소스 설치 및 실행

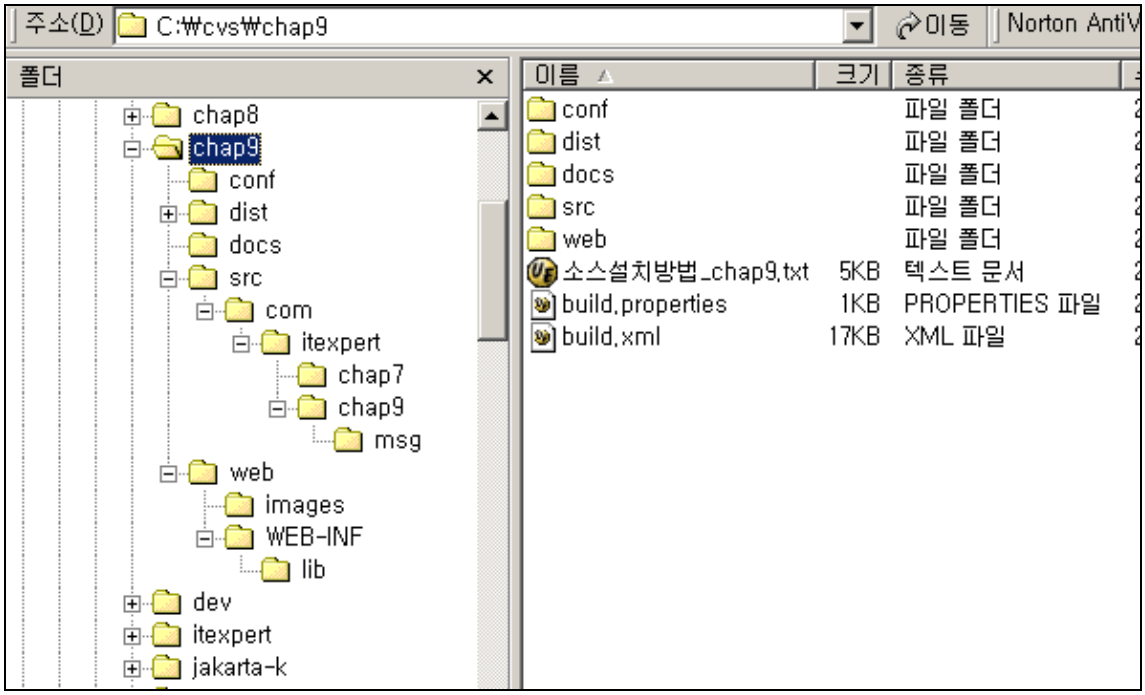
필요사항

* 일단 공통적으로 J2SDK1.4와 톰캣 4.1.18 이상 버전을 설치해야 된다. 편의상 J2SDK 가 설치된 디렉토리를 <JAVA_HOME>으로 톰캣 4.1.X가 설치된 디렉토리를 <CATALINA_HOME>으로 표시하겠다.

* 공통 설정 부분은 세 가지 방법 모두 필요한 부분이다. 주로 환경변수 설정이나 <CATALINA_HOME>/conf/server.xml 등을 변경해서 톰캣을 재시동해야 변경사항이 적용되는 경우이다.

가장 권장하는 방식은 ant를 이용하는 방식이고, 편한 방식은 war 파일을 이용하는 것이다. 그리고, 가장 까다롭고, 노가다성의 방식이 직접 컴파일하고 배치하는 방법이다. 각 장의 소스는 chap1.zip, chap2.zip 등으로 압축이 되어 있고, 압축을 풀면 필요한 jar파일들, 소스들, build.xml, 안내 파일들이 디렉토리별로 구분되어 있다.

일반적인 디렉토리 구조는 다음과 같다.



[그림 A-1] 소스의 일반적인 디렉토리 구조

공통 설치

-. 컨텍스트 추가와 DBCP 설정법

압축을 풀고 conf 디렉토리에 있는 Context.xml 파일을 편집기로 열어서 jdbc 정보를 수정한 다음 전체를 복사한다. 톰캣의 server.xml 파일을 열어서 다음 부분 뒤에 복사한 부분을 붙여넣는다.

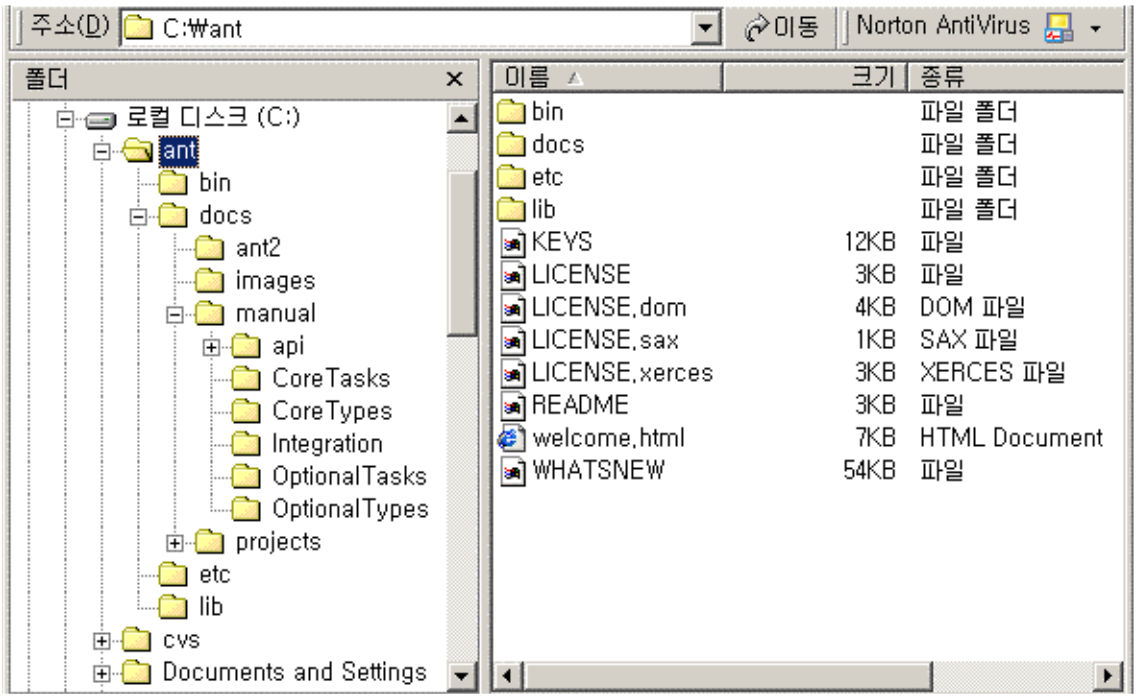
```
...
        <!-- Tomcat Root Context -->
        <!--
          <Context path="" docBase="ROOT" debug="0"/>
        -->
<!-- Tomcat chap9 Context -->
<Context path="/chap9" docBase="chap9" debug="0" reloadable="true">
    <Logger className="org.apache.catalina.logger.FileLogger"
...
```

Ant의 사용

-. Ant의 설치

1. CD의 ant 디렉토리에 있는 apache-ant-1.5.2-bin.zip파일을 적당한 위치에 압축을 풀어놓

는다. 예를 들면 c:/ant 디렉토리 같은 곳에 모든 서브디렉토리와파일을 풀어 놓는다. ant 디렉토리 구조는 다음과 같다.



[그림 A-2] ant의 디렉토리 구조

ant 최신 버전은 http://ant.apache.org/에서 구할 수 있다.

