

---

# Spring Framework

1. Web Technology
2. Spring & myBatis Framework

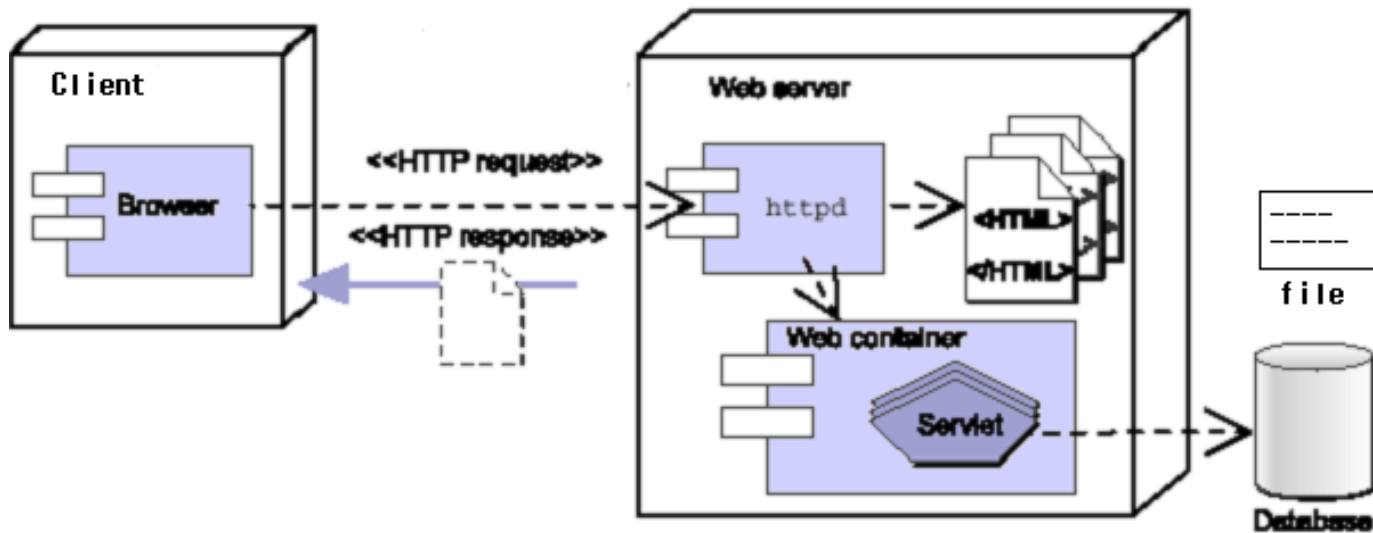
---

# ***1. Web Technology***

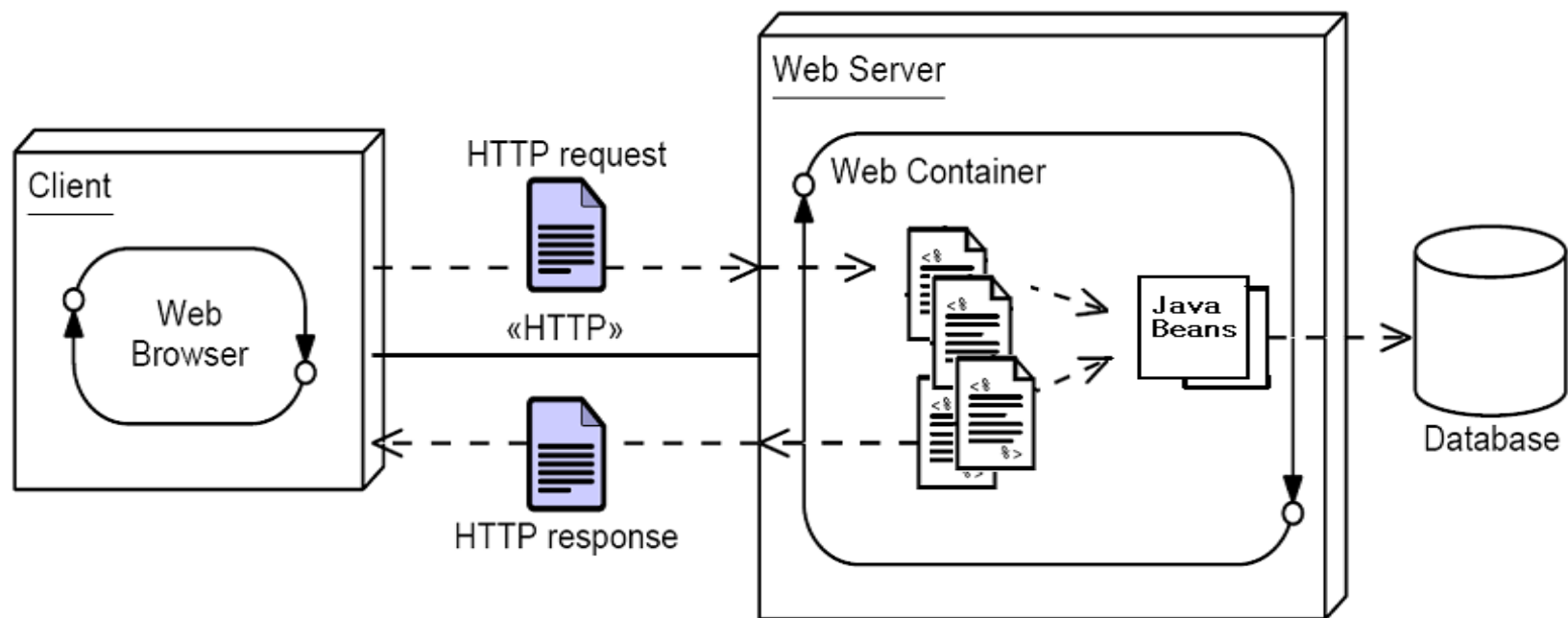
---

# Servlet

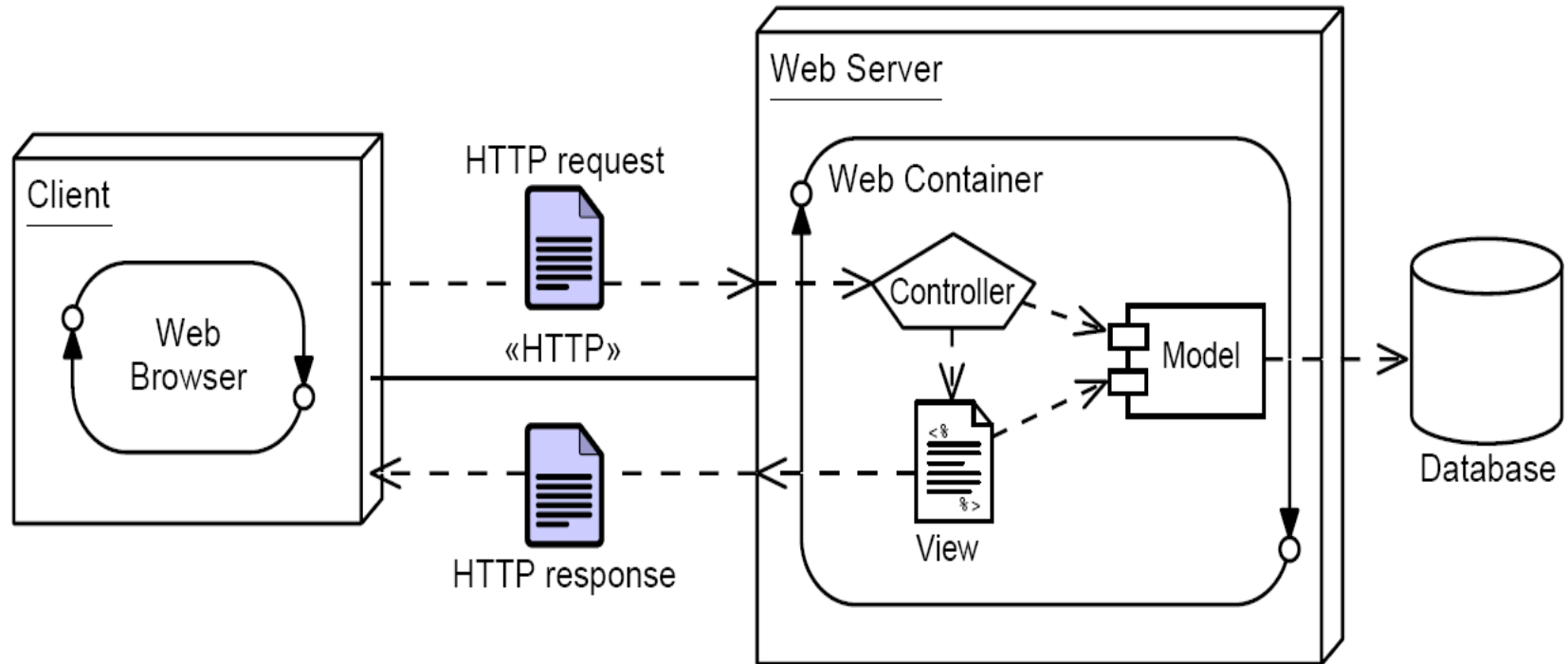
- CGI와 fast CGI의 성능저하를 개선하기 위해 Sun Microsystems에서 내놓은 Java 기반 웹 어플리케이션 기술
- 서블릿 요청 시 마다 thread를 생성, process보다 가볍고, 여러 thread가 동시에 요청을 처리함으로써 병목현상 없이 효과적인 서비스제공



