

# Spring 3.2 & MyBatis & Hibernate & JPA

Oracle Java 교육학원

# 개발환경 설정

종류	제품명
OS	Windows 7
JVM	Java 1.8
IDE	Spring Tool Suite 3.6.4.RELEASE
Build Tool	Apache Maven 3.3.9
Application Server	Pivotal tc Server Developer Edition v3.1
Web Browser	Google Chrome

# 설정 파일 준비

- Pom.xml
  - Web Library
  - Logger Library
  - Tag Library
  - Spring Framework
  - Json Library
  - Maven 리포지토리에 없는 jar파일 추가
- Web.xml (web용)
- 공통 Spring Bean 설정 파일
- Spring MVC 용 설정파일(web용)

# Web Library

```
<!-- Web Library -->  
  <dependency>  
    <groupId>javax.servlet</groupId>  
    <artifactId>servlet-api</artifactId>  
    <version>3.1</version>  
    <scope>provided</scope>  
  </dependency>  
  <dependency>  
    <groupId>javax.servlet.jsp</groupId>  
    <artifactId>jsp-api</artifactId>  
    <version>2.1</version>  
    <scope>provided</scope>  
  </dependency>
```

# Tag Library

```
<!-- Tag Library -->  
  <dependency>  
    <groupId>taglibs</groupId>  
    <artifactId>standard</artifactId>  
    <version>1.1.2</version>  
  </dependency>  
  <dependency>  
    <groupId>javax.servlet</groupId>  
    <artifactId>jstl</artifactId>  
    <version>1.2</version>  
  </dependency>
```

# Logger Library

```
<!-- Logger Library -->
```

```
<dependency>
```

```
<groupId>org.slf4j</groupId>
```

```
<artifactId>jcl-over-slf4j</artifactId>
```

```
<version>1.6.6</version>
```

```
</dependency>
```

```
<dependency>
```

```
<groupId>org.slf4j</groupId>
```

```
<artifactId>slf4j-api</artifactId>
```

```
<version>1.6.6</version>
```

```
</dependency>
```

```
<dependency>
```

```
<groupId>org.slf4j</groupId>
```

```
<artifactId>slf4j-log4j12</artifactId>
```

```
<version>1.6.6</version>
```

```
</dependency>
```

```
<dependency>
```

```
<groupId>log4j</groupId>
```

```
<artifactId>log4j</artifactId>
```

```
<version>1.2.15</version>
```

```
</dependency>
```

# Spring Framework Library

```
<!-- Sprink Framework -->
```

```
  <dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-core</artifactId>
```

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

```
  </dependency>
```

```
  <dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-web</artifactId>
```

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

```
  </dependency>
```

```
  <dependency>
```

```
    <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-webmvc</artifactId>
```

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

```
  </dependency>
```

```
</dependencies>
```

# AspectJ Library

```
<dependency>
```

```
  <groupId>org.aspectj</groupId>
```

```
  <artifactId>aspectjweaver</artifactId>
```

```
  <version>${aspectJ.version}</version>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>org.aspectj</groupId>
```

```
  <artifactId>aspectjrt</artifactId>
```

```
  <version>${aspectJ.version}</version>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>cglib</groupId>
```

```
  <artifactId>cglib-nodep</artifactId>
```

```
  <version>2.2.2</version>
```

```
</dependency>
```



# Spring Framework Library

변경하길 원하는 버전 입력

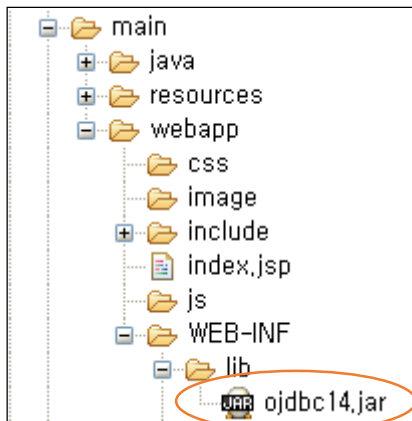
```
<properties>  
  <spring.version>3.2.13.RELEASE</spring.version>  
  <slf4j.version>1.6.6</slf4j.version>  
  <aspectJ.version>1.6.10</aspectJ.version>  
  <jackson.version>1.9.10</jackson.version>  
</properties>
```

# JSON Library

```
<dependencies>
  <!-- JSON Library -->
  <dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-core-asl</artifactId>
    <version>${jackson.version}</version>
  </dependency>
  <dependency>
    <groupId>org.codehaus.jackson</groupId>
    <artifactId>jackson-mapper-asl</artifactId>
    <version>${jackson.version}</version>
  </dependency>
</dependencies>
```

# Maven 리포지토리에 없는 jar파일 추가

- /src/main/webapp/WEB-INF/lib/ 이하에 직접 추가

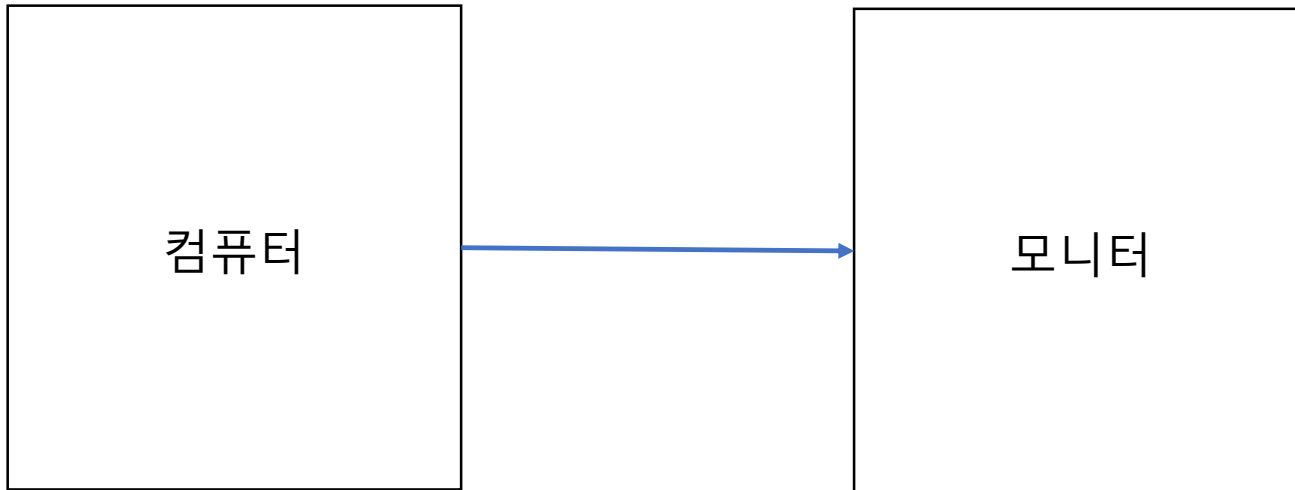


```
<dependency>
    <groupId>oracle.ojdbc</groupId>
    <artifactId>ojdbc</artifactId>
    <version>14</version>
    <scope>system</scope>
    <systemPath>${basedir}/src/main/webapp/WEB-INF/lib/ojdbc14.jar</systemPath>
</dependency>
```

# DI(Dependency Injection)

- 의존성 주입(직역)
- 의존성(Dependency)란?
  - 객체를 생성하기 위해 필요한 조건을 말함

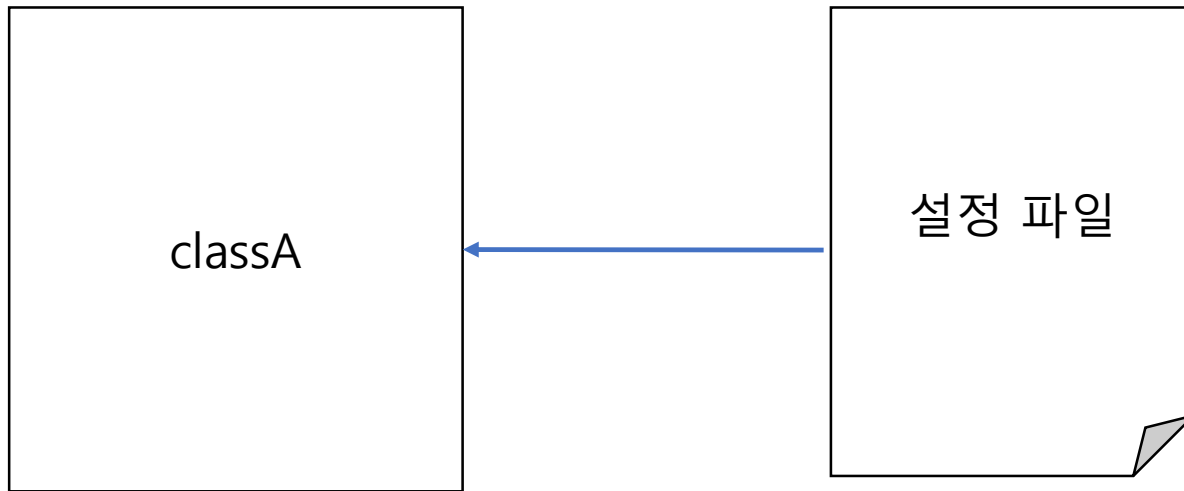
# DI(Dependency Injection)



컴퓨터는 모니터에 "의존성" 이 있다

# DI(Dependency Injection)

- 주입(Injection)이란?
  - 외부로부터 설정한다는 의미



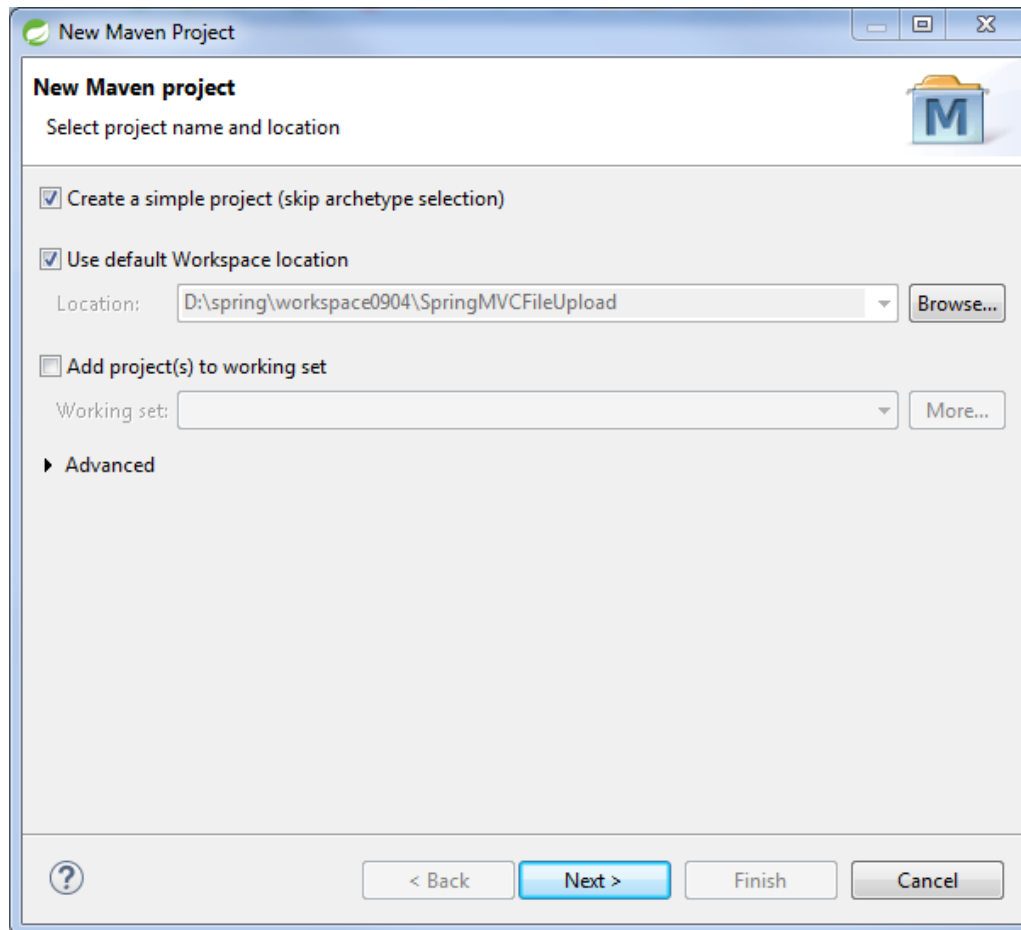
즉, DI란 "설정을 클래스사용으로부터 분리(the principal of separating configuration from use) " 한다는 의미

# Spring Framework

- Spring Framework란?
- 2002年 Rod Johnson氏가 개발한 프레임워크
- 저서: Expert One-on-One J2EE Design and Development(국내 제목: J2EE 설계와 개발) 에서 주창

# New Maven Project

Create a simple project 선택





# New Maven Project

**New Maven Project**  
Configure project

**Artifact**

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

**Parent Project**

Group Id:

Artifact Id:

Version:

▶ **Advanced**

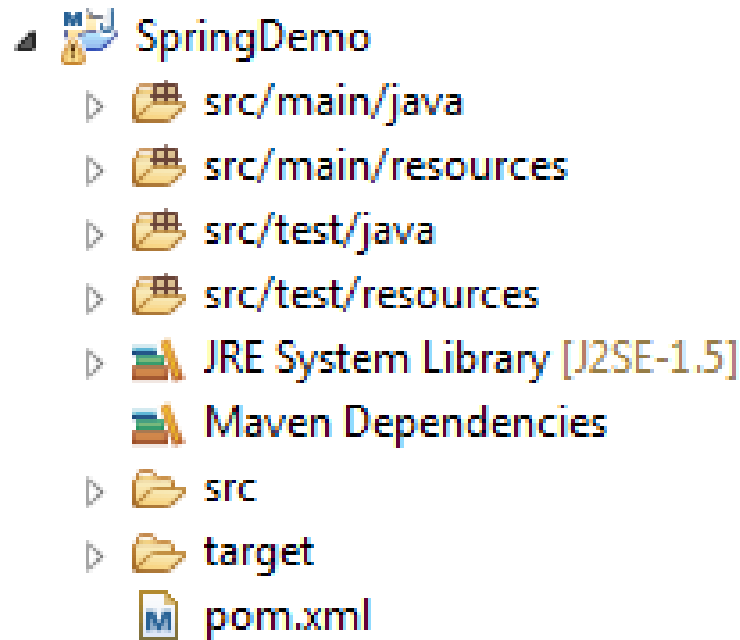
# Simple Spring Maven Project

src/main/java : 자바파일 두는 곳

src/main/resource: 스프링 설정 파일 두는 곳

src/test/java: Junit 테스트 파일(java)

src/test/resource: 테스트용 설정 파일 두는 곳



# Spring DI Test

- DrawingApp.java
- Triangle.java
- spring.xml

# Spring DI Test

```
package com.oraclejava.drawing;

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

public class DrawingApp {

    public static void main(String[] args) {
        //Triangle triangle = new Triangle();

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

        Triangle triangle = (Triangle)factory.getBean("triangle");

        triangle.draw();
    }
}
```

## Spring DI Test

```
package com.oraclejava.drawing;  
  
public class Triangle {  
  
    public void draw() {  
        System.out.println("삼각형을 그립니다");  
    }  
}
```

# Spring DI Test

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>
<bean id="triangle" class="com.oraclejava.drawing.Triangle"
/>
</beans>
```

# AOP

- AOP란?
  - 관점지향 프로그래밍
  - "횡단관심사를 분리"(Separation of Cross-Cutting Concerns)
  - "관심사"란? 하나의 처리
  - "횡단"이란? 여러 개의 모듈에 산재해 있다는 뜻

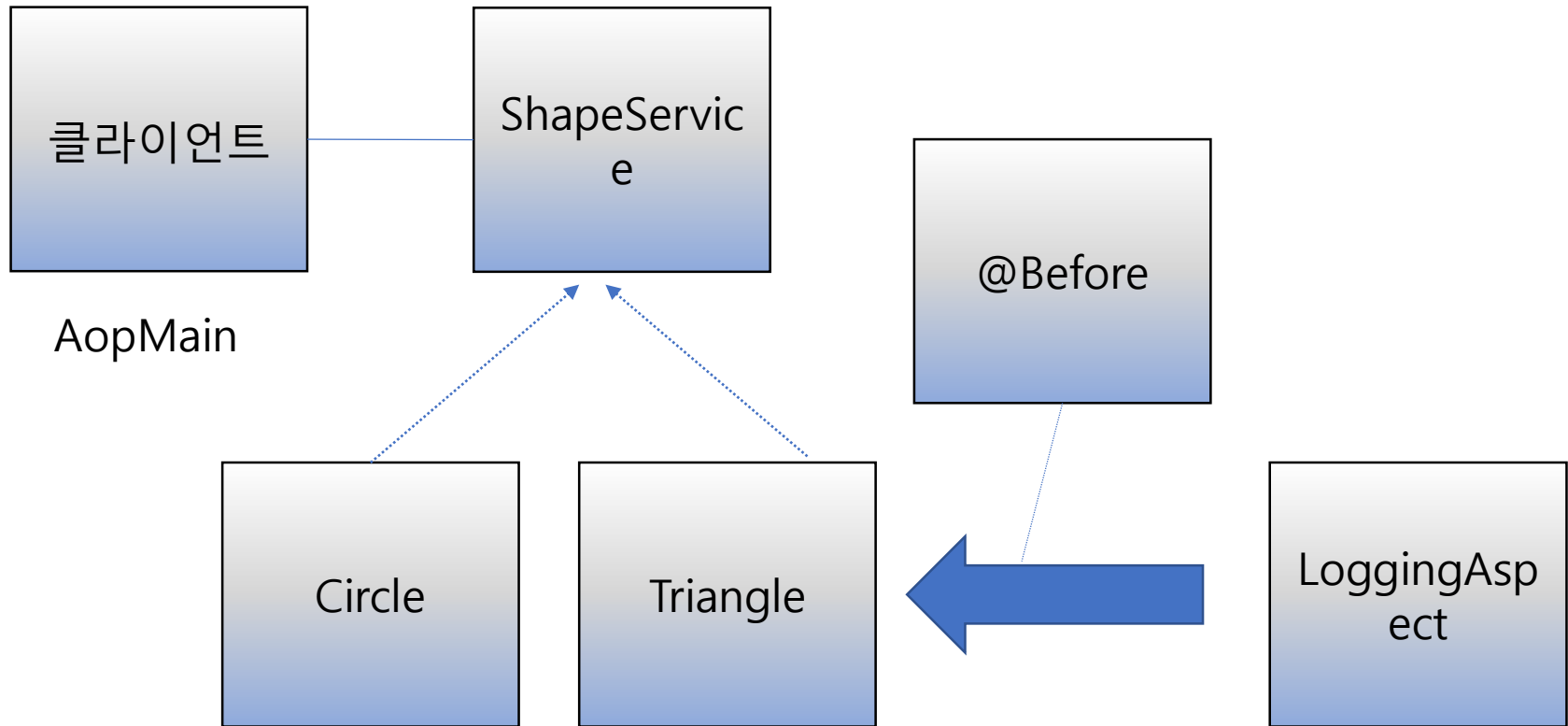
# AOP



 하나의 처리(관심사)



# AOP Sample



## Spring AOP Sample

```
package com.oraclejava.drawing.model;

public class Circle {
    private String name;

    public String getName() {
        return name;
    }

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

}
```

## Spring AOP Sample

```
package com.oraclejava.drawing.service;

import com.oraclejava.drawing.model.Circle;
import com.oraclejava.drawing.model.Triangle;

public class ShapeService {

    private Triangle triangle;
    private Circle circle;

    public Triangle getTriangle() {
        return triangle;
    }
    public void setTriangle(Triangle triangle) {
        this.triangle = triangle;
    }
    public Circle getCircle() {
        return circle;
    }
    public void setCircle(Circle circle) {
        this.circle = circle;
    }
}
```

## Spring AOP Sample

```
package com.oraclejava.drawing.aspect;

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

@Aspect
public class LoggingAspect {

    @Before("execution(public String getName())")
    public void LoggingAdvice() {
        System.out.println("Logging Advice is running");
    }
}
```

# Spring AOP Sample

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

<aop:aspectj-autoproxy />
```

## Spring AOP Sample

```
<bean name="triangle"  
class="com.oraclejava.drawing.model.Triangle">  
<property name="name" value="삼각김밥" />  
</bean>
```

```
<bean name="circle" class="com.oraclejava.drawing.model.Circle">  
<property name="name" value="보름달" />  
</bean>
```

```
<bean name="shapeService"  
class="com.oraclejava.drawing.service.ShapeService"  
autowire="byName" />
```

```
<bean name="loggingAspect"  
class="com.oraclejava.drawing.aspect.LoggingAspect" />  
</beans>
```

# Spring AOP Sample

9월 08, 2017 3:33:56 오후

org.springframework.context.support.ClassPathXmlApplicationContext  
prepareRefresh

INFO: Refreshing

org.springframework.context.support.ClassPathXmlApplicationContext@1a6c5a9e:  
startup date [Fri Sep 08 15:33:56 KST 2017]; root of context hierarchy

9월 08, 2017 3:33:57 오후

org.springframework.beans.factory.xml.XmlBeanDefinitionReader  
loadBeanDefinitions

INFO: Loading XML bean definitions from class path resource [spring.xml]

9월 08, 2017 3:33:57 오후

org.springframework.beans.factory.support.DefaultListableBeanFactory  
preInstantiateSingletons

INFO: Pre-instantiating singletons in

org.springframework.beans.factory.support.DefaultListableBeanFactory@1324409e:  
defining beans

[org.springframework.aop.config.internalAutoProxyCreator,triangle,circle,shapeService,loggingAspect]; root of factory hierarchy

Logging Advice is running

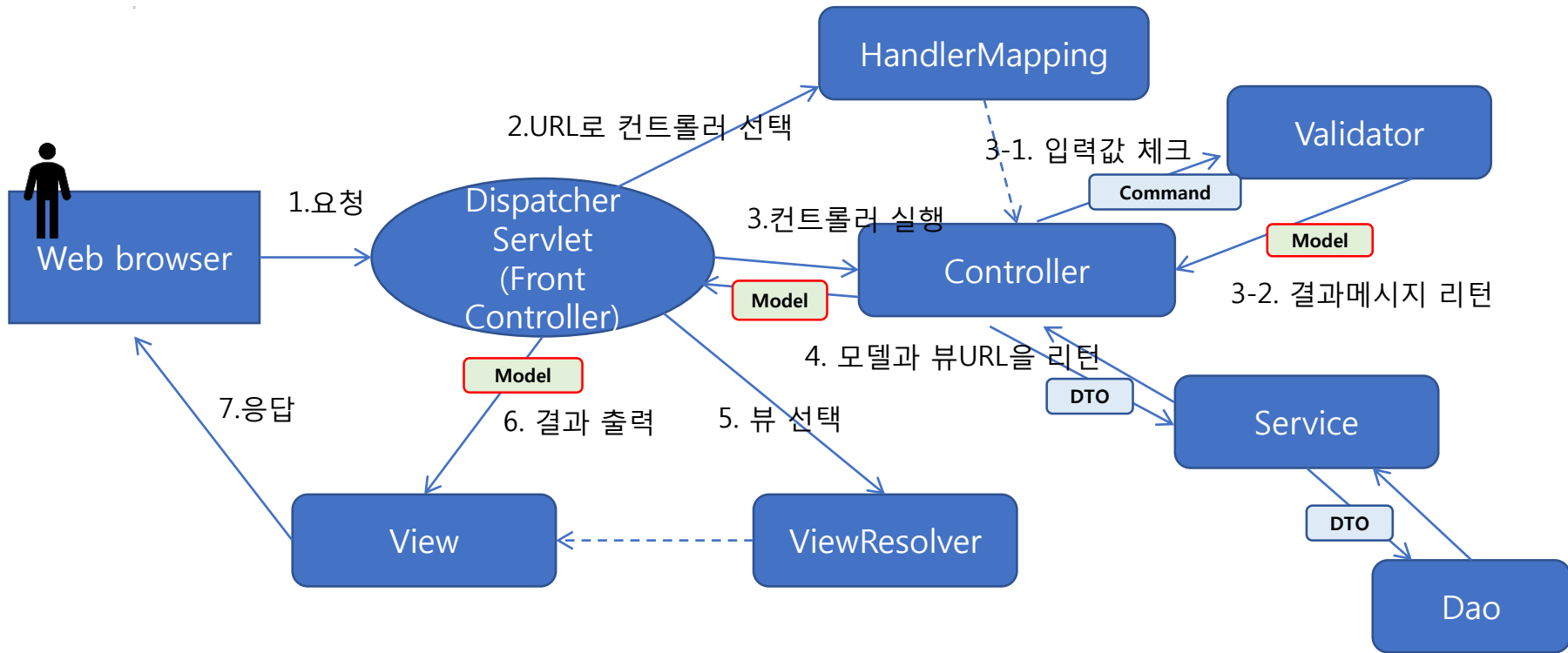
보름달

# Spring MVC 개발

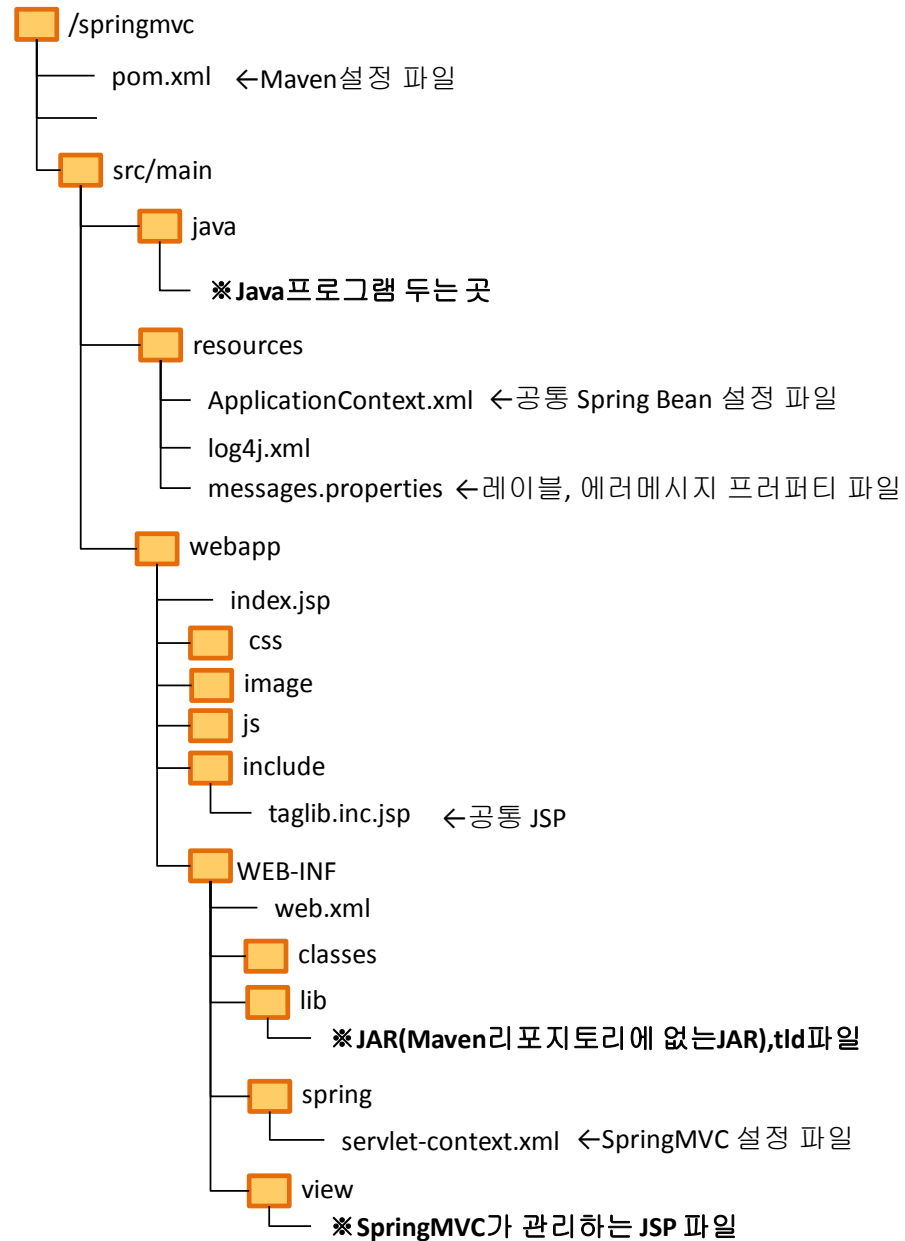
- Spring MVC 프로젝트 작성
- Spring MVC 처리 Flow
- 파일 구성
- 설정 파일 준비



