한 Crosscutting Concerns 기능을 추가할 수 있게 해 주는 핵심적인 처리 과정임

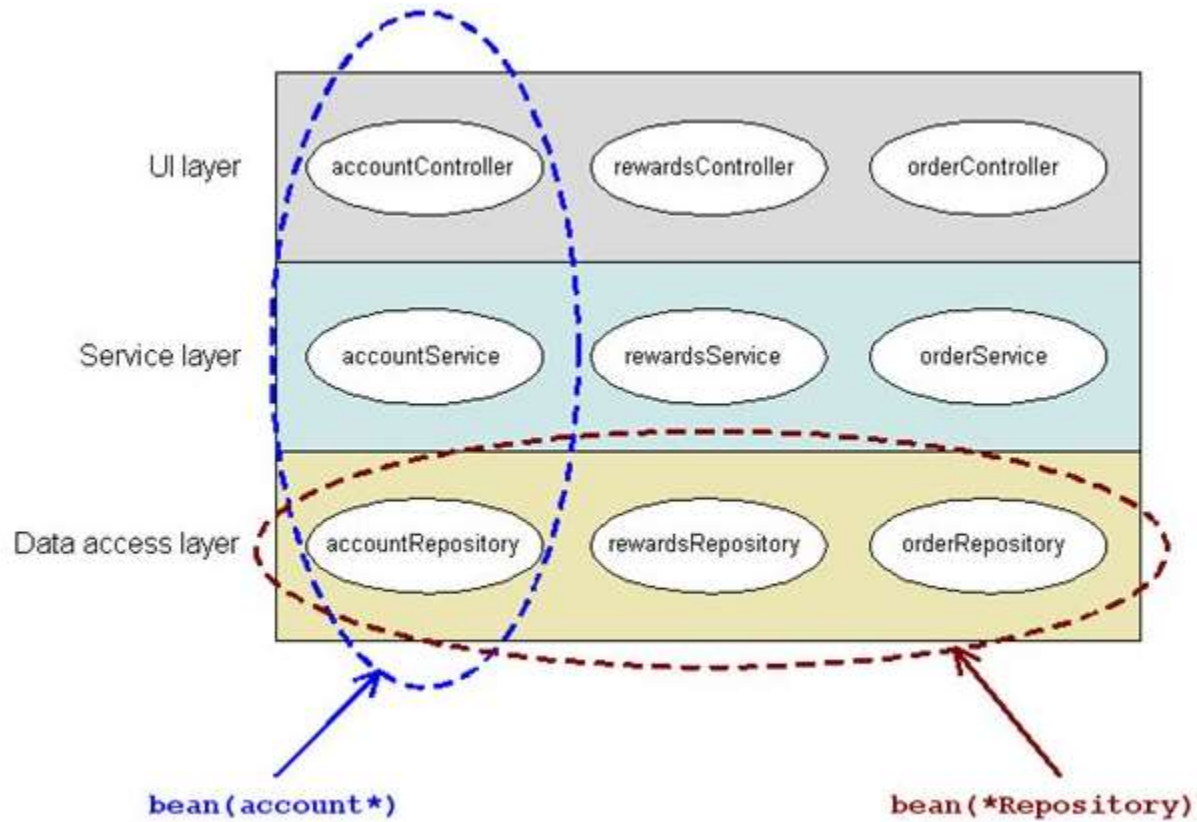
□ 주요 개념

- Advice, JoinPoint, Pointcut



□ 주요 개념

- Pointcut 예제 - bean() Pointcut을 이용하여 종적 및 횡적으로 빈을 선택



□ 주요 개념

– Weaving 방식

- 컴파일 시 엮기: 별도 컴파일러를 통해 핵심 관심사 모듈의 사이 사이에 관점(Aspect) 형태로 만들어진 횡단 관심사 코드들이 삽입되어 관점(Aspect)이 적용된 최종 바이너리가 만들어지는 방식이다. (ex. AspectJ, ...)
- 클래스 로딩 시 엮기: 별도의 Agent를 이용하여 JVM이 클래스를 로딩할 때 해당 클래스의 바이너리 정보를 변경한다. 즉, Agent가 횡단 관심사 코드가 삽입된 바이너리 코드를 제공함으로써 AOP를 지원하게 된다. (ex. AspectWerkz, ...)
- 런타임 엮기: 소스 코드나 바이너리 파일의 변경없이 프록시를 이용하여 AOP를 지원하는 방식이다. 프록시를 통해 핵심 관심사를 구현한 객체에 접근하게 되는데, 프록시는 핵심 관심사 실행 전후에 횡단 관심사를 실행한다. 따라서 프록시 기반의 런타임 엮기의 경우 메소드 호출시에만 AOP를 적용할 수 있다는 제한점이 있다. (ex. Spring AOP, ...)

– Advice 결합점 결합 타입

- Before advice: joinpoint 전에 수행되는 advice
- After returning advice: joinpoint가 성공적으로 리턴된 후에 동작하는 advice
- After throwing advice: 예외가 발생하여 joinpoint가 빠져나갈때 수행되는 advice
- After (finally) advice: join point를 빠져나가는(정상적이거나 예외적인 반환) 방법에 상관없이 수행되는 advice
- Around advice: joinpoint 전, 후에 수행되는 advice

□ 주요 기능

- 횡단 관심(CrossCutting Concern) 모듈이 삽입되어 동작할 수 있도록 지정하는 **JoinPoint** 기능
- 횡단 관심 모듈을 특정 **JoinPoint**에 사용할 수 있도록 지정하는 **Pointcut** 기능
- **Pointcut** 지정을 위한 패턴 매칭 표현식
- **Pointcut**에서 수행해야하는 동작을 지정하는 **Advice** 기능
- **Pointcut**에 의해서 결정된 **JoinPoint**에 지정된 **Advice**를 삽입하여 실제 **AOP** 방식으로 동작

□ 장점

- 중복 코드의 제거
 - 횡단 관심(CrossCutting Concerns)을 여러 모듈에 반복적으로 기술되는 현상을 방지
- 비즈니스 로직의 가독성 향상
 - 핵심기능 코드로부터 횡단 관심 코드를 분리함으로써 비즈니스 로직의 가독성 향상
- 생산성 향상
 - 비즈니스 로직의 독립으로 인한 개발의 집중력을 높임
- 재사용성 향상
 - 횡단 관심 코드는 여러 모듈에서 재사용될 수 있음
- 변경 용이성 증대
 - 횡단 관심 코드가 하나의 모듈로 관리되기 때문에 이에 대한 변경 발생시 용이하게 수행할 수 있음

□ Spring의 AOP 지원

- 스프링은 프록시 기반의 런타임 Weaving 방식을 지원한다.
- 스프링은 AOP 구현을 위해 다음 세가지 방식을 제공한다.
 - @AspectJ 어노테이션을 이용한 AOP 구현
 - XML Schema를 이용한 AOP 구현
 - 스프링 API를 이용한 AOP 구현
- 실행환경은 XML Schema를 이용한 AOP 구현 방법을 사용한다.

□ XML 스키마를 이용한 AOP 지원 (1/11)

– 개요

- Java 5 버전을 사용할 수 없거나, XML 기반 설정을 선호하는 경우 사용한다.
- AOP 선언을 한 눈에 파악할 수 있다.
- 실행환경은 XML 스키마를 이용한 AOP를 사용한다.

□ XML 스키마를 이용한 AOP 지원 (2/11)

- Aspect 정의하기

Advice 정의

```

<bean id="adviceUsingXML" class="egovframework.rte.fdl.aop.sample.AdviceUsingXML" />

<aop:config>
  <aop:pointcut id="targetMethod"
    expression="execution(* egovframework.rte.fdl.aop.sample.*Sample.*(..))" />
  <aop:aspect ref="adviceUsingXML">
    <aop:before pointcut-ref="targetMethod" method="beforeTargetMethod" />
    <aop:after-returning pointcut-ref="targetMethod"
      method="afterReturningTargetMethod" returning="retVal" />
    <aop:after-throwing pointcut-ref="targetMethod"
      method="afterThrowingTargetMethod" throwing="exception" />
    <aop:after pointcut-ref="targetMethod" method="afterTargetMethod" />
    <aop:around pointcut-ref="targetMethod" method="aroundTargetMethod" />
  </aop:aspect>
</aop:config>

<bean id="adviceSample" class="egovframework.rte.fdl.aop.sample.AdviceSample" />

```

Pointcut 정의

JoinPoint 정의

Aspect 정의

□ XML 스키마를 이용한 AOP 지원(3/11)

- Advice 정의하기 - before advice

```
public class AdviceUsingXML {  
  
    public void beforeTargetMethod(JoinPoint thisJoinPoint) {  
        Class clazz = thisJoinPoint.getTarget().getClass();  
        String className = thisJoinPoint.getTarget().getClass().getSimpleName();  
        String methodName = thisJoinPoint.getSignature().getName();  
  
        // 대상 메서드에 대한 로거를 얻어 해당 로거로 현재 class, method 정보 로깅  
        Log logger = LogFactory.getLog(clazz);  
        logger.debug(className + "." + methodName + " executed.");  
  
    }  
    ...  
}
```

□ XML 스키마를 이용한 AOP 지원(4/11)

- Advice 정의하기 – After returning advice
 - After returning advice는 정상적으로 메소드가 실행될 때 수행된다.

```
public class AdviceUsingXML {  
  
    public void afterReturningTargetMethod(JoinPoint thisJoinPoint,  
        Object retVal) {  
        System.out.println("AspectUsingAnnotation.afterReturningTargetMethod executed." +  
            " return value is [" + retVal + "]");  
    }  
    ...  
}
```


□ XML 스키마를 이용한 AOP 지원(5/11)

– Advice 정의하기 – After throwing advice

- After throwing advice 는 메소드가 수행 중 예외사항을 반환하고 종료하는 경우 수행된다.

```
public class AdviceUsingXML {  
    ...  
    public void afterThrowingTargetMethod(JoinPoint thisJoinPoint,  
        Exception exception) throws Exception{  
        System.out.println("AdviceUsingXML.afterThrowingTargetMethod executed.");  
        System.err.println("에러가 발생했습니다.", exception);  
        throw new BizException("에러가 발생했습니다.", exception);  
    }  
    ...  
}
```

□ XML 스키마를 이용한 AOP 지원(6/11)

– Advice 정의하기 – After (finally) advice

- After (finally) advice 는 메소드 수행 후 무조건 수행된다.
- After advice 는 정상 종료와 예외 발생 경우를 모두 처리해야 하는 경우에 사용된다 (예:리소스 해제 작업)

```
public class AdviceUsingXML {  
  
    public void afterTargetMethod(JoinPoint thisJoinPoint) {  
        System.out.println("AspectUsingAnnotation.afterTargetMethod executed.");  
    }  
    ...  
}
```

□ XML 스키마를 이용한 AOP 지원(7/11)

– Advice 정의하기 – Around advice

- Around advice 는 메소드 수행 전후에 수행된다.
- Return값을 가공하기 위해서는 Around를 사용해야한다.

```
public class AdviceUsingXML {  
  
    public Object aroundTargetMethod(ProceedingJoinPoint thisJoinPoint)  
        throws Throwable {  
        System.out.println("AspectUsingAnnotation.aroundTargetMethod start.");  
        long time1 = System.currentTimeMillis();  
        Object retVal = thisJoinPoint.proceed();  
  
        System.out.println("ProceedingJoinPoint executed. return value is [" + retVal + "]);  
  
        retVal = retVal + "(modified)";  
        System.out.println("return value modified to [" + retVal + "]);  
  
        long time2 = System.currentTimeMillis();  
        System.out.println("AspectUsingAnnotation.aroundTargetMethod end. Time(" +  
            + (time2 - time1) + ")");  
        return retVal;  
    }  
    ...  
}
```

□ XML 스키마를 이용한 AOP 지원(8/11)

- Aspect 실행하기 - 정상 실행의 경우

```
public class AnnotationAspectTest {  
  
    @Resource(name = "adviceSample")  
    AdviceSample adviceSample;  
  
    @Test  
    public void testAdvice () throws Exception {  
  
        SampleVO vo = new SampleVO();  
        ..  
        String resultStr = annotationAdviceSample.someMethod(vo);  
  
        assertEquals("someMethod executed.(modified)", resultStr);  
  
    }  
}
```

□ XML 스키마를 이용한 AOP 지원(9/11)

- Aspect 실행하기 - 정상 실행인 경우
 - 콘솔 로그 출력 Advice 적용 순서
 - 1.before
 - 2.around (대상 메소드 수행 전)
 - 3.대상 메소드
 - 4.after-returning
 - 5.after(finally)
 - 6.around (대상 메소드 수행 후)

□ XML 스키마를 이용한 AOP 지원(10/11)

- Aspect 실행하기 - 예외 발생의 경우

```
public class AnnotationAspectTest {

    @Resource(name = "adviceSample")
    AdviceSample adviceSample;

    @Test
    public void testAdviceWithException() throws Exception {

        SampleVO vo = new SampleVO();
        // exception 을 발생하도록 플래그 설정
        vo.setForceException(true);
        ..
        try {
            String resultStr = annotationAdviceSample.someMethod(vo);

            fail("exception을 강제로 발생시켜 이 라인이 수행될 수 없습니다.");
        } catch (Exception e) {
            ...
        }
    }
}
```

□ XML 스키마를 이용한 AOP 지원(11/11)

- Aspect 실행하기 - 예외 발생의 경우
 - 콘솔 로그 출력 Advice 적용 순서
 - 1.before
 - 2.around (대상 메소드 수행 전)
 - 3.대상 메소드 (ArithmeticException 예외가 발생한다)
 - 4.afterThrowing
 - 5.after(finally)

□ Pointcut 지정자

- execution: 메소드 실행 결합점(join points)과 일치시키는데 사용된다.
- within: 특정 타입에 속하는 결합점을 정의한다.
- this: 빈 참조가 주어진 타입의 인스턴스를 갖는 결합점을 정의한다.
- target: 대상 객체가 주어진 타입을 갖는 결합점을 정의한다.
- args: 인자가 주어진 타입의 인스턴스인 결합점을 정의한다.
- @target: 수행중인 객체의 클래스가 주어진 타입의 어노테이션을 갖는 결합점을 정의한다.
- @args: 전달된 인자의 런타임 타입이 주어진 타입의 어노테이션을 갖는 결합점을 정의한다.
- @within: 주어진 어노테이션을 갖는 타입 내 결합점을 정의한다.
- @annotation: 결합점의 대상 객체가 주어진 어노테이션을 갖는 결합점을 정의한다.

□ Pointcut 표현식 조합

- '&&' : anyPublicOperation() && inTrading()
- '||' : bean(*dataSource) || bean(*DataSource)
- '!' : !bean(accountRepository)

□ Pointcut 정의 예제

Pointcut	선택된 Joinpoints
execution(public **(..))	public 메소드 실행
execution(* set*(..))	이름이 set으로 시작하는 모든 메소드명 실행
execution(* com.xyz.service.AccountService.*(..))	AccountService 인터페이스의 모든 메소드 실행
execution(* com.xyz.service.*.*(..))	service 패키지의 모든 메소드 실행
execution(* com.xyz.service..*.*(..))	service 패키지과 하위 패키지의 모든 메소드 실행
within(com.xyz.service.*)	service 패키지 내의 모든 결합점
within(com.xyz.service..*)	service 패키지 및 하위 패키지의 모든 결합점
this(com.xyz.service.AccountService)	AccountService 인터페이스를 구현하는 프록시 개체의 모든 결합점
target(com.xyz.service.AccountService)	AccountService 인터페이스를 구현하는 대상 객체의 모든 결합점
args(java.io.Serializable)	하나의 파라미터를 갖고 전달된 인자가 Serializable인 모든 결합점
@target(org.springframework.transaction.annotation.Transactional)	대상 객체가 @Transactional 어노테이션을 갖는 모든 결합점
@within(org.springframework.transaction.annotation.Transactional)	대상 객체의 선언 타입이 @Transactional 어노테이션을 갖는 모든 결합점
@annotation(org.springframework.transaction.annotation.Transactional)	실행 메소드가 @Transactional 어노테이션을 갖는 모든 결합점
@args(com.xyz.security.Classified)	단일 파라미터를 받고, 전달된 인자 타입이 @Classified 어노테이션을 갖는 모든 결합점
bean(accountRepository)	“accountRepository” 빈
!bean(accountRepository)	“accountRepository” 빈을 제외한 모든 빈
bean(*)	모든 빈
bean(account*)	이름이 'account'로 시작되는 모든 빈
bean(*Repository)	이름이 “Repository”로 끝나는 모든 빈
bean(accounting/*)	이름이 “accounting/”로 시작하는 모든 빈
bean(*dataSource) bean(*DataSource)	이름이 “dataSource” 나 “DataSource” 으로 끝나는 모든 빈

□ 실행환경 AOP 가이드라인

- 실행환경은 예외 처리와 트랜잭션 처리에 AOP를 적용함

□ 실행환경 AOP 가이드라인- 예외 처리(1/2)

- 관점(Aspect) 정의: resources/egovframework.spring/context-aspect.xml

Advice
정의

```
<bean id="exceptionTransfer" class="egovframework.rte.fdl.cmmn.aspect.ExceptionTransfer">
...
</bean>

<aop:config>
  <aop:pointcut id="serviceMethod"
    expression="execution(* egovframework.rte.sample..impl.*Impl.*(..))" />
  <aop:aspect ref="exceptionTransfer">
    <aop:after-throwing throwing="exception"
      pointcut-ref="serviceMethod" method="transfer" />
  </aop:aspect>
</aop:config>
```

Pointcut 정의

JoinPoint
정의

Aspect 정의

□ 실행환경 AOP 가이드라인- 예외 처리(2/2)

- Advice 클래스 : egovframework.rte.fdl.cmmn.aspect.ExceptionTransfer

```
public class ExceptionTransfer {

    public void transfer(JoinPoint thisJoinPoint, Exception exception) throws Exception {
        ...
        if (exception instanceof EgovBizException) {
            log.debug("Exception case :: EgovBizException ");
            EgovBizException be = (EgovBizException) exception;
            getLog(clazz).error(be.getMessage(), be.getCause());
            processHandling(clazz, exception, pm, exceptionHandlerServices, false);
            throw be;
        } else if (exception instanceof RuntimeException) {
            log.debug("RuntimeException case :: RuntimeException ");
            RuntimeException be = (RuntimeException) exception;
            getLog(clazz).error(be.getMessage(), be.getCause());
            processHandling(clazz, exception, pm, exceptionHandlerServices, true);
            ...
            throw be;
        } else if (exception instanceof FdlException) {
            ...
            throw fe;
        } else {
            ...
            throw processException(clazz, "fail.common.msg", new String[] {}, exception, locale);
        }
    }
}
```

□ 실행환경 AOP 가이드라인- 트랜잭션 처리

- 관점(Aspect) 정의: resources/egovframework.spring/context-transaction.xml

```
<!-- 트랜잭션 관리자를 설정한다. -->
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>

<!-- 트랜잭션 Advice를 설정한다. -->
<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
    <tx:method name="*" rollback-for="Exception"/>
  </tx:attributes>
</tx:advice>

<!-- 트랜잭션 Pointcut를 설정한다.--->
<aop:config>
  <aop:pointcut id="requiredTx"
    expression="execution(* egovframework.rte.sample..impl.*Impl.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="requiredTx" />
</aop:config>
```

❑ Spring 3.2 Reference Documentation

- <http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/aop.html>
- 이전 버전 참조
 - <http://static.springframework.org/spring/docs/2.5.x/reference/aop.html>
 - <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/aop.html>

❑ The new() Pointcut

- <http://blog.springsource.com/2007/09/24/the-new-bean-pointcut/>

LAB 202-AOP 실습

□ @AspectJ 어노테이션을 이용한 AOP 지원 (1/11)

– 개요

- @AspectJ는 Java 5 어노테이션을 사용하여 일반 Java 클래스로 관점(Aspect)을 정의하는 방식이다.
- @AspectJ 방식은 AspectJ 5 버전에서 소개되었으며, Spring은 2.0 버전부터 AspectJ 5 어노테이션을 지원한다.
- Spring AOP 실행 환경은 AspectJ 컴파일러나 Weaver에 대한 의존성이 없이 @AspectJ 어노테이션을 지원한다.