2. 바탕화면의 내컴퓨터 아이콘을 오른 마우스버튼으로 클릭해서 등록정보(R) 창을 연다.
3. 고급 탭을 클릭해서 환경변수(E)... 버튼을 클릭한다.
4. 시스템 변수(S) 아래에 새로 만들기(W)... 버튼을 클릭해서 창이 뜨면, 아래와 같이 ant가 설치된 디렉토리 경로를 ANT_HOME 환경변수에 넣어준다.

변수 이름(N): ANT_HOME

변수 값(V): c:₩ant

5. 시스템 변수(S)에서 Path 값에 ant 실행파일 경로를 추가합니다.

변수 이름(N): Path

변수 값(V): 기존 값에 ;%ANT_HOME%₩bin 추가

6. 확인버튼으로 창을 모두 닫는다.
7. 시작>실행(R)을 클릭해서 cmd를 입력하고 확인 버튼을 클릭한다.
8. 명령프롬프트 창이 뜨면 ant -version을 입력해서

C:₩>ant -version

Apache Ant version 1.5.2 compiled on February 28 2003

위와 같은 메시지가 나오면 ant가 잘 설치된 것이다.

-. 톰캣과 ant의 연결(Tomcat 4.1.12 이상 기준)

1. Tomcat의 /server/lib 디렉토리에 있는 catalina-ant.jar파일을 ant가 설치된 곳의 lib 디렉토리에 복사한다.
2. Tomcat의 /conf/tomcat-users.xml파일의 내용을 열어서 설치시 관리자 아이디와 비밀번호를 확인하고, 이 아이디에 "manager" role이 주어졌는지 확인한다.
예) id와 비밀번호가 각각 admin과 admin1일 경우
/conf/tomcat-users.xm 의 내용에서 다음 줄이 있는지 확인한다.

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="admin" password="admin1" roles="admin,manager"/>
</tomcat-users>
```

만일 설치시 id와 비밀번호 입력과정을 넘겼을 경우 위 내용을 추가한다.

-. 소스의 설치(Tomcat 4.1.12 이상 기준)

1. 작업디렉토리(예: c:/cvs)에 zip 파일을 풀어 놓는다.
예) c:/cvs/chap9
2. build.properties 파일을 열어서 톰캣의 경로와 manager.username과 manager.password 값을/conf/tomcat-users.xml 파일에 있는 manager role을 갖고 있는 id와 비밀번호로 수정한다.
예) build.properties

```
catalina.home=c:/Tomcat4.1
#포트가 있을 경우 http://localhost:8080/manager
manager.url=http://localhost/manager
manager.username=admin
manager.password=admin1
```

3. build.xml파일을 열어서 다음 프로퍼티 값을 자신의 환경에 맞게 수정한다.
예) build.xml 일부

```
<property name="app.name"         value="chap9"/>     <!-- 컨텍스트 이름 -->
<property name="build.home"       value="${catalina.home}/webapps/${app.name}"/>
```

build.properties와 build.xml에 겹치는 property가 있는 경우 build.propeties에 지정된 값이

우선순위를 갖는다.
4. 저장하고 명령프롬프트 창을 띄워서 웹 애플리케이션 작업디렉토리로 이동한다.
cd c:/cvs/chap9
5. build.xml과 build.properties 파일을 확인하고, 톰캣이 운영중인지 확인한다. 톰캣이 멈춰 있다면 톰캣을 가동한다.
6. ant reload
웹 어플리케이션이 컴파일되면서 새롭게 변경된 파일을 refresh시켜준다.
manager 웹 어플리케이션을 통해 reload해준다.
7. 브라우저를 열어서 접속을 시도해본다.
예) 포트가 80이면 http://localhost/chap9
8. app.name 프로퍼티(예;chap9)에 지정된 값으로 <CATALINA_HOME>/webapps/ 아래 디렉토리가 생성된다. 이 디렉토리가 컨텍스트 루트이다.
예) c:/Tomcat4.1/webapps/chap9
9. 소스의 변경 작업은 압축을 푼 곳에서 하고, ant reload로 적용시킨다.

war파일 사용

-. 공통 라이브러리 복사
압축을 풀고 lib 디렉토리에 있는 공통 라이브러리 파일(예. jdc-pool.jar, classes12.jar이 있는 경우 <CATALINA_HOME>/common/lib 디렉토리에 복사한다.
-. war 파일 설치
압축을 풀고 dist 디렉토리에 있는 war 파일(예; chap9.war)을 <CATALINA_HOME>/webapps 디렉토리에 복사한다.
톰캣의 /conf 디렉토리의 server.xml 파일을 열어서 아래 부분을 찾아서 unpackWARs="true" 가 맞는지 확인한다.

```
<!-- Define the default virtual host -->
        <Host name="localhost" debug="0" appBase="webapps"
         unpackWARs="true" autoDeploy="true">
```

자동으로 압축이 풀리지만 혹시 안 풀릴 경우는 톰캣을 재시동한다.

브라우저를 열어서 접속을 시도해본다.
예) 포트가 80이면 http://localhost/chap9

직접 컴파일 배치

-. 디렉토리 만들기

공통 설치의 path 속성에 정한 컨텍스트 루트 디렉토리와 웹 애플리케이션에 필요한 디렉토리를 생성한다. 예를 들면 다음과 같다. WEB-INF는 모두 대문자이다.

```
C:₩>c:
C:₩>cd Tomcat4.1
C:₩Tomcat4.1>cd webapps
C:₩Tomcat4.1₩webapps>mkdir chap9
C:₩Tomcat4.1₩webapps>cd chap9
C:₩Tomcat4.1₩webapps₩chap9>mkdir WEB-INF
C:₩Tomcat4.1₩webapps₩chap9>cd WEB-INF
C:₩Tomcat4.1₩webapps₩chap9₩WEB-INF>mkdir classes
C:₩Tomcat4.1₩webapps₩chap9₩WEB-INF>mkdir lib
```

-. 라이브러리 복사

lib 디렉토리에 있는 jar 파일들을 <CATALINA_HOME>/common/lib에 복사한다.

-. 파일 복사

압축을 풀고, web 디렉토리 아래 있는 내용을 공통 설치의 path속성에 정한 컨텍스트 루트로 복사한다.

예) C:/Tomcat4.1/webapps/chap9

-. java 소스 복사와 컴파일

src 디렉토리 아래 있는 java 파일들과 properties 파일들 그리고 컴파일 배치 파일(compile.bat)을 모두 컨텍스트 루트의 /WEB-INF/classes 디렉토리 아래로 디렉토리 통채로 복사한다. 다음 예제처럼 디렉토리를 이동한 후에 compile.bat를 실행한다.