# Spring MVC 처리 Flow



# 파일 구성



# web.xml

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

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee  
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"  
version="3.1">
```

...설정

```
</web-app>
```

# 인코딩 필터

<filter>

<filter-name>CharacterEncodingFilter</filter-name>

<filter-

class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

# 공통 Spring Bean 취득

```
<listener>
```

```
    <listener-class>
```

```
        org.springframework.web.context.ContextLoaderListener
```

```
    </listener-class>
```

```
</listener>
```

```
<context-param>
```

```
    <param-name>contextConfigLocation</param-name>
```

```
    <param-value>
```

```
        /WEB-INF/applicationContext.xml
```

```
    </param-value>
```

```
</context-param>
```

# Spring Web 스코프 이용할 경우 설정 추가

```
<listener>
```

```
    <listener-class>
```

```
    org.springframework.web.context.request  
        .RequestContextListener
```

```
    </listener-class>
```

```
</listener>
```

# Spring MVC Bean 취득

```
<servlet>
    <servlet-name>webmvc-dispatcher</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</serv
let-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>webmvc-dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

# 공통Spring Bean파일

```
<?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"
```

```
xmlns:util="http://www.springframework.org/schema/util"
```

```
xmlns:tx="http://www.springframework.org/schema/tx"
```

```
xsi:schemaLocation="http://www.springframework.org/schema/mvc
```

```
http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
```

```
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
```

```
http://www.springframework.org/schema/tx
```

```
http://www.springframework.org/schema/tx/spring-tx-3.2.xsd
```

```
http://www.springframework.org/schema/util
```

```
http://www.springframework.org/schema/util/spring-util-  
3.2.xsd">
```



# 공통Spring Bean파일

```
<!-- Autowired Scan 유효화 정의 -->
<context:annotation-config />
<!-- 설정값 -->
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <value>classpath:database.properties</value>
<value>classpath:app.properties</value>            </list>
        </property>
    </bean>
```

설정값을 외부화  
SpringBean 안에서 [키]로 접근 가능

# 공통Spring Bean파일

```
<bean id="messageSource"  
class="org.springframework.context.support.ReloadableResourc  
eBundleMessageSource">  
    <property name="basenames">  
        <list>  
            <value>classpath:message/message</value>  
<value>classpath:message/label</value>          </list>  
        </property>  
    </bean>
```



공통 메시지 정의  
JSP 커스텀 태그<spring:message>으로 접근 가능

# 공통Spring Bean파일

Autowire 자동 설정 패키지 명 설정

<!-- Scan대상 패키지 정의

Scan대상 어노테이션 : @Component, @Repository, @Service, or @Controller. -->

<context:component-scan base-package="oraclejava.service" />

<!-- SpringMVC의 어노테이션 유효화 -->

<annotation-driven />

SpringMVC의 어노테이션을 유효화한다

# Spring MVC용 설정파일

```
<?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">
```

```
...
```

```
</beans>
```

# Spring MVC용 설정파일

<!-- 뷰를 정의합니다 -->

```
<bean  
class="org.springframework.web.servlet.view.InternalResourceViewResolver  
>
```

```
    <beans:property name="prefix" value="/WEB-INF/views/" />
```

```
    <beans:property name="suffix" value=".jsp" />
```

```
</bean>
```



ViewResolver를 정  
의한다

# 컨트롤러 작성

```
/**
 * 간단 컨트롤러
 */
@Controller
모델 (값을 유지하는 클래스) 작성
public class HelloWorldController {
    @RequestMapping("/hello")
    public ModelAndView helloWorld() {

        ModelAndView mav = new ModelAndView("/helloView");

        // 직접 모델에 메시지 설정
        mav.addObject("message1", " Hello World, <strong>Spring MVC!</strong> ");

        // 모델을 취득해서 메시지 설정
        mav.getModelMap().put("message2", " 메시지2");

        mav.addObject("currentDate", new Date());

        return mav;
    }
}
```

# JSP 정의(helloView.jsp)

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

```
<%@ include file="/include/taglibs.inc.jsp" %>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title>Insert title here</title>
```

```
</head>
```

```
<body>
```

```
    <ul>
```

```
        <li>message1=${message1}</li>
```

```
        <li>message1(커스텀태그)= <c:out value="${message1}" /> </li>
```

```
        <li>message2=${message2}</li>
```

```
        <li>currentDate(포맷)= <fmt:formatDate value="${currentDate}" pattern="yyyy년 MM월
dd일"/> </li>
```

```
    </ul>
```

```
</body>
```

```
</html>
```

# 공통 jsp(taglibs.inc.jsp)

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

```
<%-- JSTL 정의 --%>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
```

```
<%-- Spring 커스텀 태그 정의--%>
```

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

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



# 웹 브라우저에서 확인

<http://localhost:8080/spring/hello>



- `message1=Hello world, Spring MVC`
- `message1(커스텀태그)=Hello world, <strong>Spring MVC</strong>`
- `message2=메시지2`
- `currentTime(포맷)=2017년 01월 20일`

# 컨트롤러의 인수와 리턴값 정리

# 컨트롤러 인수정리

No.	Java형	설명	I/O
1	ServletRequest /HttpServletRequest	Servlet API Request	I
2	ServletResponse /HttpServletResponse	Servlet API Response	O
3	HttpSession	Servlet API 세션. Null이 안되고, 항상 Spring MVC에서 생성된다.	I/O
4	org.springframework.web.context.request.WebRequest /org.springframework.web.context.request.NativeWebRequest	Session, Request등을 취득할 때 Servlet API 대신에 사용 가능.	I/O
5	java.util.Locale	현재의 Locale 정보를 얻을 수 있다.	I
6	java.io.InputStream /java.io.Reader	요청된 정보의 입력 스트림을 얻을 수 있다.	I
7	java.io.OutputStream /java.io.Writer	응답할 정보의 출력 스트림을 얻을 수 있다.	O
9	@RequestParam이 붙은 매개 변수	모델에 저장되어 있는 요청 파라미터를 얻을 수 있다. <ul style="list-style-type: none"> <li>이름이 없는 경우 <ul style="list-style-type: none"> <li>@RequestParam String userId</li> </ul> </li> <li>이름이 있는 경우 <ul style="list-style-type: none"> <li>@RequestParam("userId") String userId</li> </ul> </li> </ul>	I

# 컨트롤러 인수정리

No.	Java형	설명	I/O
10	@RequestHeader가 붙은 매개변수	특정 Request Http Header를 취득	I
11	@RequestBody가 붙은 매개변수	HTTP Request본체를 취득. JSON/XML데이터를 취득할 때 사용.	I
13	java.util.Map /org.springframework.ui.Model /org.springframework.ui.ModelMap	View에 전달할 암묵적 모델을 취득함. 리턴값으로 ModelAndView사용하지 않을 경우 사용함.	O
14	@ModelAttribute가 붙은 매개변수	Form의 값을 JavaBean에 바인드된 변수에 전달함. Spring MVC에서는 Command라고 함. @InitBinder가 붙은 메서드를 통해 Bind시 세밀한 처리가 가능 ModelAttribute에 이름을 설정하지 않는 경우 초기값은 "command"가 됨. 클래스에 @SessionAttributes를 붙이면 Form의 값을 세션 스코프에 보존시킨다.	I
15	org.springframework.validation.Errors /org.springframework.validation.BindingResult	Form객체의 validation결과가 입력된다.	I
16	org.springframework.web.bind.support.SessionStatus	세션 정보를 삭제하는 등 보조적인 기능	I

# 컨트롤러 리턴값 정리

No.	Java형	설명
1	ModelAndView	View로 지정한 URL에 Model(데이터)를 지정
2	Model	View는 암묵적으로 정해짐. Model을 설정. 보통 현재의 View는 변화없음.
3	java.util.Map	Model과 같음. ModelMap은 Map을 기반으로 만들어져서, ModelMap을 리턴으로 해도 됨.
4	View	View에 지정한 URL로 이동. Model은 암묵적으로 정해짐. 인수로서 Model, @ModelAttribute가 지정되면 그 값이 Model이 됨. View를 구현한 클래스는 JSON, PDF, Excel 등 다양하다.
5	String	View의 URL을 직접 지정. 나머지 View와 같음.
6	void	자신으로 응답이 옴.
7	void (인수가 @ResponseBody인 경우)	인수로 지정한 @ResponseBody를 리턴
8	메서드에 @ResponseBody가 붙어있는 경우	리턴값을 HTTP 응답의 Body로서 리턴. JSON형식으로 리턴할 때 사용
10	임의의 객체	

# 컨트롤러 패턴

1. View(Jsp)로 이동하는 경우
2. View(Jsp)로 이동하는 경우(모델과 함께)
3. 다른 URL로 이동하는 경우
4. URL 응답에서 값을 받는 경우
5. Form에서 값을 받는 경우
6. Form에서 값을 받아, 세션에 저장하는 경우

# 컨트롤러 패턴

- 7. 파일 다운로드(HttpServletResponse사용)
- 8. 파일 다운로드(ResponseEntity사용)
- 9. 파일 다운로드(Writer/OutputStream사용)
- 10. JSON형식으로 리턴하는 경우
- 11. 보내는 곳을 생략하는 경우(메서드 리턴값이 void)
- 12. 보내는 곳을 생략하는 경우(View의 패스가 없음)

# View(Jsp)로 이동하는 경우

- 단순히 jsp로 이동
- 인수 없음
- 리턴값은 String
- 확장자 필요 없음



# View(Jsp)로 이동하는 경우

@Controller

```
public class SampleController {
```

```
    @RequestMapping("/sample")
```

```
    public String sample() {
```

```
        return "/toView";
```

```
    }
```

```
}
```

# View(Jsp)로 이동하는 경우(모델과 함께)

- 인수 없음
- 리턴 값은 ModelAndView

# View(Jsp)로 이동하는 경우(모델과 함께)

```
@Controller
public class SampleController {

    @RequestMapping("/sample")
    public ModelAndView sample() {

        // View 설정
        ModelAndView mav = new ModelAndView("/toView");

        // 모델을 취득해서, 메시지 설정
        mav.addObject("message1", " 메시지1");

        return mav;
    }
}
```

# 다른 URL로 이동하는 경우

- 리턴 값에 접두어 "forward:" 혹은 "redirect:"를 붙인다
- forward는 요청한 곳에서 "POST"메서드로 보냈을 경우, 응답하는 곳에도 "POST"가 된다.
- redirect는 다른 http:// 도 가능하다

# 다른 URL로 이동하는 경우

```
@Controller
public class SampleController {
    // 포워드
    @RequestMapping("/sample1")
    public String forward() {
        return "forward:/hello.html";
    }

    // 리다이렉트
    @RequestMapping("/sample2")
    public String redirect() {
        return "redirect:/hello.html";
    }
}
```

# URL 응답링크에서 값을 받는 경우

- 인수에 @RequestParam을 지정한다
- 인수로서 BindingResult를 지정할 수 없다
- 리턴값은 ModelAndView

# URL 응답에서 값을 받는 경우

@Controller

```
public class SampleController {
```

```
    @RequestMapping(value="/sample", method={RequestMethod.GET})
```

```
    public ModelAndView sample(@RequestParam(value="userCd", required=true)  
Integer id) {
```

```
        // 바인드 시 에러처리
```

```
        if(bindingResult.hasErrors()) {
```

```
            . . . 생략
```

```
        }
```

```
        ModelAndView mav = new ModelAndView("/toView");
```

```
        return mav;
```

```
    }
```

```
}
```

# Form에서 값을 받는 경우

- 인수로서 POST되는 값이 JavaBean에 맵핑할 값이라면, @ModelAttribute를 지정한다
- 인수로서 BindingResult를 지정한다. @ModelAttribute로 파라미터를 받을 경우 반드시 지정해야 한다.
- 리턴값은 ModelAndView



# Form에서 값을 받는 경우

@Controller

```
public class SampleController {
```

```
    @RequestMapping(value="/sample", method={RequestMethod.POST})
```

```
    public ModelAndView sample(@ModelAttribute LoginCommand  
command, BindingResult bindingResult) {
```

```
        // 바인드시 예러처리
```

```
        if(bindingResult.hasErrors()) {
```

```
            . . . 생략
```

```
        }
```

```
        ModelAndView mav = new ModelAndView("/toView");
```

```
        return mav;
```

```
    }
```

```
}
```

# Form에서 값을 받아, 세션에 저장하는 경우

- 인수로서 WebRequest를 지정한다.
- 인수로서 POST되는 값이 JavaBean에 맵핑할 값이라면, @ModelAttribute를 지정한다
- 인수로서 BindingResult를 지정한다. @ModelAttribute로 파라미터를 받을 경우 반드시 지정해야 한다.
- 리턴값은 ModelAndView

# Form에서 값을 받아, 세션에 저장하는 경우

@Controller

```
public class SampleController {
```

```
    @RequestMapping(value="/sample", method={RequestMethod.POST})
```

```
    public ModelAndView sample(WebRequest request, @ModelAttribute  
LoginCommand command, BindingResult bindingResult) {
```

```
        // 바인드시 에러처리
```

```
        if(bindingResult.hasErrors()) {
```

```
            . . . 생략
```

```
        }
```

```
        // 세션에 데이터 저장
```

```
        request.setAttribute("loginUser", "hogehoge", RequestAttributes.SCOPE_SESSION);
```

```
        ModelAndView mav = new ModelAndView("/toView");
```

```
        return mav;
```

```
    }
```

```
}
```

# 파일 다운로드(HttpServletResponse사용)

- 인수로서 HttpServletResponse 사용
- 리턴값은 void
- ServletAPI 직접 사용하므로, Content-Type등을 지정할 수 있다.