□ @AspectJ 어노테이션을 이용한 AOP 지원 (2/11)

- @AspectJ 설정하기

```
<aop:aspectj-autoproxy/>
```

- Aspect 정의하기

```
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class AspectUsingAnnotation {
    ..
}
```

- Pointcut 정의하기

```
@Aspect
public class AspectUsingAnnotation {
    ...
    @Pointcut("execution(public * egovframework.rte.fdl.aop.sample.*Sample.*(..))")
    public void targetMethod() {
        // pointcut annotation 값을 참조하기 위한 dummy method
    }
    ...
}
```


□ @AspectJ 어노테이션을 이용한 AOP 지원 (3/11)

- Advice 정의하기 – before advice
 - Before advice는 @Before 어노테이션을 사용한다.

```
@Aspect
public class AspectUsingAnnotation {

    @Before("targetMethod()")
    public void beforeTargetMethod(JoinPoint thisJoinPoint) {
        Class clazz = thisJoinPoint.getTarget().getClass();
        String className = thisJoinPoint.getTarget().getClass().getSimpleName();
        String methodName = thisJoinPoint.getSignature().getName();

        // 대상 메서드에 대한 로거를 얻어 해당 로거로 현재 class, method 정보 로깅
        Log logger = LogFactory.getLog(clazz);
        logger.debug(className + "." + methodName + " executed.");

    }
    ...
}
```

□ @AspectJ 어노테이션을 이용한 AOP 지원 (4/11)

- Advice 정의하기 – After returning advice
 - After returning advice 는 정상적으로 메소드가 실행될 때 수행된다.
 - After returning advice 는 @AfterReturning 어노테이션을 사용한다.

```
@Aspect
public class AspectUsingAnnotation {

    @AfterReturning(pointcut = "targetMethod()", returning = "retVal")
    public void afterReturningTargetMethod(JoinPoint thisJoinPoint,
        Object retVal) {
        System.out.println("AspectUsingAnnotation.afterReturningTargetMethod executed." +
            " return value is [" + retVal + "]);
    }
    ...
}
```

□ @AspectJ 어노테이션을 이용한 AOP 지원 (5/11)

– Advice 정의하기 – After throwing advice

- After throwing advice 는 메소드가 수행 중 예외사항을 반환하고 종료하는 경우 수행된다.
- After throwing advice 는 @AfterThrowing 어노테이션을 사용한다.

```
@Aspect
public class AspectUsingAnnotation {

    @AfterThrowing(pointcut = "targetMethod()", throwing = "exception")
    public void afterThrowingTargetMethod(JoinPoint thisJoinPoint,
        Exception exception) throws Exception {
        System.out.println("AspectUsingAnnotation.afterThrowingTargetMethod executed.");
        System.out.println("에러가 발생했습니다.", exception);

        throw new BizException("에러가 발생했습니다.", exception);
    }
    ...
}
```

□ @AspectJ 어노테이션을 이용한 AOP 지원 (6/11)

- Advice 정의하기 – After (finally) advice
 - After (finally) advice 는 메소드 수행 후 무조건 수행된다.
 - After advice 는 정상 종료와 예외 발생 경우를 모두 처리해야 하는 경우에 사용된다 (예:리소스 해제 작업)
 - After (finally) advice 는 @After 어노테이션을 사용한다.

```
@Aspect
public class AspectUsingAnnotation {

    @After("targetMethod()")
    public void afterTargetMethod(JoinPoint thisJoinPoint) {
        System.out.println("AspectUsingAnnotation.afterTargetMethod executed.");
    }
    ...
}
```

□ @AspectJ 어노테이션을 이용한 AOP 지원 (7/11)

- Advice 정의하기 – Around advice
 - Around advice 는 메소드 수행 전후에 수행된다.
 - Around advice 는 @Around 어노테이션을 사용한다.

```
@Aspect
public class AspectUsingAnnotation {

    @Around("targetMethod()")
    public Object aroundTargetMethod(ProceedingJoinPoint thisJoinPoint)
        throws Throwable {
        System.out.println("AspectUsingAnnotation.aroundTargetMethod start.");
        long time1 = System.currentTimeMillis();
        Object retVal = thisJoinPoint.proceed();

        System.out.println("ProceedingJoinPoint executed. return value is [" + retVal + "]);

        retVal = retVal + "(modified)";
        System.out.println("return value modified to [" + retVal + "]);

        long time2 = System.currentTimeMillis();
        System.out.println("AspectUsingAnnotation.aroundTargetMethod end. Time("
            + (time2 - time1) + ")");
        return retVal;
    }...
}
```

□ @AspectJ 어노테이션을 이용한 AOP 지원 (8/11)

- Aspect 실행하기 – 정상 실행인 경우

```
public class AnnotationAspectTest {  
  
    @Resource(name = "annotationAdviceSample")  
    AnnotationAdviceSample annotationAdviceSample;  
  
    @Test  
    public void testAnnotationAspect() throws Exception {  
  
        SampleVO vo = new SampleVO();  
        ..  
        String resultStr = annotationAdviceSample.someMethod(vo);  
  
        assertEquals("someMethod executed.(modified)", resultStr);  
    }  
}
```

□ @AspectJ 어노테이션을 이용한 AOP 지원 (9/11)

- Aspect 실행하기 – 정상 실행인 경우
 - 콘솔 로그 출력 Advice 적용 순서
 1. @Before
 2. @Around (대상 메소드 수행 전)
 3. 대상 메소드
 4. @Around (대상 메소드 수행 후)
 5. @After(finally)
 6. @AfterReturning

□ @AspectJ 어노테이션을 이용한 AOP 지원 (10/11)

- Aspect 실행하기 – 예외 발생의 경우

```
public class AnnotationAspectTest {

    @Resource(name = "annotationAdviceSample")
    AnnotationAdviceSample annotationAdviceSample;

    @Test
    public void testAnnotationAspect() throws Exception {

        SampleVO vo = new SampleVO();
        // exception 을 발생하도록 플래그 설정
        vo.setForceException(true);
        ..
        try {
            String resultStr = annotationAdviceSample.someMethod(vo);

            fail("exception을 강제로 발생시켜 이 라인이 수행될 수 없습니다.");
        } catch (Exception e) {
            ...
        }
    }
}
```


□ @AspectJ 어노테이션을 이용한 AOP 지원 (11/11)

- Aspect 실행하기 – 예외 발생의 경우
 - 콘솔 로그 출력 Advice 적용 순서
 1. @Before
 2. @Around (대상 메소드 수행 전)
 3. 대상 메소드 (ArithmeticException 예외가 발생한다)
 4. @After(finally)
 5. @AfterThrowing

LAB 201-IOC 실습

실습 개요

❑ 실습을 통해 IOC에 대하여 살펴본다.

❑ 실습 순서

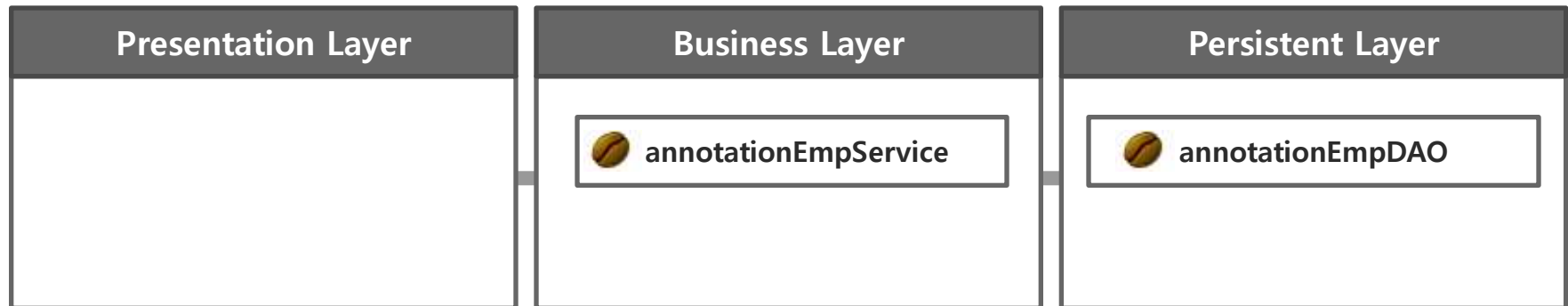
1. XML 설정 방식의 Spring Bean 서비스 작성
2. Annotation 설정 방식의 Spring Bean 서비스 작성

실습 개요

❑ XML 설정방식에서 설정할 beans



❑ Annotation 설정방식에서 설정할 beans



1. XML 설정 방식의 Spring bean 서비스 작성

❑ 1. Interface 작성

- /lab201-ioc/src/main/java/egovframework/lab/ioc/service/EmpService.java 를 작성한다.

```
public interface EmpService {  
  
    public void insertEmp(EmpVO empVO) throws Exception;  
  
    public void updateEmp(EmpVO empVO) throws Exception;  
  
    public void deleteEmp(EmpVO empVO) throws Exception;  
  
    public EmpVO selectEmp(EmpVO empVO) throws Exception;  
  
    public List<EmpVO> selectEmpList() throws Exception;  
  
}
```

- Ctrl + Shift + O (source > Organize Imports) 를 수행하여 자동 import 를 수행한다
cf.) 아직 EmpVO 를 작성하지 않아 컴파일 에러 상태일 것이다.

1. XML 설정 방식의 Spring bean 서비스 작성

□ 2. VO 작성(1/2)

(현재 실습과정에서 사용하는 DAO 에서 DB 구현 없이 자바 class 영역에 데이터를 임시 관리할 때 데이터 비교를 위해 Comparable 구현 (Generic 스타일로))

- /lab201-ioc/src/main/java/egovframework/lab/ioc/service/EmpVO.java 를 작성한다.

```
public class EmpVO implements Serializable {  
  
    private int empNo;  
  
    private String empName;  
  
    private String job;  
}
```

- Getter, setter 생성하기 : 마우스 우클릭 > Source > Generate Getters and Setters 를 실행하여 Select All 한 다음 OK 실행
- Serializable을 implements함. -> Ctrl + Shift + O (자동 import)
- EmpVO 의 마커바 상에서 quick fix 로 제공되는 기능 중 Add generated serial version ID 추가하기를 권고함.

1. XML 설정 방식의 Spring bean 서비스 작성

□ 2. VO 작성(2/2)

- Comparable 를 implements 토록 추가 - 여기서는 generic 스타일로 Comparable<EmpVO> 로 한정
- compareTo 메서드 추가 - 여기서는 EmpVO 의 empNo 속성의 크기를 비교하여 판단토록 하였음.

```
public class EmpVO implements Serializable, Comparable<EmpVO> {  
  
    . . .  
    public int compareTo(EmpVO o) {  
        if (empNo > o.getEmpNo()) {  
            return 1;  
        } else if (empNo < o.getEmpNo()) {  
            return -1;  
        } else {  
            return 0;  
        }  
    }  
}
```

- EmpService 소스에서 Ctrl+Shift+O 하여 EmpVO 자동 import

1. XML 설정 방식의 Spring bean 서비스 작성

❑ 3. Service Impl작성 (1/2)

- /lab201-ioc/src/main/java/egovframework/lab/ioc/service/impl/XmlEmpServiceImpl.java 를 작성한다.

```
public class XmlEmpServiceImpl implements EmpService {

    private XmlEmpDAO empDAO;

    public void setEmpDAO(XmlEmpDAO empDAO) {
        this.empDAO = empDAO;
    }

    public void insertEmp(EmpVO empVO) throws Exception {
        empDAO.insertEmp(empVO);
    }

    public void updateEmp(EmpVO empVO) throws Exception {
        empDAO.updateEmp(empVO);
    }

    public void deleteEmp(EmpVO empVO) throws Exception {
        empDAO.deleteEmp(empVO);
    }

    public EmpVO selectEmp(EmpVO empVO) throws Exception {
        return empDAO.selectEmp(empVO);
    }

    public List<EmpVO> selectEmpList() throws Exception {
        return empDAO.selectEmpList();
    }

}
```


1. XML 설정 방식의 Spring bean 서비스 작성

❑ 3. Service Impl작성 (2/2)

- 위에서 dependency 객체로 XmlEmpDAO 를 setEmpDAO 메서드를 통해 Container 로부터 주입받아 동작하게 되며 EmpService 자체에 복잡한 비즈니스 로직이 필요치 않은 경우로 DAO 에 단순 CRUD 기능을 위임해 처리하고 있음을 확인할 수 있음
- 목록조회 메서드에서 확인할 수 있듯이 JDK 1.5 이상의 Generic 스타일로 구현하는 것을 권고함

1. XML 설정 방식의 Spring bean 서비스 작성

□ 4. DAO 작성(1/3)

- /lab201-ioc/src/main/java/egovframework/lab/ioc/service/impl/XmlEmpDAO.java 를 작성한다.

```
public class XmlEmpDAO {

    static List<EmpVO> list;

    static {
        list = new ArrayList<EmpVO>();
        EmpVO empVO;
        for (int i = 1; i <= 100; i++) {
            empVO = new EmpVO();
            empVO.setEmpNo(i);
            empVO.setEmpName("EmpName " + i);
            empVO.setJob("SALESMAN");
            list.add(empVO);
        }
    }

    public void insertEmp(EmpVO empVO) throws Exception {
        list.add(empVO);
        Collections.sort(list);
    }

    . . . (계속)

}
```

1. XML 설정 방식의 Spring bean 서비스 작성

□ 4. DAO 작성(2/3)

```
. . .(이어서)
public void updateEmp(EmpVO empVO) throws Exception {
    int index = Collections.binarySearch(list, empVO);
    // 해당 데이터가 없는 경우 여기서는 ArrayIndexOutOfBoundsException 발생할 것임
    EmpVO orgEmpVO = list.get(index);

    orgEmpVO.setEmpName(empVO.getEmpName());
    orgEmpVO.setJob(empVO.getJob());
}

public void deleteEmp(EmpVO empVO) throws Exception {
    list.remove(Collections.binarySearch(list, empVO));
    Collections.sort(list);
}

public EmpVO selectEmp(EmpVO empVO) throws Exception {
    int index = Collections.binarySearch(list, empVO);

    // list search 결과 해당값을 찾을 수 없으면 음수값
    // (-(insertion point) - 1) 이 되돌려짐
    return index < 0 ? null : list.get(index);
}

public List<EmpVO> selectEmpList() throws Exception {
    return list;
}
}
```

1. XML 설정 방식의 Spring bean 서비스 작성

□ 4. DAO 작성(3/3)

- 현 실습과정의 위 DAO에서는 DB 연동/구현 없이 static 영역에 100 개의 EmpVO 에 대한 리스트를 생성해 두고 insert/update/delete 시에 static 하게 관리하고 있는 데이터에 대해 추가/변경/삭제가 일어나도록 간략히 구현한 예임.
- DB 가 아니므로 duplicated key 체크 등 번잡한 기능은 고려치 않았고, 데이터의 변경시에는 항상 sorting 을 새로 하여 select 시 binarySearch 로 빨리 찾을 수 있도록 하였음(EmpVO 는 Comparable 을 구현하였음).
- 목록조회는 검색조건 없이 전체 데이터를 return 하는 것으로 작성하였음.

1. XML 설정 방식의 Spring bean 서비스 작성

□ 5. XML 설정 파일 작성

- /lab201-ioc/src/test/resources/META-INF/spring/context-emp.xml 를 작성한다.

```
<!-- xml 형식 bean 정의 -->

<bean id="xmlEmpService"
class="egovframework.lab.ioc.service.impl.XmlEmpServiceImpl">
<property name="empDAO" ref="xmlEmpDAO" />
</bean>

<bean id="xmlEmpDAO" class="egovframework.lab.ioc.service.impl.XmlEmpDAO" />
```

- xmlEmpService 와 xmlEmpDAO 에 대한 bean 설정을 확인할 수 있으며 xmlEmpService 의 property 설정 요소(setter injection 방식) 로 xmlEmpDAO 를 연결하고 있음을 확인 가능.
- Spring IDE 기반의 bean 설정파일에 대한 다양한 code assist 가 지원되므로 대상 클래스에 Ctrl + 마우스 오버 --> 클릭시 대상 소스 열림 또는 class="" 속성, property name="" 속성 내에서 [일부typing] Ctrl + Space 등을 사용하여 자동 완성되는 코드를 사용하는 것이 오타 가능성을 줄일 수 있음.

1. XML 설정 방식의 Spring bean 서비스 작성

❑ 6. Testcase 작성(1/5)

- /lab201-ioc/src/test/java/egovframework/lab/ioc/service/EmpServiceTest.java 를 작성한다.

```
package egovframework.lab.ioc.service;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertTrue;

import java.util.List;

import javax.annotation.Resource;

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 = {
    "classpath*:META-INF/spring/context-common.xml",
    "classpath*:META-INF/spring/context-emp.xml"
    // , "classpath*:META-INF/spring/context-postprocessor.xml" // 이 주석을 풀고 테스트 시
    // annotationEmpService 에 대해서는 delete 메서드에 @Debug 를 설정하였으므로 trace 로그가 출력될 것임.
})
public class EmpServiceTest {

    . . . (계속)
```

1. XML 설정 방식의 Spring bean 서비스 작성

❑ 6. Testcase 작성(2/5)

```
... (이어서)
// xml 형식으로 bean 설정한 경우 - 주석을 변경해 가며 xml, annotation 에 대해 테스트 할것
@Resource(name = "xmlEmpService")
EmpService empService;

// annotation 형식으로 bean 설정한 경우
// @Resource(name = "annotationEmpService")
// EmpService empService;

public EmpVO makeVO() {
    // DAO 확인 - static 하게 관리하는 100 개 기본 데이터 있음
    return makeVO(101);
}

public EmpVO makeVO(int empNo) {
    EmpVO vo = new EmpVO();
    vo.setEmpNo(empNo);
    vo.setEmpName("홍길동" + empNo);
    vo.setJob("개발자");
    return vo;
}

public void checkResult(EmpVO vo, EmpVO resultVO) {
    assertNotNull(resultVO);
    assertEquals(vo.getEmpNo(), resultVO.getEmpNo());
    assertEquals(vo.getEmpName(), resultVO.getEmpName());
    assertEquals(vo.getJob(), resultVO.getJob());
}
... (계속)
```

1. XML 설정 방식의 Spring bean 서비스 작성

❑ 6. Testcase 작성(3/5)

```
... (이어서)
@Test
public void testInsertEmp() throws Exception {
    EmpVO vo = makeVO();

    // insert
    empService.insertEmp(vo);
    // select
    EmpVO resultVO = empService.selectEmp(vo);
    // check
    checkResult(vo, resultVO);
}

@Test
public void testUpdateEmp() throws Exception {
    EmpVO vo = makeVO(102);

    // insert
    empService.insertEmp(vo);
    // data change
    vo.setEmpName("홍길순");
    vo.setJob("설계자");

    // update
    empService.updateEmp(vo);
    // select
    EmpVO resultVO = empService.selectEmp(vo);
    // check
    checkResult(vo, resultVO);
}
... (계속)
```


1. XML 설정 방식의 Spring bean 서비스 작성

❑ 6. Testcase 작성(4/5)

```
... (이어서)
@Test
public void testDeleteEmp() throws Exception {
    EmpVO vo = makeVO(103);
    // insert
    empService.insertEmp(vo);

    // delete
    empService.deleteEmp(vo);
    // select
    EmpVO resultVO = empService.selectEmp(vo);
    // null 이어야 함
    assertNull(resultVO);
}

@Test
public void testSelectEmpList() throws Exception {

    // select list
    List<EmpVO> resultList = empService.selectEmpList();
    // check
    int firstListSize = resultList.size();
    // DAO 에서 Emp 데이터를 관리할 때 항상 sorted 된 상태임
    assertEquals(1, resultList.get(0).getEmpNo());

    // delete first data
    EmpVO empVO = new EmpVO();
    empVO.setEmpNo(1);
    ... (계속)
```

1. XML 설정 방식의 Spring bean 서비스 작성

❑ 6. Testcase 작성(5/5)

```
    . . . (이어서)

    empService.deleteEmp(empVO);

    // select List again
    resultList = empService.selectEmpList();

    assertEquals(firstListSize - 1, resultList.size());
    // DAO 에서 Emp 데이터를 관리할 때 항상 sorted 된 상태임
    assertEquals(2, resultList.get(0).getEmpNo());
    assertEquals("EmpName 2", resultList.get(0).getEmpName());
    assertEquals("SALESMAN", resultList.get(0).getJob());
}
}
```

- Spring 연동을 위해 제공하는 @RunWith(SpringJUnit4ClassRunner.class), @ContextConfiguration (...) 설정에 유의한다. 테스트에 필요한 Spring Bean 설정 파일만으로 제한하는 것이 바람직함.
- 테스트에 필요한 Spring Bean 들은 annotation 형태(여기서는 @Resource)로 injection 하여 사용한다.
- JUnit 4.4 의 Assert 관련 기능은 Ctrl+Shift+O 로 자동 import 되지 않음. static import 사용해야 함. --> 예러로 표시되는 asssertXX 사용 위치에 마우스 오버 하면 Add static imports ... 와 같은 quick fix 가 나타나 활용 가능함.