# Model 1 Architecture



# Model 2 Architecture

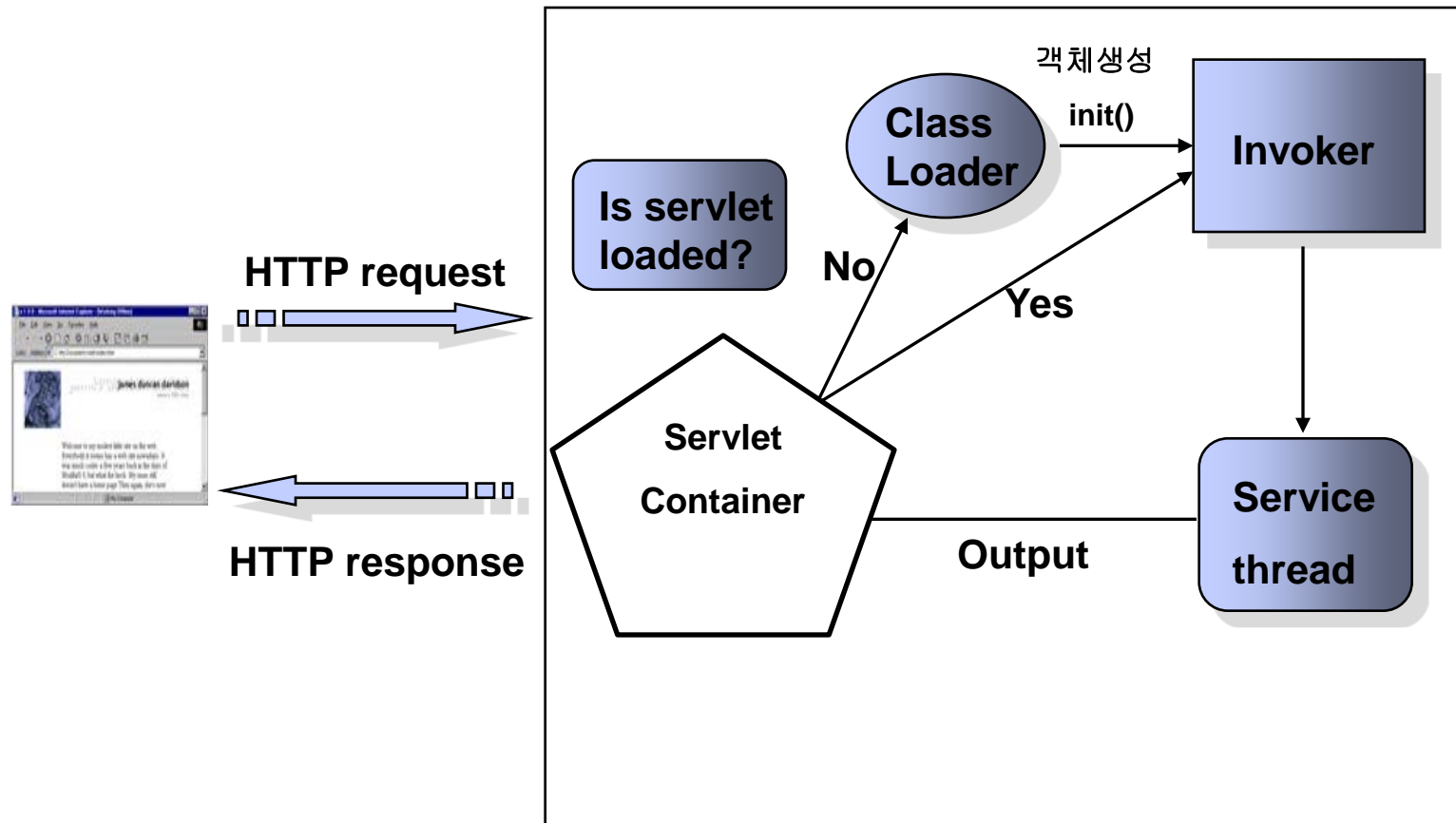


# Servlet Test

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException{
        res.setContentType("text/html");
        try{
            PrintWriter out = res.getWriter();
            out.println("<html>");
            out.println("<head><title>Servlet Test</title></head>");
            out.println("<h1>Servlet Test a a a !</h1>");
            out.println("</body></html>");
            out.close();
            //throw new IOException(); //log 파일 기록 확인
        }catch(IOException ioe){
            getServletContext().log("Error in HelloWorld",ioe);
        }
    }
}
```

# Servlet Life Cycle



# Servlet Life Cycle

## ■ init()

- 객체 생성 직후 한번 call
- 객체 초기화

## ■ service()

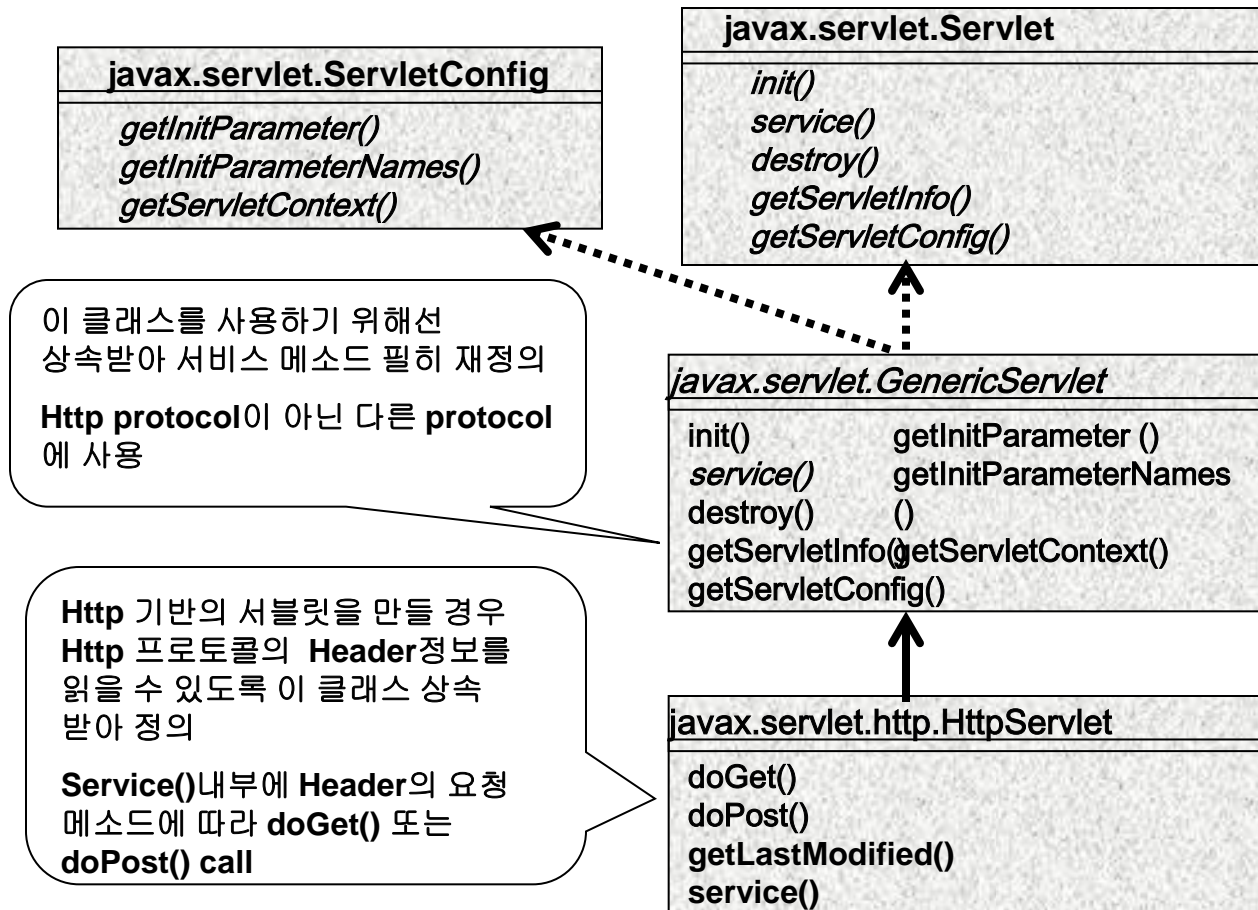
- 매 요청 시 마다 호출- thread 단위로 요청에 대한 서비스
- Multi- thread 환경
- 모든 서비스들이 공유하는 멤버 데이터의 **concurrency** 문제 고려 요
- HTTP 프로토콜을 처리하기 위해 **get, post** 방식등을 처리할 수 있도록 Overriding된다.

## ■ destroy()

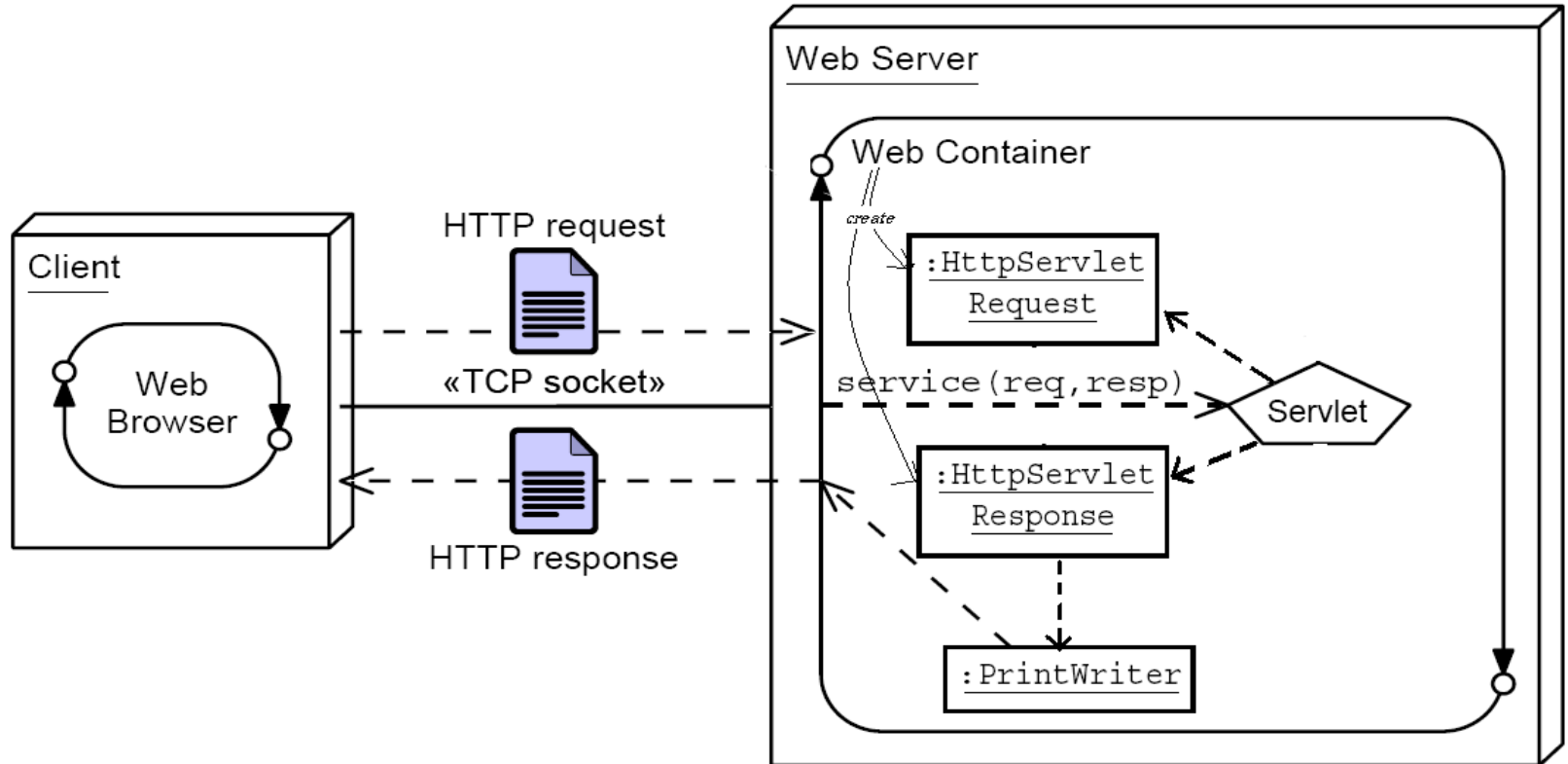
- object 제거 직전에 한번 call
- 객체가 가진 정보 반환 후 **garbage collector**에 대상 등록
- 모든 **service()**가 끝나도록 처리해 주어야 함
- **server** 마다 알아서 마지막 **request**로 부터 30분 경과하면 **container**에서 알아서 **destroy call**



# Servlet의 상속도



# Request and Response Process



# javax.servlet package의 interface

## ■ RequestDispatcher

- 현 서블릿의 정보를 context내의 다른 서블릿,JSP에게 넘기는 기능 정의
- `getServletContext().getRequestDispatcher("/a.jsp").forward(req,res);`
- `request.getRequestDispatcher("a.jsp").forward(req,res);`

## ■ ServletConfig

- 서블릿에서 필요로 하는 초기화 파라미터 정보를 접근할 수 있는 기능 정의
- `String ip=getServletConfig().getInitParameter("server_ip");`

## ■ ServletContext

- 서블릿이 실행되는 환경정보를 저장, 관리하는 기능 정의
- 서블릿 공유데이터 저장
- (서블릿버전,특정파일의 MIME 타입,Dispatcher객체,url에 대한 실제경로등)

# Servlet package의 interface

## ■ ServletRequest, HttpServletRequest

- 클라이언트가 보내온 정보를 관리하는 기능(문자열 인코드값,요청정보의 전체 길이, 언어코드,요청파라미터,헤더정보,client ip등)

## ■ ServletResponse, HttpServletResponse

- 요청에 대한 처리 결과를 되돌려주는 기능 (응답요청에 대한 버퍼 크기, 언어 지정 등)

## ■ SingleThreadModel

- service() 메소드가 동시에 한번만 호출(객체를 복사해서 각각의 요청에 응답)

# HTML Form

First Name:

Last Name:

```
<FORM METHOD="get" ACTION="/servlet/SimpleForm">  
    First Name:<INPUT TYPE="TEXT" NAME="FNAME"><br>  
    Last Name:<INPUT TYPE="TEXT" NAME="LNAME">  
    <input type="submit" value="send">  
    <input type="reset" value="clear">  
</FORM>
```



# FORM Tag

```
<FORM NAME="abc" action="/servlet/FileUpload" method="POST"  
      ENCTYPE="multipart/form-data">
```

- Name : FORM의 이름
- Method : 서블릿이나 cgi로 데이터 전송시 사용할 방식 지정  
    ↗ 생략시 GET
- Action : 요청을 처리할 servlet 이나 CGI스크립트
- ENCTYPE : encoding Type 결정

```
<INPUT TYPE=      NAME=      VALUE=      >
```

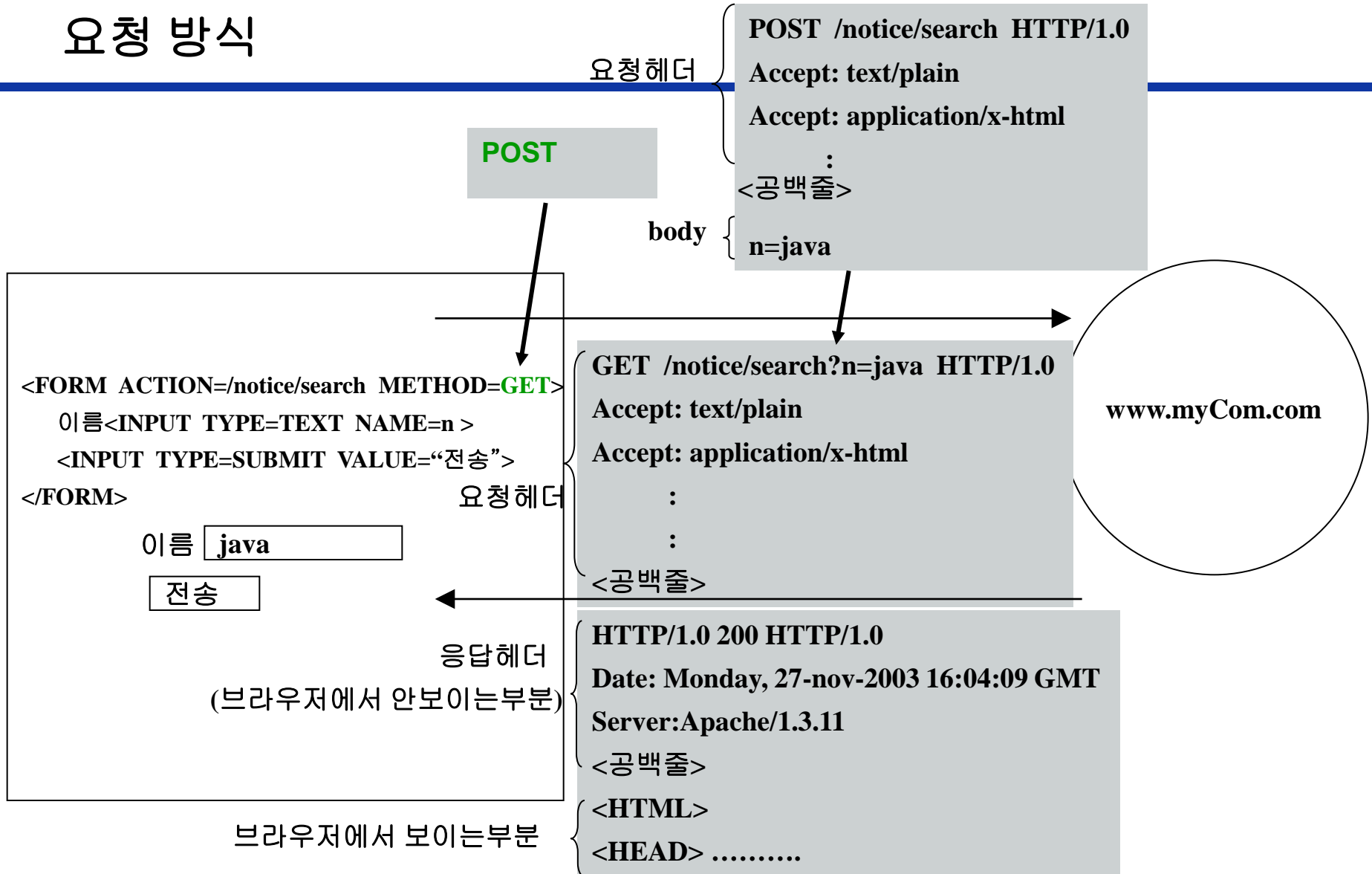
- TYPE : Text, Password, Checkbox, Radio, Button, Hidden  
          Image, Submit, Reset



# HTML Method

- GET
  - ↗ QUERY\_STRING를 URI에 덧붙여 전송
  - ↗ Data의 크기 제한
  - ↗ Default
  - ↗ 보안 취약
- POST
  - ↗ Body에 입력 데이터를 스트림으로 전송
  - ↗ Data의 크기 제한 없다.
  - ↗ GET방식에 비해 보안 우수
- HEAD : GET 방식과 비슷, **body** 없이 **Header**정보만 요구
- OPTIONS : 자원 접근할 수 있는 선택사항 설정
- PUT : 바디부분의 데이터를 **URI**에 저장
- DELETE : **URI**의 지정 자원을 지울 수 있다
- LINK : 헤더정보 서버문서와 연결
- UNLINK : 연결 해제
- TRACE : Loop Back 테스트

# 요청 방식





# Servlet에서 Method 처리

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException{
    ...

    String s=req.getQueryString();
        out.println("<p> QueyrString : " + s);

    value=req.getParameter("n");

    /* keys=req.getParameterNames();
        while(keys.hasMoreElements()){
            key=(String)keys.nextElement();
            value=req.getParameter(key);
            out.println("<p>" +key+": "+value);
        }
    */
}
```



# Servlet 수행결과 응답(Clinet) 코드

- 200 ~ 클라이언트가 요청한 자원을 정상적으로 처리했음을 의미
- 300 ~ 클라이언트가 요청은 정상적으로 했으나 자원에 접근하기위해서  
부가적인 일을 더해야 함
- 400 ~ 클라이언트가 요청한 자원은 서버에 없음(클라이언트의 요청오류)
- 500 ~ 클라이언트의 요청을 서버에서 처리 중 오류 발생(서버의 처리 오류)

# Http Protocol 장단점

- Stateless, Connectionless 통신  
( TCP/IP 통신(ex: 채팅) 컨넥션 유지)
- 장점 : 다수 사용자 상대를 하는 인터넷에 네트워크 서버의 오버헤드를 줄일 수 있다.
- 단점 : 연속되는 작업에서 이전 정보를 유지할 수 없다.  
==> 데이터 추적, 사용자 인식 등의 서비스가 어렵다.  
(사용자인증, 쇼핑카드)

# Cookie

- 특정정보를 client에 저장할 목적 사용
- 저장 : server --> client : HttpServletResponse에 의해 클라이언트에 보내짐 --> client에 저장(메모리/파일)
- 읽기 : Client --> Server : HttpServletRequest에 추가해 server에 보내짐
- javax.servlet.http.Cookie <-- Class

- Client에 저장시

```
Cookie myCookie = new Cookie(String name, String value);  
response.addCookie(myCookie);
```

- Sever에서 획득 시

```
Cookie[] allCookie = request.getCookies();  
Hashtable cookieTable = new Hashtable();  
for(int i = 0; i < allCookie.length; i++){  
    cookieTable.put(allCookie[i].getName(), allCookie[i].getValue()); }  
}
```

# Session의 사용

- 세션객체 얻기

- ↗ `HttpSession session=req.getSession(true | false);`

- 데이터 세션 객체에 저장

- ↗ `session.setAttribute(String name, Object value);`

- 데이터 세션 객체로 부터 얻기

- ↗ `session.getAttribute(String name);`

# 기타

- 현재 페이지의 정보를 가지고 다른 페이지 수행하기

- `RequestDispatcher rd= getServletContext().getRequestDispatcher("페이지명");`

- `rd.include(req,res);`//수행결과를 가져온다.

- `rd.forward(req,res);`//그 페이지로 옮겨가 수행한다.