# 파일 다운로드(HttpServletResponse사용)

@Controller

```
public class DownloadController {
```

```
    // HttpServletResponse를 사용하는 경우
```

```
    @RequestMapping(value="/downloadFile1", method =  
    {RequestMethod.GET} )
```

```
    public void downloadFile1(HttpServletResponse response) throws  
    IOException {
```

```
        // HTTP 헤더 지정
```

```
        response.setContentType("application/octet-stream");
```

```
        response.setHeader("Content-Disposition",  
        String.format("filename=₩"%s₩"", "sample.txt"));
```

```
        //...
```

```
    }
```

```
}
```

## 파일 다운로드(ResponseEntity사용)

- 직접 ServletAPI을 사용하지 않는 방법
- 리턴값은 ResponseEntity
- HTTP 헤더는 HttpHeaders를 사용

## 파일 다운로드(ResponseEntity사용)

@Controller

```
public class DownloadController {
```

```
    // ResponseEntity를 사용하는 경우
```

```
    @RequestMapping(value="/downloadFile2", method =  
{RequestMethod.GET})
```

```
    public ResponseEntity<String> downloadFile2() throws IOException {
```

```
        // HTTP헤더 지정
```

```
        HttpHeaders headers = new HttpHeaders();
```

```
        headers.setContentType(new MediaType("application", "octet-stream"));
```

```
        headers.set("Content-Disposition", String.format("filename=W"%sW",  
"sample.txt"));
```

```
        String data = "HelloWorld!";
```

```
        byte[] output = data.getBytes();
```

```
        return new ResponseEntity<String>(output, headers, HttpStatus.OK);
```

```
    }
```

```
}
```

# 파일 다운로드(Writer/OutputStream사용)

- 인수로서 Writer 나 OutputStream을 사용.
- ContentType이나, Header를 지정할 수 없기 때문에 ResponseEntity나 HttpServletResponse를 사용하는 것이 일반적
- 리턴값 없음



# JSON형식으로 리턴하는 경우

- 리턴값이 객체
- 메서드에 `@ResponseBody` 어노테이션을 붙인다. (자동적으로, Http헤더 Accept가 `application/json` 이 된다)
- 별도 라이브러리 Jackson이 필요

# JSON형식으로 리턴하는 경우

@Controller

public class JsonController {

    @RequestMapping(value="/ajax/jsonOut1", method={RequestMethod.GET, RequestMethod.POST})

    @ResponseBody

    public List<UserInfoViewDto> jsonOut (@RequestParam String cd) {

        if(StringUtils.isEmpty(departmentCd)) {  
            return new ArrayList<UserInfoViewDto>();  
        }

        List<UserInfoDto> list = /\* Servlet/DAO로부터 취득 \*/;  
        return list;

    }

}

# 보내는 곳을 생략하는 경우(메서드 리턴 값이 void)

- 자신에게 다시 돌아온다.

# 보내는 곳을 생략하는 경우(View의 패스가 없음)

- 자신에게 다시 돌아온다.

form 데이터 송수신

# Form 데이터 송수신

1. @RequestParam에 의한 데이터 송수신
2. Command(@ModelAttribute)에 의한 데이터 송수신
3. 독자 데이터 바인드(@InitBinder)

# @RequestParam에 의한 데이터 송수신

- 컨트롤러에 2종류의 @RequestMapping 설정
  - RequestMethod.GET (form의 초기화)
  - RequestMethod.POST
    - @RequestParam으로 파라미터를 받음
    - required=false (체크박스로 데이터를 받을 때 반드시 설정; 체크하지 않으면 데이터가 null이 됨)

# Command(=@ModelAttribute)에 의한 데이터 송수신

- JavaBean 형식으로 각 프로퍼티에 getter,setter
- 세션에 저장될 것을 고려해서 Serializable 인터페이스를 구현한다.
- 디버깅하기 쉽게 하기 위해 toString()을 구현한다
- 기본형을 사용하지 않는다(래퍼 클래스 사용. 이유: null을 표시할 수 없기 때문에)



# Command 클래스 작성

```
import java.io.Serializable;

public class SampleCommand implements Serializable {

    /** serialVersionUID */
    private static final long serialVersionUID = 1L;

    private String name;
    private String mail;
    private Integer age;
    private Boolean confirmed;

    public SampleCommand() { }

    @Override
    public String toString() {
        return "";
    }

    // getter,setter 생략
}
```

# 컨트롤러 클래스 작성(GET)

@Controller

@RequestMapping("/test/form2")

public class Form2Controller {

// ①command 초기 객체 취득

@ModelAttribute("**sampleCommand**")

public SampleCommand createInitCommand() {

    SampleCommand command = new SampleCommand();

    return command;

}

// ②초기값 설정

@RequestMapping(method=RequestMethod.GET)

public void setupForm(Model model) {

    SampleCommand command = createInitCommand();

    command.setAge(1);

    model.addAttribute("**sampleCommand**", command);

}

Command의 이름은  
Controller와 JSP사이에 일  
치시켜야 한다.  
Controller에서 문자열상수  
로 정의해도 된다.

# 컨트롤러 클래스 작성(POST)

// ③post로 보내는 경우

@RequestMapping(method=RequestMethod.POST)

public ModelAndView doAction(@ModelAttribute("sampleCommand") SampleCommand command, BindingResult bindingResult) {

System.out.println(command);

// command 각 항목 에러 체크

if(StringUtils.isEmpty(command.getName())) {

bindingResult.rejectValue("name", "error.required");

}

// 공통 에러 체크

if(bindingResult.hasErrors()) {

bindingResult.reject("error.message");

}

//에러가 있는 경우 (원래 화면으로 돌아감)

if(bindingResult.hasErrors()) {

ModelAndView mav = new ModelAndView();

mav.getModel().putAll(bindingResult.getModel());

return mav;

}

입력값을 체크

뭔가 에러가 발생할 경우  
에러 내용을 Model을 제공해 준  
자신의 화면으로 돌아감

# 독자 데이터타입 바인드(@InitBinder)

- InitBinder : 형변환초기화 메서드
  - 자동으로 JavaBean에 Bind되지만, 형변환이 필요할 경우 사용
- 날짜형 바인드(CustomDateEditor)
- 수치형 바인드(CustomNumberEditor)

# 날짜형 바인드(CustomDateEditor) 실습

- 메서드에 @InitBinder를 붙인다.
- 인수로 WebDataBinder를 가진다.
- registerCustomEditor 메서드에 미리 만들어진 CustomDateEditor를 이용하여 체크한다

# 수치형 바인드(CustomNumberEditor) 실 습

- 메서드에 @InitBinder를 붙인다.
- 인수로 WebDataBinder를 가진다.
- registerCustomEditor 메서드에 미리 만들어진 CustomNumberEditor를 이용하여 체크한다

spring 파일 업로드

# 파일 업로드

1. CommonsFileUpload 준비
2. 단순한 파일 업로드
3. 멀티 파일 업로드
4. 파일 사이즈 초과시 처리방법



# CommonsFileUpload 준비

라이브러리 준비

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <dependencies>
    . . . 생략

    <dependency>
      <groupId>commons-fileupload</groupId>
      <artifactId>commons-fileupload</artifactId>
      <version>1.2.2</version>
    </dependency>

    . . . 생략
  </dependencies>
</project>
```

# CommonsFileUpload 준비

root-context.xml 추가

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

```
<beans
```

```
xmlns="http://www.springframework.org/schema/beans"> . . . 생략
```

```
    <!-- use file upload -->
```

```
    <bean id="multipartResolver"  
class="org.springframework.web.multipart.commons.CommonsMultip  
artResolver">
```

```
        <!-- 업로드 가능한 최대 파일 사이즈 (단위 바이트) 20mb -->
```

```
        <property name="maxUploadSize" value="20971520"/>
```

```
    </bean>
```

```
</beans>
```

# 단순한 파일 업로드

- JSP

form속성에 enctype="multipart/form-data" 추가

```
<h4>파일 업로드</h4>
```

```
<form action="${appUrl}/test/fileupload/single.html"  
method="post" enctype="multipart/form-data">
```

```
<input type="file" name="file" />
```

```
<input type="submit"/>
```

```
</form>
```

# 단순한 파일 업로드

- 컨트롤러

```
@Controller
```

```
@RequestMapping("/fileupload")
```

```
public class FileuploadController {
```

```
    @RequestMapping(method=RequestMethod.GET)
```

```
    public void setupForm(Model model) {
```

```
    }
```

```
// 하나의 파일을 업로드함
```

```
    @RequestMapping(value="single", method=RequestMethod.POST)
```

```
    public ModelAndView doAction(@RequestParam("file") MultipartFile file)  
    throws IllegalStateException, IOException {
```

# 단순한 파일 업로드

- 컨트롤러 계속

```
if(!file.getOriginalFilename().isEmpty() && !file.isEmpty()) {  
    File uploadFile = new File("d:/upload/", file.getOriginalFilename());  
    //서버내의 다른 장소에 업로드함  
    file.transferTo(uploadFile);  
    ModelAndView mav = new ModelAndView("complete");  
    mav.addObject("filename", file.getOriginalFilename());  
    mav.addObject("filesize",  
FileUtils.byteCountToDisplaySize(file.getSize()));  
    return mav;  
  
    } else {  
        ModelAndView mav = new ModelAndView("fail");  
        return mav;  
    }  
}  
}
```

# 멀티 파일 업로드

JavaBeans에 배열이나 List형식으로 가지고 있음.

```
public class FileUpload {  
    private CommonsMultipartFile[] files;  
  
    public CommonsMultipartFile[] getFiles() {  
        return files;  
    }  
  
    public void setFiles(CommonsMultipartFile[] files) {  
        this.files = files;  
    }  
  
}
```

# 멀티 파일 업로드

- JSP 작성

```
<h4>멀티 파일 업로드</h4>
```

```
<spring:url value= "/upload" var="uploadURL" />
```

```
<form:form modelAttribute= "formUpload" method="post"  
action="${uploadURL}" enctype="multipart/form-data">
```

```
<form:input path= "files" type="file" multiple="multiple"/>
```

```
<form:errors path= "files" cssStyle="color:red;" />
```

```
<button type= "submit">Upload</button>
```

```
</form:form>
```

# 멀티 파일 업로드

- Controller 작성

@Controller

@RequestMapping(value="/")

**public class UploadController {**

@RequestMapping(value="/uploadPage",  
method=RequestMethod.**GET**)

**public ModelAndView uploadPage() {**

ModelAndView model = **new ModelAndView("upload\_page");**  
model.addObject("formUpload", **new FileUpload();**)

**return model;**

**}**



# 멀티 파일 업로드

- Controller 작성(계속)

// 멀티 파일 업로드

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

```
public ModelAndView upload(@ModelAttribute("formUpload")  
FileUpload fileUpload, BindingResult result) throws  
IOException {
```

```
if (result.hasErrors()) {
```

```
return new ModelAndView("upload_page");
```

```
}
```

```
return new ModelAndView("success", "fileNames",  
processUpload(fileUpload));
```

```
}
```

# 멀티 파일 업로드

- Controller 작성(계속)

```
private List<String> processUpload(FileUpload files) throws  
IOException {
```

```
List<String> fileNamees = new ArrayList<String>();
```

```
CommonsMultipartFile[] commonsMultipartFiles = files.GetFiles();
```

```
for (CommonsMultipartFile commonsMultipartFile :  
commonsMultipartFiles) {
```

```
FileCopyUtils.copy(commonsMultipartFile.getBytes(), new  
File("D:upload" +  
commonsMultipartFile.getOriginalFilename()));
```