```
C:₩Tomcat4.1₩webapps₩chap9₩WEB-INF>cd classes
C:₩Tomcat4.1₩webapps₩chap9₩WEB-INF₩classes>compile
```

톰캣을 재시동하고 브라우저를 열어서 접속을 시도해본다.

예) 포트가 80이면 http://localhost/chap9

9. JSTL (JSP Standard Tag Library)

1 JSTL 개요

1.1 JSTL 이란

커스텀태그를 공부하면서 jsp페이지에서 자바코드를 추출하는데 탁월한 성능을 발휘하는 모습을 보았다. 허나 이것을 만든다고 생각하면, 아직도 좀 아찔한 감이 남아있다. 개발시간은 촉박한데, 로직 부분을 빼낸다고, tld 만들고, 태그핸들러 만들고, 컴파일하고, 잘 동작하는지 테스트해보고 하려면 사실 시간이 더 많이 소요되기 때문에 망설여지는 것이 사실이다.

하지만 커스텀태그 소개하면서 계속 얘기했던 것은 자주 쓰이는 커스텀태그들을 표준으로 정해서 모아놓은 것이 있으니까, 이것들만 잘 찾아서 쓰면 된다고 얘기했다. 정말 알짜들만 모아놓았다. 최정예 커스텀태그들을 모아서 이름을 붙여준 것이 JSP Stadard Tag Library 이고, 줄여서 JSTL 이라고 부른다. JCP(http://www.jcp.org)에서 표준을 제정했다. JSR-52 에서 각계의 전문가들이 심사 후에 알짜들을 모아놓은 것이다.

1.2 JSTL의 현황

집필하는 현재 JSTL은 JSP2.0 스펙에 포함되었다. 따라서 JSP2.0 의 표준을 제일 먼저 구현해서 참고를 삼는 톰캣5.0 이 정식으로 발표된다면, 별다른 설치를 하지 않고 바로 JSTL을 사용할 수 있게 된다. (이미 발표된 톰캣5.0 알파버전에는 JSTL의 기능은 빠져 있다. EL 과 심플 커스텀태그와 태그파일의 기능은 들어가 있고, JSTL과 JSF는 빠져있다.)

JSTL 은 EL 을 이용해서 객체의 접근을 쉽게 할 수 있다. jsp 에서 객체의 접근은 먼저 선언부터 되어야 되지만, EL에서는 별다른 선언없이도 객체를 받아서 처리할 수 있게 되었다. 이에 관해서는 후에 예제를 통해서 살펴보겠다.

JSTL 에 관련된 최신 정보는 sun의 jstl 홈페이지 http://java.sun.com/products/jsp/jstl 와 자카르타 프로젝트의 taglibs http://jakarta.apache.org/taglibs 를 통해서 얻을 수 있다.

1.3 JSTL 처리영역

JSTL은 태생이 커스텀태그이기 때문에 jsp와 밀접하게 관계가 있다. application, session, request, response, pageContext 등의 내장객체에 쉽게 접근하며, 그 외에도 파라메터, 헤더, 쿠키 등을 복잡한 코드를 사용하지않고, 쉽게 직관적으로 사용할 수 있다. 또한 기본적인 연산이나 객체의 비교 등을 .equals() 메소드 등을 이용하는 대신 == 와 같이 쉽게 구현했으며, 조건, 반복, 이동에 대한 태그를 지원하기 때문에 태그만으로도 반복 기능을 구현할 수 있다.

JSTL 의 처리영역은 크게 4가지로 나누어진다. core, format, xml, sql 로 기능이 구분되고, 각각의 기능은 이름이 말해 주듯, 기본기능, 형식화, xml처리, sql처리를 담당한다.

기능	prefix	기본URI
기본기능	c	http://java.sun.com/jstl/core
XML 처리	x	http://java.sun.com/jstl/xml
i18n & 형식화	fmt	http://java.sun.com/jstl/fmt
데이터베이스 작업	sql	http://java.sun.com/jstl/sql

2 JSTL 설치

2.1 JSTL 다운로드

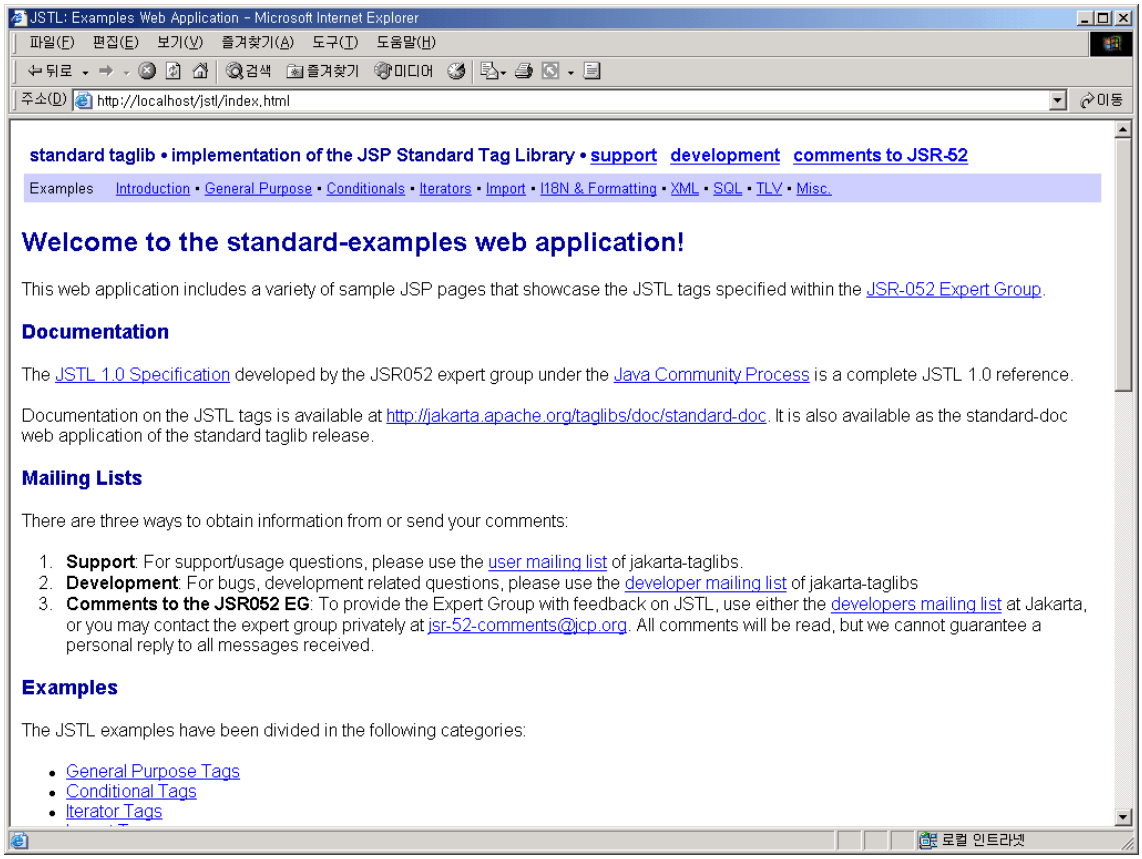
JSTL은 JSP2.0 표준 스펙에 포함이 되었지만 아직 이 스펙을 지원하는 제품이 나오지 않은 상태이기 때문에 기존의 톰캣4에서 사용할 수 있게 설치하는 법을 알아보려고 한다.

이 책의 논외가 되겠지만 sun에서 나온 JWSDP(Java Web Services Developer Pack)에는 톰캣 4.1.2와 JSTL v1.0.3 이 기본으로 들어있기 때문에 이것만 설치하면 쉽게 JSTL을 사용할 수 있다. 이에 관한 사용법은 다음 주소를 참고하기 바란다. http://java.sun.com/webservices/docs/1.1/tutorial/doc/index.html jstl의 기본 설명도 충실하게 잘 나와있고, 웹서비스에 관련된 좋은 안내서이기에 꼭 읽어보기 바란다.

JSTL은 현재 자카르타의 taglibs 서브 프로젝트에서 받아올 수 있다. http://jakarta.apache.org/taglibs 에 접속하고, Downloads 링크를 클릭한다. Release 버전과 Nightly Build 버전이 있는데, 정식 릴리스 버전은 현재 1.0.3 까지 나와있다. http://www.apache.org/dist/jakarta/taglibs/standard/ 에 가면 바이너리 빌드를 받아 볼 수 있다. jakarta-taglibs-standard-current.zip 를 다운로드 받아서 적당한 위치에 압축을 푼다.

2.2 JSTL 예제 설치

jakarta-taglibs 디렉토리 아래 standard 에서 standard-examples.war 파일을 jstl.war 파일로 이름을 바꾼 뒤에 Tomcat 4 가 설치된 디렉토리의 webapps 디렉토리에 복사한다. server.xml 의 <Host> 에 unpackWARs="true" 로 설정되어 있다면, jstl 이라는 디렉토리가 자동으로 생기면서 jstl.war 의 압축이 풀린 뒤에 브라우저에서 http://localhost:포트/jstl 로 예제 웹 어플리케이션을 실행해 볼 수 있다.



2.3 JSTL add-on 설치

기존의 컨텍스트에서 JSTL을 사용하기 위해서는 웹 어플리케이션의 WEB-INF/lib 디렉토리에 필요한 라이브러리를 복사하면 된다. JSTL 의 주된 라이브러리 파일은 jstl.jar, standard.jar 이고, xml에서 지원되는 기능을 사용하기 위해서 jaxen-full.jar, saxpath.jar, jaxp-api.jar 파일 등이 필요하다. 이 파일들을 웹어플리케이션의 WEB-INF/lib 에 복사하고, 컨텍스트를 리로드한다. 구체적인 파일들의 명세는 다음과 같다.

이름	설명	파일
JSTL 1.0 API 클래스	jstl 1.0 API 클래스	jstl.jar
JSTL 1.0 구현 클래스	JSTL 구현	standard.jar
Jaxen 1.0	Xpath 엔진	jaxen-full.jar
Saxpath 1.0	Xpath 파싱에 필요 sax 방식	saxpath.jar
Jdbc 2.0 선택 패키지	JDBC 구현 클래스 j2se1.4, 톰캣4 기본포함	jdbc2_0-stdext.jar
Jaxp 1.2 구현	jaxp 1.2 호환 파서 필요	jaxp-api.jar dom.jar sax.jar xercesImpl.jar

Xalan	아파치 xslt 변환기	xalan.jar

만일 톰캣4의 ROOT 컨텍스트에 설치를 한 경우에는 webapps/ROOT/WEB-INF/ 아래에 lib 디렉토리를 만들고 jstl 의 필요한 JAR 파일들을 복사해 놓으면 된다.

2.4 JSTL tld 파일

tld파일이 없거나, web.xml 파일에 8개의 tld 파일을 등록하지 않아도 표준태그는 사용할 수 있도록 되어 있다.

4가지의 JSTL 태그마다 EL 기반과 RT 기반의 태그로 나눠진다. 태그핸들러의 종류는 똑같지만 차이가 있다면 value 값으로 EL 을 사용하느냐 아니면 스크립트의 표현식을 허용하느냐의 차이다. 이에 관해서는 후에 살펴보겠다.

서버에 따라서 필요한 경우가 있으므로 간략히 방법을 설명하면, jakarta-taglibs 의 standard 디렉토리에 있는 tld 디렉토리의 8개의 tld 파일을 웹 어플리케이션의 WEB-INF/ 아래에 적당한 위치에 복사한 뒤에 이 파일들의 위치를 web.xml 에 등록한다.

3 표현언어 EL (Expression Language)

EL은 다양한 위치에 있는 데이터에 접근하기 위한 언어이다. 문법체계가 직관적으로 아주 쉽다. jsp 에서는 모든 변수의 생성과 선언을 반드시 표시를 해주어야 되지만 EL은 그 과정 없이 바로 접근이 가능하다.

3.1 EL 의 표시형식

EL임을 표시하는 형식은 ${} 이다. 이 안에 들어있는 것은 EL 로 처리된다. 내장객체의 접근이 가능하고, 산술연산과 비교연산이 가능하다.

자바스크립트와 비슷한 방법으로 객체 내부의 자원에 접근이 가능하다. dot(.) 과 bracket([]) 모두를 이용해서 접근할 수 있으며, 예를 들면

${header.cookie} 와 ${header['cookie']} 는 같은 결과 값을 얻을 수 있다. 만일 user-agent 처럼 - 와 같이 있을 경우는 ${header['user-agent']} 로 접근하는 것이 좋다.

[] 를 이용해서 객체에 접근할 경우에 index 를 대신해서 사용할 수도 있다.

다음 예제는 이전 내용을 요약한 것이다.

예제 1. jstlel01.jsp