- 새로운 요청으로 페이지 이동

- `res.sendRedirect("페이지명");`

- 캐쉬 되지 않도록 하기

- `res.setHeader("Cache-control","no-cache");`



# JSP문법

- Directives `<%@ ... %>`
- Declaration `<%! ... %>`
- Scriptlets `<%...%>`
- Expressions `<%= ... %>`
- Actions
- Comment

`<!-- -->` : browser실행시 comment 처리 (소스보기에 보임)

`<%-- --%>` :jsp comment, JSP 엔진이 skip(가장 강력)

`<%/* */%>` : java comment로 jsp에는 들어가나 클라이언트에선 보이지 않음



# Scriptlets – Implicit Reference

## ■ JSP에서 지원하는 객체 참조 reference들

Implicit Reference	사용 객체
request	javax.servlet.HttpServletRequest
response	javax.servlet.HttpServletResponse
pageContext	javax.servlet.jsp.PageContext
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
out	javax.servlet.JspWriter
config	javax.servlet.ServletConfig
page	java.lang.Object
exception	java.lang.Throwable



# EL ( Expression Language )

- JSP 페이지가 웹 페이지 출력 용도로 사용이 되면서, 좀더 쉬운 데이터 출력 방법이 요구되었다.
- JSP 2.0이 발표가 되면서 표현력이 좋은 **EL**이 추가 되었다.
- 자바 언어를 사용하지 않고 **JSP**페이지를 작성이 용이
- 사용법  
    `${ 변수 or 연산식 }`
- 아래와 같은 결과를 가져온다.
  - ➔ `<% out.println( 변수 or 연산식); %>`
  - ➔ `<%= 변수 or 연산식 %>`
  - ➔ `<c:out value="${식 or 연산식}" />`

# JSTL core Library

- 가장 일반적으로 많이 쓰이는 태그
- 수식, 제어 흐름, URL 처리 등에 관련된 작업
- `<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>`

기능	태그
EL 지원	<catch>, <out>, <remove>, <set>
제어 흐름	<choose>, <when>, <otherwise>, <forEach>, <forEachToken>
URL 관리	<import>, <param>, <redirect>, <url>

# JSTL core Library - if

- 조건이 맞는 경우 태그 내용 처리
- else 사용 안됨

```
<c:if test="testCondition" var="varName"  
        [scope="{page|request|session|application}"] />
```

- 태그 내용이 있는 경우

```
<c:if test="testCondition" var="varName"  
        [scope="{page|request|session|application}"] >  
    // body content  
</c:if>
```

# JSTL core Library - forEach

- 반복문
- items에 여러 데이터를 갖는 변수 설정, 데이터의 개수만큼
- 아이템들에 대해서 반복 수행

```
<c:forEach [var="varName"] items="collection"  
           [varStatus="varStatusName"]
```

```
    //body content
```

```
</c:forEach>
```

- 카운트에 대해서 반복 수행

```
<c:forEach [var="varName"] [varStatus="varStatusName"]  
           begin="begin" end="end" [step="step"]>
```

```
    //body content
```

```
</c:forEach>
```

---

## 2. Spring

## 2.1 Spring 소개

2.2 Spring DI

2.3 Spring AOP

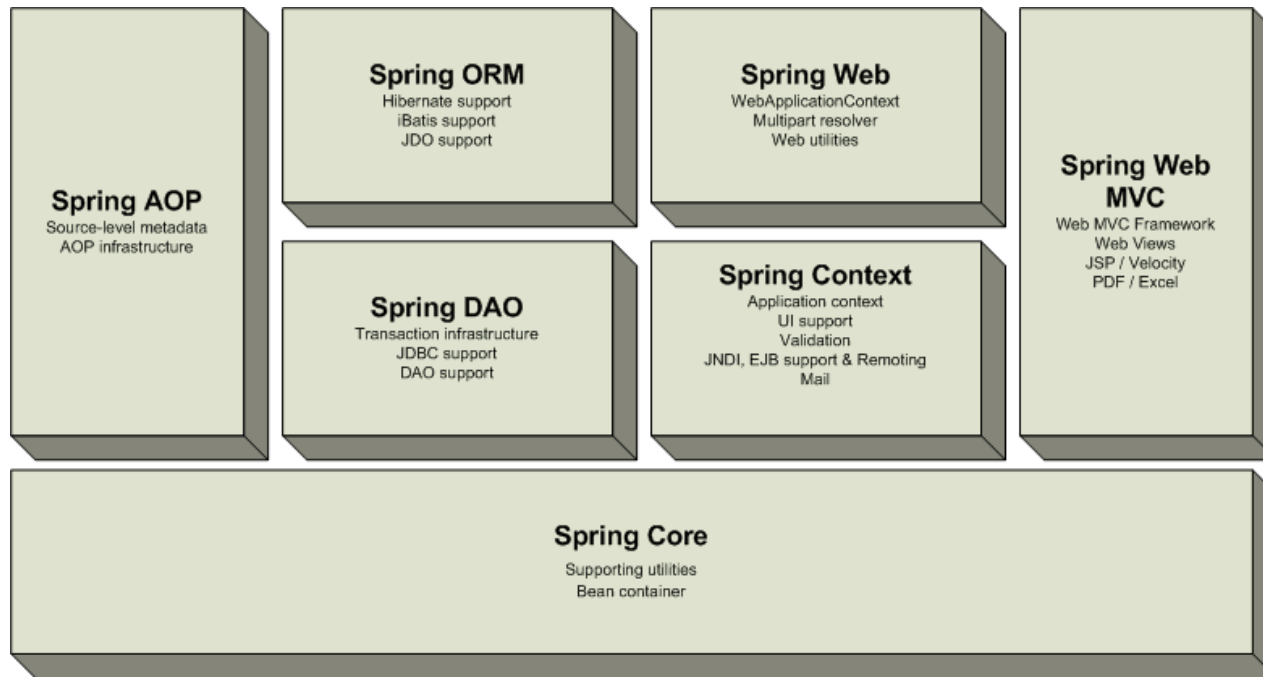
2.4 Spring MVC

2.5 Spring Database

2.6 Spring & myBatis

- 경량 컨테이너
- DI/AOP 컨테이너
- 일관적인 트랜잭션 제공
- POJO 지원
- 영속성 API 지원
- 다양한 엔터프라이즈 서비스 지원

- Rod Johnson 중심으로 개발된 Java/JavaEE용 프레임워크
- Spring 모듈





# History

버 전	출시일	특징
최초공개	2003년 6월	아파치 라이선스
1.0	2004년 3월	DI, AOP 지원
2.0	2006년 10월	어노테이션 등장
2.5	2007년 11월	어노테이션 강화(AspectJ지원, SpringMVC Annotation지원)
3.0	2009년 12월	spring EL, REST 지원
3.1	2011년 12월	스프링 캐시
3.2	2013년 Nov	Servlet 3기반, JSON2 RestTemplate사용
4.0	2013년 12월	Autowire Generic지원, Java8, Websocket ,SockJS 지원

2.1 Spring 소개

**2.2 Spring DI**

2.3 Spring AOP

2.4 Spring MVC

2.5 Spring Database

2.6 Spring & myBatis

- Dependency Injection(의존성 주입)
- 필요한 객체들( Service구현 객체,Dao 구현객체들 )을 Spring DI 컨테이너에서 객체를 만들어 관리하고 레퍼런스를 연결시켜주는 형태
- 기본적으로 Singleton 구현
- 클라이언트(사용객체)는 실제 구현된 클래스가 아니라 슈퍼타입이나 인터페이스 타입을 참조하여 호출
- 확장성이 좋음

- ▶ Service 구현 클래스에서 DAO 구현 클래스의 일반적인 사용

```
public class BoardService{  
    //...  
    public void requestInsert(...){  
        //...  
        BoardDao dao = new BoardDao();  
        dao.insert(...);  
    }  
}
```

- 향후 같은 업무에 다른 db 적용 시 새로운 클래스 작성

- Service 구현 클래스에서 인터페이스를 통한 DAO 구현 클래스의 일반적인 사용

```
public class BoardService{  
    //...  
  
    public void requestInsert(...){  
        //...  
        BoardDao dao =  
            new OracleBoardDao();  
        dao.insert(...);  
    }  
}
```

```
public class BoardService{  
    //...  
  
    public void requestInsert(...){  
        //...  
        BoardDao dao =  
            new MySQLBoardDao();  
        dao.insert(...);  
    }  
}
```

- 공통 메서드를 인터페이스로 추출
- 변수 타입은 인터페이스, 객체는 구현클래스의 인스턴스
- 하지만 상황에 따라 코드가 달라져야 한다.

- ▶ Service 구현 클래스에서 팩토리를 통한 DAO 구현 클래스의 일반적인 사용

```
public class BoardService{  
    //...  
    public void requestInsert(...){  
        //...  
        BoardDaoFactory factory = BoardDaoFactory.getInstance();  
        BoardDao dao = factory.getBoardDao();  
        dao.insert(...);  
    }  
}
```

- factory는 properties 파일과 연결
- BoardDao 구현체의 클래스명을 properties로부터 읽어들이м.
- 단점은 업무와 무관한 Factory 클래스를 참조해야 함.

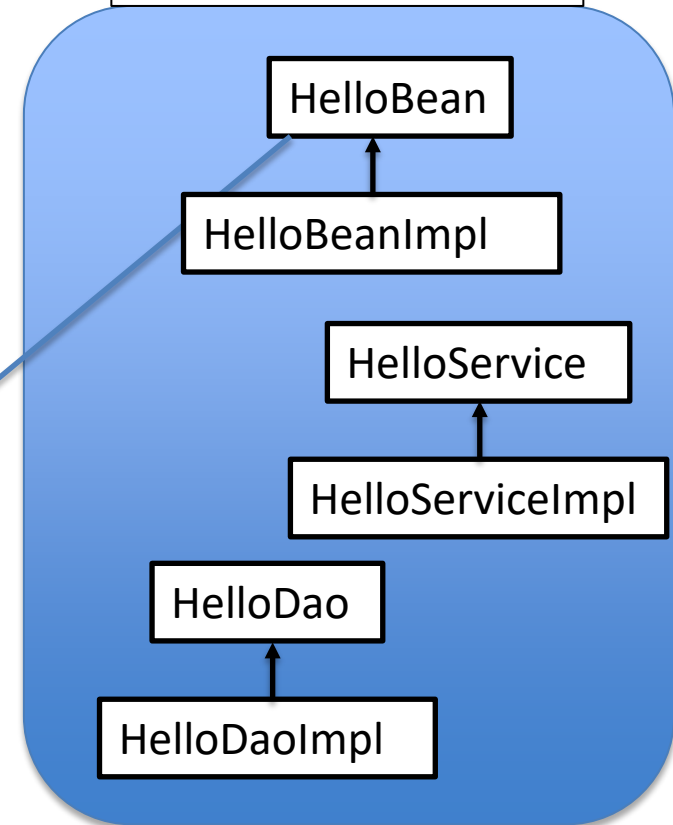
### ▶ 컨테이너를 통한 객체 조립

app.xml

```
<bean id="hello"  
      class="kr.jaen.spring.HelloBeanImpl" />
```

```
void method(){  
    BeanFactory factory = new  
        ClassPathXmlApplicationContext ("app.xml") ;  
    HelloBean bean =  
        (HelloBean)factory.getBean("hello");  
}
```

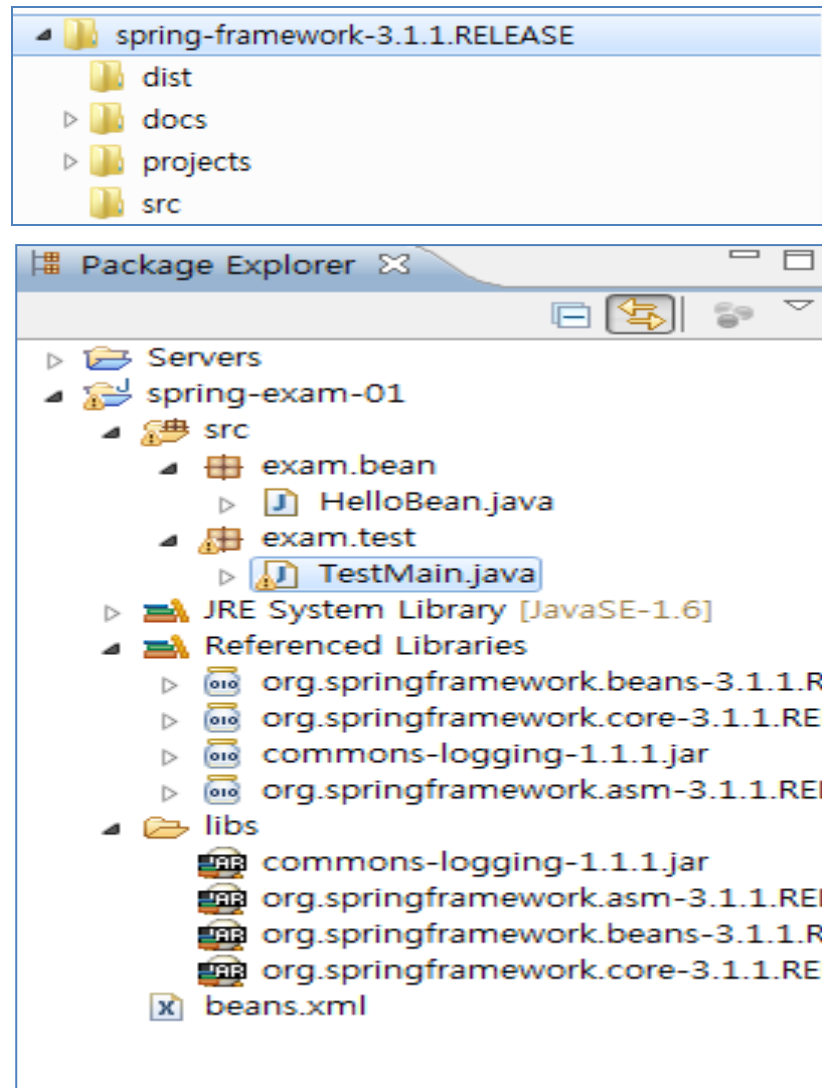
Spring DI Container



▶ 압축해제

▶ 프로젝트생성

▶ 라이브러리 등록  
build path에 추가





### ▶ HelloBean.java

```
package kr.jaen.spring;  
  
public interface HelloBean {  
    public void sayHello(String name) ;  
}
```

```
package kr.jaen.spring;  
  
public class HelloBeanImpl implements HelloBean {  
    public void sayHello(String name){  
        System.out.println("Hi~ "+name);  
    }  
}
```

### ➤ app.xml

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

<bean id="hello" class="kr.jaen.spring.HelloBeanImpl" >
</bean>

</beans>
```

### ➤ TestMain.java

```
package kr.jaen.test;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import kr.jaen.spring.HelloBean;