2. Annotation 설정 방식의 Spring bean 서비스 작성

❑ 1. 동일한 Interface

- EmpService

❑ 2. 동일한 VO

- EmpVO

❑ 3. Annotation 을 적용한 Impl (1/2)

- /lab201-ioc/src/main/java/egovframework/lab/ioc/service/impl/AnnotationEmpServiceImpl.java 를 작성한다.

... (계속)

2. Annotation 설정 방식의 Spring bean 서비스 작성

...(이어서)

```
@Service("annotationEmpService")
public class AnnotationEmpServiceImpl implements EmpService {

    @Resource(name = "annotationEmpDAO")
    private AnnotationEmpDAO empDAO;

    public void insertEmp(EmpVO empVO) throws Exception {
        empDAO.insertEmp(empVO);
    }

    public void updateEmp(EmpVO empVO) throws Exception {
        empDAO.updateEmp(empVO);
    }

    public void deleteEmp(EmpVO empVO) throws Exception {
        empDAO.deleteEmp(empVO);
    }

    public EmpVO selectEmp(EmpVO empVO) throws Exception {
        return empDAO.selectEmp(empVO);
    }

    public List<EmpVO> selectEmpList() throws Exception {
        return empDAO.selectEmpList();
    }
}
```

- @Service 스테레오 타입 Annotation 을 사용하여 bean 설정하였음.
- @Resource (JSR250 표준) Annotation 을 사용하여 Dependency Bean(여기서는 AnnotationEmpDAO) 를 injection 하였음.
- 기타 CRUD 관련 비즈니스 메서드는 동일함.

2. Annotation 설정 방식의 Spring bean 서비스 작성

❑ 4. Annotation을 적용한 DAO (1/3)

```
@Repository("annotationEmpDAO")
public class AnnotationEmpDAO {

    static List<EmpVO> list;

    static {
        list = new ArrayList<EmpVO>();
        EmpVO empVO;
        for (int i = 1; i <= 100; i++) {
            empVO = new EmpVO();
            empVO.setEmpNo(i);
            empVO.setEmpName("EmpName " + i);
            empVO.setJob("SALESMAN");
            list.add(empVO);
        }
    }

    public void insertEmp(EmpVO empVO) throws Exception {
        list.add(empVO);
        Collections.sort(list);
    }

    ... (계속)
```

2. Annotation 설정 방식의 Spring bean 서비스 작성

❑ 4. Annotation을 적용한 DAO(2/3)

```
... (이어서)
public void updateEmp(EmpVO empVO) throws Exception {
    int index = Collections.binarySearch(list, empVO);
    // 해당 데이터가 없는 경우 여기서는 ArrayIndexOutOfBoundsException 발생할 것임
    EmpVO orgEmpVO = list.get(index);

    orgEmpVO.setEmpName(empVO.getEmpName());
    orgEmpVO.setJob(empVO.getJob());
}

public void deleteEmp(EmpVO empVO) throws Exception {
    list.remove(Collections.binarySearch(list, empVO));
    Collections.sort(list);
}

public EmpVO selectEmp(EmpVO empVO) throws Exception {
    int index = Collections.binarySearch(list, empVO);

    // list search 결과 해당값을 찾을 수 없으면 음수값
    // (-(insertion point) - 1) 이 되돌려짐
    return index < 0 ? null : list.get(index);
}

public List<EmpVO> selectEmpList() throws Exception {
    return list;
}
}
```

2. Annotation 설정 방식의 Spring bean 서비스 작성

❑ 4. Annotation을 적용한 DAO(3/3)

- xml 설정 방식의 예와 마찬가지로 DB 연동 없이 테스트를 위한 static 한 내부 데이터를 관리하며 CRUD 하는 예임
- @Repository 스테레오 타입 Annotation 을 사용하여 bean 설정 하였음. (DAO 인 경우)

2. Annotation 설정 방식의 Spring bean 서비스 작성

❑ 5. common 설정 파일 - component scan

- /lab201-ioc/src/test/resources/META-INF/spring/context-common.xml 를 작성한다.

```
<!-- annotation 형식 bean 정의한 것에 대해 자동적으로 scan 하여 등록함. 여기서는  
AnnotationEmpServiceImpl, AnnotationEmpDAO -->  
<context:component-scan base-package="egovframework" />
```

❑ 6. Testcase 작성 (기존 Testcase 에서 DI 하는 서비스만 변경)

- /lab201-ioc/src/test/java/egovframework/lab/ioc/service/EmpServiceTest.java 를 작성한다. (이미 작성하였음.)

```
// annotation 형식으로 bean 설정한 경우  
@Resource(name = "annotationEmpService")  
EmpService empService;
```

- annotation 형식으로 설정한 annotationEmpService 를 테스트 대상 서비스로 사용토록 주석 변경하였음.

LAB 202-AOP 실습

실습 개요

❑ 실습을 통해 AOP에 대하여 살펴본다.

❑ 실습 순서

– Lab 202-aop 프로젝트

XML 설정 방식의 AOP 테스트 서비스 작성

❑ lab202-aop프로젝트의 beans



AOP



1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 1. Interface 작성

- /lab202-aop/src/main/java/egovframework/lab/aop/service/EmpService.java 를 확인한다.
(lab201-ioc 와 동일함)

❑ 2. VO 작성

- /lab202-aop/src/main/java/egovframework/lab/aop/service/EmpVO.java 를 확인한다.
(lab201-ioc 의 VO 와 동일함)

1. XML 설정 방식의 AOP 테스트 서비스 작성

□ 3. Impl 작성

- /lab202-aop/src/main/java/egovframework/lab/aop/service/impl/XmlEmpServiceImpl.java 를 확인한다.

```
. . .  
public EmpVO selectEmp(EmpVO empVO) throws Exception {  
    EmpVO resultVO;  
    resultVO = empDAO.selectEmp(empVO);  
  
    if(resultVO == null) {  
        throw new Exception("no data found!");  
    }  
  
    return resultVO;  
}  
. . .
```

- lab201-ioc 와 대부분 동일하며 selectEmp 메서드에서 해당 데이터가 없는 경우 exception 을 throw 하도록 비즈니스 로직을 추가하였음.

1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 4. DAO 작성

- /lab202-aop/src/main/java/egovframework/lab/aop/service/impl/XmlEmpDAO.java 를 확인한다. (lab201-ioc 의 DAO 와 동일함.)

❑ 5. xml 설정 파일 작성

- /lab202-aop/src/test/resources/META-INF/spring/context-emp.xml 를 확인한다. (lab201-ioc 의 설정과 동일함.)

1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 6. Advice 작성 (1/3)

- /lab202-aop/src/main/java/egovframework/lab/aop/xml/AdviceUsingXML.java 를 작성한다.

```
public class AdviceUsingXML {

    private final Log sysoutLogger = LogFactory.getLog(AdviceUsingXML.class);

    public void beforeTargetMethod(JoinPoint thisJoinPoint) {
        sysoutLogger.debug("\nAdviceUsingXML.beforeTargetMethod executed.");

        @SuppressWarnings("unchecked")
        Class clazz = thisJoinPoint.getTarget().getClass();
        String className = thisJoinPoint.getTarget().getClass().getSimpleName();
        String methodName = thisJoinPoint.getSignature().getName();

        // 현재 class, method 정보 및 method arguments 로깅
        StringBuffer buf = new StringBuffer();
        buf.append("\n== AdviceUsingXML.beforeTargetMethod : [" + className
            + "." + methodName + "() ] ==");
        Object[] arguments = thisJoinPoint.getArgs();
        int argCount = 0;
        for (Object obj : arguments) {
            buf.append("\n - arg ");
            buf.append(argCount++);
            buf.append(" : ");
            // commons-lang 의 ToStringBuilder 를
            // 통해(reflection 을 이용)한 VO 정보 출력
            buf.append(ToStringBuilder.reflectionToString(obj));
        }

        // 대상 클래스의 logger 를 사용하여 method arguments 로깅하였음.
        Log logger = LogFactory.getLog(clazz);
        if (logger.isDebugEnabled())
            logger.debug(buf.toString());
    }
}
...(계속)
```

1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 6. Advice 작성 (2/3)

```
...(이어서)
    public void afterTargetMethod(JoinPoint thisJoinPoint) {
        sysoutLogger.debug("AdviceUsingXML.afterTargetMethod executed.");
    }

    @SuppressWarnings("unchecked")
    public void afterReturningTargetMethod(JoinPoint thisJoinPoint,
        Object retVal) {
        sysoutLogger
            .debug("AdviceUsingXML.afterReturningTargetMethod executed.");

        Class clazz = thisJoinPoint.getTarget().getClass();
        String className = thisJoinPoint.getTarget().getClass().getSimpleName();
        String methodName = thisJoinPoint.getSignature().getName();

        // 현재 class, method 정보 및 method arguments 로깅
        StringBuffer buf = new StringBuffer();
        buf.append("\n== AdviceUsingXML.afterReturningTargetMethod : ["
            + className + "." + methodName + "()] ==");
        buf.append("\n");

        // 결과값이 List 이면 size 와 전체 List 데이터를 풀어 reflection 으로 출력 - 성능상 사용않는 것이 좋음
        if (retVal instanceof List) {
            List resultList = (List) retVal;
            buf.append("resultList size : " + resultList.size() + "\n");
            for (Object oneRow : resultList) {
                buf.append(ToStringBuilder.reflectionToString(oneRow));
                buf.append("\n");
            }
        } else {
            buf.append(ToStringBuilder.reflectionToString(retVal));
        }
    }
...(계속)
```


1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 6. Advice 작성 (3/3)

```
...(이어서)
// 대상 클래스의 logger 를 사용하여 결과값 로깅하였음.
Log logger = LogFactory.getLog(clazz);
if (logger.isDebugEnabled())
    logger.debug(buf.toString());
// return value 의 변경은 불가함에 유의!
}

public void afterThrowingTargetMethod(JoinPoint thisJoinPoint,
    Exception exception) throws Exception {
    sysoutLogger.debug("AdviceUsingXML.afterThrowingTargetMethod executed.");
    sysoutLogger.error("에러가 발생했습니다.", exception);

    // 원본 exception 을 wrapping 하고 user-friendly 한
    // 메시지를 설정하여 새로운 Exception 으로 re-throw
    throw new BizException("에러가 발생했습니다.", exception);
    // 여기서는 간단하게 작성하였지만 일반적으로 messageSource 를 사용한
    // locale 에 따른 다국어 처리 및 egov.exceptionHandler
    // 를 확장한 Biz. (ex. email 공지 등) 기능 적용이 가능함.
}

public Object aroundTargetMethod(ProceedingJoinPoint thisJoinPoint)
    throws Throwable {
    sysoutLogger.debug("AdviceUsingXML.aroundTargetMethod start.");
    long time1 = System.currentTimeMillis();
    Object retVal = thisJoinPoint.proceed();

    // Around advice 의 경우 결과값을 변경할 수도 있음!
    // 위의 retVal 을 가공하거나 심지어 전혀 다른 결과값을 대체하여 caller 에 되돌려줄 수 있음
    long time2 = System.currentTimeMillis();
    sysoutLogger.debug("AdviceUsingXML.aroundTargetMethod end. Time("
        + (time2 - time1) + ")");

    return retVal;
}
}
```

1. XML 설정 방식의 AOP 테스트 서비스 작성

□ 7. xml 설정 방식의 AOP정의 작성

- /lab202-aop/src/test/resources/META-INF/spring/context-advice-xml.xml 를 작성한다.

```
<!-- TODO [Step 1-7] xml 설정 방식의 AOP 설정 -->
<bean id="adviceUsingXML" class="egovframework.lab.aop.xml.AdviceUsingXML" />

<aop:config>
  <aop:pointcut id="targetMethod"
    expression="execution(* egovframework.lab.aop..Xml*Impl.*(..))" />
  <aop:aspect ref="adviceUsingXML">
    <aop:before pointcut-ref="targetMethod" method="beforeTargetMethod" />
    <aop:after-returning pointcut-ref="targetMethod"
      method="afterReturningTargetMethod" returning="retVal" />
    <aop:after-throwing pointcut-ref="targetMethod"
      method="afterThrowingTargetMethod" throwing="exception" />
    <aop:after pointcut-ref="targetMethod" method="afterTargetMethod" />
    <aop:around pointcut-ref="targetMethod" method="aroundTargetMethod" />
  </aop:aspect>
</aop:config>
```

- 별도의 클래스로 작성한 Advice bean 을 등록하고, aop 네임스페이스 빈 설정 태그(aop:config, aop:pointcut, aop:aspect, before/after-returning/after-throwing/after/around 정의 등) 를 사용하여 AOP 설정. 여기서는 모든 비즈니스 메서드(Impl 로 끝나는 모든 class 의 모든 메서드)에 대해 다양한 Advice 기능을 동시에 적용하였음에 유의할 것.

1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 8. Testcase 작성 (1/3)

- /lab202-aop/src/test/java/egovframework/lab/aop/service/EmpServiceTest.java 를 작성한다.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {
    "classpath*:META-INF/spring/context-advice.xml",
    "classpath*:META-INF/spring/context-aspectj-annotation.xml",
    "classpath*:META-INF/spring/context-common.xml",
    "classpath*:META-INF/spring/context-emp.xml"})
public class EmpServiceTest {

    // xml 형식으로 bean 설정한 경우 - 주석을 변경해 가며 xml,
    // annotation 에 대해 테스트 할것
    @Resource(name = "xmlEmpService")
    EmpService empService;
    // annotation 형식으로 bean 설정한 경우
    // @Resource(name = "annotationEmpService")
    // EmpService empService;

    public EmpVO makeVO() {
        // DAO 확인 - static 하게 관리하는 100 개 기본 데이터 있음
        return makeVO(101);
    }

    public EmpVO makeVO(int empNo) {
        EmpVO vo = new EmpVO();
        vo.setEmpNo(empNo);
        vo.setEmpName("홍길동" + empNo);
        vo.setJob("개발자");
        return vo;
    }

    ...(계속)
```

1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 8. Testcase 작성 (2/3)

```
...(이어서)
public void checkResult(EmpVO vo, EmpVO resultVO) {
    assertNotNull(resultVO);
    assertEquals(vo.getEmpNo(), resultVO.getEmpNo());
    assertEquals(vo.getEmpName(), resultVO.getEmpName());
    assertEquals(vo.getJob(), resultVO.getJob());
}

@Test
public void testInsertEmp() throws Exception {
    EmpVO vo = makeVO();
    // insert
    empService.insertEmp(vo);
    // select
    EmpVO resultVO = empService.selectEmp(vo);
    // check
    checkResult(vo, resultVO);
}

@Test
public void testUpdateEmp() throws Exception {
    EmpVO vo = makeVO(102);
    // insert
    empService.insertEmp(vo);
    // data change
    vo.setEmpName("홍길순");
    vo.setJob("설계자");
    // update
    empService.updateEmp(vo);
    // select
    EmpVO resultVO = empService.selectEmp(vo);
    // check
    checkResult(vo, resultVO);
}
...(계속)
```

1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 8. Testcase 작성 (3/3)

```
...(이어서)
@Test
public void testDeleteEmp() throws Exception {
    EmpVO vo = makeVO(103);
    // insert
    empService.insertEmp(vo);
    // delete
    empService.deleteEmp(vo);
    // select
    try {
        @SuppressWarnings("unused")
        EmpVO resultVO = empService.selectEmp(vo);
        fail("Biz Exception 이 발생해야 합니다.");
    } catch (Exception e) {
        assertNotNull(e);
        // aop 를 적용하여 after-throwing 에서 가공한 BizException 으로 넘어올 것임.
        // cf.) ServiceImpl 원본에서는 그냥 Exception 이었음.
        assertTrue(e instanceof BizException);
        assertEquals("에러가 발생했습니다.", e.getMessage());
        assertEquals("no data found!", e.getCause().getMessage());
    }
}

@Test
public void testSelectEmpList() throws Exception {
    // select list
    List<EmpVO> resultList = empService.selectEmpList();
    // check
    int firstListSize = resultList.size();
    // DAO 에서 Emp 데이터를 관리할 때 항상 sorted 된 상태임
    assertEquals(1, resultList.get(0).getEmpNo());

    ...(계속)
```

1. XML 설정 방식의 AOP 테스트 서비스 작성

❑ 8. Testcase 작성 (3/3)

```
...(이어서)