```
<% response.setContentType("text/html;"); %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<h3>header</h3>
<c:forEach items="${header}" var="h">
  <c:out value="${h}"/><br/>
</c:forEach>
<h3>header.cookie</h3>
<c:out value="${header.cookie}"/><br/>
<c:out value='${header["cookie"]}'/>
```

header 라는 객체가 갖고 있는 값들을 불러서 보여주는 소스이다. request.getHeader() 로 가져올 수 있는 것인데, 훨씬 깔끔한 소스로 보여진다.

JSTL 태그에서 기본적으로 EL을 사용한다. var 속성은 변수를 지정하는데, 여기에는 문자열 상수로 지정되며, 이 후에 이 변수를 참조하기 위해서는 EL 을 사용한다.

3.2 EL 내장객체

EL 로 접근할 수 있는 내장 객체들은 다음과 같다.

pageScope	page scope 의 변수들
requestScope	request scope 의 변수들

sessionScope	session scope 의 변수들
applicationScope	application scope 의 변수들
param	parameter 변수들 문자열
paramValues	parameter 변수들 문자열 배열
header	HTTP request 헤더
headerValues	HTTP request 헤더 문자열 배열
initParam	컨텍스트 초기 변수 web.xml 에서 지정
cookie	쿠키 변수들
pageContext	현재 페이지의 pageContext 객체

pageScope 의 변수는 스크립틀릿의 변수와는 틀리며, 이를 이용하려면 조금은 번잡한 과정을 거치게 된다. 따라서 가능하면 스크립틀릿은 사용하지 않기를 권장한다. 이를 활용하는 소스이다.

예제 2. jstlel02.jsp

```
<% response.setContentType("text/html;"); %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="c_rt" uri="http://java.sun.com/jstl/core_rt" %>
<% String[] abc = {"빨강","파랑","노랑"}; %>

<c:set var="t" value="hello"/>
<c_rt:set var="color" value="<%=abc%>"/>

<h3>pageScope</h3>
<c:forEach items="${pageScope}" var="h">
    <c:out value="${h}"/><br/>
</c:forEach>

<h3>colors</h3>
<c:forEach items="${color}" varStatus="i">
    <c:out value="${i.count}"/>.
    <c:out value="${color[i.index]}"/><br/>
</c:forEach>
```

<c:forEach/> 를 통해서 pageScope 의 값을 모두 출력해보면, String 배열변수인 abc 는 접근할 수 없게 되어있다. <c:set/> 과 <c_rt:set/> 으로 정해놓은 t와 color 만 나타나는 것

이 보인다.
color 의 값을 출력하는 코드를 보면 color 의 인덱스를 이용해 배열 내용을 뽑아 왔다. varStatus 대신 var 변수를 통해서 바로 내용을 출력할 수도 있다.

value 가 들어가는 자리에 표현식을 쓰기 위해서는 RT 기반의 태그를 사용해야 된다. 만일 <c_rt:set var="color" value="<%=abc%>"/> 에서 <c_rt:set/> 대신 <c:set/> 를 사용하면 color 가 갖게 되는 값은 "<%=abc%>" 문자열이 되어버린다.

3.3 EL 연산자

null 인지 아닌지 판단하기 위해서는 empty 라는 키워드를 사용한다. 세션 값이 없으면 세션에 값을 저장하고, 있을 경우 하나를 더한 후에 출력하는 코드이다

예제 3. jstlel03.jsp