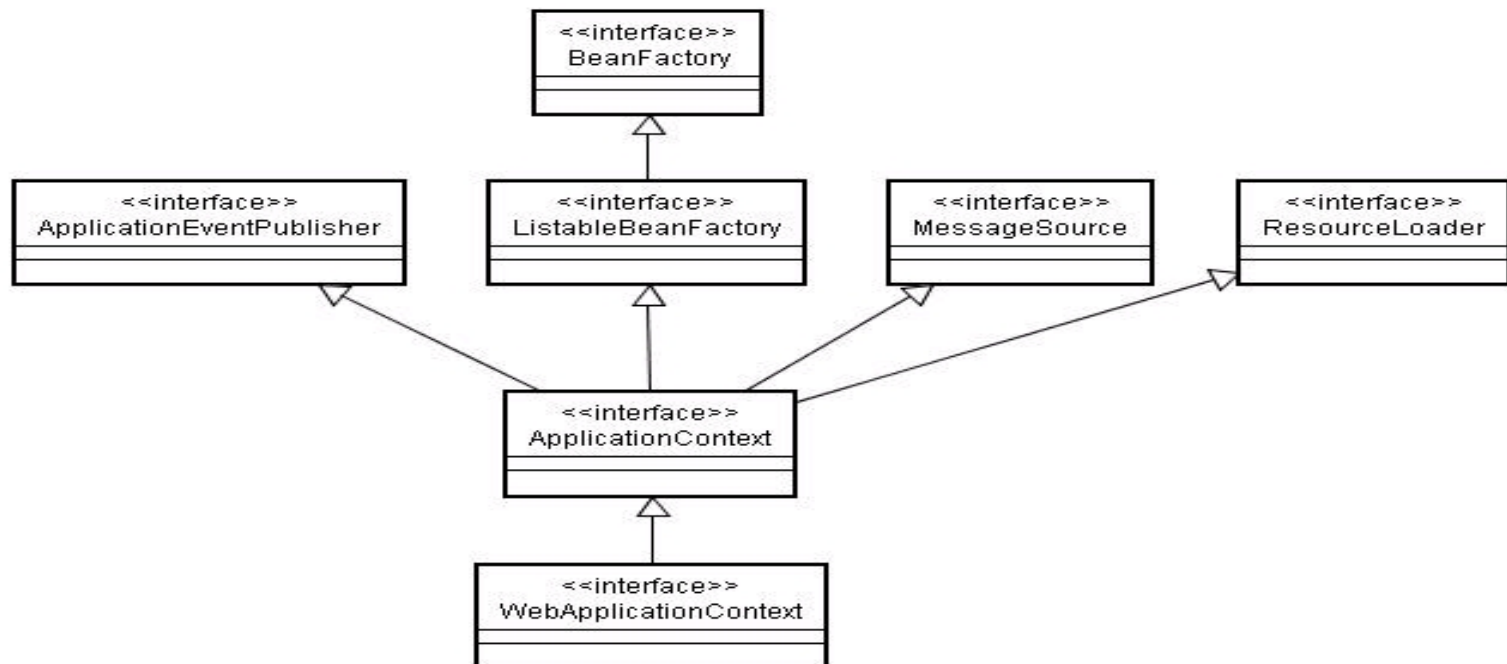
public class TestMain {
    public static void main(String[] args) {
        BeanFactory ctx=new ClassPathXmlApplicationContext("/app.xml");
        HelloBean bean = (HelloBean) factory.getBean("hello");
        bean.sayHello("Diana");
    }
}
```

## ➤ BeanFactory 인터페이스

리턴타입	메서드명	설명
boolean	<a href="#">containsBean</a> ( <a href="#">String</a> name)	인자로 넘겨준 이름의 빈이 정의되어있는지
<a href="#">String</a> []	<a href="#">getAliases</a> ( <a href="#">String</a> name)	인자로 넘겨준 이름의 빈이름에 정의된 별명 모두 리턴
<T> T	<a href="#">getBean</a> ( <a href="#">Class</a> <T> requiredType)	인자로 넘겨준 타입의 빈을 리턴
<a href="#">Object</a>	<a href="#">getBean</a> ( <a href="#">String</a> name)	인자로 넘겨준 이름의 빈을 리턴
<T> T	<a href="#">getBean</a> ( <a href="#">String</a> name, <a href="#">Class</a> <T> requiredType)	인자로 넘겨준 이름과 타입의 빈을 리턴
<a href="#">Object</a>	<a href="#">getBean</a> ( <a href="#">String</a> name, <a href="#">Object</a> ... args)	인자로 넘겨준 이름의 빈에 args를 저장 후 리턴
boolean	<a href="#">isPrototype</a> ( <a href="#">String</a> name)	인자로 넘겨준 이름의 빈이 프로토타입인지
boolean	<a href="#">isSingleton</a> ( <a href="#">String</a> name)	인자로 넘겨준 이름의 빈이 싱글톤인지
boolean	<a href="#">isTypeMatch</a> ( <a href="#">String</a> name, <a href="#">Class</a> <?> targetType)	인자로 넘겨준 이름의 빈이 targetType인지

## ➤ ApplicationContext 인터페이스

- org.springframework.context.ApplicationContext
- bean의 관리기능외 빈 라이프사이클, 자원처리기능,메시지지원,국제화지원, 등의 추가기능 제공



### ▶ 빈 검색

```
<bean id="hello" name="a b c" class="kr.jaen.spring.HelloBeanImpl" ></bean>  
<bean id="hello" name="a,b,c" class="kr.jaen.spring.HelloBeanImpl" ></bean>  
<bean id="hello" name="a;b;c" class="kr.jaen.spring.HelloBeanImpl" ></bean>  
<bean id="hello" name="a,b;c d" class="kr.jaen.spring.HelloBeanImpl" ></bean>
```

```
HelloBean bean = (HelloBean) factory.getBean("hello");  
HelloBean bean = factory.getBean("hello", kr.jaen.spring.HelloBean);  
HelloBean bean = factory.getBean("a", kr.jaen.spring.HelloBean);  
HelloBean bean = factory.getBean("b", kr.jaen.spring.HelloBean);  
HelloBean bean = factory.getBean("c", kr.jaen.spring.HelloBean);  
HelloBean bean = factory.getBean(kr.jaen.spring.HelloBean.class);
```

- 의존성주입(Dependency Injection) 방법
  - 생성자 설정
  - 프로퍼티 설정

### ▶ 생성자를 통한 DI

```
public class BoardService {  
    private BoardDao boardDao;  
    public BoardService(BoardDao boardDao) {  
        this.boardDao = boardDao;  
    }  
}
```

### ■ 설정파일

```
<bean id="boardService" class="kr.jaen.spring.BoardService">  
    <constructor-arg ref="boardDao" />  
</bean>  
<bean id="boardDao" class="kr.jaen.spring.BoardDao" />
```



### ▶ <constructor-arg> 속성

속성명	설명
index	넘길 인자의 순번
type	인자를 넘길 데이터 타입
ref	다른 빈의 id값, <value ref=".." />와 같다
value	인자로 넘길 값. <value>...</value>와 같다.

```
<constructor-arg value="100" type="int" />
```

```
<constructor-arg>
```

```
    <value type="long">값</value>
```

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

### ▶ 프로퍼티를 통한 DI

```
public class BoardService {  
    private BoardDao boardDao;  
    public void setBoardDao(BoardDao boardDao) {  
        this.boardDao = boardDao;  
    }  
}
```

```
<bean id="boardService" class="kr.jaen.spring.BoardService">  
    <property ref="boardDao" />  
</bean>  
<bean id="boardDao" class="kr.jaen.spring.BoardDao"></bean>
```

### ➤ xml namespace를 통한 약식형태

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

  <bean id="boardService" class="kr.jaen.spring.BoardService"
    p:boardDao-ref="boardDao"
    p:portno="100" />
</bean>
<bean id="boardDao" class="kr.jaen.spring.BoardDao"></bean>

</beans>
```

### ➤ xml namespace를 통한 약식형태

- `xml:p="http://www.springframework.org/schema/p"`
- `<property name="" value="값" />` ➔ `p:프로퍼티명="값"`
- `<property name="" ref="객체명" />` ➔ `p:프로퍼티명-ref="객체명"`

### ▶ 임의의 빈

- 호출 할 때마다 새로운 객체를 생성,전달함
- <property>나 <constructor-arg>내부에 <bean> 정의함.

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

  <bean id="boardService" class="kr.jaen.spring.BoardService" p:portno="100" />
    <bean id="boardDao" class="kr.jaen.spring.BoardDao" />
  </bean>

</beans>
```

### ▶ 프로퍼티 타입

태그명	타입
<list>	java.util.List 배열
<map>	java.util.Map
<set>	java.util.Set
<props>	java.util.Properties

### ➤ <list>

```
import java.util.List;
public class HelloService {
    private List<BoardDao> daos;
    public void setDaos(List<BoardDao> daos) {
        this.daos = daos;
    }
}
```

```
<bean id="hello" class="kr.jaen.spring.HelloService" >
  <property name="daos">
    <list>
      <ref bean="boardDao"/>
      <bean class="kr.jaen.spring.BoardDao"></bean>
    </list>
  </property>
</bean>
```

### ➤ <list>

```
import java.util.List;
public class GradeService {
    private List<Integer> grades;
    public void setGrades(List<Integer> grades) {
        this.grades = grades;
    }
}
```

```
<bean id="hello" class="kr.jaen.spring.GradeService" >
  <property name="nos">
    <list>
      <value>10 </value>
      <value> 20</value>
    </list>
  </property>
</bean>
```

### ➤ <list> 자식태그로 가능한 것들

<ref>,<bean>,<value>

<list>,<map>,<props>,<set>,<null>



➤ <map>

```
import java.util.List;
public class GradeService {
    private Map<String, BoardDao> daos;
    public void setDaos(Map<String, BoardDao> daos) {
        this.daos = daos;
    }
}
```

```
<bean id="hello" class="kr.jaen.spring.HelloService" >
  <property name="daos">
    <map>
      <entry>
        <key><value>oracle</value></key>
        <ref bean="oracleBoardDao"/>
      </entry>
      <entry>
        <key><value>mysql</value></key>
        <ref bean="mysqlBoardDao"/>
      </entry>
    </map>
  </property> </bean>
```

## ➤ 의존성 자동설정

- 복잡해진 설정정보를 이름이나 타입을 통해 자동으로 정의하기
- 4가지 방식을 지원함.
  - `byName` : 프로퍼티 이름과 같은 빈이름을 설정
  - `byType` : 프로퍼티 타입과 같은 빈타입을 설정
  - `constructor` : 생성자 인자 타입과 같은 빈 타입을 설정
  - `autodetect` : `constructor` 먼저 적용 후 `byType`을 적용한다.
- 주의: `byType`인 경우 같은 타입이 여럿 존재할 경우 예외발생함.

### ▶ 의존성 자동설정

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

<bean id="hello" class="kr.jaen.spring.HelloService" autowire="byName">

...

</beans>
```

### ➤ 설정의 상속

- 반복, 중복되는 속성에 대해 부모 빈으로 정의하고 이를 이용한다.
- `abstract="true"`로 정의하면 해당 빈은 객체 생성하지 않는다.
- `parent` 속성으로 내려받고 싶은 빈의 `id`를 지정한다.
- 필요하면 기존의 `property`를 재정의 하거나 추가할 수 있다.

```
<bean id="hello" class="kr.jaen.spring.HelloService" abstract="true">
    <property name="name" value="James">
        <property name="greeting" value="HI~">
    </bean>
<bean id="hello" class="kr.jaen.spring.MyHelloService" parent="hello"></bean>
<bean id="hello" class="kr.jaen.spring.YourHelloService" parent="hello"></bean>
<bean id="hello" class="kr.jaen.spring.OurHelloService" parent="hello">
    <property name="greeting" value="HI, Boys~">
</bean>
```

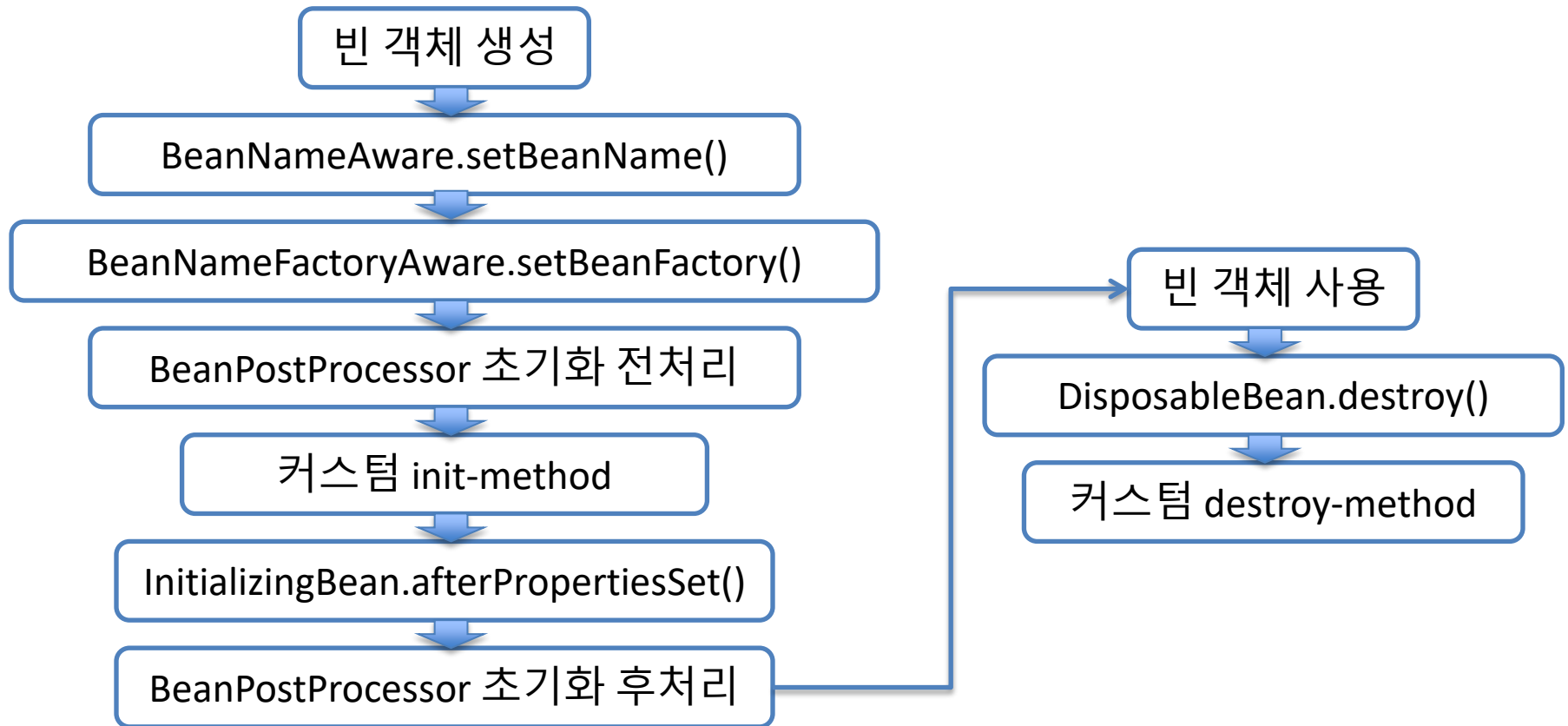
### ▶ 빈 객체의 범위 설정

- 기본적으로 컨테이너당 한개씩의 빈을 생성한다.
- 상황에 따라 매번 빈을 생성할 수 있다.

```
<bean id="boardDao" class="kr.jaen.spring.BoardDao" scope="prototype"></bean>
```

범위값	설명
singleton	컨테이너당 한 개씩만 생성한다(기본값)
prototype	빈을 요청 할 때마다 객체를 생성한다.
request	HTTP요청마다 빈 객체를 생성한다.
session	HTTP session마다 빈 객체를 생성한다.
global-session	portlet지원하는 컨텍스트에서 사용한다.