```
fileNamees.add(commonsMultipartFile.getOriginalFilename());  
}
```

```
return fileNamees;  
}
```

# 파일 사이즈 초과시 처리방법

- MaxUploadSizeExceededException 발생
- 공통 예외처리 클래스 HandlerExceptionResolver로 처리한다.

rest 서비스 작성

# REST 서비스 작성

- 서버측: @RequestMapping 메서드로 응답.
  - 데이터 형식: XML이나 JSON
- 클라이언트측: jQuery로 구현

# REST 서비스 작성

1. JSON/XML 데이터 송수신
2. RESTful 시스템 설계

# JSON/XML 데이터 송수신

1. 서버에서 JSON데이터 출력
2. 클라이언트에서 JSON데이터 송신
3. 서버에서 XML데이터 출력
4. 클라이언트에서 XML데이터 송신

# JSON/XML 데이터 송수신

- 서버에서 JSON데이터 출력
  - 어노테이션 @ResponseBody 추가
    - http header accept가 application/json 이 됨.
- 명시적으로 Response 타입 설정가능
  - **Accept=application/json**
  - **produces="application/json"**
    - Spring 3.1 전용



# JSON/XML 데이터 송수신

- 서버에서 JSON데이터 출력
  - 컨트롤러 소스

@Controller

```
public class JsonController {
```

```
    @RequestMapping(value="/ajax/jsonOut1", method={RequestMethod.GET, RequestMethod.POST})
```

```
        //@RequestMapping(value="/ajax/jsonOut1", headers="Accept=application/json")
```

```
        //@RequestMapping(value="/ajax/jsonOut1", produces="application/json")
```

```
//미디어타입을 명시적으로 지정
```

```
//“produces”는 Spring 3.1부터 이용 가능
```

```
    @ResponseBody    public List<UserInfoViewDto> jsonOut (@RequestParam String cd) {
```

```
        if(StringUtils.isEmpty(departmentCd)) {
```

```
            return new ArrayList<UserInfoViewDto>();
```

```
        }
```

```
        List<UserInfoDto> list = /* Servlet/DAO로부터 취득함 */;
```

```
        //리턴값 Java객체는 자동적으로 JSON으로 변경 가능함.
```

```
        return list;
```

```
    }
```

```
}
```

# JSON/XML 데이터 송수신

- 클라이언트: jquery 이용

```
<script type="text/javascript">
```

```
$(document).ready(function(){
```

```
    $('#jsonOut1').click(function(){
```

```
        $.ajax({
```

```
            type: "POST",
```

```
            url : "${appUrl}/ajax/jsonOut1.html",
```

```
            data : {"cd": "sawon"},
```

```
            // 수신시(응답)의 미디어타입
```

```
            dataType: "json",
```

```
            success:function(data){
```

```
                //TODO:
```

```
                alert(data);
```

```
            }
```

```
        });
```

```
    });
```

```
});
```

```
</script>
```

```
<ol>
```

```
    <li> <a href="${appUrl}/ajax/jsonOut1.html?cd=aaa">JSON형식으로 취득(GET사용)</a> </li>
```

```
    <li> <span id="jsonOut1">JSON형식으로 취득(jQuery사용)</span> </li>
```

```
</ol>
```

# JSON/XML 데이터 송수신

- 클라이언트에서 데이터 송신시 처리(서버 수신)

```
@Controller public class JsonController {  
  
    @RequestMapping(value="/ajax/jsonIn1")  
    //@RequestMapping(value="/ajax/jsonIn1", headers="Content-  
Type=application/json")  
    //@RequestMapping(value="/ajax/jsonIn1",  
consumes="application/json")  
    public ModelAndView jsonIn1(@RequestBody  
    JsonSampleCommand1 command) {  
  
        ModelAndView mav = new ModelAndView("/ajax/sample1");  
        mav.addObject("data", command);  
        return mav;  
    }  
}
```

# JSON/XML 데이터 송수신

- 클라이언트측 처리(jQuery)

```
<script type="text/javascript"> $(document).ready(function(){
    $('#jsonIn1').click(function(){
        $.ajax({
            type: "POST",
            url : "${appUrl}/ajax/jsonIn1.html",
            // 송신시 미디어타입
            contentType: "application/json;charset=UTF-8",
            data : '{"name": " 고양이", "age": "2"}',
            // 수신시 미디어타입
            dataType: "html",
            success:function(data){
                alert(data);
            },
            //입력 에러나 기타 통신에러가 발생시 에러처리
            error:function(XMLHttpRequest, textStatus, errorThrown){
                // 에러 타입 (timeout, error, notmodified, parsererror)
                alert("textStatus = " + textStatus);

                // 에러 메시지
                alert("errorThrown=" + errorThrown);
            }
        });
    });
});
```



[{"name": " 관리자", "id": "1"}, {"name": " 일반사용자", "id": "2"}]

# 서버에서 XML 출력 처리(JAXB)

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

// • JAXB의 프리퍼티 속성 정의
@XmlAccessorType(XmlAccessType.FIELD) @XmlType(name = "", propOrder = {
    "id",
    "value"
})
@XmlRootElement(name = "SampleJaxb1")
public class SampleJaxb1 {

    protected String id;
    protected Integer value;

    public String getId() {
        return id;
    }

    public void setValue(Integer value) {
        this.value = value;
    }
}
```

# 서버에서 XML 출력 처리

컨트롤러 소스

@Controller

```
public class XmlJaxbController {
```

```
    @RequestMapping(value="/ajax/xmlOut1")
```

```
//    @RequestMapping(value="/ajax/xmlOut1", headers="Accept=application/xml")
```

```
//    @RequestMapping(value="/ajax/xmlOut1", produces="application/xml")
```

```
    @ResponseBody
```

```
    public SampleJaxb1 jsonOut1(@RequestParam String cd) {
```

```
        System.out.printf("cd=%s\n", cd);
```

```
        SampleJaxb1 outData = new SampleJaxb1();
```

```
        outData.setId( " 고양이");
```

```
//XML은 JAXB용 객체를 리턴해야만 한다.
```

```
        outData.setValue(2);
```

```
        return outData;
```

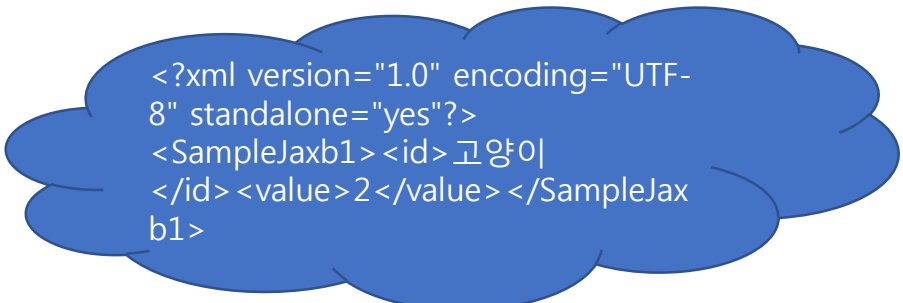
```
    }
```

```
}
```

# 서버에서 XML 출력 처리

jQuery 소스

```
<script type="text/javascript">
$(document).ready(function(){
    $('#xmlOut1').click(function(){
        $.ajax({
            type: "POST",
            url : "${appUrl}/ajax/xmlOut1.html",
            data : {"cd": "syasin"},
            // 수신시 미디어타입
            dataType: "xml",
            success:function(data){
                alert(data);
            }
        });
    });
});
```



<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<SampleJaxb1> <id> 고양이  
</id> <value>2</value> </SampleJaxb1>

```
});
});
</script>
<ol>
    <li><a href="${appUrl}/ajax/xmlOut1.html?cd=aaa">XML형식취득(GET방식)</a> </li>
    <li><span id="xmlOut1">XML형식취득(jQuery로 취득)</span> </li>
</ol>
```

# 클라이언트에서 XML송신/서버수신

- 클라이언트(jQuery)

```
<script type="text/javascript">
```

```
//송신시 미디어 타입을 "contentType"에 명시적으로 지정 가능
```

```
//XML데이터를 송신할 경우, 텍스트 형식으로 XML를 작성 필요
```

```
$(document).ready(function(){
```

```
    $('#xmlIn1').click(function(){
```

```
        $.ajax({
```

```
            type: "POST",
```

```
            url : "${appUrl}/ajax/xmlIn1.html",
```

```
            // 송신시 미디어 타입
```

```
contentType: "application/xml;charset=UTF-8",
```

```
data : '<SampleJaxb1> <id> 고양이 </id> <value>2 </value> </SampleJaxb1>',
```

```
            dataType: "html",
```

```
            success:function(data){
```

```
                alert(data);
```

```
            },
```

```
            error:function(XMLHttpRequest, textStatus, errorThrown){
```

```
                alert("textStatus=" + textStatus);
```

```
                alert("errorThrown=" + errorThrown);
```

```
            }
```

```
        });
```

```
    });
```



# 클라이언트에서 XML송신/서버수신

@Controller

```
public class XmlJaxbController {
```

```
    @RequestMapping(value="/ajax/xmlIn1")
```

```
// @RequestMapping(value="/ajax/xmlIn1",  
headers="ContentType=application/xml")
```

```
// @RequestMapping(value="/ajax/xmlIn1",  
consumes="application/xml")
```

```
public ModelAndView xmlIn1(@RequestBody SampleJaxb1 command) {  
    System.out.printf("cd=%s\n", command);
```

```
    ModelAndView mav = new ModelAndView("/ajax/sample1");  
    mav.addObject("xmlData", command);  
    return mav;
```

```
}
```

```
}
```

# REST 서비스에서의 에러 처리

No.	HTTP 상태 코드	처리방법
1	1xx(Informational)	사용하지 않음
2	2xx(Successful)	Controller에서 처리
3	3xx(Redirection)	리다이렉트 처리(ex. 로그인시)
4	4xx(Client Error)	Controller에서 주로 처리(클라이언트에서 문제발생)
5	5xx(Server Error)	서버에서 처리(ex. 톰캣 서버)

# EL 식

- JSP 2.0부터 도입
  - 실사용은 tomcat 6.0부터
- 식언어(Expression Language)
- JSP안에서 연산, 값비교등을 간단히 처리할 경우 사용
- `${식}`
  - { } 안에서 계산해서, 결과를 출력한다.

# EL 식

- `<%-- 변수 호출 --%>`
- `${var}`
- 
- `<%-- Bean의 name프러퍼티 호출 --%>`
- `${loginUser.name}`
- 
- `<%-- JSTL에 의한 값 출력 --%>`
- `<c:out value="${loginUser.name}" />`
- 
- `<%-- EL Function에 의한 값 출력 --%>`
- `${fn:escapeXml(loginUser.name)}`
- `${f:h(loginUser.name)}`

# 스cope(Scope)에서의 EL식

우선도	스cope 종류	EL식 기술법
1	Page 스cope	pageScope
2	Request 스cope(Model과 같음), Flash 스cope(리다이렉트하는 곳에 Model전달)	requestScope
3	Session 스cope	sessionScope
4	Application 스cope	applicationScope

```
<%-- session스cope 참조 --%>  
${sessionScope.loginUser.name}
```

```
<%-- application스cope 참조 --%>  
${applicationScope.loginUser.name}
```

# EL 계산식

- `<%-- 산술연산자 --%>`
- `${5*2}`
- `<%-- 비교연산자 --%>`
- `${3 > 5}`
- `<%-- 논리연산자 --%>`
- `${!(3==5)}`
- `<%-- 삼항연산자 --%>`
- `${3 > 2 ? 100:200}`
- `<%-- 비었는지, null인지?--%>`
- `${empty data}`

# EL 리스트(배열), 맵 참조

- 리스트(배열) 참조
  - sample[0]
  - sample[1].name
- 맵 참조
  - sample[red]
  - sample[red].name
  - sample[red][0].age

# JSTL 함수

- JSP 설정
- `<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>`
- `<%-- escape해서 출력 --%>`
- `${fn:escapeXml(message1)}`



# JSTL 함수

No.	EL 함수	설명
1	fn:contains	문자열 중에, 지정한 문자열이 존재하느냐
2	fn:containsIgnoreCase	문자열 중에, 지정한 문자열이 존재하느냐(대소문자 구별 없음)
3	fn:endsWith	문자열 중에, 지정한 문자열로 끝나느냐
4	fn:escapeXml	XML로 해석되는 (<, >, &, ', ""')을 (&lt;、&gt;、&amp;、&#039;、&#034;)로 바꿈
5	fn:indexOf	문자열 중에, 지정한 문자열이 존재하면, 그 인덱스 리턴(없다면, -1)
6	fn:join	배열을 문자열로 리턴(separator로 구분)
7	fn:length	문자열의 문자수를 카운트
8	fn:replace	문자열에서 바꿀문자열을 지정하여 바꿈
9	fn:split	문자열에서 구별자로 구분해서 배열 리턴
10	fn:startsWith	문자열 중에, 지정한 문자열로 시작하느냐

# JSTL 함수

No.	EL 함수	설명
11	fn:substring	index를 이용해서 문자열 추출(start, end)
12	fn:substringAfter	index를 이용해서 index이후의 문자열 추출
13	fn:substringBefore	index를 이용해서 index이전의 문자열 추출
14	fn:toLowerCase	소문자로 변환
15	fn:toUpperCase	대문자로 변환
16	fn:trim	앞뒤 공백문자 제거

# Spring Expression Language(SpEL)

- SpEL상수
- SpEL연산자
- 속성에 접근
- 메서드 호출

# SpEL 상수

No	종류	사용예	설명
1	문자열	'Hello World'	작은 따옴표로 감싼다
2	수치(정수)	1234, -1234	자바와 같다
3	수치(실수)	12.3233	자바와 같다
4	16진수	0x7FFFF	자바와 같다
5	불린값	true, false	자바와 같다
6	null값	null	자바와 같다
7	리스트	{1,2,3,4}, {'a','b'},{'c','d'}}	중괄호로 감싸고, 구분은 쉼표(,)로 한다
8	배열	new int[4] new int[]{1,2,3} new int[4][5]	자바와 같다
9	변수	#name	앞에 #을 붙인다