```
<% response.setContentType("text/html;"); %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<c:if test="${!empty hit}">
    <c:set var="hit" value="${hit+1}" scope="session"/>
    <c:out value="${hit}"/> 번 리로드되었습니다.
</c:if>
<c:if test="${empty hit}">
    <c:set var="hit" value="1" scope="session"/>
    세션에 값을 저장했습니다.
</c:if>
<br>
<a href="jstlel03.jsp">reload</a>
```

EL 의 장점은 객체의 접근이 쉽다고 했다. ${empty 객체} 를 통해서, 객체의 null 을 확인할 수 있다. ${!empty hit} 는 ${!empty sessionScope.hit} 로 대치해도 같은 결과가 나온다.
EL 의 연산자는 관계연산자, 산술연산자, 논리연산자, empty 연산자가 있고 다음 표와 같다.

연산자 구분	연산자
관계	< lt > gt <= le >= ge == eq != ne
산술	+ - * / div % mod
논리	&& and || or ! not
Empty	empty

연산자들간의 우선순위는 다음과 같다.

1. [] .
2. ()
3. - (단항) not ! empty
4. * / div % mod
5. + - (이항)
6. < > <= >= lt gt le ge
7. == != eq ne
8. && and
9. || or

4 JSTL Core

core 태그를 사용하기 위해서 페이지 상단에 다음과 같이 선언되어야 된다.

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

가장 빈번하고 자주 쓰이는 것이다. Core 의 태그들은 다음과 같이 정리된다.

기능	태그	prefix
EL 지원	catch , out , remove , set	c
흐름 제어	choose (when , otherwise) , forEach , forTokens , if	
URL 관리	Import (param) , redirect (param) , url (param)	

4.1 <c:out/>

가장 많이 쓰게 되는 것은 JSP 의 표현식을 대체하는 <c:out/> 이다. 다음과 같은 형식을 갖고 있다.

```
body 없는 경우
<c:out value="value" [escapeXml="{true|false}"] [default="기본값"] />

body 있는 경우
<c:out value="value" [escapeXml="{true|false}"] />
기본값
</c:out>
```

[] 으로 둘러 쌓인 부분은 생략 가능한 부분이다. value 와 default 값은 일반 문자열이나 EL 이 들어간다. value 안에 JSP의 표현식을 사용하려 한다면 RT 기반의 <c_rt:out> 을 사용해야 된다는 것은 이전 예제에서 살펴 보았다.

escapeXml 속성은 값 중에 포함된 < > & ' " 문자들을 각각 < > & ' "

로 출력한다. 생략될 경우 true 가 기본 값이다.
null 값의 처리에 대해서 JSP 의 expression 의 경우는 "null" 문자열로 출력이 되었던 것을 jstl의 스펙에서는 이 경우 빈 문자열("")또는 기본값으로 처리한다고 명시되어있다.

4.2 <c:set/>, <c:remove/>

<c:set/> 의 기본형식은 다음과 같다. scope 속성이 생략될 경우 기본값은 page 이다.

```
Syntax 1: scope 에 해당하는 변수에 속성 값을 정한다.
<c:set value="value"
        var="varName" [scope="{page|request|session|application}"]/>

Syntax 2: scope 에 해당하는 변수에 body 값을 정한다.
<c:set var="varName" [scope="{page|request|session|application}"]>
        body content
</c:set>

Syntax 3: 속성 값으로 target 객체의 프로퍼티 값을 정한다.
<c:set value="value"
        target="target" property="propertyName"/>

Syntax 4: body 값으로 target 객체의 프로퍼티 값을 정한다.
<c:set target="target" property="propertyName">
        body content
</c:set>
```

변수에 값을 할당한다. 빈과 같은 객체에 할당하기 위해서는 target 과 property속성을 이용한다.

<c:remove/> 는 JSP 의 removeAttribute() 와 같은 역할을 한다. 해당 scope 에 있는 변수를 제거하는 역할을 한다.

4.3 <c:catch/>

<c:catch/> 는 body 위치에서 실행되는 코드의 예외를 잡아내는 역할을 담당한다. var 속성을 지정해서 변수를 선언하면 그 변수에 예외의 내용이 들어가게 된다.

다음 예제는 이상의 태그를 사용한 예제이다.

예제 4. jstlcore01.jsp

```
<% response.setContentType("text/html"); %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<h3>코어 </h3>
<h4>&lt;c:out></h4>
<pre>
${1+2} <c:out value="${1+2}"/>
${1>3} <c:out value="${1>3}"/>
${1 gt 3} <c:out value="${1 gt 3}"/>

${ 표시 <c:out value="${'${'}test}"/>

escapeXml 속성; 기본값은 false
false: <c:out value="<b>bold</b> <,>,&,',₩" " escapeXml="false"/>
true:  <c:out value="<b>bold</b> <,>,&,',₩" " escapeXml="true"/>

" 큰따옴표 사용주의; ' 작은따옴표로 대치
&lt;c:out value='&lt;font color="blue">파랑&lt;/font>'/>
<c:out value='<font color="blue">파랑</font>' escapeXml="false"/>

<hr><h4>&lt;c:set></h4>
set session scope var "name": <c:set var="name" value="하늘" scope="session"/>
c:out name: <c:out value="${name}"/>
expression name: <%= session.getAttribute("name")%>

set page scope var "name": <c:set var="name">
  hello
</c:set>
c:out name: <c:out value="${pageScope.name}"/>
c:out sessionScope.name: <c:out value="${sessionScope.name}"/>
expression name: <%= session.getAttribute("name")%>

<hr><h4>&lt;c:remove></h4>
remove session scope var "name": <c:remove var="name" scope="session"/>
expression name: <%= session.getAttribute("name")%>
c:out sessionScope.name: <c:out value="${sessionScope.name}"/>
```

```
<hr><h4>&lt;c:catch></h4>
<c:catch var="errmsg">
line1
<%=1/0 %>
line2
</c:catch>
<c:out value="${errmsg}"/>
</pre>
```

value 속성에 들어가는 값은 문자열과 EL 이다. EL의 경우 ${} 안에서 나온 결과값을 표시한다. 부등호 > 과 gt 는 같은 뜻이다. "${" 를 표시하기 위해서는 EL 로 감싸야 된다. 즉 ${'${'} 로 해야 표시된다.
escapeXml 속성은 브라우저에서 특수한 기능을 하는 문자 표시 여부를 결정한다. false 값일 경우는 태그가 먹힌 굵은 글씨의 bold 가 나오고, true 일 경우는 '<'과 '>'를 각각 '<','>' 로 변환한다. 결과는 bold 로, 브라우저에서 태그가 보이도록 나온다. 큰따옴표와 작은따옴표는 바꿔서 쓸 수 있다. 대신 짝이 맞아야 된다. 또한 하나로 다 이어서 쓸 경우 변환과정에서 에러가 나기 때문에 주의해야된다.

따옴표 사용 예	외부	문자열내부	사용
<c:out value='파랑' />	작은	큰	가능
<c:out value="파랑" />	큰	작은	가능
<c:out value="파랑" />	큰	큰	불가

session 스코프에 name 이라는 key 로 "하늘"을 넣는다. <c:out value="${name}"/> 으로 scope 를 지정하지 않아도 내장 객체를 훑어서 sessionScope에서 걸리는 "name"키를 찾아서 출력한다. 이 값은 스크립틀릿에서도 참고할 수 있다.
page 스코프에 같은 key 에 "hello" 라는 값을 넣으면 <c:out value="${name}"/> 은 더 이상 session 에 있는 값을 가져오지 않는다.
<c:remove/> 를 통해서 scope 속성에 지정된 key 값을 제거한다.

예제에서 <c:catch/> 태그는 body 실행 도중에 <%=1/0 %> 에서 예외가 발생한 것을 errmsg 라는 변수에 넣는다. 이 후에 <c:out/> 을 통해서 에러 메시지를 표시한다.

4.4 <c:if/>

<c:if/> 는 흔히 보는 조건문이다. 형식은 다음과 같다.