## ▶ 라이프사이클-BeanFactory



### ▶ 커스텀 초기화 메서드

- 사용자 정의 초기화 작업이나 종료작업을 호출하고자 할 때 사용한다.

```
<bean id="boardDao" class="kr.jaen.spring.BoardDao"  
    init-method="my_init"  
    destroy-method="my_destroy">  
</bean>
```

```
public class BoardService {  
  
    public void my_init(){ ..... }  
    public void my_destroy(){ ..... }  
}
```

### ▶ 설정 프로퍼티 외부파일 지정

```
jdbc.driver=oracle.jdbc.driver.OracleDriver  
jdbc.url=jdbc:oracle:thin:@localhost:1521:XE  
jdbc.username=scott  
jdbc.password=tiger
```

```
<bean  
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  
  <property name="locations">  
    <value>classpath:exam/config/db.properties</value>  
  </property>  
</bean>  
<bean id="ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-  
method="close">  
  <property name="driverClassName" value="${jdbc.driverName}"></property>  
  <property name="url" value="${jdbc.url}"></property>  
  <property name="username" value="${jdbc.username}"></property>  
  <property name="password" value="${jdbc.password}"></property>  
</bean>
```



### ➤ 설정 프로퍼티 외부파일 지정

- 여러 개의 property파일을 정의할때엔 <list>태그를 이용한다.

```
<bean  
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">  
  <property name="locations">  
    <list>  
      <value>classpath:exam/config/db.properties</value>  
      <value>classpath:exam/config/log.properties</value>  
    </list>  
  </property>  
</bean>
```

### ➤ 설정 프로퍼티 외부파일 지정

- <context:property-placeholder> 태그를 사용한 외부 프로퍼티 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xml:context="http://www.springframework.org/schema/context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/beans/spring-context-3.0.xsd "
>
<context:property-placeholder
    location="classpath:/config/db.properties" />
<!-- or
<context:property-placeholder
location="classpath:/config/db.properties, classpath:/config/db2.properties" />
-->
</beans>
```

## ➤ 메시지 처리

- 다국어 지원
- `org.springframework.context.MessageSource` 인터페이스 제공
  - `public String getMessage(String code, Object[] args, String defaultMessage, Locale locale);`
  - `public String getMessage(String code, Object[] args, Locale locale) throws NoSuchMessageException;`
  - `public String getMessage(MessageSourceResolvable resolvable, Locale locale) throws NoSuchMessageException;`

### ➤ 메시지 처리

```
<bean id="messageSource"  
  class="org.springframework.context.support.ResourceBundleMessageSource">  
  <property name="basename" value="message.board"></property>  
</bean>
```

- messageSource 빈 id가 있어야 함.
- basename은 FQN이어야 한다.
- 위코드에서 message.my\_messages는 message는 패키지명 board는 프로퍼티 파일명
- 언어에 따른 프로퍼티 파일명
  - board.properties – 기본 메시지. 해당 언어의 파일이 없는 경우
  - board\_en.properties – 영어 메시지
  - board\_ko.properties – 한국어 메시지

```
insert_title=INSERT  
list_title=Board LIST
```

### ➤ 메시지 처리

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

- 파일이 여러 개 인 경우 <list>를 이용한다.

## ▶ 빈에서 메시지 이용하기

- ApplicationContextAware 인터페이스 구현
  - .getMessage(...)
- MessageSourceAware 인터페이스 구현
  - .getMessage(...)

```
public interface MessageSourceAware{  
    public void setMessageSource(MessageSource messageSource);  
}
```

## ➤ MessageSourceAware

```
public class BoardService implements MessageSourceAware{
    private MessageSource messageSource;
    public void setMessageSource(MessageSource messageSource){
        this.messageSource= messageSource;
    }
    ...
    public void insert(...){
        String message=messageSource.getMessage("insert");
        ... ..
    }
}
```

## ➤ 어노테이션 사용하기

- @Required
  - RequiredAnnotationBeanPostProcessor
- @Autowired
  - AutowiredAnnotationBeanPostProcessor
- @Resource
  - CommonAnnotationBeanPostProcessor
- @PostConstruct
  - CommonAnnotationBeanPostProcessor
- @PreDestroy
  - CommonAnnotationBeanPostProcessor
- @Configuration
  - ConfigurationClassPostProcessor



## ▶ 어노테이션 사용하기

- @Required - 필수 프로퍼티
- RequiredAnnotationBeanPostProcessor를 빈으로 등록해야 함.

```
<bean class="
org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor "/>
```

```
public class BoardService {
    private BoardDao boardDao;

    @Required
    public void setBoardDao(BoardDao boardDao) {
        this.boardDao = boardDao;
    }
}
```

### ➤ 어노테이션 사용하기

- RequiredAnnotationBeanPostProcessor를 빈으로 등록대신 <context:annotation-config> 사용해도 됨
- 이것을 사용하면 빈을 정의하지 않아도 됨.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xml:context="http://www.springframework.org/schema/context"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www
http://www
http://www
<bean class="
org.springframework.beans.factory.annotation.AutowiredAnnotation
nBeanPostProcessor "/>
>
  <context:annotation-config />
... ..
</beans>
```

- @Autowired 어노테이션을 사용한 자동 설정
  - 타입으로 의존관계 자동 설정
    - 생성자
    - 메서드
    - 필드

### ➤ @Autowired 어노테이션을 사용한 자동 설정

```
public class BoardService {  
  
    private BoardDao boardDao;  
  
    @Autowired  
    public void setBoardDao(BoardDao boardDao) {  
        this.boardDao = boardDao;  
    }  
}
```

```
public class BoardService {  
  
    @Autowired  
    private BoardDao boardDao;  
}
```

## ▶ 어노테이션을 이용한 자동 등록

- @Component – 자동으로 빈에 등록 됨.
- 등록되는 빈 이름은 클래스명 첫글자를 소문자로 변경하여 등록
- @Component("beanName") : 이름을 명시적으로 지정할 수 있다.
- @Scope("prototype") : 빈 범위 지정. 기본은 singleton

```
@Component
public class BoardService {

    @Autowired
    private BoardDao boardDao;
    //..
}
```

### ▶ 어노테이션을 이용한 자동 등록

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

### ➤ @PostConstruct 어노테이션

```
public class BoardService {  
    @PostConstruct  
    public void init(){  
        //초기화코드  
    }  
    ...  
    @PreDestroy  
    public void close(){  
        //자원정리코드  
    }  
}
```

2.1 Spring 소개

2.2 Spring DI

**2.3 Spring AOP**

2.4 Spring MVC

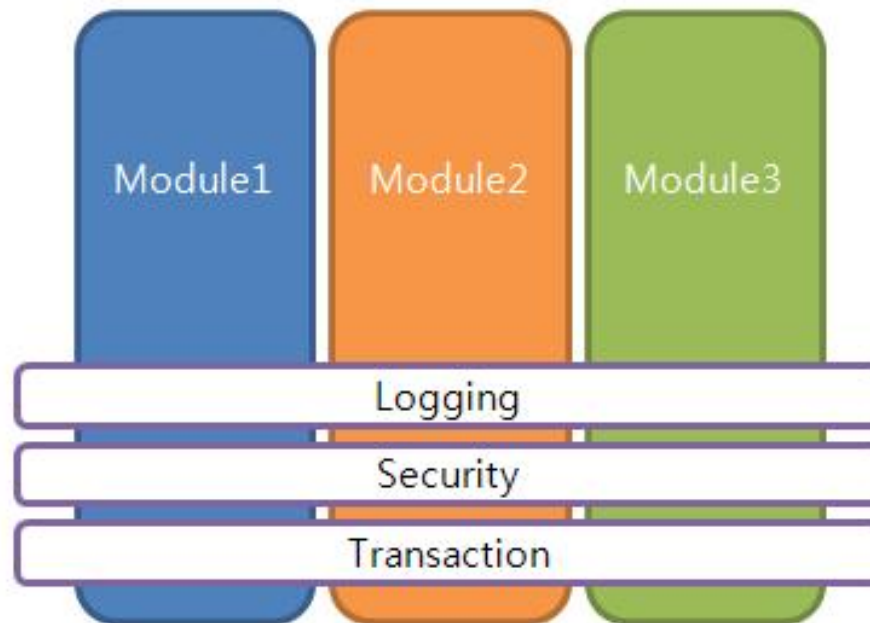
2.5 Spring Database

2.6 Spring & myBatis



## ➤ Aspect Oriented Programming

- 많은 곳에서 공통적으로 사용되는 코드를 모듈화 하여 원래코드를 수정하지 않고 원하는 곳에 코드를 적용시키는 기능



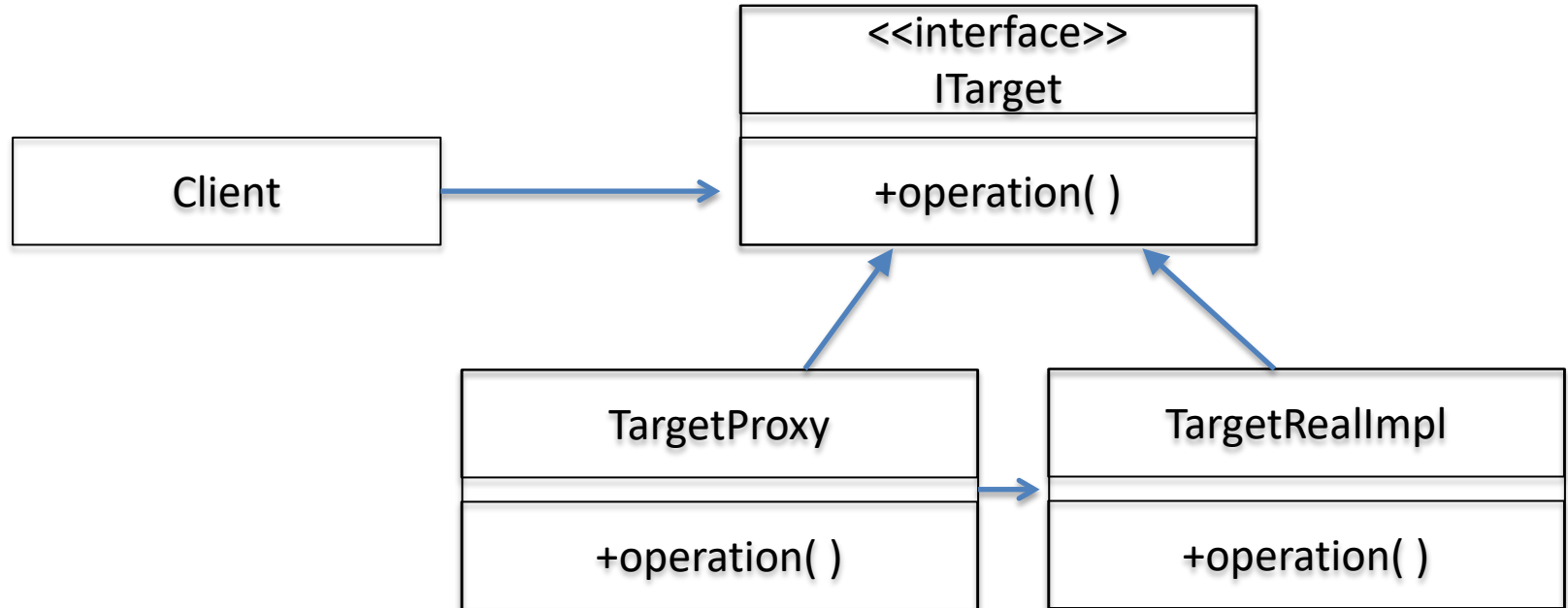
## ➤ AOP 용어

- Advice : 적용할 공통관심사항의 기능을 정의
- Joinpoint : advice 적용 가능한 시점들을 의미
- Pointcut : advice를 적용하는 joinpoint를 의미
- Weaving : advice를 핵심로직에 적용하는 것
- Aspect : 여러 객체에 공통으로 적용하는 공통관심 사항.

## ➤ Weaving

- 컴파일시 Weaving
  - 클래스 로딩시 Weaving
  - 실행시 Weaving
- 
- AOP 라이브러리는 타겟 객체의 원하는 위치에 공통 코드가 삽입된 새로운 클래스를 사용하도록 한다.
  - 소스코드를 변경하지 않는 대신 프록시를 이용하여 AOP를 구현한다.

## ▶ Proxy 기반의 AOP



### ➤ Advice 종류

인터페이스	종류	설명
MethodBeforeAdvice	before	대상 객체의 메서드 호출전에 공통기능을 실행
AfterReturningAdvice	after-returning	대상 객체의 메서드가 정상 처리 처리된 후 공통 기능 실행
ThrowsAdvice	after-throwing	대상 객체의 메서드가 예외 발생하면 공통 기능 실행
	after	대상 객체의 메서드가 예외 발생하든 안 하든 무조건 공통 기능 실행
MethodInterceptor	around	대상 객체의 메서드 실행 전후 또는 예외 발생시 공통 기능 실행

### ➤ 클라이언트

- 클라이언트는 원하는 객체를 호출한다.

```
public class Test {  
    public static void main(String[] args) {  
        BeanFactory ctx=new ClassPathXmlApplicationContext("/config/app.xml");  
        IService ser=ctx.getBean(IService.class);  
        Employee e=ser.search(1111);  
        System.out.println(e);  
    }  
}
```

### ▶ 실제 구현 클래스

```
public interface IService {  
  
    public abstract String getMsg();  
    public abstract Employee search(int num);  
}
```

```
public class MyService implements IService {  
    @Override  
    public String getMsg() {  
        return "AOP Test";  
    }  
    @Override  
    public Employee search(int num) {  
        return new Employee(1111,"김철수","영업",3000000);  
    }  
}
```

### ➤ Advice 파일

```
public class MyFirstAspect {

    public void before(JoinPoint jp) {
        System.out.println("Hello Before! ");
        Signature sig = jp.getSignature();
        System.out.println("-----> " + sig.getName());
        Object[] o = jp.getArgs();
        System.out.println("----->" + o[0]) ;
    }

    public void after() {
        System.out.println("Hello After! *");
    }

    public void afterReturning(JoinPoint jp, Employee e) {
        System.out.println("Hello AfterReturning! ");
        Signature sig = jp.getSignature();
        System.out.println("-----> " + sig.getName());
        Object[] o = jp.getArgs();
        System.out.println("-----> " + o[0]);
    }

}
```



### ➤ Advice 파일

```
public Employee around(ProceedingJoinPoint pjp) throws Throwable {
    Signature sig = pjp.getSignature();
    System.out.println("Hello Around! Before :"+ sig.getName() );
    Employee p = (Employee) pjp.proceed();
    System.out.println("Hello Around! after "+p);
    return p;
}

public void afterThrowing(Throwable ex) {
    System.out.println("Hello Throwing! ");
    System.out.println("exception value = " + ex.toString());
}
}
```

## ➤ Pointcut

- AspectJ의 포인트컷

- execution : 메서드나 생성자 실행을 pointcut으로 한다.

**[modifier] return\_type [패키지.클래스명.]메서드명 ( type or ..)**

execution( public void set\*() )

execution( \* kr.jaen.exam.\*.\*() )

execution( \* \*..\*exam() )

execution( \* \*..MemberDao.\*(..) )

- 와일드카드

- \* : .을 포함하지 않는 임의의 문자열
  - .. : .을 포함하는 임의의 문자열
  - + : 하위클래스 또는 하위 인터페이스

- 논리연산자

- && , || , !

## ➤ 설정파일

- 네임스페이스 추가 AOP

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

... ..