# SpEL 연산자

종류	사용예	설명
산술 연산자	+, -, *, /(div), %(mod), ^	괄호안은 xml에서 사용시
비교 연산자	==(eq), !=(ne), <(lt), >(gt), <=(le), >=(ge)	상동
논리 연산자	and, or, !(not)	상동
삼항 연산자	a?b:c	
기타	=(대입), instanceof(인스턴스 비교), matches(정규표현식)	예) '5.00' matches '^-?\\Wd+(\\W\\W\\.\\Wd{2})?\\\$'

# SpEL속성 접근

No	종류	사용예	설명
1	중첩된 객체접근	command.name command.member.name	점(.)으로 접근하면, 해당 getter 메서드가 호출됨
2	중첩된 객체접근(Null 처리)	command?.name command?.member?.name	부모객체가 Null일 경우, NullPointerException 안나고 null 리턴
3	리스트, 배열	member[0].name	
4	맵	officers['president']	president는 키

# SpEL메서드 호출

No	종류	사용예	설명
1	인스턴스 메서드	'abc'.substring(2, 3) isMember('oraclejava')	자바와 같다.
2	static 메서드	T(java.lang.Math).abs(-100) T(Math).abs(-100) T(org.apache.commons.lang.StringUtils).isEmpty(#name)	클래스 명앞에 "T"를 붙임. 패키지가 "java.lang"인 경우 생략가능
3	static 변수	T(java.lang.Math).PI T(java.math.RoundingMode).CEILING	
4	캐스트	T(String) 10 (T(java.util.Date) #obj).time	
5	템플릿	random number is #{T(java.lang.Math).random() }	출력예) random number is 0.1200323242324

# SpEL사용예

- Spring XML 설정파일

```
<bean id="numberGuess"  
class="org.springframework.samples.NumberGuess">  
    <property name="randomNumber"  
value="#{ T(java.lang.Math).random() * 100.0 }"/>  
  
    <!-- other properties -->  
</bean>
```



# SpEL사용예

- Spring XML 설정파일(시스템 프로퍼티 호출)

```
<bean id="taxCalculator"  
class="org.springframework.samples.TaxCalculator">  
    <property name="defaultLocale"  
value="#{ systemProperties['user.region'] }"/>  
  
    <!-- other properties -->  
</bean>
```

# SpEL사용예

- 자바파일(어노테이션)

```
public static class FieldValueTestBean
```

```
    @Value("#{ systemProperties['user.region'] }")
```

```
    private String defaultLocale;
```

```
    public void setDefaultLocale(String defaultLocale) {  
        this.defaultLocale = defaultLocale;  
    }
```

```
    public String getDefaultLocale() {  
        return this.defaultLocale;  
    }  
}
```

# 스코프의 종류

No.	스코프	설명
1	페이지(page)	페이지 내에서만 유효한 스코프
2	리퀘스트(request)	요청이 발생해서 종료될 때까지 유효. 다음 페이지까지 유효. 단 redirect되면 무효화됨. Controller에서 Model객체와 같음.
3	플래시(flash)	redirect되어도 유효함. Spring 3.1에서 표준기능이 됨.
4	세션(session)	페이지간에 유효함. (ex. 로그인시 사용자정보 담고 있음)
5	애플리케이션(application)	애플리케이션 전체에서 유효함. 세션이 끝나도 남아있음. (공통변수 저장시 사용)

# 정적 리소스 이동

- web.xml에서 "/"로 시작하는 URL은 모두 SpringMVC의 DispatcherServlet으로 처리된다.
- 이미지나 자바스크립트 파일등을 처리하기 위해 스프링 설정파일에 <mvc:resources>를 등록한다.
  - <mvc:resources location="/WEB-INF/resources/images/" mapping="/images/\*\*"/>
  - <mvc:resources location="/WEB-INF/resources/js/" mapping="/js/\*\*"/>
  - <mvc:resources location="/WEB-INF/resources/css/" mapping="/css/\*\*"/>

# MyBatis

# MyBatis 3.3.0

- ORM
- iBATIS(Apache에서 발표)
- Google Code로 전환
- XML을 이용하여, SQL과 OR 매핑 수행
- Mapper를 이용하여, iBATIS보다 Type Safe

# MyBatis 구성요소

구성요소	역할
MyBatis 설정파일	데이터베이스 연결 정보, 매핑파일 패스 등 고정된 환경 정보 설정. XML파일로 구성
SqlSessionFactoryBuilder	MyBatis 설정파일로 SqlSessionFactory를 생성한다. 애플리케이션 시작할 때 사용하고 일단 SqlSessionFactory가 만들어지면 필요없다. 애플리케이션 만들고 나서는 버리는 개념.
SqlSessionFactory	SqlSession을 생성한다. SqlSessionFactory는 스레드세이프로서, Singleton패턴으로서 공유해서 사용해야 한다. Spring과 연결해서 사용할 때는 DI 컨테이너에서 관리된다.
SqlSession	SQL 실행이나, 트랜잭션 처리를 함. SqlSession은 스레드세이프가 아니므로, 스레드마다 필요할 경우 만들고 없애야 한다.
Mapper 인터페이스	매핑파일에 기재되어있는 SQL을 불러오기 위한 인터페이스. 애플리케이션에서 직접 SQL, OR매핑을 기술하는 경우도 가능하다. 객체는 MyBatis가 자동으로 생성해 준다.
매핑파일	SQL과 OR매핑을 설정한다. XML파일로 구성

# MyBatis 샘플(테이블정의, 기본데이터)

```
create table item
(
    item_code char(3),
    item_name varchar2(20),
    item_price number(5),
    primary key(item_code)
);
```

ITEM_C	ITEM_NAME	ITEM_PRICE
91	라면	3000
94	된장라면	4000
95	돈코츠라면	5000
102	규동	6000
200	카레	5500
201	드라이카레	6500
500	콘스프	3500
502	야채스프	3000
505	콘소메스프	3200



# MyBatis 설정파일 샘플

```
<?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>
  <environments default="dev">
    <environment id="dev">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver"
value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url"
value="jdbc:oracle:thin:@localhost:1521:XE"/>
        <property name="username" value="scott"/>
        <property name="password" value="tiger"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="item_mapper.xml"/>
  </mappers>
</configuration>
```

# MyBatis 맵핑파일 샘플

```
<?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="item">
    <select id="selectList" resultType="map">
      select * from item
    </select>
  </mapper>
```

# MyBatis Dao 샘플

```
public class ItemDao {
    public List<ItemVO> getItemList() {
        List<ItemVO> itemList = new ArrayList<ItemVO>();

        try {
            InputStream in = ItemDao.class.getResourceAsStream("/mybatis-
            config.xml");
            SqlSessionFactory factory = new
            SqlSessionFactoryBuilder().build(in);

            SqlSession session = factory.openSession();
            List<Map<String, Object>> result =
            session.selectList("item.selectList");
            for(int i=0; i<result.size(); i++) {
                Map<String, Object> map = result.get(i);
                ItemVO item = new ItemVO();
                item.setItemCode(Integer.parseInt(map.get("ITEM_CODE").to
                String().trim()));
                item.setItemName(map.get("ITEM_NAME").toString());
                item.setItemPrice(Integer.parseInt(map.get("ITEM_PRICE").
                toString()));
                itemList.add(item);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

# MyBatis Dao 샘플

```
}  
} catch (Exception e) {  
e.printStackTrace();  
}  
  
return itemList;  
}  
}
```

# MyBatis Dao Test 샘플

```
package com.oraclejava.dao;

import java.util.List;

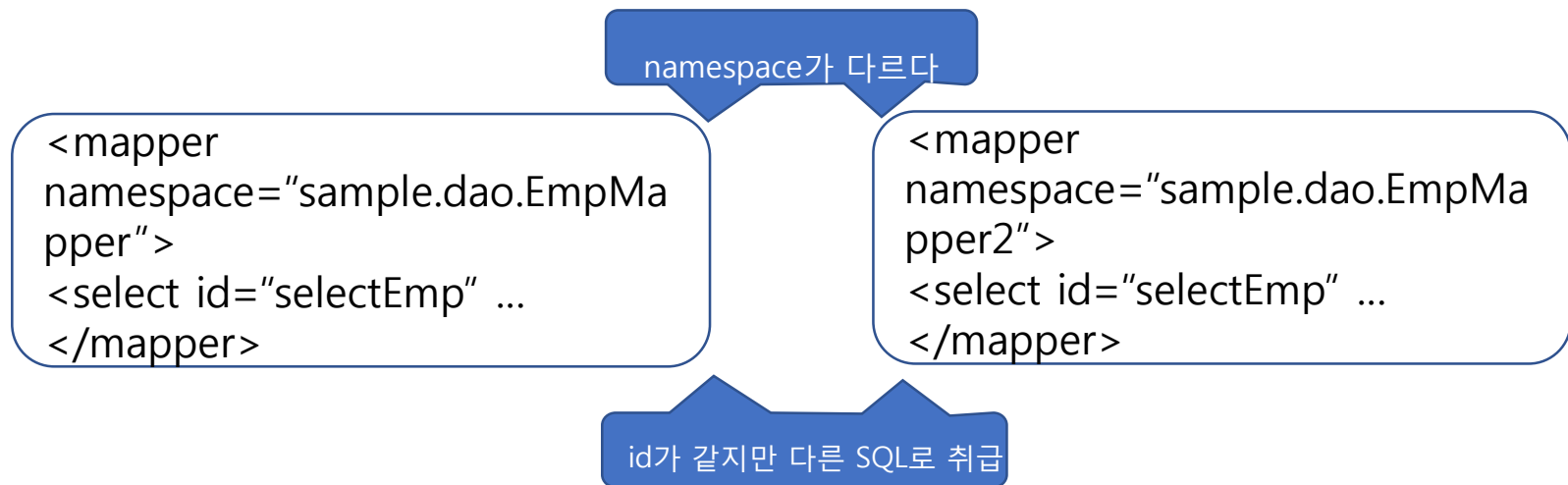
import com.oraclejava.vo.ItemVO;

public class ItemDaoTest {

    public static void main(String[] args) {
        ItemDao dao = new ItemDao();
        List<ItemVO> itemList = dao.getItemList();
        for(int i=0; i<itemList.size(); i++) {
            ItemVO item = itemList.get(i);
            System.out.println(item.getItemCode() + " " +
                               item.getItemName());
        }
    }
}
```

# 매핑파일

- namespace
  - SQL 그룹핑을 수행
  - namespace가 다르다면 SQL ID가 같아도 다른 SQL로 취급된다.



# SQL 파라미터 바인딩

- 파라미터가 하나일 경우
  - `{empNo}`
    - "empNo"의 이름은 아무거나 써도 된다.
- JavaBeans를 사용한 파라미터 바인딩
  - `parameterType`을 JavaBeans의 이름
  - 각각의 이름은 JavaBeans의 속성명 지정
  - `{empNo}`, `{ename}` ...

# SQL 파라미터 바인딩

- Map을 사용한 파라미터 바인딩
  - `parameterType="hashmap"`
  - map의 키를 사용함.
  - 호출시
  - `Map<String, Object> map = new HashMap<String, Object>();`
  - `map.put("empNo", 9999);`
  - ...



# SELECT 결과 매핑

- `resultType="Emp"`
  - JavaBeans 형식으로 변환
- 레코드 1건
  - `Emp emp = (Emp)session.selectOne("sample.dao.EmpMapper.selectEmp", id);`
- 레코드 여러건
  - `List<Emp> list = session.selectList("sample.dao.EmpMapper.selectEmp", id);`

# 동적 SQL 샘플

```
<select id="findEmp" parameterType="Emp"
resultMap="empResultMap">
    SELECT * FROM EMP
    <where>
        <if test="ename != null ">
            ENAME = #{ename}
        </if>
        <if test="mgr != null ">
            AND MGR = #{mgr}
        </if>
    </where>
</select>
```

# 동적 SQL 태그

태그	설명
if	조건에 따라 문자열 추가. test가 필수조건
choose	Java Switch문과 유사. 복수의 조건에서 하나의 문자열 채택.
when	조건에 일치하면 처리. test가 필수조건
otherwise	어떠한 조건도 일치하지 않는 경우 처리
trim	SQL 부분치환
where	where구를 동적으로 작성
set	update의 set구를 동적으로 작성
foreach	반복문으로 문자열 추가

# INSERT시 키 생성

- DB가 auto increment을 사용하는 경우

```
<insert id="insertPet" parameterType="sample.biz.domain.Pet"
useGeneratedKeys="true" keyProperty="petId">
INSERT INTO PET (PET_NAME, OWNER_NAME, PRICE,
BIRTH_DATE)
        VALUES ({petName}, {ownerName}, {price},
#{birthDate})
</insert>
```

# INSERT시 키 생성

- DB가 시퀀스를 사용하는 경우

```
<insert id="insertPet"
parameterType="sample.biz.domain.Pet">
```

```
    <selectKey keyProperty="petId" resultType="int"
order="BEFORE">
```

```
        SELECT PET_SEQ.NEXTVAL FROM DUAL
```

```
    </selectKey>
```

```
    INSERT INTO PET (PET_ID, PET_NAME, OWNER_NAME,
PRICE, BIRTH_DATE)
```

```
        VALUES (#{petId}, #{petName}, #{ownerName}, #{price},
#{birthDate})
```