```
Syntax 1: Body 없는 경우
<c:if test="testCondition"
```

```
        var="varName" [scope="{page|request|session|application}"]/>

Syntax 2: Body 있는 경우
<c:if test="testCondition"
        [var="varName"] [scope="{page|request|session|application}"]>
    body content
</c:if>
```

<c:if/> 에서 나온 결과를 varName 변수에 넣고, 나중에 활용이 가능하다. 변수의 scope는 임의로 지정할 수 있고, 생략될 경우 기본값은 page 이다.

4.5 <c:choose/>, <c:when/>, <c:otherwise/>

<c:choose/> 태그는 java 의 switch 문과 같지만, 조건에 문자열 비교도 가능하고 쓰임의 범위가 넓다. 또한 <c:if/> 태그에 else 가 없기 때문에 이의 대체 기능도 수행한다.
형식은 다음과 같다.

```
<c:choose>
body content
(하나 이상의 <when> 과 하나 이하의 <otherwise> 서브태그)
    <c:when test="조건">
        body content
    </c:when>

    <c:otherwise>
        conditional block
    </c:otherwise>
</c:choose>
```

조건 판단을 수행하는 간단한 예제이다.

예제 5. jstlcore02.jsp

```
<% response.setContentType("text/html"); %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<h3>조건</h3>
파라메터 없음:<c:out value="${empty param.name}" />
<h4>&lt;c:if test=""></h4>
<c:if test="${empty param.name}">
```

```
<form>
이름을 적어주세요.<br>
    <input type="text" name="name">
    <input type="submit" value="확인">
</form>
</c:if>
<c:if test="${!empty param.name}">
    안녕하세요. <c:out value="${param.name}"/>님.
</c:if>

<h4>&lt;c:choose> &ltc:when test=""> &ltc:otherwise></h4>
<c:choose>
    <c:when test="${empty param.name}">
        <form>
        이름을 적어주세요.<br>
            <input type="text" name="name">
            <input type="submit" value="확인">
        </form>
    </c:when>
    <c:when test="${param.name=='admin'}">
        안녕하세요. 관리자님.
    </c:when>
    <c:otherwise>
        안녕하세요. <c:out value="${param.name}"/>님.
    </c:otherwise>
</c:choose>
```

파라메터 name 값이 없는 경우 입력 폼을 출력한다. 파라메터 name 의 값이 "admin"일 경우 관리자를 표시하고, 그 외에는 파라메터 값을 그대로 출력한다.
파라메터의 유무는 empty 와 !empty 연산자를 통해서 확인할 수 있다.

4.6 <c:forEach/>, <c:forTokens/>

<c:forEach/> 는 강력한 반복실행 태그이다. 형식은 다음과 같다.

```
Syntax 1: 객체 전체에 걸쳐서 반복
<c:forEach [var="varName"] items="collection"
           [varStatus="varStatusName"]
           [begin="begin"] [end="end"] [step="step"]>
```

```
    body content
</c:forEach>

Syntax 2: 지정한 횟수만큼 반복
<c:forEach [var="varName"]
           [varStatus="varStatusName"]
           begin="begin" end="end" [step="step"]>
    body content
</c:forEach>
```

<c:forEach/> 태그는 여러가지로 활용이 가능하다. 원하는 구간만큼 반복할 수도 있고, 객체를 받아와서 그 객체의 길이만큼 반복할 수도 있다. begin , end 속성은 시작번호와 끝번호를 지정하고, step 속성을 이용해서 증가 구간을 정할 수 있다. var 속성에서 정한 변수로 반복되는 내부 구간에서 사용할 수 있다.

<c:forTokens/> 는 java.util.StringTokenizer 를 이용한 것이다. 형식은 다음과 같다.

```
Syntax
<c:forTokens items="stringOfTokens" delims="delimiters"
        [var="varName"]
        [varStatus="varStatusName"]
        [begin="begin"] [end="end"] [step="step"]>
    body content
</c:forEach>
```

<c:forEach/> 와 <c:forTokens/> 를 활용한 예제이다.

예제 6. jstlcore03.jsp

```
<% response.setContentType("text/html"); %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<h3>반복</h3>
<h4>&lt;c:forEach></h4>

<c:forEach var="one" begin="1" end="10">
    <c:out value="${one}"/>
</c:forEach>
<p><b>header</b></p>
<c:forEach var="h" items="${header}">
```

```
    <c:out value="${h.key}:${h.value}"/><br>
</c:forEach>

<h4>&lt;c:forTokens></h4>
<c:forTokens var="one"
            items="서울|인천,대전,대구,부산,광주,평양"
            delims="," varStatus="sts">
    <c:out value="${sts.count}:${one}"/>&middot;
</c:forTokens>
<hr>
<c:forTokens var="one"
            items="서울|인천,대전,대구,부산,광주,평양"
            delims=",|" varStatus="sts">
    <c:out value="${sts.count}:${one}"/>&#149;
</c:forTokens>
```

예제의 첫번째 <c:forEach/> 에서는 1에서 10까지 반복하면서 값을 출력한다. 이때 var속성의 one변수에는 진행중인 값이 저장된다.

두번째 <c:forEach/> 반복문을 보면 items 속성에 header 객체를 받아온다. header 객체는 Map 형태이고, getKey() 와 getValue() 메소드의 사용이 가능하기 때문에 h.key 와 h.value 를 통해서 출력할 수 있다.

<c:forTokens/> 태그는 StringTokenizer 와 동일한 기능을 한다. delims 속성에 정해진 char 로 나뉘어지게 된다. 마지막 반복구간에서 마디로 나누는 기준은 , 과 | 두 가지이다. varStatus 속성에서 정해준 변수는 .index 와 .count 를 사용할 수 있고, 시작 번호는 각각 0 과 1 이다.

4.7 <c:import/>

이제 소개할 <c:import/> 는 아주 강력한 도구이다. 웹 어플리케이션 내부의 자원 접근은 물론이고, http, ftp 같은 외부에 있는 자원도 가져와서 페이지 내에 귀속시킨다. 자유롭게 가공할 수도 있고, 편집도 가능하다. <c:import/> 의 형식은 다음과 같다.