</beans>
```

### ▶ 설정파일

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

<bean id="mySer" class="kr.jaen.di.ser.EmployeeService" />

<bean id="myFirstAspect" class="kr.jaen.aop.MyFirstAspect" />
<aop:config>
  <aop:aspect id="myAspect" ref="myFirstAspect">
    <aop:pointcut id="pc" expression="execution(* search(int))" />
    <aop:before pointcut-ref="pc" method="before" />
    <aop:after pointcut-ref="pc" method="after" />
    <aop:after-returning pointcut-ref="pc"
      method="afterReturning" returning="e" />
    <aop:around pointcut-ref="pc" method="around" />
    <aop:after-throwing pointcut-ref="pc" method="afterThrowing" throwing="ex" />
  </aop:aspect>
</aop:config></beans>
```

## ➤ aop 관련 태그

- <aop:config> AOP 설정정보임을 명시
- <aop:aspect> Aspect를 설정
- <aop:pointcut> Pointcut을 설정
- <aop:around> Around Advice를 설정

태그명	설 명
<aop:before>	메서드 실행 전에 적용되는 advice 정의
<aop:after-returning>	메서드 정상실행 후 적용되는 advice 정의
<aop:after-throwing>	메서드 예외발생 후 적용되는 advice 정의
<aop:after>	메서드 무조건 실행 후 적용되는 advice 정의
<aop:around>	메서드 호출 전후 실행 후 적용되는 advice 정의

➤ @Aspect 어노테이션 : Advice 파일에 포인트컷 직접 명시

```
@Aspect
@Component
public class MyFirstAspect {

    @Before("execution(* getMsg())")
    public void before(JoinPoint jp) {...}

    @After("execution(* getMsg())")
    public void after() {...}

    @AfterReturning(value = "execution(* search(int))", returning = "e")
    public void afterReturning(JoinPoint jp, Employee e) {...}

    @Around("execution(* search(int))")
    public Employee around(ProceedingJoinPoint pjp) throws Throwable {...}

    @AfterThrowing(value = "execution(* getMsg())", throwing = "ex")
    public void afterThrowing(Throwable ex) {...}
}
```

## ➤ @Aspect 어노테이션

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

<context:annotation-config />
<context:component-scan base-package="kr.jaen.*" />
<aop:aspectj-autoproxy />
</beans>
```

2.1 Spring 소개

2.2 Spring DI

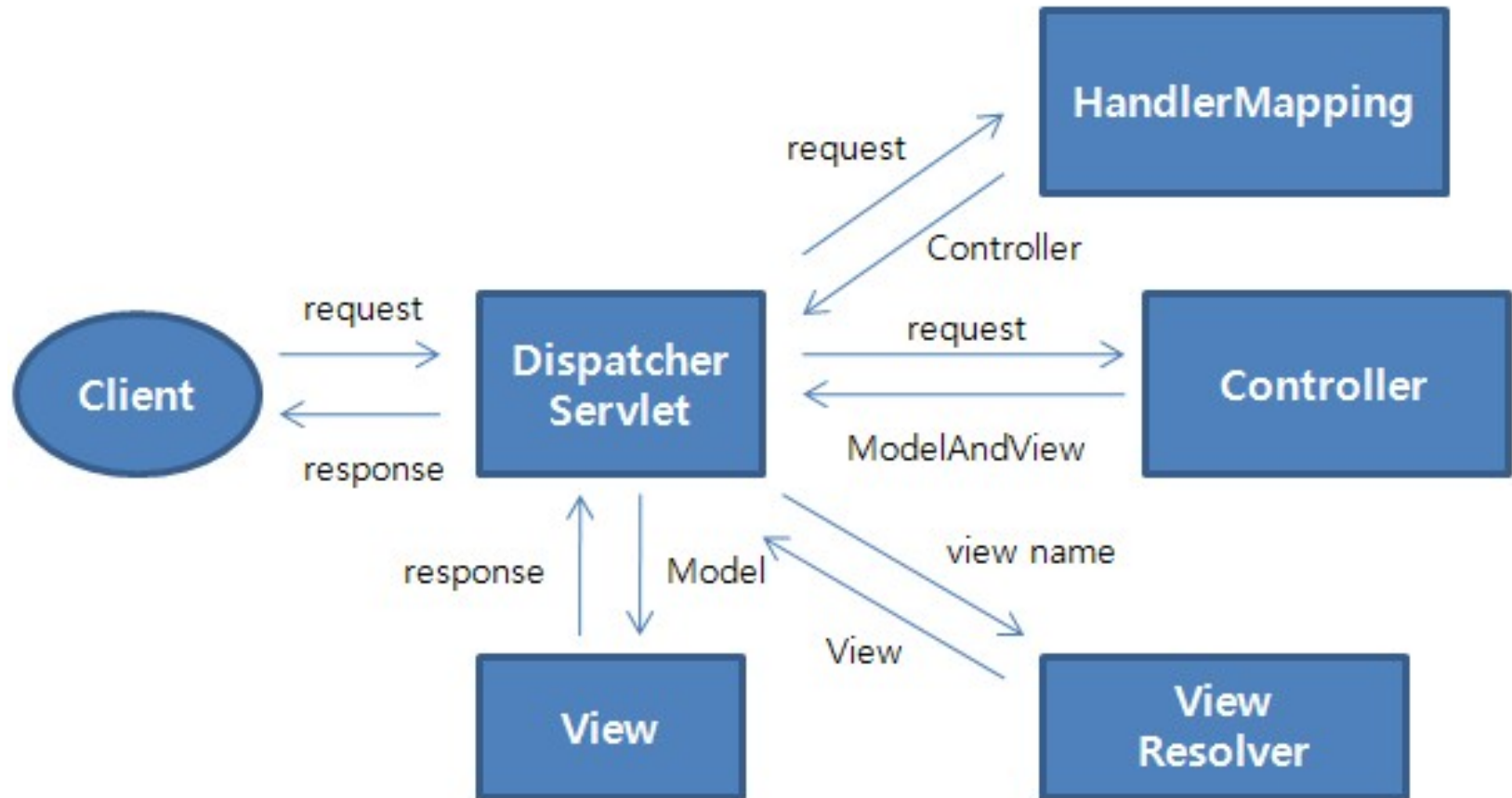
2.3 Spring AOP

**2.4 Spring MVC**

2.5 Spring Database

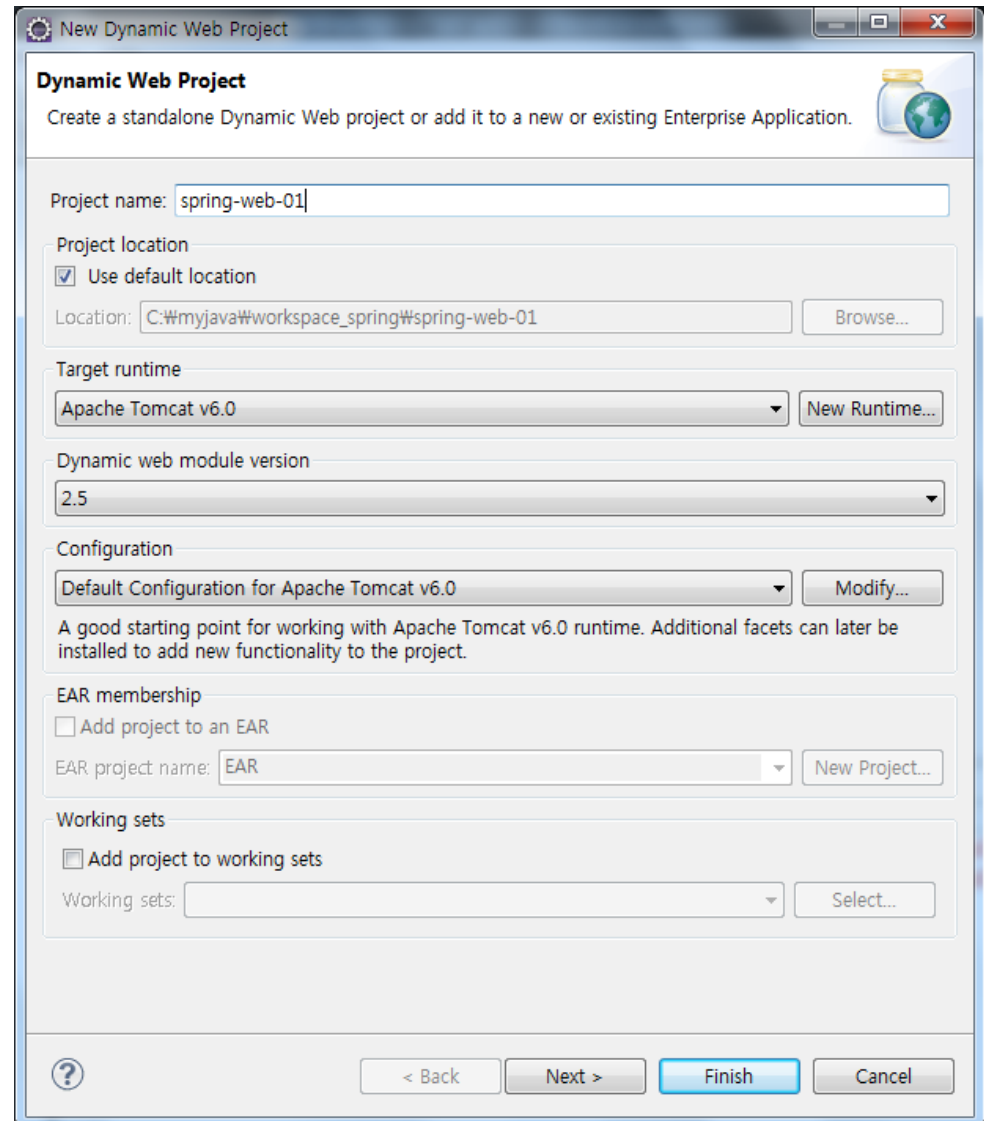
2.6 Spring & myBatis





- Dynamic Web Project
- Spring 모듈준비
- web.xml
- beanMapping파일
- 컨트롤러 작성
- 뷰 작성

### ➤ (1) Dynamic Web Project

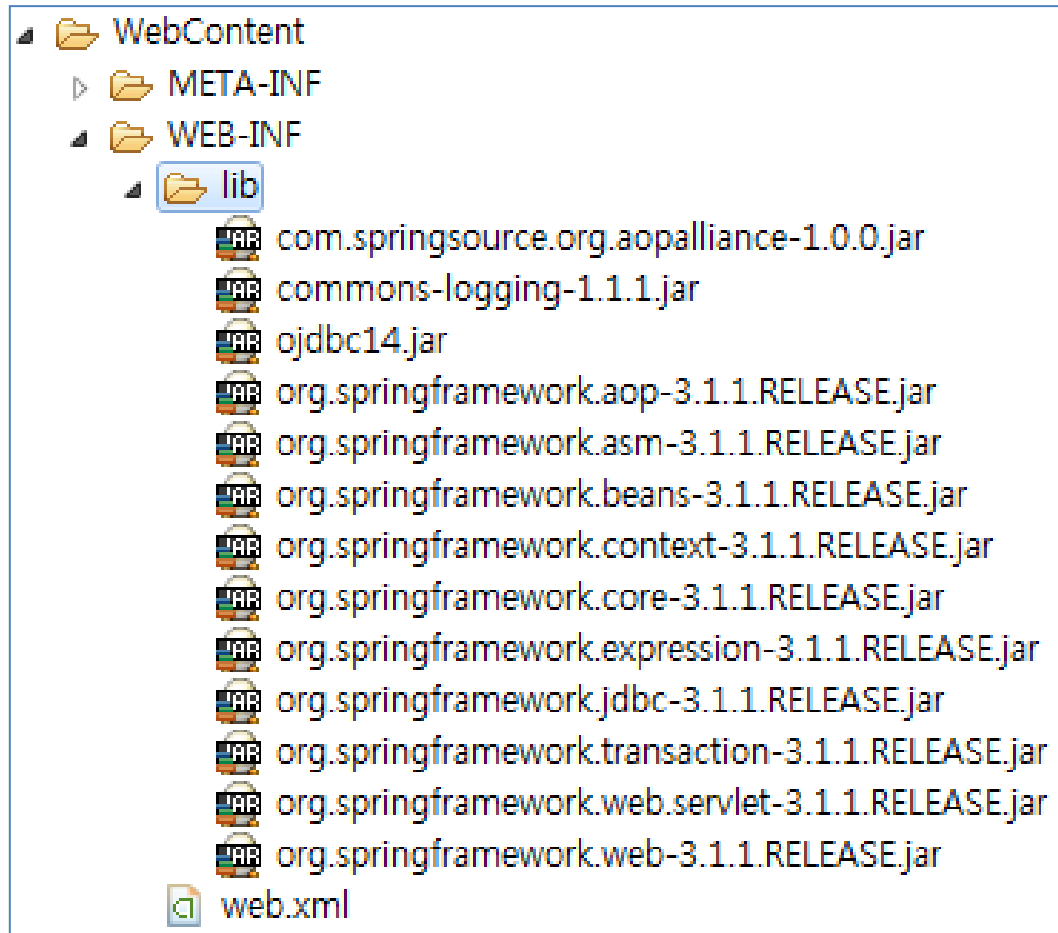


The screenshot shows the 'New Dynamic Web Project' dialog box in the Eclipse IDE. The dialog is titled 'New Dynamic Web Project' and has a subtitle 'Dynamic Web Project'. Below the subtitle, it says 'Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.' The dialog contains several sections with input fields and checkboxes:

- Project name:** A text field containing 'spring-web-01'.
- Project location:** A section with a checked checkbox 'Use default location' and a text field 'Location:' containing 'C:\myjava\workspace\_spring\spring-web-01'. There is a 'Browse...' button next to the text field.
- Target runtime:** A dropdown menu showing 'Apache Tomcat v6.0' and a 'New Runtime...' button.
- Dynamic web module version:** A dropdown menu showing '2.5'.
- Configuration:** A dropdown menu showing 'Default Configuration for Apache Tomcat v6.0' and a 'Modify...' button. Below this, there is a descriptive text: 'A good starting point for working with Apache Tomcat v6.0 runtime. Additional facets can later be installed to add new functionality to the project.'
- EAR membership:** A section with an unchecked checkbox 'Add project to an EAR', a text field 'EAR project name:' containing 'EAR', and a 'New Project...' button.
- Working sets:** A section with an unchecked checkbox 'Add project to working sets', a text field 'Working sets:', and a 'Select...' button.

At the bottom of the dialog, there is a help icon (question mark) and four buttons: '< Back', 'Next >', 'Finish' (highlighted in blue), and 'Cancel'.

### ➤ (2) library 복사



## ➤ (3) web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ... version="2.5">
  <display-name>Spring_MVC_1_an</display-name>
  <welcome-file-list>  <welcome-file>index.html</welcome-file> </welcome-file-list>
  <servlet>
    <servlet-name>springDispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

## ➤ (4) 컨트롤러 클래스

```
package kr.jaen.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;

public class HelloController implements Controller{
    public ModelAndView handleRequest(HttpServletRequest req,
                                     HttpServletResponse res) throws Exception {
        ModelAndView mv=new ModelAndView();
        mv.addObject("msg", "안녕하세요~~~~");
        mv.setViewName("result.jsp");
        return mv;
    }
}
```

### ➤ (5) bean 설정파일 – /WEB-INF/config.xml

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

<bean id="/hello.do" class="kr.jaen.controller.HelloController"/>
</beans>
```

➤ (6) 뷰처리 - /index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="EUC-KR">
    <title>Insert title here</title>
  </head>
  <body>
    <h1> Spring MVC Page</h1>
    <p/>
    <a href="hello.do"> 컨트롤러 호출</a>
  </body>
</html>
```



## ➤ (6) 뷰처리 - /result.jsp