```
    </insert>
```

# Mapper 인터페이스

- Mapper 인터페이스를 사용하지 않는 경우
  - 직접 namespace + "." + SQL ID 사용가능
  - 단점:SQL파일이 많고, 복잡할 경우 처리어려움. 이유->이클립스에서 코드어시스트가 지원 안됨.
  - ```
public interface EmpMapper {  
    Emp selectEmp(int id);  
}
```
- 호출시:
  - ```
EmpMapper mapper = session.getMapper(EmpMapper.class);
```
  - ```
Emp emp = mapper.selectEmp(7369);
```

# Mapper Interface Annotation

- Mapper 인터페이스의 특징이 직접 SQL을 지정가능하다는 점.
- `@Select("SELECT * FROM EMP WHERE EMPNO = #{empNo}")`
- `Emp selectEmp(int id);`
  - 한계 있다.

# 트랜잭션 처리

- SqlSession에 준비되어 있는 4가지 메서드
  - void commit();
  - void commit(boolean force);
  - void rollback();
  - void rollback(boolean force);
- **boolean force의 의미:**
  - commit(), rollback()은 INSERT, UPDATE, DELETE문이 실행되었을 경우에만 실제로 처리된다.
  - 만일 INSERT, UPDATE, DELETE문의 실행여부와 상관없이 트랜잭션이 이루어져야 할 경우 commit(true), roleback(true) 이런 형식으로 호출한다.



# 트랜잭션 처리 샘플

```
SqlSession session = sessionFactory.openSession();
try {
    session.insert("sample.dao.EmpMapper.insertEmp", emp);
    session.commit();
} catch(Exception e) {
    session.rollback();
    throw new RuntimeException(e);
} finally {
    session.close();
}
```

# MyBatis와 Spring 연동

- SqlSessionFactory를 Bean으로서 정의

```
<bean id="sqlSessionFactory"  
class="org.mybatis.spring.SqlSessionFactoryBean">  
    <property name="dataSource" ref="dataSource" />  
    <property name="configLocation"  
value="classpath:sample/mybatis.Spring.xml" />  
    <property name="mapperLocations">  
        <list>  
            <value>sample/dao/EmpMapper.xml</value>  
        </list>  
    </property>  
</bean>
```

# SqlSession 인젝션

- SqlSessionTemplate 이용

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

```
public class MybatisEmpDao implements EmpDao {  
    @Autowired  
    private SqlSession session;  
}
```

실제로  
SqlSessionTemplate  
객체가 사용된다.

SqlSessionTemplate는  
Thread Safe이므로 이  
렇게 사용해도 됨.

# SqlSession 인젝션

- SqlSessionDaoSupport 이용하는 방법
  - `public class MyatisSpringEmpDao extends SqlSessionDaoSupport implements EmpDao {`
    - `public Emp findEmpById(int id) {`
      - `return getSqlSession().selectOne("sample.dao.EmpMapper.selectEmp", 7369);`
  - `}`

# Spring 트랜잭션과 연동

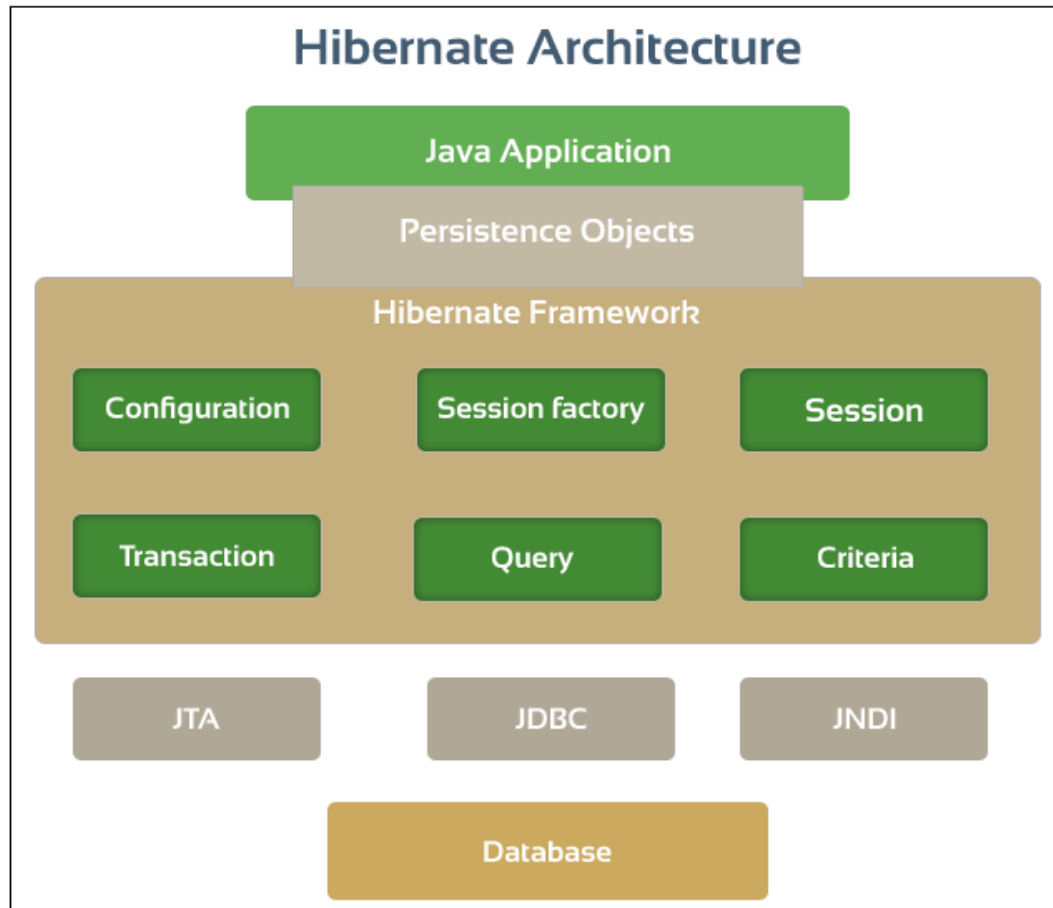
```
<bean id="transactionManager"  
class="org.springframework.jdbc.datasource.DataSourceTransactio  
nManager">  
    <property name="dataSource">  
        <ref local="dataSource" />  
    </property>  
</bean>
```

# Hibernate

# Hibernate의 구조

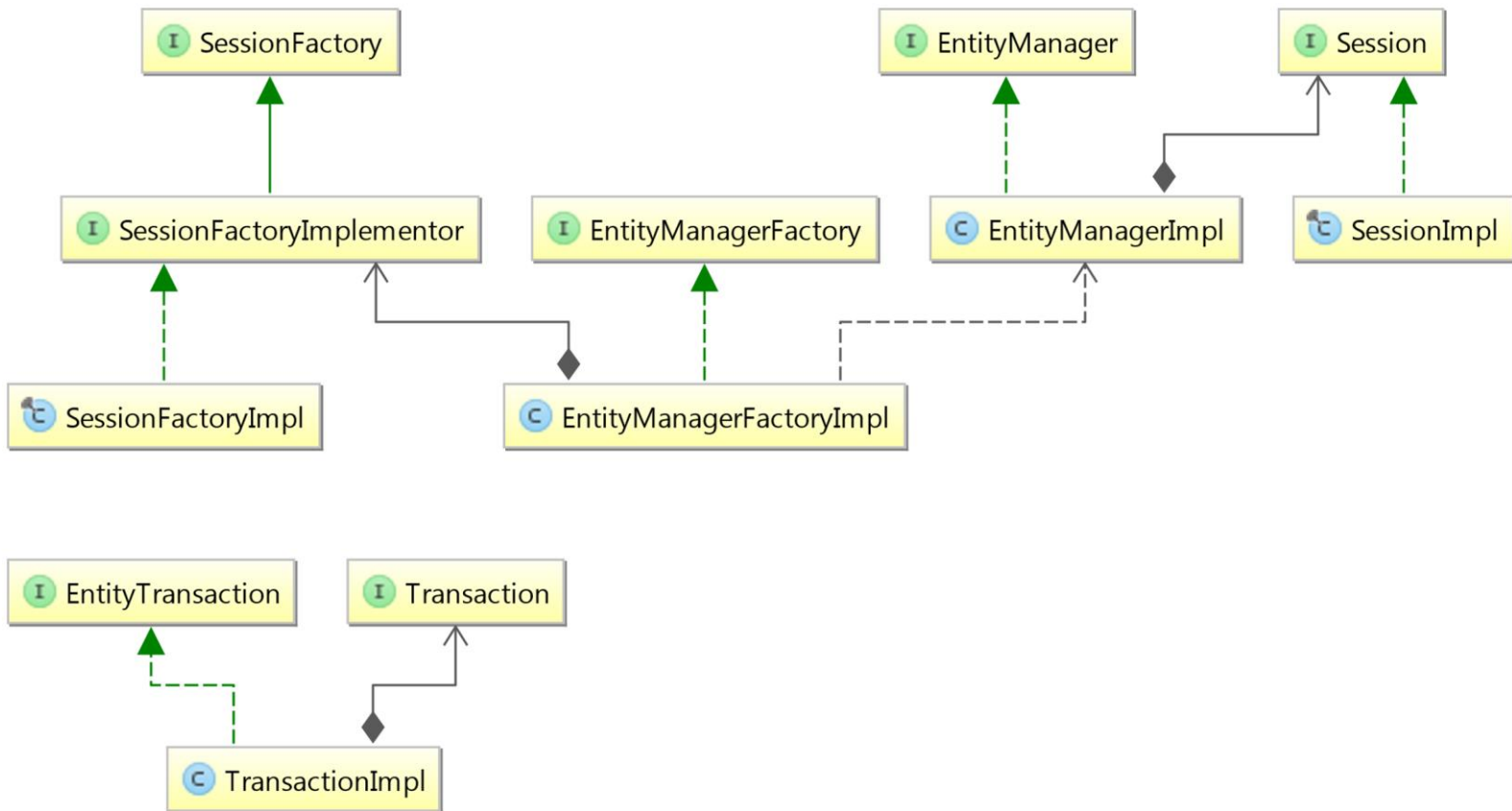


# Hibernate의 구조





# SessionFactory와 Session



# Hibernate 환경설정

```
<dependency>  
<groupId>org.hibernate</groupId>  
<artifactId>hibernate-core</artifactId>  
<version>4.1.0.Final</version>  
</dependency>
```

```
<dependency>  
<groupId>org.hibernate.javax.persistence</groupId>  
<artifactId>hibernate-jpa-2.0-api</artifactId>  
<version>1.0.0.Final</version>  
</dependency>
```

# applicationContext.xml

```
<!-- Session Factory Declaration -->
<bean id="sessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
<property name="dataSource" ref="dataSource" />
<property name="packagesToScan" value="com.oraclejava.entity" />
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
<prop key="hibernate.show_sql">true</prop>
<prop key="hibernate.enable_lazy_load_no_trans">true</prop>
<prop key="hibernate.default_schema">test</prop>
<prop key="format_sql">true</prop>
<prop key="use_sql_comments">true</prop>
<!-- <prop key="hibernate.hbm2ddl.auto">create</prop> -->
</props>
</property>
</bean>
```

# Entity 클래스 작성 예

```
@Entity
@Table(name="users")
public class Users {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="user_id")
    private Integer id;

    @Column(name="user_name")
    private String name;

    @Column(name="email")
    private String email;
```

# DaoImpl 작성 예

```
@Repository
@Transactional

public class UserDaoImpl implements UserDao{

    @Autowired
    private SessionFactory session;

    @Override
    public List<Users> list() {
        return session.getCurrentSession().createQuery("from Users").list();
    }

    @Override
    public boolean delete(Users users) {
        try {
            session.getCurrentSession().delete(users);
        } catch (Exception ex) {
            return false;
        }
        return true;
    }