```
Syntax 1: 해당 주소를 바로 출력하거나 String 에 담아놓는다.
<c:import url="url" [context="context"]
        [var="varName"] [scope="{page|request|session|application}"]
        [charEncoding="charEncoding"]>
   <c:param> 서브 태그 위치
</c:import>
```

```
Syntax 2: 해당 주소의 컨텐츠를 Reader 객체로
<c:import url="url" [context="context"]
        varReader="varReaderName"
        [charEncoding="charEncoding"]>
    varReader 를 사용하는 액션
</c:import>
```

스트림으로 받아와서 파일로 저장하거나, DB에 입력할 수도 있도록 되어있다.

다음은 간단한 예제이다.

예제 7. jstlcore04.jsp

```
<% response.setContentType("text/html"); %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>

<c:set var="url" value="http://www.google.co.kr/"/>
<c:import url="${url}" var="u"/>
<c:out value="${url}"/> 가져옵니다.
<hr>
<base href="<c:out value="${url}"/>">
    <c:out value="${u}" escapeXml="false"/>
</base>
<hr>

<c:set var="url" value="http://www.okjsp.pe.kr"/>
<c:import url="${url}" var="u"/>
<c:out value="${url}"/> 가져옵니다.
<hr>
<pre><c:out value="${u}"/></pre>
<hr>

<c:set var="url" value="ftp://ftp.dacom.co.kr"/>
<c:import url="${url}" var="u"/>
<c:out value="${url}"/> 가져옵니다.
<hr>
<pre><c:out value="${u}"/></pre>
<hr>
```

```
<c:set var="url" value="jstlcore02.jsp"/>
<c:import url="${url}" var="u">
    <c:param name="name" value="admin"/>
</c:import>
<c:out value="${url}"/> 가져옵니다.
<hr>
<c:out value="${u}" escapeXml="false"/>
<hr>
```

4개의 주소를 불러오는 예제이다.

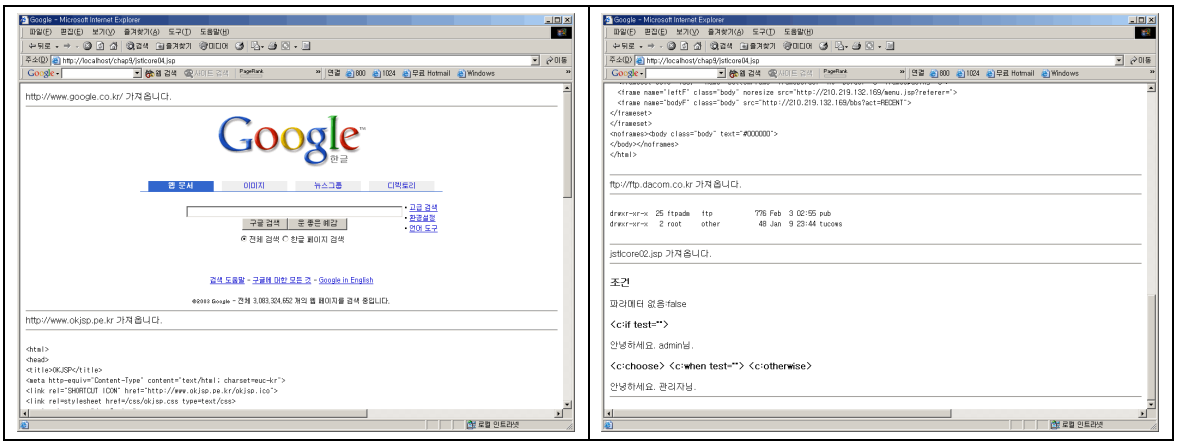
처음에는 구글사이트를 불러서 그대로 표시한다. escapeXml 속성을 false 로 해야 브라우저에 태그가 나타나지 않는다.

<base href=""> 태그는 소스를 가져왔을 때 상대경로에 있는 주소의 기준점이 된다. 따라서 상대경로로 표시된 이미지 등을 깨지지 않고 표시할 수 있다.

두번째는 okjsp 사이트를 가져와서 소스를 보여준다. escapeXml 속성이 생략되면 true값으로 지정되고, 태그를 볼 수 있게 했다.

세번째는 ftp.dacom.co.kr 에 있는 내용을 가져온다.

마지막에는 앞에서 보았던 jstlcore02.jsp 를 불러오는 예제이다. <c:param/> 태그를 내부에 두어서 파라메터값을 보낼 수 있다.



4.8 <c:url/>

<c:url/> 태그는 컨텍스트를 자동으로 추가해서 주소를 자동으로 생성해준다. context 속성이 지정되었을 경우 value 와 context 의 값은 / 로 시작을 해야된다. context 속성이 생략되면 당연히 현재의 컨텍스트가 적용된다. <c:url/> 의 형식은 다음과 같다.

```
Syntax 1: Body 없는 경우
<c:url value="value" [context="context"]
        [var="varName"] [scope="{page|request|session|application}"]/>
```

```
Syntax 2: Body 있는 경우 쿼리 스트링 파라메터 지정
<c:url value="value" [context="context"]
        [var="varName"] [scope="{page|request|session|application}"]>
    <c:param> 서브태그
</c:url>
```

다음은 간단한 예제이다.

예제 8. jstlcore05.jsp

```
<% response.setContentType("text/html"); %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>

<c:url value="/images/tomcat.gif"/>
<img src="<c:url value="/images/tomcat.gif"/>">
```

코드를 실행하면, <img src="/chap9/images/tomcat.gif"> 와 같이 출력이 된다. ROOT 컨텍스트일 경우는 <img src="/images/tomcat.gif"> 와 같이 출력된다.
다른 컨텍스트를 사용하는 경우 정적인 컨텐츠의 주소를 쓰기 위해서는 jsp 에서는 <img src="<%=request.getContextPath()%>/images/tomcat.gif"> 와 같이 처리를 해주어야 되었다. 이런 경우를 대체하는 태그이다.

4.9 <c:redirect/>

response.sendRedirect() 를 대체하는 태그이다. 컨텍스트를 지정해서 다른 컨텍스트로 이동이 가능하다.
<c:redirect/> 의 형식은 다음과 같다.

```
Syntax 1: Body 없는 경우
<c:redirect url="value" [context="context"]/>

Syntax 2: Body 있는 경우 쿼리 스트링 파라메터 지정
<c:redirect url="value" [context="context"]/>
    <c:param> 서브태그
</c:redirect>
```

다음은 간단한 예제이다.

예제 9. jstlcore06.jsp

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

<c:redirect url="/jstlcore01.jsp"/>
```

<c:url/> 태그와 마찬가지로 context 속성을 지정하면, value 와 context 의 값은 / 로 시작을 해야된다. <c:param/> 태그를 중첩시켜서 사용할 수도 있다.

4.10 <c:param/>

<c:param/> 은 다음과 같이 url 에 바로 붙여서 쓸 수도 있다.

```
<c:import url="/exec/dolt">
  <c:param name="action" value="register"/>
</c:import>
이 방법은 아래 태그와 같은 효과가 있다.
<c:import url="/exec/dolt?action=register"/>
```

이제까지 기본적인 태그인 core 태그들에 대해서 살펴보았다. 다음 섹션에서 출력 형식을 지정하는 fmt 태그를 살펴보도록 하겠다.

5 JSTL 국제화 지역화 태그

다국어 문서를 처리할 때 유용하고, 날짜와 숫자 형식을 다루는 fmt 태그는 다음과 같은 종류가 있다.

기능	태그	prefix
Locale 설정	setLocale, requestEncoding	fmt
메시지 처리	bundle, message(param), setBundle	
숫자 날짜 형식	formatNumber, formatDate, parseDate, parseNumber, setTimeZone, timeZone	

fmt 태그를 사용하기 위해서 페이지 상단에 다음과 같이 선언되어야 된다.

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

5.1 <fmt:setLocale/>

<fmt:setLocale/> 의 형식은 다음과 같다.

```
Syntax
<fmt:setLocale value="locale"
        [variant="variant"]
        [scope="{page|request|session|application}"]/>
```

다국어 페이지를 만들 경우 사용할 경우 ResourceBundle 로 불러오는 *.properties 파일들

과 연계되어서 사용할 수 있다. value 속성에 들어가는 locale 값은 [1]언어코드와 [2]국가코드로 이루어진다. 생략될 경우 톰캣 서버의 기본값으로 설정이 되고, 둘 중에 하나만 사용할 수도 있다. http://java.sun.com/j2se/1.4.1/docs/api/java/util/Locale.html api 에 보다 상세한 설명이 있다.