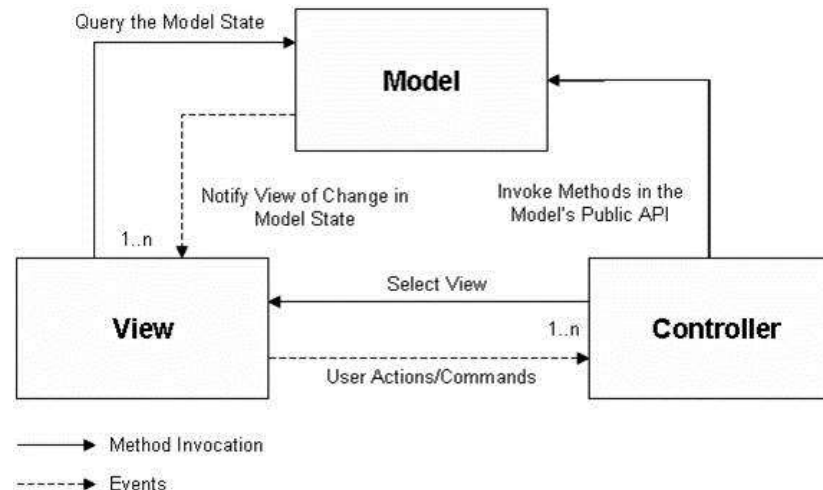
// delete first data
EmpVO empVO = new EmpVO();
empVO.setEmpNo(1);

empService.deleteEmp(empVO);
// select List again
resultList = empService.selectEmpList();
assertEquals(firstListSize - 1, resultList.size());
// DAO 에서 Emp 데이터를 관리할 때 항상 sorted 된 상태임
assertEquals(2, resultList.get(0).getEmpNo());
assertEquals("EmpName 2", resultList.get(0).getEmpName());
assertEquals("SALESMAN", resultList.get(0).getJob());
    }
}
```

- lab201-ioc 의 테스트케이스와 유사하며 ContextConfiguration 설정 파일 로딩에 유의할것.
- delete 테스트의 경우 삭제 후 재조회 시 null 인 경우 강제 Exception 처리를 하였고 이를 다시 after-throwing AOP 처리로 BizException 으로 재처리한 것을 테스트 하고 있음.

□ 서비스 개요

- MVC(Model-View-Controller) 패턴은 코드를 기능에 따라 Model, View, Controller 3가지 요소로 분리한다.
 - **Model** : 어플리케이션의 데이터와 비즈니스 로직을 담는 객체이다.
 - **View** : Model의 정보를 사용자에게 표시한다. 하나의 Model을 다양한 View에서 사용할 수 있다.
 - **Controller** : Model과 View의 중계역할로 view를 선택한다. 사용자의 요청을 받아 Model에 변경된 상태를 반영하고, 응답을 위한 V
- MVC 패턴은 UI 코드와 비즈니스 코드를 분리함으로써 종속성을 줄이고, 재사용성을 높이고, 보다 쉬운 변경이 가능하도록 한다.
- 전자정부프레임워크에서 "MVC 서비스"란 MVC 패턴을 활용한 Web MVC Framework를 의미한다.



□ 오픈소스 Web MVC Framework

- Spring MVC, Struts, Webwork 등이 있다.
- 전자정부프레임워크에서는 **Spring MVC**를 채택하였다.
 - Framework내의 특정 클래스를 상속하거나, 참조, 구현해야 하는 등의 제약사항이 비교적 적다.
 - IOC Container가 Spring 이라면 연계를 위한 추가 설정 없이 Spring MVC를 사용할 수 있다.
 - 오픈소스 프로젝트가 활성화(꾸준한 기능 추가, 빠른 bug fix와 Q&A) 되어 있으며 로드맵이 신뢰할 만 하다.
 - 국내 커뮤니티 활성화 정도, 관련 참고문서나 도서를 쉽게 구할 수 있다.

□ Spring MVC

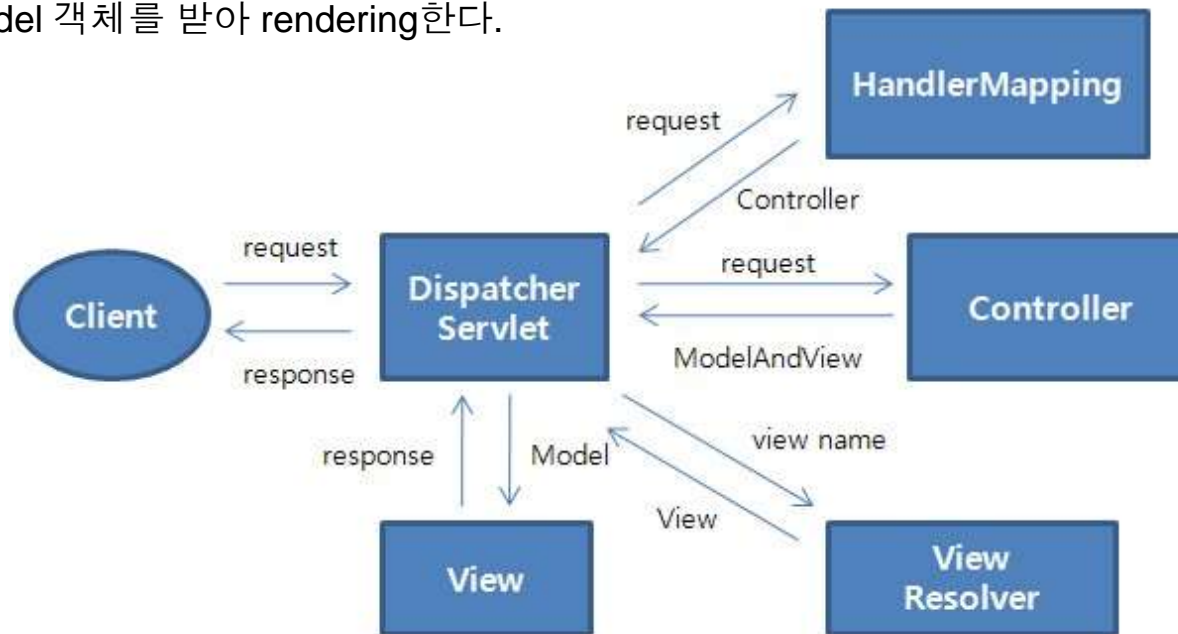
- DispatcherServlet, HandlerMapping, Controller, Interceptor, ViewResolver, View 등 각 **컴포넌트들의 역할이 명확하게 분리**된다.
- HandlerMapping, Controller, View 등 컴포넌트들에 **다양한 인터페이스 및 구현 클래스를 제공**한다.
- Controller(@MVC)나 폼 클래스(커맨드 클래스) 작성시에 특정 클래스를 상속받거나 참조할 필요 없이 **POJO 나 POJO-style의 클래스를 작성함으로써 비즈니스 로직에 집중한 코드를 작성**할 수 있다.
- 웹요청 파라미터와 커맨드 클래스간에 데이터 매핑 기능을 제공한다.
- 데이터 검증을 할 수 있는, Validator와 Error 처리 기능을 제공한다.
- JSP Form을 쉽게 구성하도록 Tag를 제공한다.

❑ Spring MVC의 핵심 Component

- **DispatcherServlet**
 - Spring MVC Framework의 Front Controller, 웹요청과 응답의 Life Cycle을 주관한다.
- **HandlerMapping**
 - 웹요청시 해당 URL을 어떤 Controller가 처리할지 결정한다.
- **Controller**
 - 비즈니스 로직을 수행하고 결과 데이터를 ModelAndView에 반영한다.
- **ModelAndView**
 - Controller가 수행 결과를 반영하는 Model 데이터 객체와 이동할 페이지 정보(또는 View객체)로 이루어져 있다.
- **ViewResolver**
 - 어떤 View를 선택할지 결정한다.
- **View**
 - 결과 데이터인 Model 객체를 display한다.

□ Spring MVC 컴포넌트간의 관계와 흐름

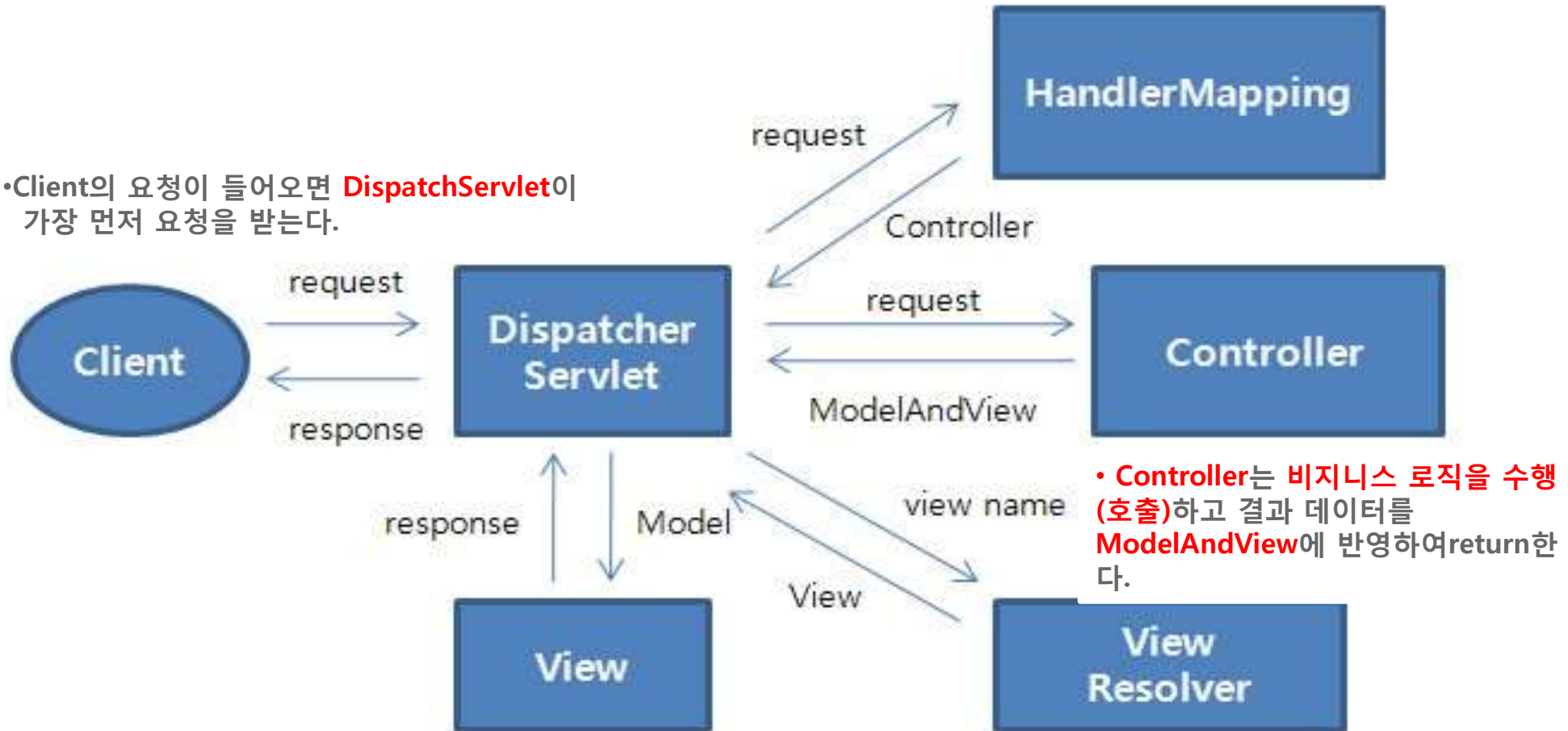
- Client의 요청이 들어오면 DispatcherServlet이 가장 먼저 요청을 받는다.
- HandlerMapping이 요청에 해당하는 Controller를 return한다.
- Controller는 비즈니스 로직을 수행(호출)하고 결과 데이터를 ModelAndView에 반영하여 return한다.
- ViewResolver는 view name을 받아 해당하는 View 객체를 return한다.
- View는 Model 객체를 받아 rendering한다.



□ Spring MVC 컴포넌트간의 관계와 흐름

• **HandlerMapping**이 요청에 해당하는 **Controller**를 return한다.

• Client의 요청이 들어오면 **DispatcherServlet**이 가장 먼저 요청을 받는다.

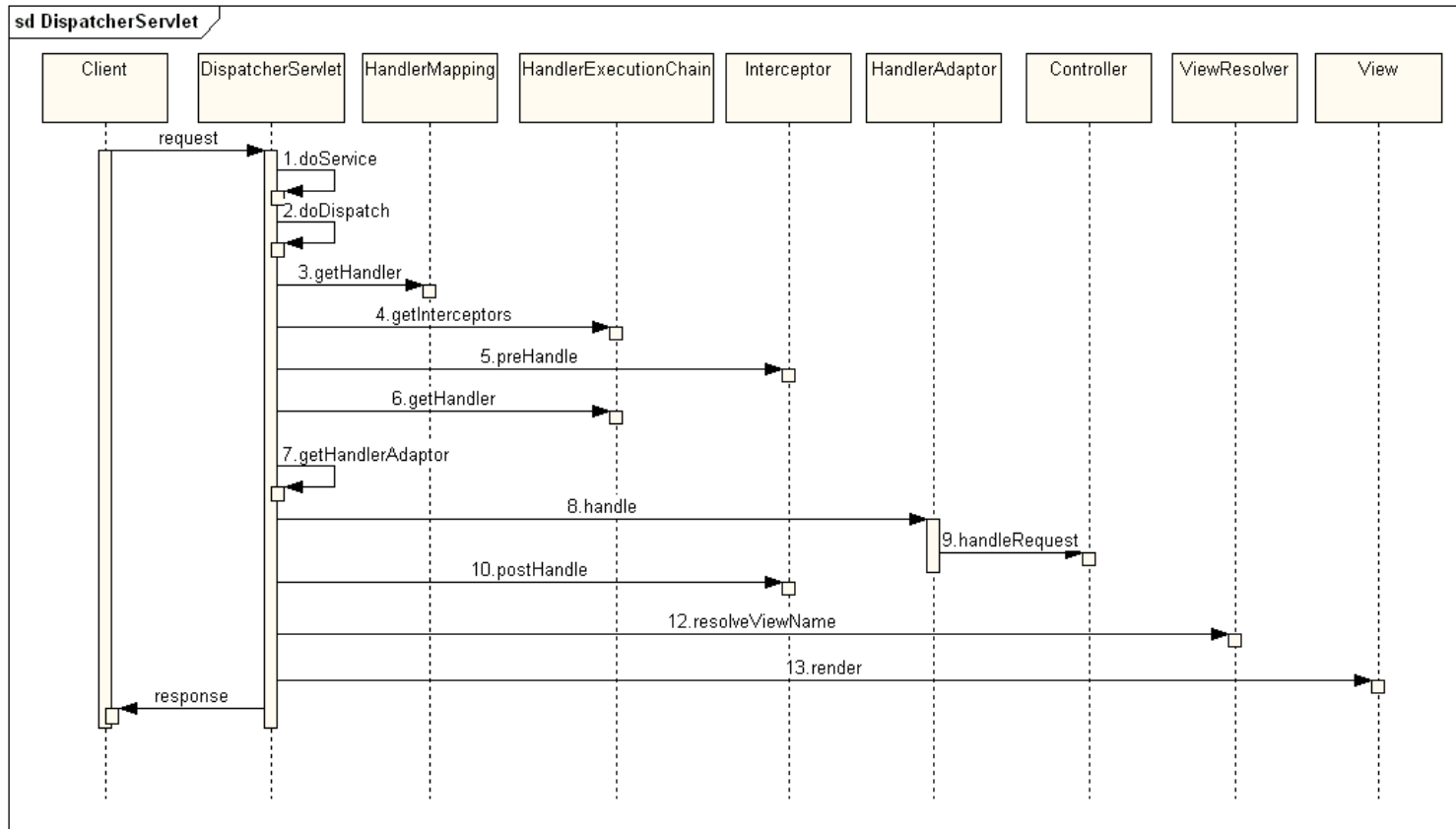


• **View**는 Model 객체를 받아 rendering한다.

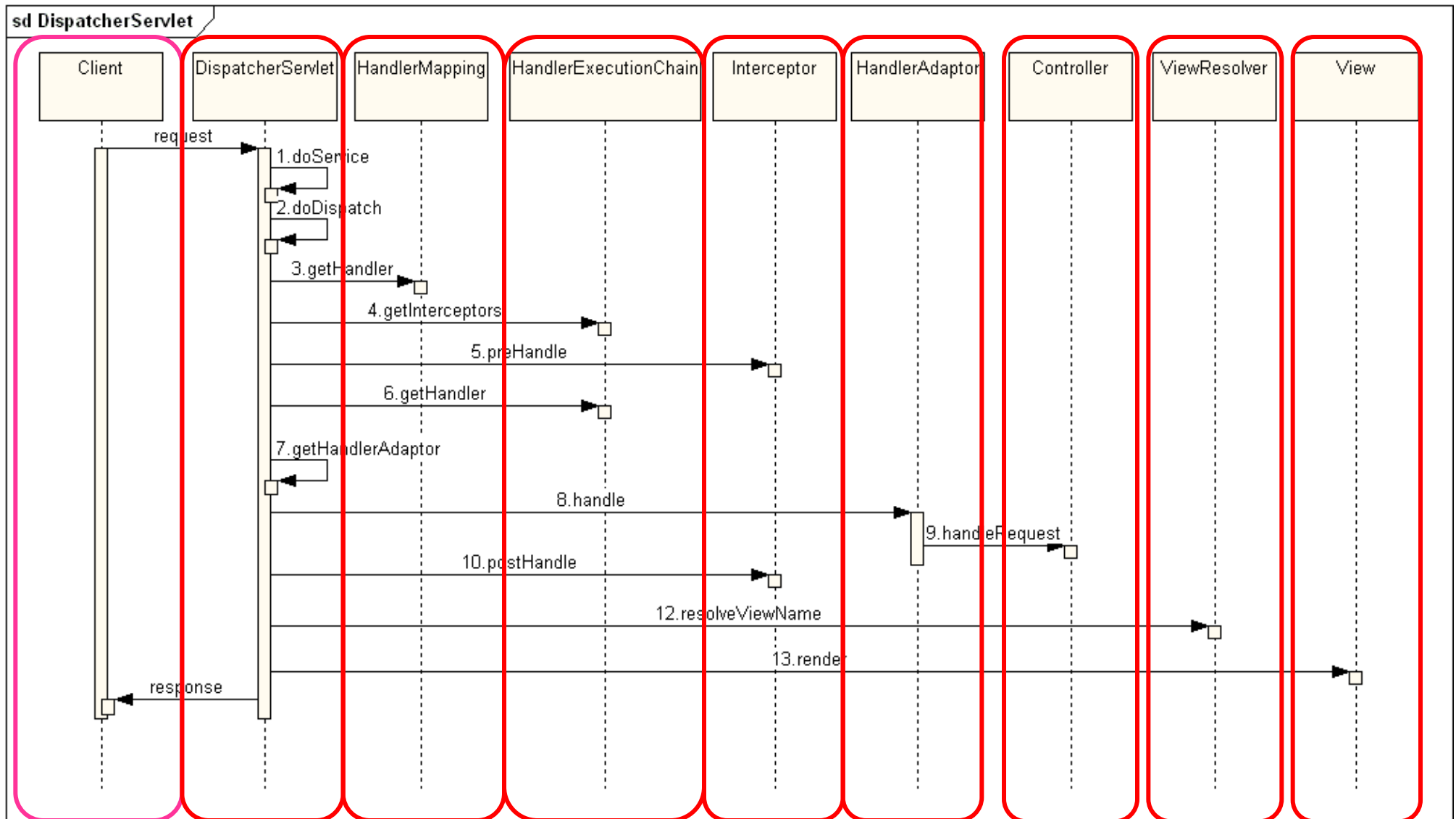
• **ViewResolver**는 view name을 받아 해당하는 **View** 객체를 return한다.

❑ DispatcherServlet

- **Controller로 향하는 모든 웹요청의 진입점**이며, 웹요청을 처리하며, 결과 데이터를 Client에게 응답 한다.
- **Spring MVC의 웹요청 Life Cycle을 주관**

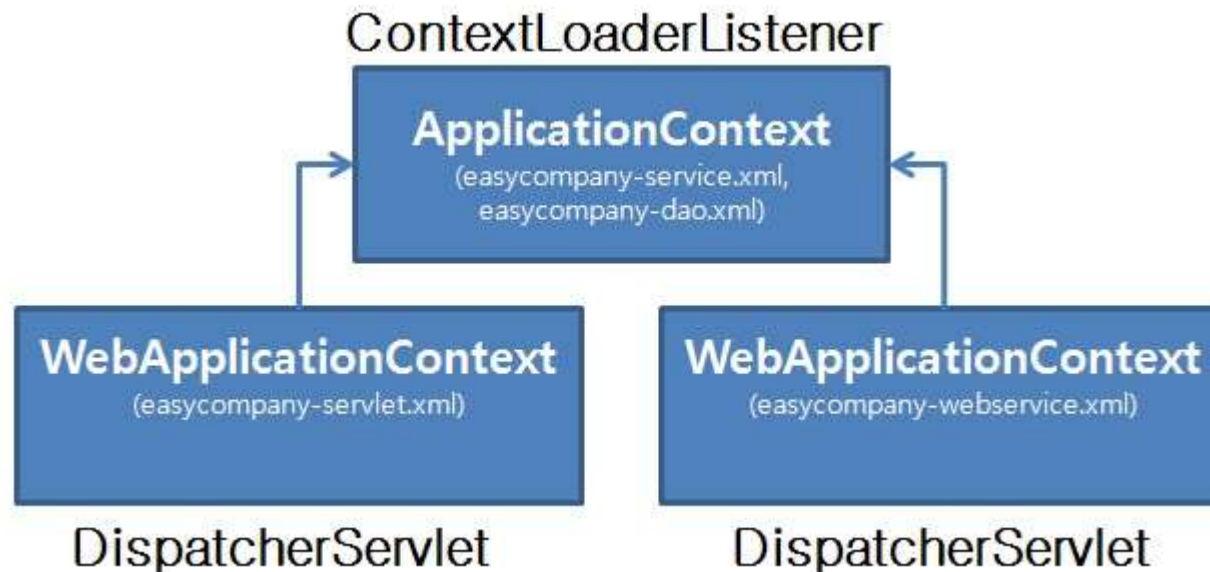


❑ Spring MVC의 웹요청 Life Cycle 을 주관하는 DispatcherServlet



❑ DispatcherServlet, ApplicationContext, WebApplicationContext

- 하나의 빈 설정파일에 모든 빈을 등록할 수도 있지만, 아래와 같이 **Layer 별로 빈파일을 나누어 등록하고 ApplicationContext, WebApplicationContext 사용하는것을 권장.**
- **ApplicationContext** : **ContextLoaderListener에 의해 생성. persistance, service layer의 빈**
- **WebApplicationContext** : **DispatcherServlet에 의해 생성. presentation layer의 빈**
- ContextLoaderListener는 웹 어플리케이션이 시작되는 시점에 ApplicationContext를 만들고, 이 ApplicationContext의 빈 정보는 모든 WebApplicationContext들이 참조할 수 있다.



❑ web.xml에 DispatcherServlet 설정하기