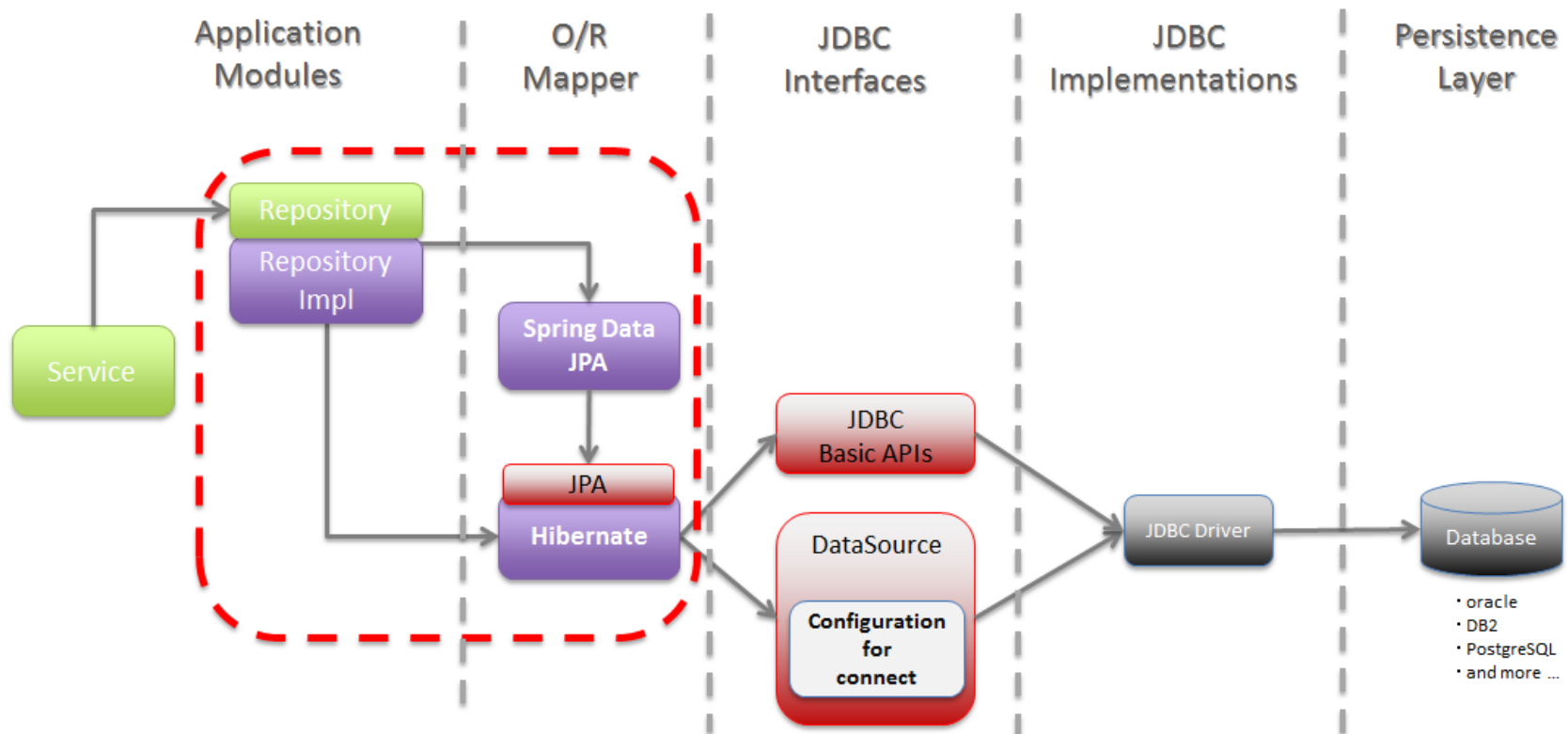
    @Override
    public boolean saveOrUpdate(Users users) {
        session.getCurrentSession().saveOrUpdate(users);
        return true;
    }
}
```

# Spring Data JPA

# JPA란?

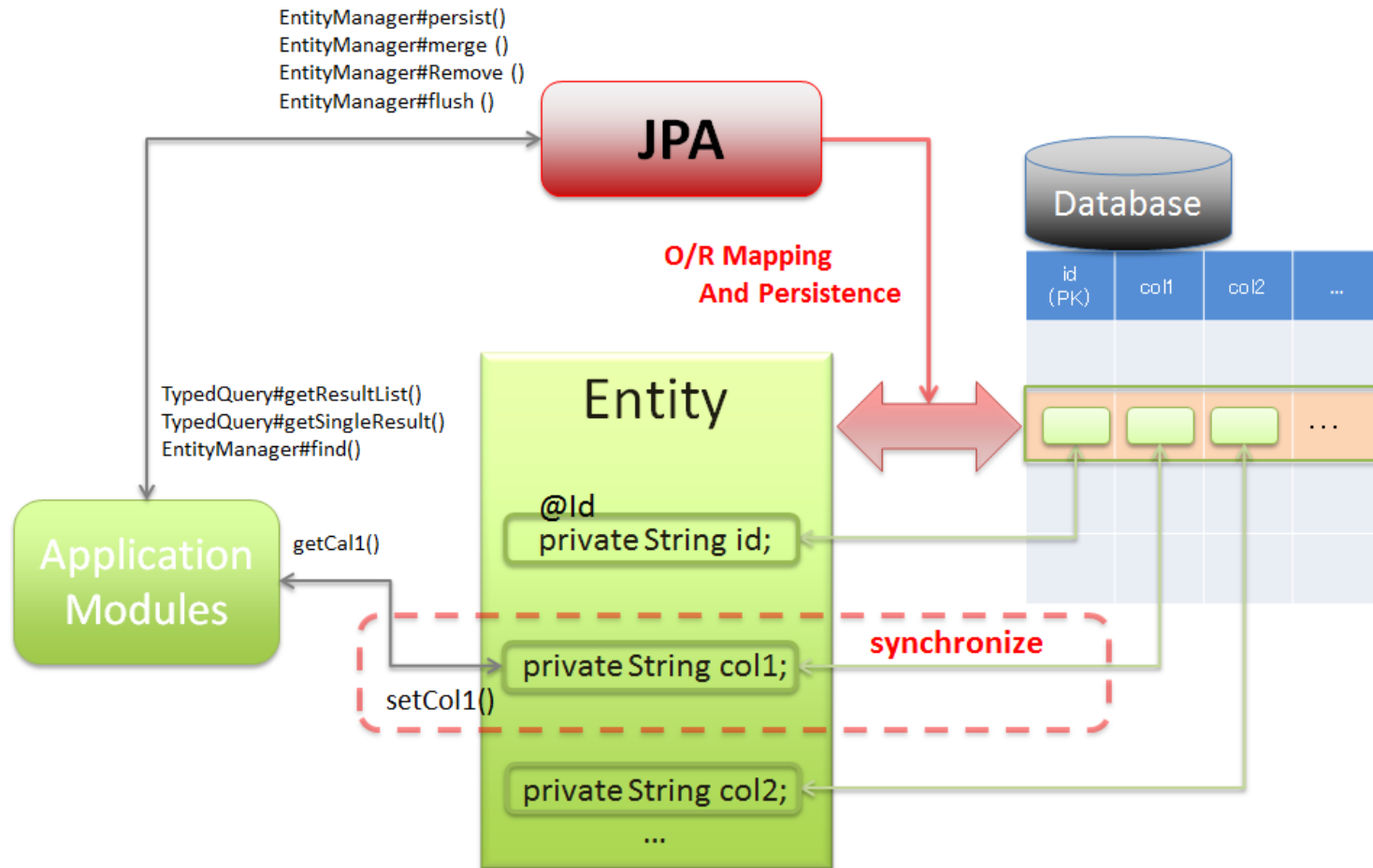
- Java Persistence API의 약자. RDBMS(관계형 데이터베이스)에서 관리되는 레코드를 Java 객체로 맵핑하거나, 맵핑된 객체에 대해 조작한 내용을 DB에 반영하는 방법을 Java API의 사양으로 정의해 놓은 것.
- 사양만 정의해 놓을 것을 뿐 실제 구현은 없음. JPA의 구현은 각 O/R Mapper 프로바이더가 담당. 대표적인 프로바이더로 Hibernate가 있음.

# Spring Data JPA의 전체 구조





# JPA의 O/R Mapping

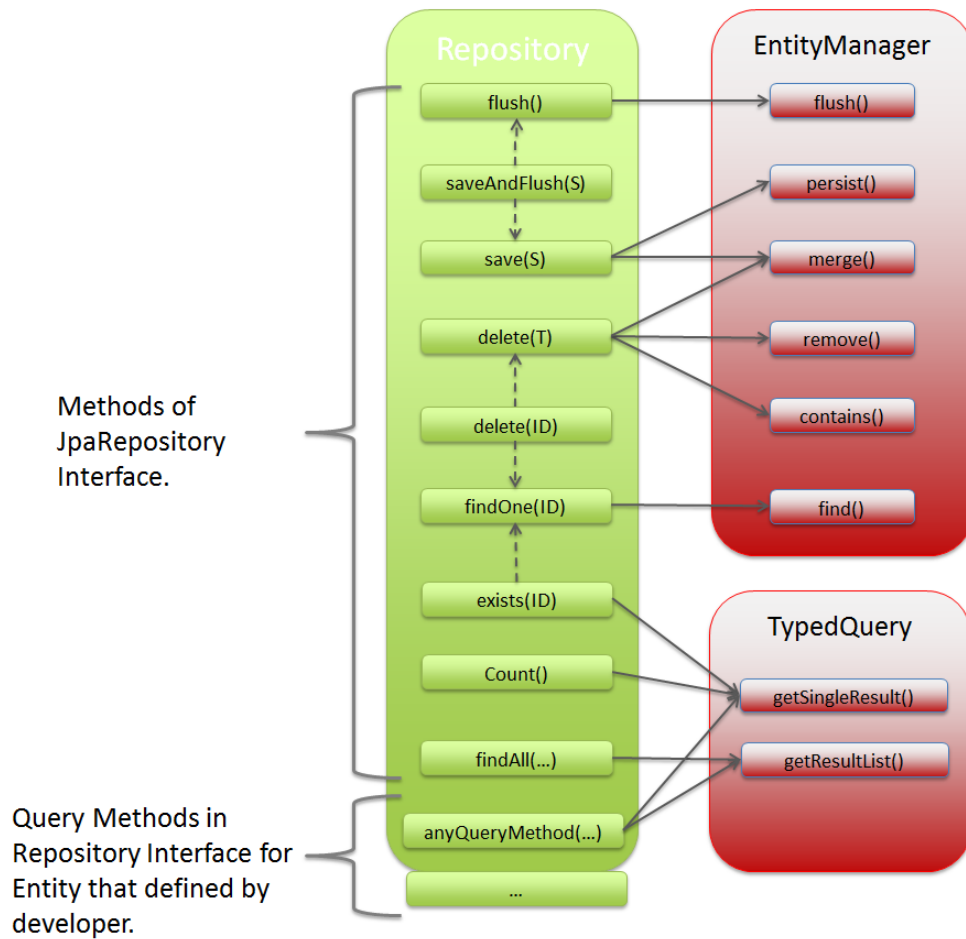


# Entity 클래스

- RDB에서 관리되는 레코드를 표현하는 클래스.  
@javax.persistence.Entity 어노테이션을 붙이면 Entity 클래스가 됨

# Spring Data JPA

- Spring Data JPA란? JPA를 사용하여 Repository(DAO클래스 같은 것-인터페이스로 작성)를 작성하기 위한 라이브러리
- Spring Data JPA를 이용하면 Query메서드를 부르는 메서드를 Repository인터페이스에 정의하는 것만으로 조건에 일치하는 Entity를 취득 가능.



# uplus라는 db

	id	name	price
▶	1	갤럭시S8	26300
	2	갤럭시S8+ (정품)	28600
	3	LG G6	29200
	4	LG X400	1450
	5	갤럭시 A5	12560
	6	IPHONE 7 128GB	38260
	7	LG X300	2490
	8	IPHONE 6S 128GB	21340
	9	갤럭시 노트5 64gb	14590
	10	LG V20	21820
	11	IPHONE 7 Plus 1...	44630
	12	갤럭시 On7	7820
	13	갤럭시 폴더	1590
	14	갤럭시 A7	6180
	15	Galaxy S7 32G	21620
★	NULL	NULL	NULL

# 기존의 Phone클래스를 Entity화

- @Entity
- **public class** Phone {
  - @Id
  - **private int** id;

# JPA 환경설정

- LocalContainerEntityManagerFactoryBean
- DataSource 설정
- JpaTransactionManager
- Tx:annotation-driven 설정

# LocalContainerEntityManagerFactoryBean

- `<bean id= "myEMF"`
- `class= "org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">`
- `<property name= "dataSource" ref= "dataSource" />`
- `<property name= "packagesToScan" value= "com.oraclejava.todo.domain.model"/>`
- `<property name= "jpaVendorAdapter">`
- `<bean class= "org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />`
- `</property>`
- `<property name= "jpaProperties">`
- `<props>`
- `<!-- 개발시: update, 운용시: none, 개발환경 새로 만들때 : create-drop -->`
- `<prop key= "hibernate.hbm2ddl.auto">create-drop</prop>`
- `<prop key= "hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</prop>`
- `<prop key= "hibernate.show_sql">true</prop>`
- `</props>`
- `</property>`
- `</bean>`



# *JpaTransactionManager*

- `<bean id= "transactionManager"`
- `class= "org.springframework.orm.jpa.JpaTransactionManager">`
- `<property name= "entityManagerFactory"`  
`ref= "myEMF"> </property>`
- `</bean>`

# AbstractDao 작성

- EntityManager 작성
- @PersistenceContext 로 의존성 주입

- **public class AbstractDao<PK extends Serializable, T> {**
- **private final Class<T> persitentClass;**
- **@SuppressWarnings("unchecked")**
- **public AbstractDao() {**
- **//제네릭스로 부모클래스가 운용하는 파라미터의 클래스 취득**
- **this.persitentClass =**  
**(Class<T>)((ParameterizedType)this.getClass().**
- **getGenericSuperclass()).getActualTypeArguments()[1];**
- **}**

- @PersistenceContext
- EntityManager entityManager;
- **protected EntityManager getEntityManager() {**
- **return this.entityManger;**
- **}**
- **protected T getByKey(PK key) {**
- **return (T)entityManager.find(persitentClass, key);**
- **}**

- **protected void persist(T entity) {**
- entityManager.persist(entity);
- }

- **protected void upd(T entity) {**
- entityManager.merge(entity);
- }

- **protected void del(T entity) {**
- entityManager.remove(entity);
- }
- }

# Dao작성

- EntityManager 사용
- JPQL 사용

- @Repository
- **public class TodoRepositoryImpl2 extends AbstractDao<Integer, Todo> implements TodoRepository {**
- @Override
- **public Todo findOne(int todoId) {**
- **return getByKey(todoId);**
- }
- @Override
- **public Collection<Todo> findAll() {**
- List<Todo> todos =
- getEntityManager()
- .createQuery("SELECT t FROM Todo t")
- .getResultList();
- **return todos;**
- }

- @Override
- **public void create(Todo todo) {**
- persist(todo);
- }

- @Override
- **public boolean update(Todo todo) {**
- upd(todo);
- **return true;**
- }

- @Override
- **public void delete(Todo todo) {**
- del(todo);
- }



- @Override
- **public long countByFinished(boolean finished) {**
- **long cnt = (Long)getEntityManager().**
- **createNativeQuery("SELECT count(\*) FROM Todo where**  
**finished = :finished")**
- **.setParameter("finished", finished)**
- **.getSingleResult();**
- **return cnt;**
- **}**
- **}**