```
<%@ page language="java" contentType="text/html; charset=EUC-KR" pageEncoding="EUC-KR"%>
<!DOCTYPE html >
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
        <title>Insert title here</title>
    </head>
    <body>
        <h1> Result Page....</h1>
        <p/>
        결과 메세지 : ${msg}
        <p/>
        <a href="second.do"> 두 번째 페이지</a>
    </body>
</html>
```

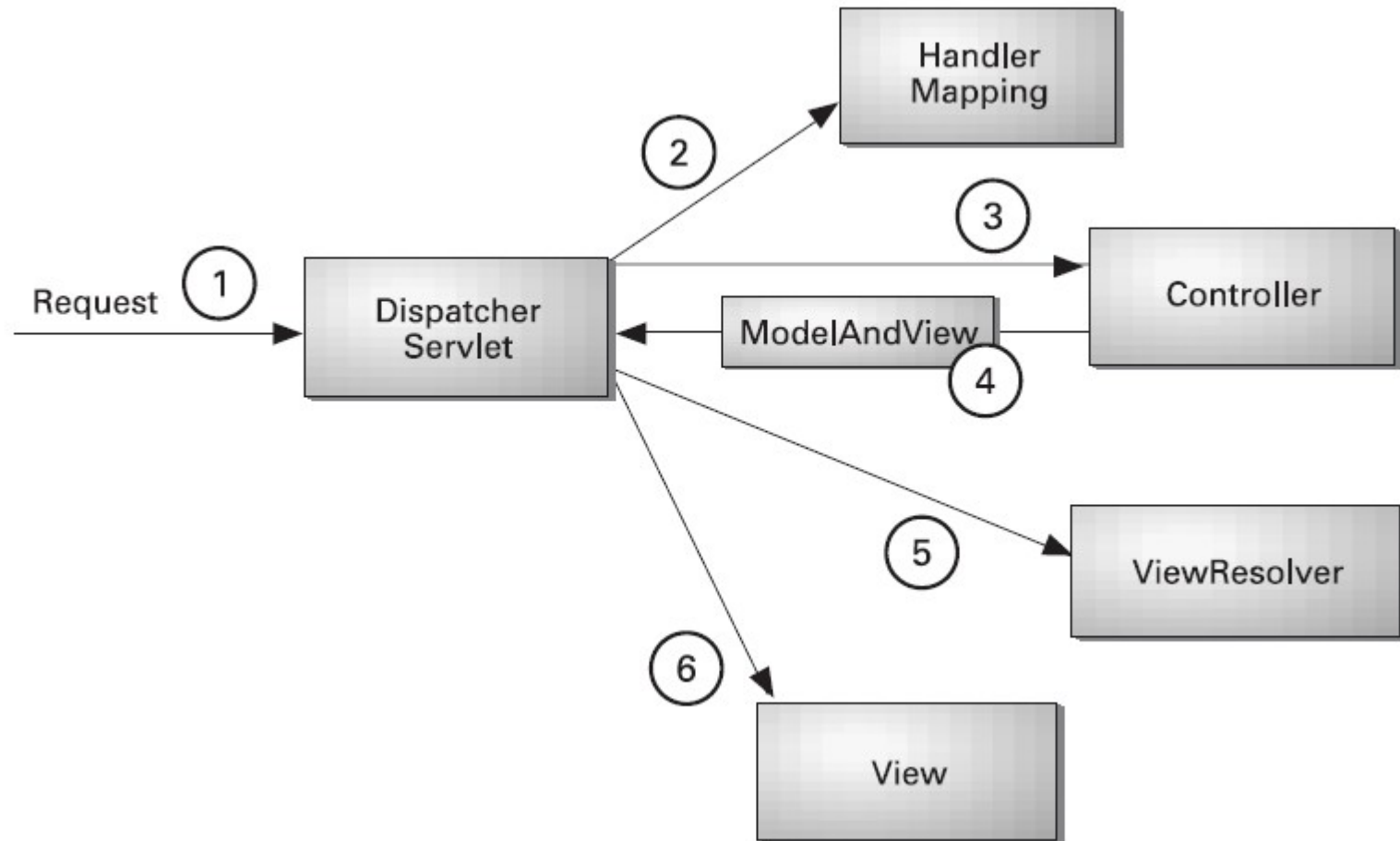
➤ 실행 결과

**Spring MVC Page**  
[컨트롤러 호출](#)



**Result Page....**  
결과 메시지 : 안녕하세요~~~~~  
[두번째 페이지](#)

## ▶ 실행흐름



## ➤ web.xml

- DispatcherServlet은 클라이언트 요청 처리 핵심이다.
- 설정파일은 config.xml 이다.

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

```
<servlet-mapping>
  <servlet-name>springDispatcherServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

- DispatcherServlet
- 기본 설정파일은 /WEB-INF/<servlet-name>값-servlet.xml이다.
  - 설정파일 구분자는 공백, 콤마, 탭, 줄바꿈, 세미콜론

```
<servlet>
  <servlet-name>springapp</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath:/config/app.xml, classpath:/config/notice.xml
      /WEB-INF/message.xml
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

- ▶ ContextLoaderListener 를 활용하여 설정파일 지정하기
  - 기본설정파일은 /WEB-INF/applicationContext.xml
  - classpath:/config/applicationContext.xml

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/app.xml , /WEB-INF/message.xml
  </param-value>
</context-param>
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

## ▶ Spring 한글처리

```
<filter>
  <filter-name>CharacterEncodingFilter</filter-name>
  <filter-class> org.springframework.web.filter.CharacterEncodingFilter </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>EUC-KR</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>CharacterEncodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

## ➤ (4) 컨트롤러 클래스

```
package kr.jaen.controller;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;

public class HelloController implements Controller{
    public ModelAndView handleRequest(HttpServletRequest req,
                                     HttpServletResponse res) throws Exception {
        ModelAndView mv=new ModelAndView();
        mv.addObject("msg", "안녕하세요~~~~");
        mv.setViewName("result.jsp");
        return mv;
    }
}
```



## ▶ Controller 종류

타입	설명
Controller AbstractController	간단한 구현
AbstractCommandController	요청 파라미터를 객체에 저장
SimpleFormController	폼출력및 폼입력 데이터 처리기능
AbstractWizardFormController	여러 페이지에 걸친 데이터 처리시
ParameterizableViewController UrlFilenameViewController	아무 작업없이 단순히 부만 출력할 때
MultiActionController	여러 개의 기능을 하나의 클래스에서 제공

## ➤ 리다이렉트 뷰

- “redirect:” 를 접두어로 붙인다.

```
ModelAndView mav=new ModelAndView();  
mav.setViewName("redirect:/board/list.do");  
return mav;
```

```
ModelAndView mav=new ModelAndView();  
mav.setViewName("redirect:http://www.daum.net/");  
return mav;
```

- 어노테이션을 이용한 컨트롤러
  - 컨트롤러 클래스를 빈으로 등록
  - 컨트롤러 클래스에 @Controller 적용
  - 요청 처리 메서드에 @RequestMapping 적용

## ▶ 어노테이션 으로 Controller 클래스 만들기

```
package kr.jaen.controller;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class HelloController {
    @RequestMapping("/hello.do")
    public String bye(Model m){
        m.addAttribute("msg", "안녕하세요~~~~");
        System.out.println("Controller.....");
        return "result.jsp";
    }
    @RequestMapping("/second.do")
    public String byebye(Model m){
        m.addAttribute("msg", "안녕하세요2~~~~");
        System.out.println("Controller2.....");
        return "result.jsp";
    }
}
```

➤ 컨트롤러 클래스를 빈 등록

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

<context:component-scan base-package="kr.jaen.controller"/>
</beans>
```

## ➤ HTTP 요청 매핑

```
@RequestMapping("/list.do")
```

```
@RequestMapping({"/detail.do","/read.do"})
```

```
@RequestMapping(value="/list.do")
```

```
@RequestMapping(value={"/detail.do", "/read.do"})
```

```
@RequestMapping(value="/list.do", method=Request.GET )
```

```
@Controller
public class HelloController {
    @RequestMapping(value="/board/insert.do",method=RequestMethod.GET);
    public String form(){
        return "board/insert_form";
    }
    @RequestMapping(value="/board/insert.do",method=RequestMethod.POST);
    public String insert_action(){
        // 게시물 등록작업코드 ...
        return "board/insert_form";
    }
}
```

➤ 컨트롤러 메서드의 클라언트 값 전달

파라미터타입	설 명
HttpServletRequest, HttpServletResponse, HttpSession	서블릿 API
Java.util.Locale	요청에 대한 Locale
InputStream, Reader	요청 콘텐츠에 직접접근할때
OutputStream, Writer	응답 콘텐츠를 생성할때
@RequestParam 적용	HTTP요청 파라미터에 매핑
Map, ModelMap	뷰에 전달할 모델 데이터 설정
커맨드 객체	HTTP요청 파라미터를 저장한 객체.모델명은 클래스 이름을 사용.@ModelAttribute를 이용하여 모델명을 설정할 수 있다.

## ➤ @RequestParam

```
<body>
<header><h1>직원 등록 화면</h1></header>
<section>
<b>form action="empSave.do" method="post">
    <table>
        <tr><th>직원번호</th><td> <input type="number" name="snum" /></td></tr>
        <tr><th>직원이름</th><td> <input type="text" name="sname" /></td></tr>
        <tr><th>직원급여</th><td> <input type="number" name="salary" /></td></tr>
        <tr><td colspan="2"> <input type="submit" value="등록" />
        <input type="reset" value="취소" /></td></tr>
    </table>
</b>
</form>
</section>
</body>
```

### @RequestMapping("/empSave.do")

```
public String empSave(int snum, @RequestParam("sname") String name, int salary, Model m){
    ser.save(new Employee(snum, name, salary));
    m.addAttribute("msg", "직원 등록 완료!!!");
    return "Result";
}
```



## ➤ @RequestParam

- 필수파라미터 없으면 400오류발생
- 파라미터값 없으면 null저장
- 파라미터값 없으면 숫자인 경우 예외발생(400오류)
- 기본값 정의

@RequestParam(value="age", required=false)

@RequestParam(value="age", required=false, defaultValue="0")

## ▶ 파라미터값 vo로 바로 전달하기

```
<body>
<header><h1>직원 등록 화면</h1></header>
<section>
<form action="/empSave.do" method="post">
    <table>
        <tr><th>직원번호</th> <td><input type="number" name="snum" /></td></tr>
        <tr><th>직원이름</th> <td><input type="text" name="sname" /></td></tr>
        <tr><th>직원급여</th> <td><input type="number" name="salary" /></td></tr>
        <tr><td colspan="2"><input type="submit" value="등록" />
        <input type="button" value="취소" />
    </table>
</form>
</section>
</body>
```

```
@RequestMapping("/empSave.do")
public String empSave( Employee e, Model m){
    System.out.println("controller:"+e);
    ser.save(e);
    m.addAttribute("msg", "직원 등록 완료!!!");
    return "Result";
}
```

### ➤ 컨트롤러 메서드의 리턴타입

리턴타입	설 명
ModelAndView	모델정보와 뷰 정보를 담고 있는 객체
Model	뷰에 전달할 객체정보를 담고 있는 객체. 뷰 이름은 요청 url에 의해 결정된다. RequestToViewNameStranlator
Map	뷰에 전달할 객체정보를 담고 있는 Map. 뷰 이름은 요청 url에 의해 결정된다. RequestToViewNameStranlator
String	뷰 이름을 리턴한다.
View	뷰 객체를 직접 리턴한다.
void	메서드가 RequestResponse나 HttpServletResponse타입 파라미터를 갖는 경우 메서드가 직접 응답을 처리한다

### ➤ 리다이렉트/포워드뷰

- forward:/list.do
- redirect:/error.do

## ➤ ViewResolver

구현 클래스명	설 명
InternalResourceViewResolver	뷰 이름과 jsp, jstl, tiles 연동을 위한 View 객체를 리턴한다.
VelocityViewResolver	뷰 이름과 velocity 연동을 위한 View 객체를 리턴한다.
BeanNameViewResolver	뷰 이름과 동일한 빈을 View 객체로 사용한다.
ResourceBundleViewResolver	뷰 이름과 View객체간의 매칭정보를 resource 파일을 사용한다.
XmlViewResolver	뷰 이름과 View객체간의 매칭정보를 xml파일을 사용한다.

### ➤ InternalResourceViewResolver

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

```
@RequestMapping("/empSave.do")  
public String empSave( Employee e, Model m){  
    System.out.println("controller:"+e);  
    ser.save(e);  
    m.addAttribute("msg", "직원 등록 완료!!!");  
    return "Result"; // return "/WEB-INF/jsp/Result.jsp"  
}
```

# Exception

## ➤ Controller별 예외 처리

```
@ExceptionHandler(EmptyResultDataAccessException.class)
public String handleException(){
    System.out.println("EmptyResultDataAccessException");
    return "Error1";
}

@ExceptionHandler
public String handleException(Exception e){
    System.out.println(e.getMessage());
    return "Error2";
}
```

- 해당 클래스내 @RequestMapping 선언된 메서드내 RecordNotFoundException 발생시 이 메서드 실행됨.

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
...
    <h1>사용에 불편을 드려 죄송합니다.</h1>
    ${exception.message}
...
```

# Exception

- 웹 어플리케이션별 공통 예외 처리
- 스프링 설정XML 파일에 정의한다.

```
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
  <property name="exceptionMappings">
    <props>
      <prop key="kr.jaen.exception.RecordNotFoundException">error/notfound</prop>
      <prop key="kr.jaen.exception.DuplicatedException">error/dupld</prop>
      <prop key="java.lang.FileNotFoundException">error/exception</prop>
    </props>
  </property>
</bean>
```

# Spring - File Upload

메이븐 의존 추가 또는 라이브러리 추가

```
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.1</version>
</dependency>
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.4</version>
</dependency>
```

파일업로드 빈 등록

```
<mvc:annotation-driven />
<bean id="multipartResolver"
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
</bean>
```



# Spring - File Upload

form.jsp

```
<form action="upload.do" method="post" enctype="multipart/form-data">
    title :      <input name="title">          <br>
    업로드파일 : <input type="file" name="upfile"> <br>
    <input type="submit" value="업로드">
</form>
```

Controller

```
@RequestMapping(" upload.do ")
public void upload_action(@RequestParam("upfile") MultipartFile upfile,
                          HttpServletRequest req){
    String img = upfile.getOriginalFilename();
    File f= new File(req.getServletContext().getRealPath("upload"), img);
    upfile.transferTo(f);
    //..
}
```

# JSON/XML

- Jackson 라이브러리 메이븐 의존 추가(jar)또는 라이브러리 추가
- 메서드에 `@ResponseBody` 추가
  - 메서드가 객체를 반환하면 JSON 형태로 바뀌어 클라이언트 전달
  - 단, 반환되는 객체의 클래스 위에 `@XmlRootElement` 가 선언되어 있으면 XML로 변환하여 클라이언트에 전달

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.2.3</version>
</dependency>
```

```
@ResponseBody
@RequestMapping("/board/list_json.do")
public List<BoardVo> list_json(ModelMap map ){
    List<BoardVo> list = session.selectList("selectAll");
    return list;
}
```

# JSON/XML

## ➤ ajax.jsp

```
...
<script src="//code.jquery.com/jquery-1.11.3.min.js"></script>
<script type="text/javascript">
    function loadList() {
        $.ajax({
            url:"list_json.do",
            success:function(data){
                $("#info").empty();
                $.each(data,function(){
                    var txt = this.no+" : "+this.title+"<br>";
                    $("#info").append(txt)
                });
            }
        });
    }
</script>
...
<button onclick="loadList()">list</button>
<div id="info"></div>
```

2.1 Spring 소개

2.2 Spring DI

2.3 Spring AOP

2.4 Spring MVC

**2.5 Spring Database**

2.6 Spring & myBatis

- JDBC: JdbcTemplate
- Hibernate: hibernate3.LocalSessionFactoryBean
- IBatis / Mybatis:
- JPA: EntityManagerFactory
- JDO: Java Data Objects

### ➤ DataSource : DB Connection 관리

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
  <context:property-placeholder location="classpath:/config/jdbc.properties"/>
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
    <property name="maxActive" value="${jdbc.maxPoolSize}"/>
  </bean>
  <bean class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg ref="dataSource"/>
  </bean>
</beans>
```

```
jdbc.driverClassName=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:XE
jdbc.username=jaen
jdbc.password=jaen
jdbc.maxPoolSize=5
```

## ▶ JDBC 지원 클래스

클래스명	설명
JdbcTemplate	기본적인 jdbc 템플릿 클래스
NamedParameterJdbcTemplate	PreparedStatement에서 이름을 가진 파라미터를 사용할 수 있도록 지원
SimpleJdbcTemplate	자바 5의 가변인자를 이용하도록 지원
SimpleJdbcInsert	데이터 삽입을 위한 인터페이스 제공
SimpleJdbcCall	프로시저 호출을 위한 인터페이스 제공

### ▶ DAO 클래스 구현

```
package kr.jaen.spring;

@Repository("dao")
public class EmpDAOImpl implements EmpDAO {
    @Autowired
    JdbcTemplate temp;

    @Override
    public void save(Employee e) {
        String sql="insert into Employee values(?,?,?)";
        temp.update(sql,e.getSnum(),e.getSname(),e.getSalary());
    }
}
```



## ➤ 조회를 위한 메서드

- `query(String sql, RowMapper<T> rowMapper)`
- `List<T> query(String sql, Object[] args, RowMapper<T> rowMapper)`
- `List<T> query(String sql, Object[] args, int[] argTypes, RowMapper<T> rowMapper)`
- `List<T> queryForList(String sql, Class<T> elementType)`
- `List<T> queryForList(String sql, Object[] args, Class<T> elementType)`
- `List<T> queryForList(String sql, Object[] args, int[] argTypes, Class<T> elementType)`

```
public List<Employee> search() {  
    List<Employee> list;  
    String sql="Select * from Employee";  
    list=temp.query(sql,new BeanPropertyRowMapper<>(Employee.class));  
    return list;  
}
```