```
<!-- ApplicationContext 빈 설정 파일-->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/config/service/easycompany-service.xml <!--서비스 빈 정의-->
    /WEB-INF/config/service/easycompany-dao.xml <!--Dao 빈 정의-->
  </param-value>
</context-param>
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>

<!-- WebApplicationContext 빈 설정 파일-->
<servlet>
  <servlet-name>servlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/config/easycompany-servlet.xml <!--web layer 관련 빈 선언-->
    </param-value>
  </init-param>
</servlet>

<!-- WebApplicationContext 빈 설정 파일-->
<servlet>
  <servlet-name>webservice</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/config/easycompany-webservice.xml
    </param-value>
  </init-param>
</servlet>
```

LAB 301-mvc 실습 (1)

□ @MVC

- **어노테이션을 이용한 설정** : XML 기반으로 설정하던 정보들을 어노테이션을 사용해서 정의한다.
- **유연해진 메소드 시그니처** : Controller 메소드의 파라미터와 리턴 타입을 좀 더 다양하게 필요에 따라 선택할 수 있다.
- **POJO-Style의 Controller** : Controller 개발시에 특정 인터페이스를 구현 하거나 특정 클래스를 상속해야할 필요가 없다. 하지만, 폼 처리, 다중 액션등 기존의 계층형 Controller가 제공하던 기능들을 여전히 쉽게 구현할 수 있다.
- **Bean 설정파일 작성** : @Controller만 스캔하도록 설정한다.

❑ <context:component-scan/> 설정

- @Component, @Service, @Repository, @Controller 가 붙은 클래스들을 읽어들이어 ApplicationContext, WebApplicationContext에 빈정보를 저장, 관리한다.
- @Controller만 스캔하려면 <context:include-filter>나 <context:exclude-filter>를 사용해야 한다.

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

  <context:component-scan base-package="com.easycompany.controller.annotation">
    <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>
    <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Service"/>
    <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Repository"/>
  </context:component-scan>
</beans>
```

❑ RequestMappingHandlerMapping (DefaultAnnotationHandlerMapping는 deprecated)

- @MVC 개발을 위한 HandlerMapping. 표준프레임워크 3.0(Spring 3.2.9)에서 사용가능.
- 기존 DefaultAnnotationHandlerMapping이 deprecated되면서 대체됨.
- **@RequestMapping에 지정된 url과 해당 Controller의 메소드 매핑**
- RequestMappingHandlerMapping은 기본 HandlerMapping이며, 선언하기 위해서는 다음과 같이 세가지 방법이 있다.
- 선언하지 않는 방법 : 기본 HandlerMapping이므로 지정하지 않아도 사용가능하다.
- <mvc:annotation-driven/>을 선언하는 방법 : @MVC사용 시 필요한 빈들을 등록해주는 <mvc:annotation-driven/>을 설정하면 내부에서 RequestMappingHandlerMapping, RequestMappingHandlerAdapter 이 구성된다.
- RequestMappingHandlerMapping을 직접 선언하는 방법 : 다른 HandlerMapping과 함께 사용할 때 선언한다.

❑ SimpleUrlAnnotationHandlerMapping(deprecated됨 mvc 태그로 변경)

- DefaultAnnotationHandlerMapping은 특정 url에 대해 interceptor를 적용할수 없음. -> 확장 HandlerMapping
- DefaultAnnotationHandlerMapping과 함께 사용. (order 프로퍼티를 SimpleUrlAnnotationHandlerMapping에 준다.)

```
<bean id="selectAnnotaionMapper"
      class="egovframework.rte.ptl.mvc.handler.SimpleUrlAnnotationHandlerMapping" p:order="1">
  <property name="interceptors">
    <list>
      <ref local="authenticInterceptor"/>
    </list>
  </property>
  <property name="urls">
    <list>
      <value>/admin/*.do</value>
      <value>/user/userInfo.do</value>
      <value>/development/**/code*.do</value>
    </list>
  </property>
</bean>
<bean id="annotationMapper"
      class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" p:order="2"/>
```

□ 관련 어노테이션

@Controller	해당 클래스가 Controller임을 나타내기 위한 어노테이션
@RequestMapping	요청에 대해 어떤 Controller, 어떤 메소드가 처리할지를 맵핑하기 위한 어노테이션
@RequestParam	Controller 메소드의 파라미터와 웹요청 파라미터와 맵핑하기 위한 어노테이션
@ModelAttribute	Controller 메소드의 파라미터나 리턴값을 Model 객체와 바인딩하기 위한 어노테이션
@SessionAttributes	Model 객체를 세션에 저장하고 사용하기 위한 어노테이션
@CommandMap	Controller메소드의 파라미터를 Map형태로 받을 때 웹요청 파라미터와 맵핑하기 위한 어노테이션(egov 3.0부터 추가)

□ @Controller

- @MVC에서 Controller를 만들기 위해서는 작성한 클래스에 @Controller를 붙여주면 된다. 특정 클래스를 구현하거나 상속할 필요가 없다.

```
import org.springframework.stereotype.Controller;
```

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

□ @RequestMapping

- 요청에 대해 어떤 Controller, 어떤 메소드가 처리할지를 매핑하기 위한 어노테이션이다
- 관련속성

이름	타입	매핑 조건	설명
value	String[]	URL 값	<ul style="list-style-type: none"> - @RequestMapping(value="/hello.do") - @RequestMapping(value={"/hello.do", "/world.do" }) - @RequestMapping("/hello.do") - Ant-Style 패턴매칭 이용 : "/myPath/*.do"
method	Request Method[]	HTTP Request 메소드값	<ul style="list-style-type: none"> - @RequestMapping(method = RequestMethod.POST) - 사용 가능한 메소드 : GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE
params	String[]	HTTP Request 파라미터	<ul style="list-style-type: none"> - params="myParam=myValue" : HTTP Request URL중에 myParam이라는 파라미터가 있어야 하고 값은 myValue이어야 매핑 - params="myParam" : 파라미터 이름만으로 조건을 부여 - "!myParam" : myParam이라는 파라미터가 없는 요청 만을 매핑 - @RequestMapping(params={"myParam1=myValue", "myParam2", "!myParam3"})와 같이 조건을 주었다면, HTTP Request에는 파라미터 myParam1이 myValue값을 가지고 있고, myParam2 파라미터가 있어야 하고, myParam3라는 파라미터는 없어야함.

❑ @RequestMapping 설정

- @RequestMapping은 클래스 단위(type level)나 메소드 단위(method level)로 설정할 수 있다.

type level

/hello.do 요청이 오면 HelloController의 hello 메소드가 수행된다.

type level에서 URL을 정의하고 Controller에 메소드가 하나만 있어도 요청 처리를 담당할 메소드 위에 @RequestMapping 표기를 해야 제대로 맵핑이 된다.

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

@Controller
@RequestMapping("/hello.do")
public class HelloController {

    @RequestMapping
    public String hello(){
        ...
    }
}
```

method level

/hello.do 요청이 오면 hello 메소드,

/helloForm.do 요청은 GET 방식이면 helloGet 메소드, POST 방식이면 helloPost 메소드가 수행된다.

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

@Controller

```
public class HelloController {
```

```
    @RequestMapping(value="/hello.do")
    public String hello(){
        ...
    }
```

```
    @RequestMapping(value="/helloForm.do", method = RequestMethod.GET)
    public String helloGet(){
        ...
    }
```

```
    @RequestMapping(value="/helloForm.do", method = RequestMethod.POST)
    public String helloPost(){
        ...
    }
```

```
}
```


type + method level

type level, method level 둘 다 설정할 수도 있는데,

이 경우엔 type level에 설정한 @RequestMapping의 value(URL)를 method level에서 재정의 할수 없다.

/hello.do 요청시에 GET 방식이면 helloGet 메소드, POST 방식이면 helloPost 메소드가 수행된다.

```
@Controller
@RequestMapping("/hello.do")
public class HelloController {

    @RequestMapping(method = RequestMethod.GET)
    public String helloGet(){
        ...
    }

    @RequestMapping(method = RequestMethod.POST)
    public String helloPost(){
        ...
    }
}
```

❑ @RequestParam

- @RequestParam은 Controller 메소드의 파라미터와 웹요청 파라미터와 매핑하기 위한 어노테이션이다.
- 관련 속성

이름	타입	설명
value	String	파라미터 이름
required	boolean	해당 파라미터가 반드시 필수 인지 여부. 기본값은 true이다.

- 해당 파라미터가 Request 객체 안에 없을때 그냥 null값을 바인드 하고 싶다면, 아래 예제의 pageNo 파라미터 처럼 required=false로 명시해야 한다.
- name 파라미터는 required가 true이므로, 만일 name 파라미터가 null이면 org.springframework.web.bind.MissingServletRequestParameterException이 발생한다.

```
@Controller
public class HelloController {

    @RequestMapping("/hello.do")
    public String hello(@RequestParam("name") String name,
                       @RequestParam(value="pageNo", required=false) String pageNo){
        ...
    }
}
```

❑ @ModelAttribute

- @ModelAttribute은 Controller에서 2가지 방법으로 사용된다.
 1. Model 속성(attribute)과 메소드 파라미터의 바인딩.
 2. 입력 폼에 필요한 참조 데이터(reference data) 작성. - SimpleFormContrller의 referenceData 메소드와 유사한 기능.
- 관련 속성

이름	타입	설명
value	String	바인드하려는 Model 속성 이름.

❑ @SessionAttributes

- @SessionAttributes는 model attribute를 session에 저장, 유지할 때 사용하는 어노테이션이다.
- @SessionAttributes는 클래스 레벨(type level)에서 선언할 수 있다.
- 관련 속성

이름	타입	설명
value	String[]	session에 저장하려는 model attribute의 이름
required	Class[]	session에 저장하려는 model attribute의 타입

❑ @CommandMap

- 실행환경 3.0부터 추가되었으며 Controller에서 Map형태로 웹요청 값을 받았을 때 다른 Map형태의 argument와 구분해 주기 위한 어노테이션이다.
- @CommandMap은 파라미터 레벨(type level)에서 선언할 수 있다.
- 사용 방법은 다음과 같다.

```
@RequestMapping("/test.do")
public void test(@CommandMap Map<String, String> commandMap, HttpServletRequest request){
    //생략
}
```

- CommandMap을 이용하기 위해서는 반드시 EgovRequestMappingHandlerAdapter와 함께 AnnotationCommandMapArgumentResolver를 등록해 주어야 한다.

```
<bean class="egovframework.rte.ptl.mvc.bind.annotation.EgovRequestMappingHandlerAdapter">
    <property name="customArgumentResolvers">
        <list>
            <bean class="egovframework.rte.ptl.mvc.bind.AnnotationCommandMapArgumentResolver" />
        </list>
    </property>
</bean>
```

□ @Controller 메소드 시그니처

- 기존의 계층형 Controller(SimpleFormController, MultiAction..)에 비해 유연한 메소드 파라미터, 리턴값을 갖는다.

□ 메소드 파라미터

- Servlet API - ServletRequest, HttpServletRequest, HttpServletResponse, HttpSession 같은 요청,응답,세션관련 Servlet API.
- org.springframework.web.context.request.WebRequest, org.springframework.web.context.request.NativeWebRequest
- java.util.Locale
- java.io.InputStream / java.io.Reader
- java.io.OutputStream / java.io.Writer
- **@RequestParam** - HTTP Request의 파라미터와 메소드의 argument를 바인딩하기 위해 사용하는 어노테이션.
- java.util.**Map** / org.springframework.ui.**Model** / org.springframework.ui.**ModelMap** - 뷰에 전달할 모델데이터.
- **Command/form 객체** - HTTP Request로 전달된 parameter를 바인딩한 커맨드 객체, @ModelAttribute을 사용하면 alias를 줄수 있다.
- org.springframework.validation.Errors / org.springframework.validation.**BindingResult** - 유효성 검사 후 결과 데이터를 저장한 객체.
- org.springframework.web.bind.support.**SessionStatus** - 세션폼 처리시에 해당 세션을 제거하기 위해 사용된다.

□ 메소드 리턴 타입

- **ModelAndView** - 커맨드 객체, @ModelAttribute 적용된 메소드의 리턴 데이터가 담긴 Model 객체와 View 정보가 담겨 있다.
- **Model(또는 ModelMap)** - 커맨드 객체, @ModelAttribute 적용된 메소드의 리턴 데이터가 Model 객체에 담겨 있다. View 이름은 RequestToViewNameTranslator가 URL을 이용하여 결정한다.
- **Map** - 커맨드 객체, @ModelAttribute 적용된 메소드의 리턴 데이터가 Map 객체에 담겨 있으며, View 이름은 역시 RequestToViewNameTranslator가 결정한다
- **String** - 리턴하는 String 값이 곧 View 이름이 된다. 커맨드 객체, @ModelAttribute 적용된 메소드의 리턴 데이터가 Model(또는 ModelMap)에 담겨 있다. 리턴할 Model(또는 ModelMap)객체가 해당 메소드의 argument에 선언되어 있어야 한다
- **void** - 메소드가 ServletResponse / HttpServletResponse등을 사용하여 직접 응답을 처리하는 경우이다. View 이름은 RequestToViewNameTranslator가 결정한다.

❑ @Controller 로 폼처리 구현하기

- 부서정보를 수정하고 저장하는 폼페이지를 @Controller로 구현해 보자. 메소드의 이름은 폼처리를 담당하는 기존의 Form Controller인 SimpleFormController와의 비교를 위해 기능별로 동일하게 지었다.

❑ 화면 & 시나리오

부서번호	1100
부서이름	<input type="text" value="회식매뉴얼팀"/>
상위부서	<input type="text" value="경영기획실"/>
설명	<div> <p>매번 삼겹살 지겹지 않으세요? 저희 회식매뉴얼팀에서는 지속적인 즐거운 회사문화 조성을 위해...</p> </div>
<input type="button" value="저장"/> <input type="button" value="리스트페이지"/>	

1. 파라미터 부서번호의 해당 부서정보 데이터를 불러와 입력폼을 채운다.
2. 상위부서(selectbox)는 부서정보 데이터와는 별도로, 상위부서에 해당하는 부서리스트 데이터를 구해서 참조데이터로 구성한다.
3. 사용자가 데이터 수정을 끝내고 저장 버튼을 누르면 수정 데이터로 저장을 담당하는 서비스(DB)를 호출한다.
4. 저장이 성공하면 부서리스트 페이지로 이동하고 에러가 있으면 다시 입력폼페이지로 이동한다.

❑ Controller 작성하기

```

package com.easycompany.controller.annotation;
...
@Controller
public class UpdateDepartmentController {

    @Autowired
    private DepartmentService departmentService;

    //상위부서(selectbox)는 부서정보 데이터와는 별도로 상위부서에 해당하는 부서리스트 데이터를 구해서 참조데이터로 구성한다.
    @ModelAttribute("deptInfoOneDepthCategory")
    public Map<String, String> referenceData() {
        Map<String, String> param = new HashMap<String, String>();
        param.put("depth", "1");
        return departmentService.getDepartmentIdNameList(param);
    }

    // 해당 부서별 상위 부서정보 데이터를 부러와 입력폼을 채운다
    @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.GET)
    public String formBackingObject(@RequestParam("deptid") String deptid, Model model) {
        Department department = departmentService.getDepartmentInfoById(deptid);
        model.addAttribute("department", department);
        return "modifydepartment";
    }

    //사용자가 데이터 수정을 끝내고 저장 버튼을 누르면 수정 데이터로 저장을 담당하는 서비스(DB)를 호출한다.
    //저장이 성공하면 부서리스트 페이지로 이동하고 에러가 있으면 다시 입력폼페이지로 이동한다.
    @RequestMapping(value = "/updateDepartment.do", method = RequestMethod.POST)
    public String onSubmit(@ModelAttribute("department") Department department) {
        try {
            departmentService.updateDepartment(department);
            return "redirect:/departmentList.do?depth=1";
        } catch (Exception e) {
            return "modifydepartment";
        }
    }
}

```


□ JSP

- 폼 필드와 모델 데이터의 편리한 데이터 바인딩을 위해 스프링 폼 태그를 사용한다.
- commandName에는 model attribute를 적어주면 된다. "department"

```

...
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<form:form commandName="department">
  <table>
    <tr>
      <th>부서번호</th> <td><c:out value="${department.deptid}" /></td>
    </tr>
    <tr>
      <th>부서이름</th> <td><form:input path="deptname" size="20" /></td>
    </tr>
    <tr>
      <th>상위부서</th>
      <td>
        <form:select path="superdeptid">
          <option value="">상위부서를 선택하세요.</option>
          <form:options items="${deptInfoOneDepthCategory}" />
        </form:select>
      </td>
    </tr>
    <tr>
      <th>설명</th> <td><form:textarea path="description" rows="10" cols="40" /></td>
    </tr>
  </table>
  ...
  <input type="submit" value="저장"/>
  <input type="button" value="리스트페이지" onclick="location.href=/easycompany/departmentList.do?depth=1"/>
  ...
</form:form>

```

LAB 301-mvc 실습 (2)

❑ Spring Framework API

- <http://docs.spring.io/spring/docs/3.2.x/javadoc-api/>
- 이전 버전 참조
 - <http://static.springsource.org/spring/docs/2.5.x/api/index.html>
 - <http://static.springsource.org/spring/docs/3.0.x/javadoc-api/>

❑ The Spring Framework - Reference Documentation

- <http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/spring-web.html>
- 이전 버전 참조
 - <http://static.springsource.org/spring/docs/2.5.x/reference/spring-web.html>
 - <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/spring-web.html>

LAB 301-mvc 실습 (1)

Exercise 1-1-1. “/hello.do” 에 동작하는 Controller 메소드 만들기

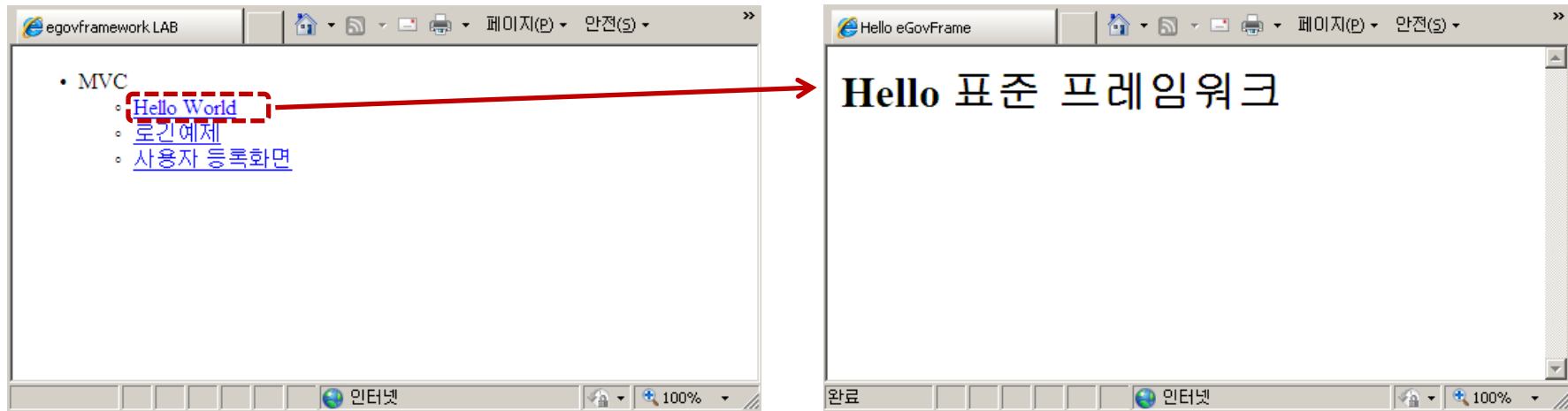
```
@RequestMapping(value = "/hello.do")
public String helloworld() {
    return getViewName();
}
```

Exercise 1-1-2. helloworld.jsp 만들기 (위치 : src\main\webapp\WEB-INF\jsp\hello)

```
<%@ page contentType="text/html; charset=utf-8" pageEncoding="utf-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Hello eGovFrame</title>
</head>
<body>
<h1>Hello 표준 프레임워크 </h1>
</body>
</html>
```

Exercise 1-1-3. 'Hello World' 예제 실행결과 확인

- 프로젝트 선택 마우스 우클릭 > Run As > Run On Server 실행
- 예제 실행 결과 확인 (<http://127.0.0.1:8080/lab301-mvc/>)



LAB 301-mvc 실습 (2)

Exercise 1-2-1. context-servlet.xml 설정 변경하기 : messageSource 활성화

```
<bean id="messageSource"
      class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>messages.message-common</value>
        </list>
    </property>
</bean>
```

Exercise 1-2-2. LoginController.java 메소드 추가하기

```
@RequestMapping(value = "/loginProcess1.do", method = RequestMethod.GET)
public String loginFormSetUp() {
    return getFormView();
}

@RequestMapping(value = "/loginProcess1.do", method = RequestMethod.POST)
public String loginProcess(@ModelAttribute("login") LoginCommand loginCommand) {

    return getSuccessView();
}

@ModelAttribute("loginTypes")
protected List<LoginType> referenceData() throws Exception {
    List<LoginType> loginTypes = new ArrayList<LoginType>();
    loginTypes.add(new LoginType("A", "개인회원"));
    loginTypes.add(new LoginType("B", "기업회원"));
    loginTypes.add(new LoginType("C", "관리자"));
    return loginTypes;
}

@ModelAttribute("login")
protected Object referenceData4login() throws Exception {
    return new LoginCommand();
}
```


Exercise 1-2-3. LoginCommand.java 완성하기

```
private String id;
private String password;
private String loginType;

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getPassword() {
    return password;
}

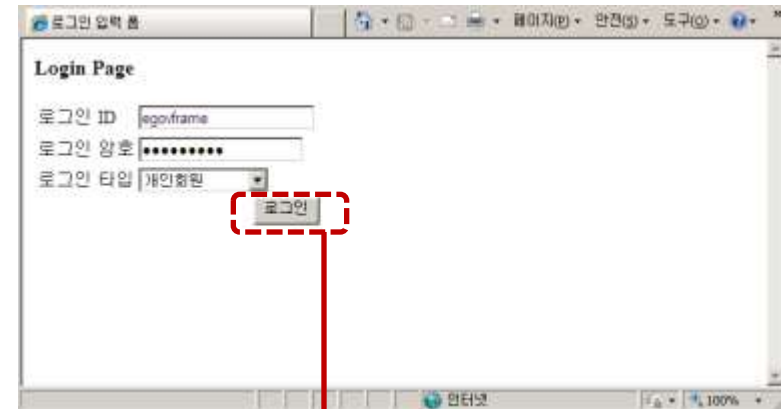
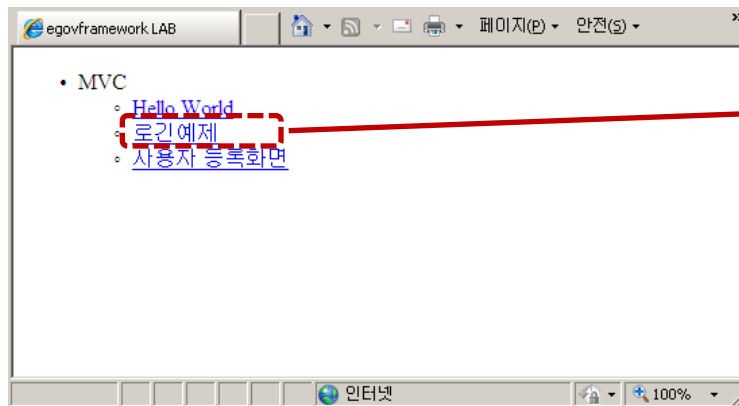
public void setPassword(String password) {
    this.password = password;
}

public String getLoginType() {
    return loginType;
}

public void setLoginType(String loginType) {
    this.loginType = loginType;
}
```

Exercise 1-2-4. . 'Hello World' 예제 실행결과 확인

- 프로젝트 선택 마우스 우클릭 > Run As > Run On Server 실행
- 예제 실행 결과 확인 (<http://127.0.0.1:8080/lab301-mvc/>)



LAB 301-mvc 실습 (3)

Step 1-3-1. context-servlet.xml 추가 설정하기

```
<!-- [Exercise 1-3-3] mvc-servlet.xml 추가 설정하기 : SessionLocaleResolver를 이용한 locale 설정 -->
<!-- setting Locale -->
<bean id="localeChangeInterceptor"
      class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor"
      p:paramName="lang" />

<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver" />

<!-- Locale Interceptor 설정하기 -->
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">
    <property name="interceptors">
        <list>
            <ref bean="localeChangeInterceptor"/>
        </list>
    </property>
</bean>
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter" />
```

Step 1-3-2. LoginController.java 에 @SessionAttributes 설정 하기

```
@Controller
@SessionAttributes("login")
public class LoginController {
    ...
}
```

Step 1-3-3. LoginController.java 추가 메소드 작성하기

```
@RequestMapping(value = "/memberInfo.do")
public ModelAndView memberInfo(HttpSession httpSession) {
    ModelAndView mav = new ModelAndView("login/memberInfo");

    if (httpSession.getAttribute("login") != null) {
        mav.addObject("login", httpSession.getAttribute("login"));
    }

    return mav;
}

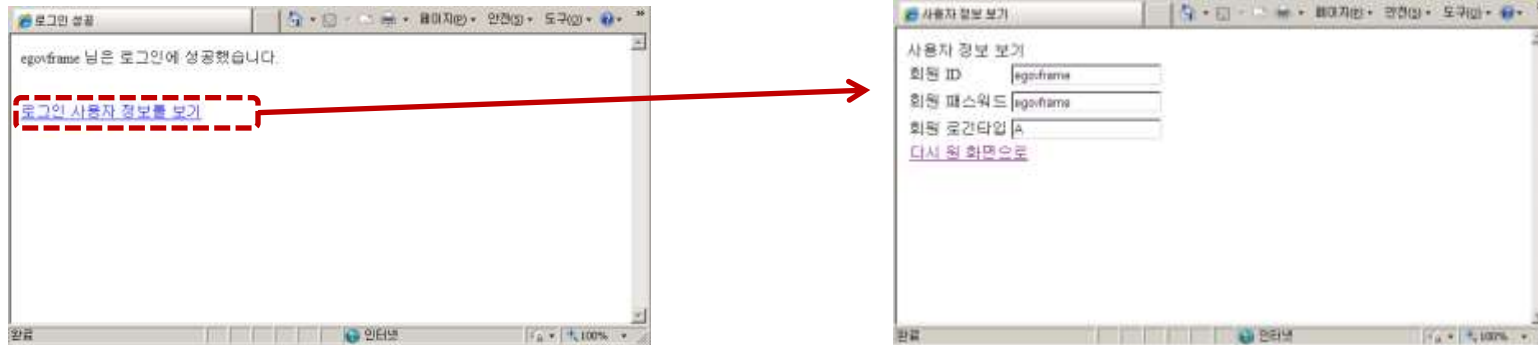
@RequestMapping(value = "/loginOut.do", method = RequestMethod.GET)
public String logOut(SessionStatus sessionStatus) {

    if (!sessionStatus.isComplete())
        sessionStatus.setComplete();

    return getFormView();
}
```

Exercise 1-3-4. . '사용자 정보보기' 예제 실행결과 확인

- 예제 실행 결과 확인

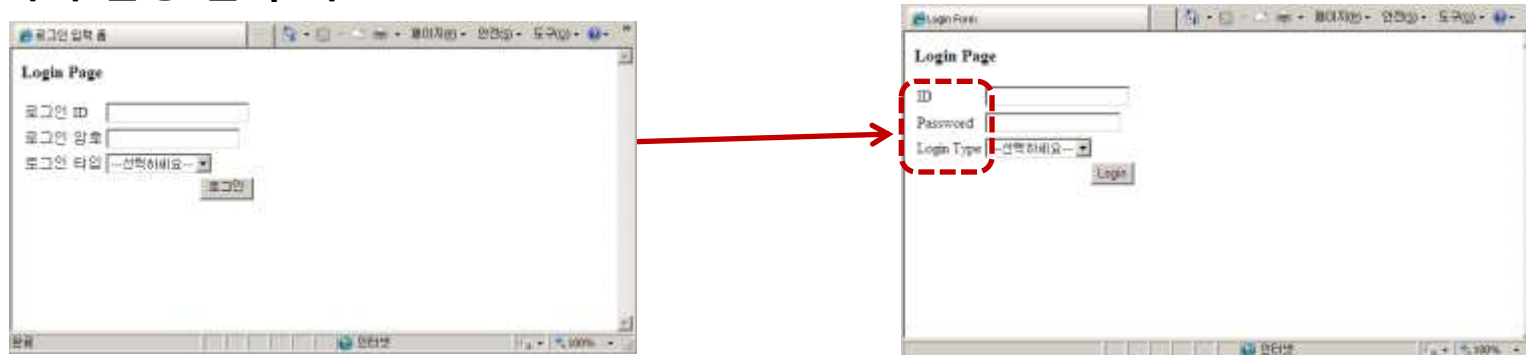


Exercise 1-3-5. 국제화 적용 결과 확인

- 로그인 페이지 locale 변경(en)

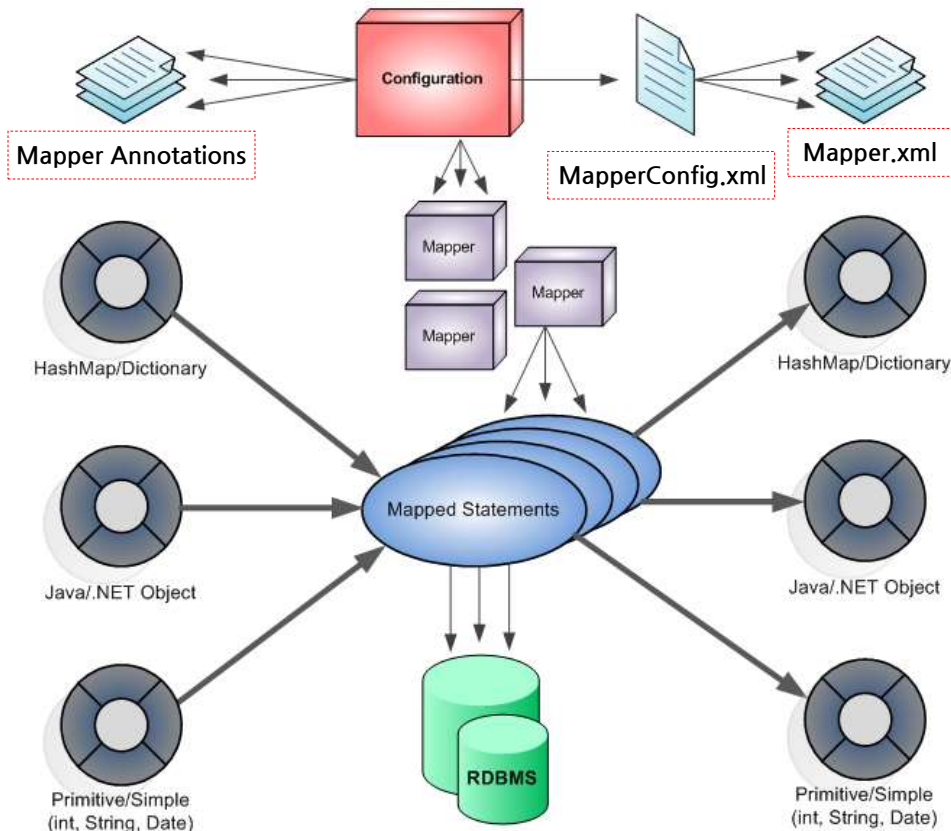
(<http://127.0.0.1:8080/lab301-mvc-tutor/loginProcess1.do?lang=en>)

- 예제 실행 결과 확인



❑ MyBatis 데이터 매퍼 서비스

- 개발자가 작성한 SQL문 혹은 저장프로시저 결과값을 자바 오브젝트에 자동 매핑하는 서비스
- 수동적인 JDBC 방식의 데이터 처리 작업 코드와는 달리 쿼리결과와 오브젝트 간 자동 매핑을 지원
- SQL문과 저장프로시저는 **XML 혹은 어노테이션 방식으로 작성 가능**



구성요소	설명
MapperConfig XML File	- MyBatis 동작을 위한 기본적인 설정을 공통으로 정의
Mapper XML File	- 실행할 SQL문 및 매핑 정보를 XML 방식으로 정의
Mapper Annotations	- 자바 코드 내에서 실행할 SQL문 및 매핑 정보를 어노테이션을 이용하여 정의
Parameter Object	- SQL문의 조건절에서 값을 비교하거나 INSERT, UPDATE절 등에서 필요한 입력값을 받아오기 위한 오브젝트
Result Object	- 쿼리 결과를 담아 리턴하기 위한 오브젝트

□ 주요 변경 사항

- iBatis의 SqlMapClient → SqlSession 변경

- SqlSession 인터페이스
 - MyBatis를 사용하기 위한 기본적인 인터페이스로, SQL문 처리를 위한 메서드를 제공
 - 구문 실행 메서드, 트랜잭션 제어 메서드 등 포함
 - selectList(), selectOne(), insert(), update(), delete(), commit(), rollback(), ...
 - SqlSessionFactory 클래스를 통해 MyBatis Configuration 정보에 해당 SqlSession 인스턴스를 생성

- 어노테이션 방식 설정 도입

- MyBatis는 본래 XML 기반의 프레임워크였으나, Mybatis 3.x 부터 어노테이션 방식의 설정을 지원
- Mapper XML File 내 SQL문 및 매핑 정보를, 자바 코드 내에서 어노테이션으로 그대로 적용 가능

- iBatis의 RowHandler → ResultHandler 변경

- ResultHandler 인터페이스
 - Result Object에 담겨 리턴된 쿼리 결과를 핸들링할 수 있도록 메서드 제공
 - 사용 예시) 대량의 데이터 처리 시, 처리 결과를 File로 출력하고자 할 때 혹은 Result Object의 형태를 Map 형태로 가져올 때

❑ Migrating from iBatis (1/2)

변경 또는 추가사항			iBatis 사용 시	MyBatis 사용 시
소스	패키지	변경	com.ibatis.*	org.apache.ibatis.*
	클래스	변경	SqlMapClient	SqlSession
		변경	SqlMapClientFactory	SqlSessionFactory
		변경	RowHandler	ResultHandler
Configuration XML File	DTD	변경	<!DOCTYPE sqlMapConfig PUBLIC "-//iBatis.com//DTD SQL Map Config 2.0//EN" "http://www.ibatis.com/dtd/sql-map-config-2.dtd">	<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
	요소와 속성	변경	< sqlMapConfig >	< configuration >
		변경	<settings ... />	<settings> <setting /> </settings>
		변경	<typeHandler callback="..." />	<typeHandlers> <typeHandler handler="..." /> </typeHandlers>
		추가	<transactionManager type="..."> <dataSource type="..." /> </transactionManager>	<environment id="..."> <transactionManager type="..." /> <dataSource type="..." /> </environment>
		추가		< typeAliases > <typeAlias alias="..." type="..." /> </typeAliases>
		변경	< sqlMap resource="..." />	< mappers > < mapper resource="..." /> </mappers>

❑ Migrating from iBatis (2/2)

변경 또는 추가사항			iBatis 사용 시	MyBatis 사용 시
Mapper XML File	DTD	변경	<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN" "http://www.ibatis.com/dtd/sql-map-2.dtd">	<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
	요소와 속성	변경	< sqlMap namespace="...">	< mapper namespace="...">
		변경	<cacheModel />	<cache />
		변경	-- SQL Statement <select id="..." resultClass resultMap ="..."> SELECT * FROM EMP </select> <insert id="..." parameterClass parameterMap ="..."> INSERT INTO EMP VALUES (#empNo# , #empName#); </insert>	<select id="..." resultType resultMap ="..."> SELECT * FROM EMP </select> <insert id="..." parameterType parameterMap ="..."> INSERT INTO EMP VALUES (#{empNo} , #{empName}); </insert>
		변경/추가	< dynamic > <isEqual /> <isNull /> ... </dynamic>	<if test="..." /> <choose> <when /> ... <otherwise /> <trim prefixOverrides suffixOverrides="..." /> <foreach collection="..." item="..." />
스프링연동	빈생성	변경	<bean id="sqlMapClient" class=" org.springframework.orm.ibatis.SqlMapClientFactoryBean ">	<bean id="sqlSession" class=" org.mybatis.spring.SqlSessionFactoryBean ">
쿼리 호출	파라미터	추가	-- Statement ID로 실행할 SQL문을 호출 List list = ...selectList(queryId, parameterObject);	-- Statement ID로 실행할 SQL문을 호출 List list = selectList(queryId, parameterObject); -- 메서드명으로 실행할 쿼리호출 List list = ...selectList(parameterObject);

❑ MyBatis를 활용한 Persistence Layer 개발

1) [MyBatis 설정 1] SQL Mapper XML 파일 작성 설정

- 실행할 SQL문과 관련 정보 설정
- SELECT/INSERT/UPDATE/DELETE, Parameter/Result Object, Dynamic SQL 등

2) [MyBatis 설정 2] MyBatis Configuration XML 파일 작성

- MyBatis 동작에 필요한 옵션을 설정
- <mapper>: SQL Mapper XML 파일의 위치

3) [스프링연동 설정] SqlSessionFactoryBean 정의

- Spring와 MyBatis 연동을 위한 설정
- 역할) MyBatis 관련 메서드 실행을 위한 SqlSession 객체를 생성
- dataSource, configLocation, mapperLocations 속성 설정

4) DAO 클래스 작성

- 방법1) SqlSessionDaoSupport를 상속하는 EgovAbstractMapper 클래스를 상속받아 확장/구현
 - 실행할 SQL문을 호출하기 위한 메서드 구현: SQL Mapping XML 내에 정의한 각 Statement id를 매개변수로 전달
- 방법2) DAO 클래스를 Interface로 작성하고, 각 Statement id와 메서드명을 동일하게 작성 (Mapper Interface 방식)
 - Annotation을 이용한 SQL문 작성 가능
 - 메서드명을 Statement id로 사용하기 때문에, 코드 최소화 가능

❑ [MyBatis 설정 1] SQL Mapper XML 파일 작성 (1/5)

- 실행할 SQL문과 Parameter Object와 Result Object 정보 등을 설정

SQL Mapper XML 설정