## ➤ 조회를 위한 메서드

- T queryForObject(String sql, RowMapper<T> rowMapper)
- T queryForObject(String sql, Object[] args, RowMapper<T> rowMapper)
- T queryForObject(String sql, Object[] args, int[] argsTypes, RowMapper<T> rowMapper)
- T queryForObject(String sql, Class<T> requiredType)
- T queryForObject(String sql, Object[] args, Class<T> requiredType)
- T queryForObject(String sql, Object[] args, int[] argsTypes, Class<T> requiredType)

```
public Employee search(int snum){  
    Employee e=temp.queryForObject("Select * from Employee where snum=?"  
                                   , new BeanPropertyRowMapper<>(Employee.class) , snum);  
    return e;  
}
```

## ➤ 조회를 위한 메서드

- `int queryForInt(String sql)`
- `int queryForInt(String sql, Object... args)`
- `int queryForInt(String sql, Object[] args, int[] argTypes)`
- `int queryForLong(String sql)`
- `int queryForLong(String sql, Object... args)`
- `int queryForLong(String sql, Object[] args, int[] argTypes)`

```
String sql="Select count(*) from product";
```

```
int count=temp.queryForInt(sql);
```

```
System.out.println("count:"+count);
```

```
String sql2="Select price from product where pnum=?";
```

```
int price=temp.queryForInt(sql2,8);
```

```
System.out.println("price:"+price);
```

## ▶ 삽입/수정/삭제를 위한 메서드

- `int update(String arg)`
- `int update(String arg, Object... args)`
- `int update(String arg, Object[] args, int[] argTypes)`

```
public void save(Employee e) {
```

```
    String sql="insert into Employee values(?,?,?)";
```

```
    temp.update(sql,e.getSnum(),e.getSname(),e.getSalary());
```

```
    //temp.update("Update product set price=? where pnum=?", 6000, 1233);
```

```
    //temp.update("delete from product where pnum=?",1233);
```

```
}
```

## ➤ NamedParameterJdbcTemplate

```
BeanFactory f= new ClassPathXmlApplicationContext("/config/config.xml");  
NamedParameterJdbcTemplate  
    ntemp=f.getBean(NamedParameterJdbcTemplate.class);  
  
String sql="Insert into Product(pnum,pname,price,pdesc)  
           values(:PNUM, :PNAME, :PRICE, :PDESC)";  
  
ntemp.update(sql,new MapSqlParameterSource()  
              .addValue("PNUM",1222)  
              .addValue("PNAME","SmartTV")  
              .addValue("PRICE",5000)  
              .addValue("PDESC","Good~")  
);
```

## ➤ NamedParameterJdbcTemplate

- `List<T> query(String sql, Map<String, ?> paramMap, RowMapper<T> rowMapper)`
- `List<T> queryList(String sql, Map<String, ?> paramMap, Class<T> elementType)`
- `T queryForObject(String sql, Map<String, ?> paramMap, RowMapper<T> rowMapper)`
- `T queryForObject(String sql, Map<String, ?> paramMap, Class<T> elementType)`
- `int queryForInt(String sql, Map<String, ?> paramMap)`
- `long queryForLong(String sql, Map<String, ?> paramMap)`
- `int update(String sql, Map<String, ?> paramMap)`

2.1 Spring 소개

2.2 Spring DI

2.3 Spring AOP

2.4 Spring MVC

2.5 Spring Database

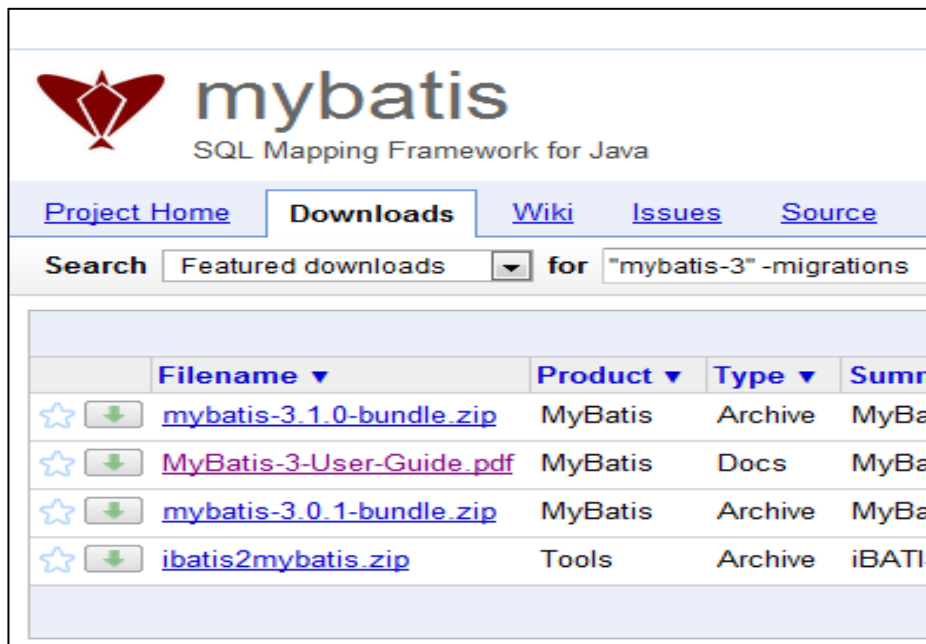
**2.6 Spring & myBatis**

➤ 가장 널리 쓰이는 데이터 매핑 프레임워크

➤ Persistence Framework

➤ 특징

- 외부로 뺀 XML
- 동적 SQL
- 캡슐화된 SQL





## ➤ myBatis 환경설정 파일 (mybatis.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
PUBLIC "-//mybatis.org//DTD config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
<settings> <setting name="jdbcTypeForNull" value="NULL"/></settings>
<typeAliases>
    <typeAlias alias="employee" type=" kr.jaen.spring.Employee"/>
</typeAliases>
</configuration>
```

- DB와 애플리케이션 사이에 데이터를 전달한 vo 클래스를 등록한다.

## ▶ 매핑파일요소 ( EmpMapper.xml )

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="emp">
  <select id="selectSnum" parameterType="int" resultType="employee">
    select * from Employee where snum = #{value}
  </select>
  <select id="selectAll" resultType="employee">
    select * from Employee
  </select>
  <insert id="empInsert" parameterType="employee">
    Insert into Employee values( #{snum},#{sname},#{salary} )
  </insert>
  <delete id="empDelete" parameterType="int">
    Delete from Employee where snum=#{value}
  </delete>
  <update id="empUpdate" parameterType="map">
    update Employee set salary=#{salary} where snum=#{snum}
  </update>
</mapper>
```

- namespace :sql매핑정보들을 구분하기 위한 값
- <select> : select태그를 기술하기 위한 태그
- id : 매핑정보를 구분하기 위한 id값
- parameterType : 쿼리 실행시 넘겨받는 인수의 타입
- resultType : 실행결과를 저장하기 위한 타입
- #{no} :인라인 파라미터. 파라미터와 대체되는 요소

## ▶ Spring 환경 설정에 myBatis 추가

```
<context:property-placeholder location="classpath:/config/jdbc.properties" />
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}" />
    <property name="url" value="${jdbc.url}" />
    <property name="username" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
    <property name="maxActive" value="${jdbc.maxPoolSize}" />
</bean>
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:config/mybatis.xml" />
    <property name="mapperLocations">
        <list>
            <value> classpath:config/EmpMapper.xml </value>
        </list>
    </property>
</bean>
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg ref="sqlSessionFactory" />
</bean>
```

## ▶ DAO 클래스 구현

```
package kr.jaen.spring;
@Repository("dao")
public class EmpDAOImpl implements EmpDAO {
    @Autowired
    SqlSession sess;
    public void save(Employee e) {
        int co=sess.insert("emp.empInsert", e);
        System.out.println("save:"+co);
    }
    public List<Employee> search() {
        return sess.selectList("emp.selectAll");
    }
    public Employee search(int snum){
        return sess.selectOne("emp.selectSnum",snum);
    }
    public void update(int snum, int salary){
        Map<String,Integer> m=new HashMap<>();
        m.put("snum",snum);
        m.put("salary",salary);
        sess.update("emp.empUpdate", m);
    }
}
```

## ➤ SqlSessionFactoryBuilder

- SqlSessionFactory 생성기능 제공
- Factory 패턴을 통해 필요할 때 마다 생성하는것을 권장

## ➤ SqlSessionFactory

- SqlSession객체 생성기
- 한 개만 유지토록 권장. 즉 공유해서 사용토록

## ➤ SqlSession

```
SqlSession session = sqlSessionFactory.openSession();  
try {  
    // 로직 코드 실행  
} finally {  
    session.close();  
}
```

➤ 하나의 VO 객체에 List 담기

```
<select id="selectOrders" parameterType="int" resultMap="ordersMap">
  select e.snum, sname, onum,quant,price from employee e, orders o
  where o.snum=e.snum and e.snum=#{value}
</select>

<resultMap type="Employee" id="ordersMap">
  <result property="snum" column="snum"/>
  <result property="sname" column="sname"/>
  <collection property="olist" ofType="Orders">
    <result property="onum" column="onum"/>
    <result property="quant" column="quant"/>
    <result property="price" column="price"/>
  </collection>
</resultMap>
```

## ➤ 동적 SQL이란

- if
- choose (when, otherwise)
- trim (where, set)
- foreach

### ➤ if

```
<select id="selectList" parameterType="string" resultType="Employee">
  select * from Employee
  <where>
    <if test="value !=null">
      sname= #{value}
    </if>
  </where>
  order by snum
</select>
```

```
<select id="selectList" parameterType="string" resultType="Employee">
  select * from Employee
  <where>
    <if test="sname !=null">
      sname like #{sname}
    </if>
    <if test="salary !=null">
      and salary >= #{salary}
    </if>
  </where>
  order by snum
</select>
```



➤ choose, when, other

```
<select id="findNotice" parameterType="Notice" resultType="Notice">
    SELECT * FROM Notice WHERE state = 'ACTIVE'
    <choose>
        <when test="title != null">
            AND title like #{title}
        </when>
        <when test="author != null and author.name != null">
            AND author_name like #{author.name}
        </when>
        <otherwise>
            AND featured = 1
        </otherwise>
    </choose>
</select>
```

➤ trim, where, set

```
<update id="updateAuthorIfNecessary" parameterType="domain.Notice.Author">
update Author
<set>
    <if test="username != null">username=#{username},</if>
    <if test="password != null">password=#{password},</if>
    <if test="email != null">email=#{email},</if>
    <if test="bio != null">bio=#{bio}</if>
</set>
where id=#{id}
</update>
```

위와 동일한 효과

```
<trim prefix="SET" suffixOverrides=",">
```

...

```
</trim>
```

접두사는 추가하고, 접미사를 삭제한다.

## ➤ foreach

- in을 적용할 경우
- 파라미터 객체로 MyBatis 에 List 인스턴스나 배열을 전달 할 수 있다.
- MyBatis 는 Map으로 자동으로 감싸고 이름을 키로 사용
- List 인스턴스는 "list" 를 키로 사용하고, 배열 인스턴스는 "array" 를 키로 사용

```
<select id="selectPostIn" resultType="domain.Notice.Post">
SELECT * FROM POST P
WHERE ID in
<foreach item="item" index="index" collection="list" open="(" separator="," close=")">
    #{item}
</foreach>
</select>
```

# Transaction

## ➤ Tx란?

- long transaction
- short transaction
- global transaction

## ➤ 속성

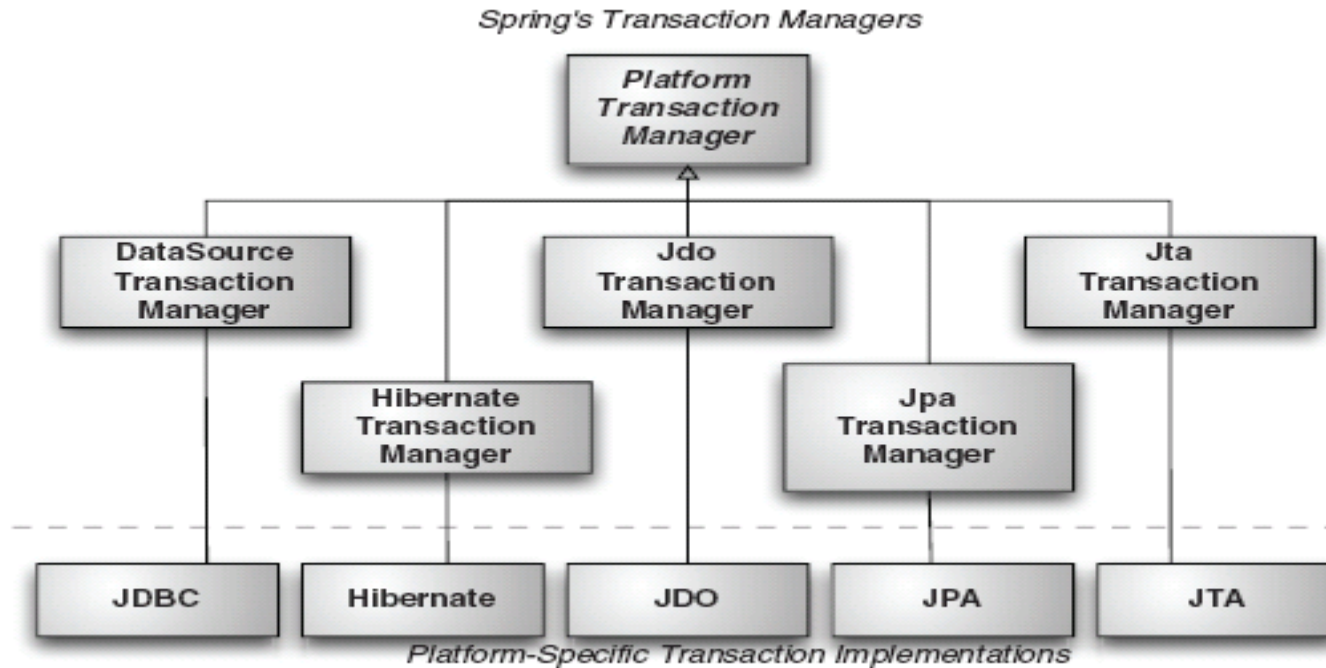
- 전파(propagation)속성
- 독립성(isolation) 수준
- 시간만료(timeout)
- 읽기전용상태
- 롤백대상
- 커밋대상

## ➤ 업무코드단위(Service 메서드)에 적용

## ➤ AOP이용

- transaction advice

# TransactionManager



```
<bean id="transactionManager"  
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
  <property name="dataSource" ref="dataSource" />  
</bean>
```

# 선언적 트랜잭션 설정 XML

```
<!-- TX를 적용시킬 대상을 지정 -->
<aop:config>
    <aop:advisor advice-ref="txAdvice"
        pointcut="execution(* *..*Service.insert*(..))"/>
</aop:config>
<!-- TX작업시 기능을 정의 -->
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*" />
    </tx:attributes>
</tx:advice>
```

# 트랜잭션 설정 XML

```
<tx:advice id="transactionAdvice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="get*" read-only="true" />
    <tx:method name="update*"
      propagation="REQUIRED"
      isolation="READ_COMMITTED"
      timeout="10"
      read-only="false"
      rollback-for="sample.biz.exception.BusinessException" />
  </tx:attributes>
</tx:advice>
```

propagation(전파속성) REQUIRED

isolation(독립성수준) READ\_COMMITTED

timeout(시간만료) 10초

읽기전용선택(read-only) false

콜백대상 예외(rollback-for) RuntimeException이면 자동 롤백, 그외 exception이면  
반드시 기술해야 함. ...BizException

# 트랜잭션 설정 어노테이션

## @Transactional

- 클래스에 지정: 클래스내 전체 메서드
- 메서드에 지정: 특정 메서드만

```
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource" />
</bean>

<tx:annotation-driven transaction-manager="transactionManager"/>
```

```
@Transactional(
    propagation=Propagation.REQUIRED,
    isolation=Isolation.READ_COMMITTED,
    timeout=10, readOnly=false,
    rollbackFor = BusinessException.class )
public void updatePet(Pet pet) throws BusinessException {
    petDao.updatePet(pet);
}
```