```

<?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="Dept"><----- 루트요소 <sqlMap>, namespace는 각 mapper 파일을 구분하기 위한 속성

  <resultMap id="deptResult" type="deptVO"><----- "deptVO"에 대한 Alias 설정은 Configuration 파일에 정의되어 있음
    <result property="deptNo" column="DEPT_NO" />
    <result property="deptName" column="DEPT_NAME" />
    <result property="loc" column="LOC" />
  </resultMap>

  <sql id="columns"> DEPT_NO, DEPT_NAME, LOC </sql>

  <select id="selectDept" parameterType="deptVO" resultMap="deptResult">
    <![CDATA[
      select <include refid="columns" />
      from DEPT
      where DEPT_NO = #{deptNo}<----- 파라미터 바인딩 시, #property# → #{property} 변경됨
    ]]>
  </select>
</mapper>

```

parameter/resultClass → parameter/resultType 변경됨

□ [MyBatis 설정 1] SQL Mapper XML 파일 작성 (2/5) - Dynamic SQL

- If

- if는 가장 많이 사용되는 Dynamic 요소로, test문의 true, false값에 따라 다양한 조건 설정이 가능
- SQL문의 다양한 위치에서 사용 가능하고, 선언된 if 조건에 따라 순서대로 test문을 수행

SQL Mapper XML 설정

```
<select id="selectEmpList" parameterType="empVO" resultType="empVO">
  <![CDATA[
    select EMP_NO as empNo,
           EMP_NAME as empName
    from EMP
    where JOB = 'Engineer'
  ]]>
  <if test="empNo != null">
    AND EMP_NO = #{empNo}
  </if>
  <if test="empName != null">
    AND EMP_NAME LIKE '%' || #{empName} || '%'
  </if>
</select>
```

❑ [MyBatis 설정 1] SQL Mapper XML 파일 작성 (3/5) - Dynamic SQL

- choose (when, otherwise)
 - 모든 조건을 적용하는 대신 한 가지 조건 만을 적용해야 할 필요가 있는 경우, choose 요소를 사용하며 이는 자바의 switch 구문과 유사한 개념임

SQL Mapper XML 설정

```
<select id="selectEmpList" parameterType="empVO" resultType="empVO">
  select *
  from EMP
  where JOB = 'Engineer'
  <choose>
    <when test="mgr != null">
      AND MGR like #{mgr}
    </when>
    <when test="empNo != null and empName != null">
      AND EMP_NAME like #{empName}
    </when>
    <otherwise>
      AND HIRE_STATUS = 'Y'
    </otherwise>
  </choose>
</select>
```

❑ [MyBatis 설정 1] SQL Mapper XML 파일 작성 (4/5) - Dynamic SQL

- trim (where, set)

- AND, OR, '와 같이 반복되는 문자를 자동적으로 trim(제거)
- 아래 예제의 <trim prefix="WHERE" prefixOverrides="AND|OR">은 <where>와 동일하게 동작

SQL Mapper XML 설정

```

<select id="selectEmpList" parameterType="empVO" resultType="empVO">
  select *
  from EMP
  where
  <if test="empNo != null">
    EMP_NO = #{empNo}
  </if>
  <if test="empName != null">
    AND EMP_NAME LIKE '%' || #{empName} || '%'
  </if>
</select>

<update id="updateEmp" parameterType="empVO">
  update EMP
  <trim prefix="SET" suffixOverrides=","> -- <set> 요소로 대체 가능
    <if test="empNo != null"> EMP_NO = #{empNo}, </if>
    <if test="empName != null"> EMP_NAME = #{empName} </if>
  </trim>
  ...
</update>

```

첫 번째 조건이 false, 두 번째 조건이 true일 경우, SQL Syntax Error!!

변경

<trim prefix="WHERE" prefixOverrides="AND|OR ">...</trim>
or
<where>

❑ [MyBatis 설정 1] SQL Mapper XML 파일 작성 (5/5) - Dynamic SQL

- foreach

- Map, List, Array에 담아 넘긴 값을 꺼낼 때 사용하는 요소

MyBatis Configuration XML 설정

```
<select id="selectforForeachFromList" parameterType="map" resultMap="egovMap">
  select
    <!-- List를 넘긴 경우, collection="list" / Array를 넘긴 경우, collection="array" 지정 -->
    <foreach collection="list" item="item" separator=", ">
      '${item}' as ${item}
    </foreach>

    <!-- Map 객체에 "collection"이라는 키로 List를 넘긴 경우 -->
    <foreach collection="collection" item="item" separator=", ">
      '${item}' as ${item}
    </foreach>
  from dual
</select>
```


□ [MyBatis 설정 2] MyBatis Configuration XML 파일 작성

- MyBatis 공통 설정 파일로, SqlSession 설정관련 상세 내역을 제어할 수 있는 메인 설정

MyBatis Configuration 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>
  <typeAliases>
    <typeAlias alias="deptVO" type="x.y.z.service.DeptVO" />
    <typeAlias alias="empVO" type="x.y.z.service.EmpVO" />
  </typeAliases>

  <mappers>
    <mapper resource="META-INF/sqlmap/mappers/lab-dept.xml" />
    <mapper resource="META-INF/sqlmap/mappers/lab-emp.xml" />
  </mappers>
</configuration>
```

요소	설명
properties	설정 파일내에서 \${key} 와 같은 형태로 외부 properties 파일을 참조할 수 있다.
settings	런타임시 MyBatis의 행위를 조정하기 위한 옵션 설정을 통해 최적화할 수 있도록 지원한다
typeAliases	타입 별칭을 통해 자바타입에 대한 좀더 짧은 이름을 사용할 수 있다. 오직 XML 설정에서만 사용되며, 타이핑을 줄이기 위해 사용된다.
typeHandlers	javaType과 jdbcType 일치를 위해 TypeHandler 구현체를 등록하여 사용할 수 있다.
environments	환경에 따라 MyBatis 설정을 달리 적용할 수 있도록 지원한다.
mappers	매핑할 SQL 구문이 정의된 파일을 지정한다.

□ [스프링연동 설정] SqlSessionFactoryBean 정의

- Spring와 MyBatis 연동을 위한 설정으로, MyBatis 관련 메서드 실행을 위해 SqlSession 객체가 필요
- 스프링에서 SqlSession 객체를 생성하고 관리할 수 있도록, SqlSessionFactoryBean을 정의
 - id와 class는 고정값
 - dataSource : 스프링에서 설정한 DataSource Bean id를 설정하여 MyBatis가 DataSource를 사용하게 한다.
 - configLocation : MyBatis Configuration XML 파일이 위치하는 곳을 설정한다.
 - mapperLocations : SQL Mapper XML 파일을 일괄 지정할 수 있다. 단, Configuration 파일에 중복 선언할 수 없다.

SqlSessionFactoryBean 설정

```
<!-- SqlSession setup for MyBatis Database Layer -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="configLocation" value="classpath:/META-INF/sqlmap/config/sql-mapper-config.xml" />
  <!-- <property name="mapperLocations" value="classpath:/META-INF/sqlmap/mappers/*.xml" /> -->
</bean>
```

❑ MyBatis를 활용한 자바클래스 작성 1

- EgovAbstractMapper 클래스를 상속받아 DAO 클래스를 작성

DAO class 샘플 코드

```
@Repository("empMapper")
public class EmpMapper extends EgovAbstractMapper {

    public void insertEmp(EmpVO vo) {
        insert("insertEmp", vo);
    }

    public int updateEmp(EmpVO vo) {
        return update("updateEmp", vo);
    }

    public int deleteEmp(EmpVO vo) {
        return delete("deleteEmp", vo);
    }

    public EmpVO selectEmp(EmpVO vo) {
        return (EmpVO) selectByPk("selectEmp", vo);
    }

    public List<EmpVO> selectEmpList(EmpVO searchVO) {
        return list("selectEmpList", searchVO);
    }
}
```

EgovAbstractMapper는 SqlSessionDaoSupport의 하위 클래스로, SqlSession 설정과 메소드 호출의 편리함을 제공한다

SQL Mapper XML 내에 정의한 각 Statement id를 이용하여 실행할 SQL문을 호출하도록 작성한다

❑ MyBatis를 활용한 자바클래스 작성 2

- DAO 클래스 대신 Interface 작성 (Mapper Interface 방식) (1/4)

- 기존 DAO 클래스의 MyBatis 메소드 호출 코드를 최소화시킨 방법으로, 각 Statement id와 메서드명을 동일하게 작성하면 MyBatis가 자동으로 SQL문을 호출한다.
- 실제 내부적으로 MyBatis는 풀네임을 포함한 메서드명을 Statement id로 사용한다.

Mapper Interface 샘플 코드

```
package x.y.z.mapper;
```

```
@Mapper("deptMapper")  
public interface DeptMapper {
```

```
    public void insertDept(DeptVO vo);
```

```
    public int updateDept(DeptVO vo);
```

```
    public int deleteDept(DeptVO vo);
```

```
    public DeptVO selectDept(DeptVO vo);
```

```
    public List<DeptVO> selectDeptList(DeptVO vo);
```

```
}
```

Mapper Interface를 자동 스캔하기 위한 기능으로,
MapperConfigurer 빈설정과 함께 사용한다

Service 클래스에서 메소드를 호출하면,
메서드명과 일치하는 SQL문을 자동 실행한다

❑ MyBatis를 활용한 자바클래스 작성 2

- DAO 클래스 대신 Interface 작성 (Mapper Interface 방식) (2/4)

- 이 때 SQL Mapper XML 파일의 namespace값을 해당 Mapper의 풀네임으로 설정해야 한다.
- MyBatis는 해당 Mapper의 풀네임과 일치하는 namespace에서 메서드명과 동일한 id를 가진 Statement를 호출한다.
- namespace : 각 SQL Mapper XML을 구분

SQL Mapper XML 설정 1

```
<mapper namespace="x.y.z.mapper.DeptMapper">
  <insert id="insertDept" parameterType="deptVO">...</insert>
  <update id="updateDept" parameterType="deptVO">...</update>
  <delete id="deleteDept" parameterType="deptVO">...</delete>
  <select id="selectDept" parameterType="deptVO" resultMap="deptResult">...</select>
  <select id="selectDeptList" parameterType="deptVO" resultMap="deptResult">...</select>
</mapper>
```

SQL Mapper XML 설정 2

```
<mapper namespace="x.y.z.mapper.EmpMapper">
  <insert id="insertEmp" parameterType="empVO">...</insert>
  <update id="updateEmp" parameterType="empVO">...</update>
  <delete id="deleteEmp" parameterType="empVO">...</delete>
  <select id="selectEmp" parameterType="empVO" resultMap="empResult">...</select>
  <select id="selectEmpList" parameterType="empVO" resultMap="empResult">...</select>
</mapper>
```

❑ MyBatis를 활용한 자바클래스 작성 2

- DAO 클래스 대신 Interface 작성 (Mapper Interface 방식) (3/4)

- @Mapper를 사용하여 Mapper Interface가 동작하도록 하려면, MapperConfigurer 클래스를 빈으로 등록한다.
- MapperConfigurer는 @Mapper를 자동 스캔하고, MyBatis 설정의 편리함을 제공한다.
- basePackage : 스캔 대상에 포함시킬 Mapper Interface가 속한 패키지를 지정

MapperConfigurer 설정

```
<!-- MapperConfigurer setup for MyBatis Database Layer -->
<bean class="egovframework.rte.psl.dataaccess.mapper.MapperConfigurer">
  <property name="basePackage" value="x.y.z.mapper" />
</bean>
```

❑ MyBatis를 활용한 자바클래스 작성 2

- DAO 클래스 대신 Interface 작성 (Mapper Interface 방식) (4/4) - 어노테이션을 이용한 SQL문 작성
 - 인터페이스 메소드 위에 @Statement(Select, Insert, Update, Delete ...)를 선언하여 쿼리를 작성한다.
 - SQL Mapper XML을 작성할 필요가 없으나, Dynamic 쿼리를 사용하지 못하고 쿼리의 유연성이 떨어진다.

Mapper Interface 샘플 코드

```
@Mapper("deptMapper")
public interface DeptMapper {

    @Select("select DEPT_NO as deptNo, DEPT_NAME as deptName, LOC as loc from DEPT where DEPT_NO = #{deptNo}")
    public DeptVO selectDept(BigDecimal deptNo);

    @Insert("insert into DEPT(DEPT_NO, DEPT_NAME, LOC) values (#{deptNo}, #{deptName}, #{loc})")
    public void insertDept(DeptVO vo);

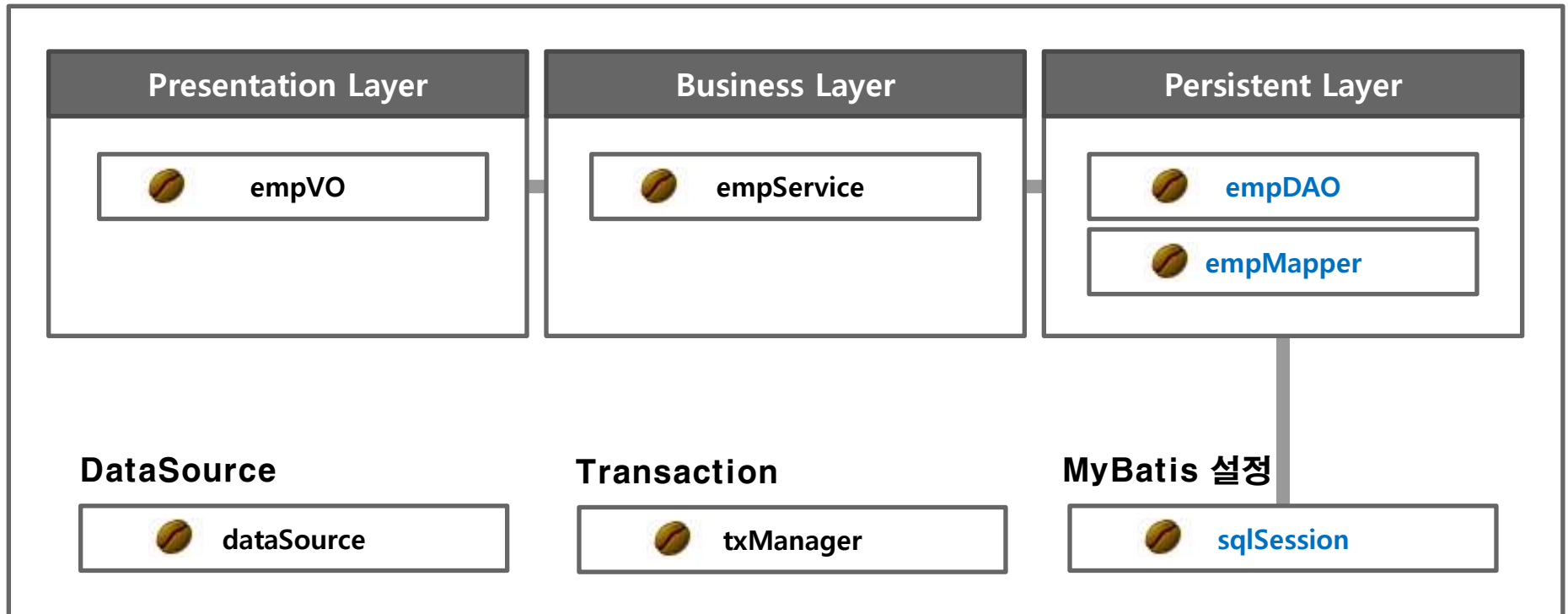
    @Update("update DEPT set DEPT_NAME = #{deptName}, LOC = #{loc} WHERE DEPT_NO = #{deptNo}")
    public int updateDept(DeptVO vo);

    @Delete("delete from DEPT WHERE DEPT_NO = #{deptNo}")
    public int deleteDept(BigDecimal deptNo);

}
```

LAB 205-MyBatis 실습

❑ lab205-mybatis 프로젝트의 beans



1. 구동 환경 설정

❑ 1. Hsqldb 초기화 스크립트

- /lab205-mybatis/src/test/resources/META-INF/testdata/sample_schema_hsql.sql 를 확인한다.
- 현 실습 프로젝트에서는 편의상 매 테스트 케이스 재실행 시 관련 table 을 drop/create 하고 있음.

❑ 2. dataSource 설정 (테스트 편의성을 위해 memory DB 사용)

- <bean id="dataSource" .../>를 이용한 방법
- <jdbc:embedded-database .../>를 이용한 방법

1. 구동 환경 설정

❑ 2. dataSource 설정

- /lab205-mybatis/src/test/resources/META-INF/spring/context-datasource.xml 에 dataSource 빈 설정을 추가한다.
- 스프링에서 제공하는 EmbeddedDataBase를 생성하여 테스트한다.

```
<!-- TODO [Step 1-2] DataSource 설정 -->
<jdbc:embedded-database id="dataSource" type="HSQL">
<jdbc:script location= "META-INF/testdata/sample_schema_hsql.sql"/>
</jdbc:embedded-database>
```

1. 구동 환경 설정

❑ 3. transaction 설정

- /lab205-mybatis/src/test/resources/META-INF/spring/context-transaction.xml 를 작성한다.

```
<!-- TODO [Step 1-3] transaction 설정 -->
<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
<tx:annotation-driven transaction-manager="txManager" />
```

cf.) 여기서는 transaction manager 와 메소드 혹은 클래스 레벨에 @Transactional 을 선언하여 트랜잭션 서비스를 이용한다. @Transactional Annotation 스캔을 위해서는 <tx:annotation-driven />을 선언해야 한다.
@Transactional을 이용하면 대상 메소드에 개별적으로 트랜잭션을 지정할 수 있다는 장점이 있으나, 보통 AOP 형식(tx:aop)으로 선언하여 트랜잭션 대상 메서드들에 일괄 지정하는 경우가 많다.

1. 구동 환경 설정

❑ 4. Spring 의 MyBatis 연동 설정

- /lab205-mybatis/src/test/resources/META-INF/spring/context-mybatis.xml 를 작성한다.

```
<!-- TODO [Step 1-4] MyBaits와 Spring 연동 설정 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:/META-INF/sqlmap/sql-mybatis-
config.xml" />
    <!-- <property name="mapperLocations" value="classpath:*/lab-*.xml" /> -->
</bean>
```

- 최신 프레임워크 환경에서는 sql-mybatis-config.xml 내에 개별 sql 맵핑 파일을 일일이 지정하는 것이 아니라, 위의 mapperLocations 영역을 주석 해제하여 Spring 의 ResourceLoader 형식으로 패턴 매칭에 의거한 일괄 로딩으로 처리가 가능하다. (단, 테스트 결과 CacheModel 등의 일부 기능에서 문제가 발생하는 경우가 있으므로 사용에 유의할것!!)

1. 구동 환경 설정

❑ 5. MyBatis 의 sql-mybatis-config 설정 파일 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/sql-mybatis-config.xml 를 작성한다.

```
<?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>
<!-- TODO [Step 1-5] MyBatis Configuration File 작성 -->
<typeAliases>
<typeAlias alias="empVO" type="egovframework.lab.dataaccess.service.EmpVO" />
</typeAliases>

<!-- MyBatis 연동을 위한 SqlSessionFactoryBean 정의 시 mapperLocations 속성으로
한 번에 모든 Mapper XML File을 설정할 수 있다.
(<property name="mapperLocations" value="classpath:*/lab-*.xml" /> 추가)
단, 아래 <mappers> 설정과 mapperLocations 설정 중 한가지만 선택해야 한다.
-->
<mappers>
<mapper resource="META-INF/sqlmap/mappers/lab-dao-class.xml" />
<mapper resource="META-INF/sqlmap/mappers/lab-mapper-interface.xml" />
</mappers>

</configuration>
```

1. 구동 환경 설정

❑ 6. ID Generation Service 설정 확인

- /lab205-mybatis/src/test/resources/META-INF/spring/context-idgen.xml 를 확인한다.

```
<!-- [Step 1-6] Id Generation Service 설정 -->
<bean name="primaryTypeSequenceIds"
class="egovframework.rte.fdl.idgnr.impl.EgovSequenceIdGnrService"
destroy-method="destroy">
<property name="dataSource" ref="dataSource" />
<property name="query" value="SELECT NEXT VALUE FOR empseq FROM DUAL" />
</bean>
```

- 여기서는 Hsqldb 를 사용하여 Oracle 의 DUAL 테이블 역할을 할 수 있도록 초기화 스크립트 sql 에 create 하였으며, DB Sequence 기반의 Id Generation 을 사용한 예이다. (위에서 select next value for seq_id from xx 는 Hsqldb 의 특화된 sequence 사용 문법임에 유의!)

1. 구동 환경 설정

□ 7. common설정 확인

- /lab205-mybatis/src/test/resources/META-INF/spring/context-common.xml 을 확인한다.
- **PropertyPlaceholderConfigurer 설정** : 외부 properties 파일을 Container 구동 시 미리 Spring Bean 설정 파일의 속성값으로 대체하여 처리해주는 PropertyPlaceholderConfigurer 설정
- **MessageSource 설정** : Locale 에 따른 다국어 처리를 쉽게 해주는 messageSource 설정. 여기서는 전자정부 실행환경의 id generation 서비스와 properties 서비스의 메시지 파일과 업무 어플리케이션을 위한 사용자 메시지(/message/message-common - message-common_en_US.properties, message-common_ko_KR.properties 를 확인할 것) 를 지정하였다.
- **전자정부 TraceHandler 설정 관련** : exception 처리 Handler 와 유사하게 특정한 상황에서 사용자가 Trace Handler 를 지정하여 사용할 수 있도록 전자정부 프레임워크에서 가이드하고 있는 TraceHandler 설정
- **component-scan 설정** : 스테레오 타입 Annotation 을 인식하여 Spring bean 으로 자동 등록하기 위한 component-scan 설정

1. 구동 환경 설정

❑ 8. aspect 설정 확인 (1/2)

- /lab205-mybatis/src/test/resources/META-INF/spring/context-aspect.xml 를 확인한다.

```
<aop:config>
  <aop:pointcut id="serviceMethod"
    expression="execution(* egovframework.lab..impl.*Impl.*(..))" />

  <aop:aspect ref="exceptionTransfer">
    <aop:after-throwing throwing="exception"
      pointcut-ref="serviceMethod" method="transfer" />
  </aop:aspect>
</aop:config>

<bean id="exceptionTransfer" class="egovframework.rte.fdl.cmmn.aspect.ExceptionTransfer">
  <property name="exceptionHandlerService">
    <list>
      <ref bean="defaultExceptionHandlerManager" />
    </list>
  </property>
</bean>

... (계속)
```

1. 구동 환경 설정

□ 8. aspect 설정 확인 (2/2)

... (이어서)

```
<bean id="defaultExceptionHandlerManager"
class="egovframework.rte.fdl.cmmn.exception.manager.DefaultExceptionHandlerManager">
    <property name="reqExpMatcher" ref="antPathMater" />
    <property name="patterns">
        <list>
            <value>**service.impl.*</value>
        </list>
    </property>
    <property name="handlers">
        <list>
            <ref bean="egovHandler" />
        </list>
    </property>
</bean>

<bean id="egovHandler"
class="egovframework.lab.dataaccess.common.JdbcLoggingExcepHndlr" />
```

- Spring AOP(xml 설정 방식) 를 사용하여 비즈니스 메서드에서 exception 이 발생한 경우 일괄적으로 ExceptionTransfer 의 transfer 메서드 기능(Advice) 를 수행해 주게 됨. --> Exception logging 및 BizException 형태로 wrapping 하여 재처리하는 Exception 공통처리 후 ExceptionHandleManager 에 의해 관리(설정) 되는 Handler (ex. exception 내용을 메일링 한더던지.. 사용자 구현 가능) 가 자동적으로 추가 수 행될 수 있음.

2. 자바 클래스 작성

❑ 1. Service Interface 확인

- /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/EmpService.java 를 확인한다.

❑ 2. VO 확인

- /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/EmpVO.java 를 확인한다.
- (참고) Getter와 Setter 메서드는 다음과 같이 생성할 수 있다.

우클릭 > Source > Generate getters and setters > Select All 선택 > OK

2. 자바 클래스 작성

□ 3. Annotation을 적용한 Impl 작성

- /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpServiceImpl.java 에 아래 내용을 추가한다.

```
@Service("empService")
public class EmpServiceImpl extends EgovAbstractServiceImpl implements EmpService {

    // TODO [Step 2-1] EmpServiceImpl 작성 추가

    @Resource(name = "empDAO")
    private EmpDAO empDAO;

    // TODO [Step 3-1] EmpServiceImpl 변경
    // EmpMapper를 사용하도록 주석 변경
    // @Resource(name = "empMapper")
    // EmpMapper empDAO;
```

- Ctrl + Shift + O 를 눌러서 import 되지 않은 클래스들 import 시킨다.
- 위 방법을 통해서도 import 되지 않은 클래스가 남아있으면, 이클립스 상단에 Project > Clean을 수행한다.

2. 자바 클래스 작성

□ 4-1. DAO 작성

- /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpDAO.java 를 작성한다.

```
@Repository("empDAO")
public class EmpDAO extends EgovAbstractMapper {
    // TODO [Step 2-2] EmpDAO 작성 (EgovAbstractMapper 상속한 DAO)

    public void insertEmp(EmpVO vo) {
        insert("Emp.insertEmp", vo);
    }
    public int updateEmp(EmpVO vo) {
        return update("Emp.updateEmp", vo);
    }
    public int deleteEmp(EmpVO vo) {
        return delete("Emp.deleteEmp", vo);
    }
    public EmpVO selectEmp(EmpVO vo) {
        return (EmpVO) selectByPk("Emp.selectEmp", vo);
    }
    @SuppressWarnings("unchecked")
    public List<EmpVO> selectEmpList(EmpVO searchVO) {
        return (List<EmpVO>) list("Emp.selectEmpList", searchVO);
    }
}
```

2. 자바 클래스 작성

❑ 4-2-1. Mapper Interface 작성

- /lab205-mybatis/src/main/java/egovframework/lab/dataaccess/service/impl/EmpMapper.java 를 작성한다.
- 아래와 같이 MyBatis에서는 Dao 클래스 대신 Interface를 이용해 데이터 처리가 가능하다.

```
@Mapper("empMapper")
public interface EmpMapper {

    // TODO [Step 2-2] EmpMapper 작성 (Mapper Interface)

    public void insertEmp(EmpVO vo);

    public int updateEmp(EmpVO vo);

    public int deleteEmp(EmpVO vo);

    public EmpVO selectEmp(EmpVO vo);

    public List<EmpVO> selectEmpList(EmpVO searchVO);

}
```

2. 자바 클래스 작성

❑ 4-2-2. @Mapper 스캔 설정

- /lab205-mybatis/src/test/resources/META-INF/spring/context-mybatis.xml 에 다음을 추가한다.

```
<!-- MapperConfigurer setup for @Mapper -->
<!-- TODO [Step 2-3] MyBatis의 Mapper Interface 자동스캔 설정 -->

<bean class="egovframework.rte.psl.dataaccess.mapper.MapperConfigurer">
<property name="basePackage" value="egovframework.lab.dataaccess.service.impl" />
</bean>
```

3. SQL 매핑 파일 작성

❑ 1. mapping xml 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<?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="Emp">

<!-- TODO [Step 3-1] lab-dao-class.xml 작성 (EgovAbstractMapper 상속한 DAO) -->
<resultMap id="empResult" type="empVO">
<id property="empNo" column="EMP_NO" />
<result property="empName" column="EMP_NAME" />
<result property="job" column="JOB" />
<result property="mgr" column="MGR" />
<result property="hireDate" column="HIRE_DATE" />
<result property="sal" column="SAL" />
<result property="comm" column="COMM" />
<result property="deptNo" column="DEPT_NO" />
</resultMap>
```

- 다음 슬라이드에서 계속...

3. SQL 매핑 파일 작성

❑ 1. mapping xml 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<insert id="insertEmp" parameterType="empVO">
<![CDATA[
insert into EMP (EMP_NO, EMP_NAME, JOB, MGR, HIRE_DATE, SAL, COMM, DEPT_NO)
values("#{empNo}", "#{empName}", "#{job}", "#{mgr}", "#{hireDate}", "#{sal}", "#{comm}", "#{deptNo}")
]]>
</insert>

<update id="updateEmp" parameterType="empVO">
<![CDATA[
updateEMP
setEMP_NAME = #{empName},
JOB = #{job},
MGR = #{mgr},
HIRE_DATE = #{hireDate},
SAL = #{sal},
COMM = #{comm},
DEPT_NO = #{deptNo}
whereEMP_NO = #{empNo}
]]>
</update>
```

- 다음 슬라이드에서 계속...

3. SQL 매핑 파일 작성

❑ 1. mapping xml 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<delete id="deleteEmp" parameterType="empVO">
<![CDATA[
delete from EMP
where EMP_NO = #{empNo}
]]>
</delete>

<select id="selectEmp" parameterType="empVO" resultMap="empResult">
<![CDATA[
selectEMP_NO, EMP_NAME, JOB, MGR, HIRE_DATE, SAL, COMM, DEPT_NO
fromEMP
whereEMP_NO = #{empNo}
]]>
</select>
```

- 다음 슬라이드에서 계속...

3. SQL 매핑 파일 작성

❑ 1. mapping xml 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<select id="selectEmpList" parameterType="empVO" resultMap="empResult">
<![CDATA[
selectEMP_NO, EMP_NAME, JOB, MGR, HIRE_DATE, SAL, COMM, DEPT_NO
fromEMP
where 1 = 1
]]>
<if test="empNo != null">
AND EMP_NO = #{empNo}
</if>
<if test="empName != null">
AND EMP_NAME LIKE '%' || #{empName} || '%'
</if>
</select>
</mapper>
```

- 다음 슬라이드에서 계속...

3. SQL 매핑 파일 작성

❑ 1. mapping xml 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-dao-class.xml 를 작성한다.

```
<select id="selectEmpList" parameterClass="empVO" resultMap="empResult">
<![CDATA[
select EMP_NO,
      EMP_NAME,
      JOB,
      MGR,
      HIRE_DATE,
      SAL,
      COMM,
      DEPT_NO
from EMP
where 1 = 1
]]>
<isNull prepend="and" property="empNo">
EMP_NO = #empNo#
</isNull>
<isNull prepend="and" property="empName">
EMP_NAME LIKE '%' || #empName# || '%'
</isNull>
</select>

</sqlMap>
```

3. SQL 매핑 파일 작성

❑ 2. mapping xml 작성

- /lab205-mybatis/src/test/resources/META-INF/sqlmap/mappers/lab-mapper-interface.xml 를 작성한다.
- /lab-dao-class.xml과 <mapper>의 name 속성값만 다르다.

```
<mapper namespace="egovframework.lab.dataaccess.dao.service.impl.EmpMapper">
```

```
<!-- TODO [Step 3-2] lab-mapper-interface.xml 작성 (Mapper Interface) -->
```

- DAO 클래스의 Statement 호출 방식: 사용자가 직접 지정해준 ID 파라미터 값과 일치하는 Statement를 호출. 동일한 Statement ID가 있으면, <mapper>의 namespace를 지정한다.
 - namespace=A, statement id=insertEmp → A.insertEmp으로 호출
 - namespace=B, statement id=insertEmp → B.insertEmp으로 호출
- Mapper 인터페이스의 Statement 호출 방식: 메소드명과 일치하는 Statement를 자동 호출. 이 때 MyBatis는 호출된 메서드가 포함된 인터페이스의 풀네임을 namespace 값으로 사용하기 때문에, 반드시 namespace 값을 지정해주어야 한다.
 - namespace=x.y.z.EmpMapper, statement id=insertEmp → 내부적으로 x.y.z.EmpMapper.insertEmp을 호출

4. 테스트 케이스 확인

❑ 1. EmpServiceTest 확인 – EmpDAO 테스트

- /lab205-mybatis/src/test/java/egovframework/lab/dataaccess/service/EmpServiceTest.java 를 확인하고 실행한다.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "classpath*:META-INF/spring/context-*" })
@TransactionConfiguration(transactionManager = "txManager", defaultRollback = false)
@Transactional
public class EmpServiceTest {

    // TODO [Step 4-1] EmpServiceTest 실행
    @Resource(name = "dataSource")
    DataSource dataSource;

    @Resource(name = "empService")
    EmpService empService;

    @Before
    public void onSetUp() throws Exception {
        // 편의상 각 테스트 메서드 수행 전에
        // 외부의 스크립트 파일(sample_schema_hsql.sql)로 DB를 초기화하도록 설정
        JdbcTestUtils.executeSqlScript(new JdbcTemplate(dataSource), new ClassPathResource("META-INF/testdata/sample_schema_hsql.sql"), true);
    }
}
```

4. 테스트 케이스 확인

```
/**
 * 사원정보 생성
 *
 * @throws ParseException
 */
public EmpVO makeVO() throws ParseException {
    EmpVO vo = new EmpVO();

    // empNo는 Biz. 서비스 내에서 IDGeneration Service 에 의해 key를 생성하고 설정
    vo.setEmpName("홍길동");
    vo.setJob("개발자");
    vo.setMgr(new BigDecimal(7902));
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd",
        java.util.Locale.getDefault());
    vo.setHireDate(sdf.parse("2009-07-09"));
    vo.setSal(new BigDecimal(1000));
    vo.setComm(new BigDecimal(0));
    vo.setDeptNo(new BigDecimal(20));

    return vo;
}

public void checkResult(EmpVO vo, EmpVO resultVO) {
    assertNotNull(resultVO);
    assertEquals(vo.getEmpNo(), resultVO.getEmpNo());
    assertEquals(vo.getEmpName(), resultVO.getEmpName());
    assertEquals(vo.getJob(), resultVO.getJob());
    assertEquals(vo.getMgr(), resultVO.getMgr());
    assertEquals(vo.getHireDate(), resultVO.getHireDate());
    assertEquals(vo.getSal(), resultVO.getSal());
    assertEquals(vo.getComm(), resultVO.getComm());
    assertEquals(vo.getDeptNo(), resultVO.getDeptNo());
}
```

4. 테스트 케이스 확인

```
/** 사원정보 입력 */
@Test
public void testInsertEmp() throws Exception {
    EmpVO vo = makeVO();

    // insert
    BigDecimal empNo = empService.insertEmp(vo);
    vo.setEmpNo(empNo);

    // select
    EmpVO resultVO = empService.selectEmp(vo);

    // check
    checkResult(vo, resultVO);
}

/** 사원정보 수정 */
@Test
public void testUpdateEmp() throws Exception {
    EmpVO vo = makeVO();

    // insert
    BigDecimal empNo = empService.insertEmp(vo);
    vo.setEmpNo(empNo);

    // data change
    vo.setEmpName("홍길순");
    vo.setJob("설계자");

    // update
    empService.updateEmp(vo);

    // select
    EmpVO resultVO = empService.selectEmp(vo);

    // check
    checkResult(vo, resultVO);
}
```


4. 테스트 케이스 확인

```
/** 사원정보 삭제 */
@Test
public void testDeleteEmp() throws Exception {
    EmpVO vo = makeVO();

    // insert
    BigDecimal empNo = empService.insertEmp(vo);
    vo.setEmpNo(empNo);

    // delete
    empService.deleteEmp(vo);

    // select
    try {
        @SuppressWarnings("unused")
        EmpVO resultVO = empService.selectEmp(vo);
        fail("EgovBizException 이 발생해야 합니다.");
    } catch (Exception e) {
        assertNotNull(e);
        // 여기서는 비즈니스 단에서 명시적으로 exception 처리하였음
        // AbstractServiceImpl 을 extends 하고
        // processException("info.nodata.msg"); 과 같이 메서드 콜 형태로 처리
        assertTrue(e instanceof EgovBizException);
        assertEquals("info.nodata.msg",
            ((EgovBizException) e).getMessageKey());
        assertEquals("해당 데이터가 없습니다.", e.getMessage());
    }
}
```

4. 테스트 케이스 확인

```
/** 사원정보 목록조회 */
@Test
public void testSelectEmpList() throws Exception {
    EmpVO vo = makeVO();

    // insert
    BigDecimal empNo = empService.insertEmp(vo);
    vo.setEmpNo(empNo);

    // 검색조건으로 empNo 설정
    EmpVO searchVO = new EmpVO();
    searchVO.setEmpNo(vo.getEmpNo());

    // selectList
    List<EmpVO> resultList = empService.selectEmpList(searchVO);

    // empNo 조건에 대한 결과는 1건일 것임
    assertNotNull(resultList);
    assertTrue(resultList.size() > 0);
    assertEquals(1, resultList.size());
    checkResult(vo, resultList.get(0));

    // 검색조건으로 empName 설정 - '%' || #{empName} || '%'
    EmpVO searchVO2 = new EmpVO();
    searchVO2.setEmpName(""); // '%' || '' || '%' // --> '%%'

    // selectList
    List<EmpVO> resultList2 = empService.selectEmpList(searchVO2);

    // like 조건에 대한 결과는 1건 이상일 것임
    assertNotNull(resultList2);
    assertTrue(resultList2.size() > 0);
}
}
```

4. 테스트 케이스 확인

❑ 2. EmpServiceTest 확인 – EmpMapper 테스트

- /lab205-mybatis/src/test/java/egovframework/lab/dataaccess/service/impl/EmpServiceImpl.java 를 다음과 같이 수정한 후, EmpServiceTest를 다시 실행해본다.

```
@Service("empService")
public class EmpServiceImpl extends EgovAbstractServiceImpl implements EmpService {

    // TODO [Step 2-1] EmpServiceImpl 추가 작성
    // @Resource(name = "empDAO")
    // public EmpDAO empDAO;

    // TODO [Step 4-2] EmpServiceImpl 변경
    // EmpMapper를 사용하도록 주석 변경
    @Resource(name = "empMapper")
    EmpMapper empDAO;
```