예제 10. jstlfmt01.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<pre>
default locale : <%= response.getLocale() %>
set locale : ko <fmt:setLocale value="ko" />
now: <%= response.getLocale() %>
set locale : ja <fmt:setLocale value="ja" />
now: <%= response.getLocale() %>
set locale : en <fmt:setLocale value="en" />
now: <%= response.getLocale() %>
</pre>
```

예제를 값을 변경시켜서 실행하면, response 쪽에 영향을 주는 것을 알 수 있다.

5.2 <fmt:requestEncoding/>

다음으로 볼 것은 request.setCharacterEncoding() 역할을 하는 <fmt:requestEncoding/> 태그이다. 형식은 다음과 같다.

Syntax

```
<fmt:requestEncoding [value="charsetName"]/>
```

파라메터를 MS949 로 인코딩하는 경우 다음과 같이 사용하면 된다.

```
<fmt:requestEncoding value="MS949"/>
```

예제 11. jstlfmt02.jsp

```
<%@ page contentType="text/html;charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<fmt:requestEncoding value="euc-kr"/>
파라메터:<c:out value="${param.id}"/>
<form method="post">
```

[1] http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt

[2] http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

```
    <input type="text" name="id">
    <input type="submit">
</form>
```

페이지 인코딩이 적용된 경우 request 에서 가져오는 parameter 와 맞지 않는 경우에 사용한다. 서버마다 차이가 있기 때문에, 테스트를 통해서 자신의 환경에 맞게 사용해야 된다. 톰캣4.1.19 의 경우 인코딩을 적용하지 않고도, 해결되는데 다음과 같은 형태로 사용할 수 있다.

예제 12. jstlfmt02b.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<% response.setContentType("text/html;"); %>
파라메터:<c:out value="${param.id}"/>
<form method="post">
    <input type="text" name="id">
    <input type="submit">
</form>
```

5.3 <fmt:bundle/>

properties 확장자를 사용하는 자원 파일을 읽어오는 역할을 하는 <fmt:bundle/> 에 대해서 알아보자. 형식은 다음과 같다.

```
Syntax
<fmt:bundle basename="basename"
        [prefix="prefix"]>
    body content
</fmt:bundle>
```

basename 속성에 지정된 properties 파일을 찾아서 locale 에 따라 읽어들인다.
properties 파일은 보통 WEB-INF/classes 아래에 위치하며 디렉토리의 깊이에 따라서 패키지형식의 이름을 취한다. TestBundle.properties 파일이 com/itexpert/chap9/msg 디렉토리에 있다면 basename="com.itexpert.chap9.msg.TestBundle" 이라고 지정하면 된다.
locale 이 ko 라면 TestBundle_ko.properties 파일을 읽어오게 되며, locale 이 맞지 않는 경우에는 TestBundle.properties 처럼 코드가 붙지 않은 파일을 읽어온다.
prefix 속성은 key 명칭이 공통적인 부분을 지정해서 body 에서 표현되는 key 를 단축시킨다. import 에서 패키지명을 지정하면 클래스명만 쓸 수 있는 것과 같이 생각할 수 있다.
properties 파일의 경우 j2sdk의 /bin/native2ascii.exe 를 이용해서 유니코드로 변환될 필

요가 있으나, 파일 수가 많을 경우 ant 의 <native2ascii> 태스크를 이용해서 쉽게 처리할 수 있다. chap9.zip 소스에 포함된 build.xml 파일에 보면 다음과 같은 태스크가 지정되어 있다.

```
<native2ascii encoding="EUC-KR"
            src="${src.home}"
            dest="${build.home}/WEB-INF/classes"
            includes="**/*_ko.properties"/>
```

5.4 <fmt:message/>

번들 태그에서 정한 값들을 가져오는 태그는 <fmt:message/>이다. 다음은 <fmt:message/> 태그의 형식이다.

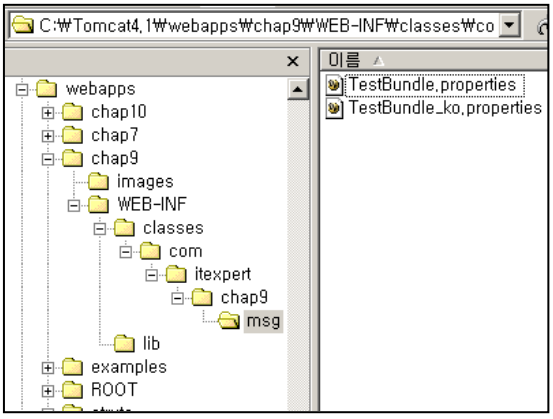
```
Syntax 1: body 없는 경우
<fmt:message key="messageKey"
        [bundle="resourceBundle"]
        [var="varName"]
        [scope="{page|request|session|application}"]/>

Syntax 2: 메시지 파라메터를 지정하는 body가 있는 경우
<fmt:message key="messageKey"
        [bundle="resourceBundle"]
        [var="varName"]
        [scope="{page|request|session|application}"]>
    <fmt:param> 서브태그
</fmt:message>

Syntax 3: 키와 선택적 메시지 파라메터를 지정하는 body가 있는 경우
<fmt:message [bundle="resourceBundle"]
        [var="varName"]
        [scope="{page|request|session|application}"]>
    key
    선택적 <fmt:param> 서브태그
</fmt:message>
```

번들에 있는 key 값을 불러온다. bundle 속성으로 번들을 직접 설정할 수도 있고, <fmt:bundle/> 태그 사이에 중첩되어서 키값만 받아서 출력할 수 있다.

예제 13. TestBundle.properties
greeting=Hello. admin=kenu



예제 14. TestBundle_ko.properties
greeting=안녕하세요. admin=허광남

예제 15. jstlfmt03.jsp
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%> <%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%> <fmt:setLocale value="ko"/> <fmt:bundle basename="com.itexpert.chap9.msg.TestBundle"> <fmt:message>greeting</fmt:message> <fmt:message>admin</fmt:message> </fmt:bundle>

locale 을 "ko" 로 지정했기 때문에 TestBundle_ko.properties 파일을 읽어온다. "en"으로 정하거나 맞는 프로퍼티가 없을 경우는 기본프로퍼티인 TestBundle.properties 파일을 읽는다.

<fmt:message> 를 통해서 greeting 키와 admin 키를 읽어와서 표시한다.

5.5 <fmt:setBundle/>

페이지 전체에서 사용할 수 있는 번들을 지정할 수 있는데, 이에 대한 지정은 <fmt:setBundle/> 태그가 담당한다. var 속성에서 정한 변수를 이후에 나오는 <fmt:message/> 태그에서 basename 속성에 변수명으로 대체할 수 있다. <fmt:setBundle/> 의 형식은 다음과 같다.

Syntax
<fmt:setBundle basename="basename" [var="varName"] [scope="{page|request|session|application}"]/>

예제 15. jstlfmt03.jsp 는 다음과 같이 수정할 수 있다.

예제 16. jstlfmt04.jsp
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>

```
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<fmt:setLocale value="ko"/>
<fmt:setBundle var="testBundle"
    basename="com.itexpert.chap9.msg.TestBundle"/>

<fmt:message bundle="${testBundle}" key="greeting"/><br>
<fmt:message bundle="${testBundle}" key="admin"/><br>
```

var 에서 변수를 설정할 때는 EL 을 사용하지 않는다. 다른 태그에서 이것을 불러오는 경우는 EL 을 사용해서 불러온다. testBundle 변수의 사용을 눈여겨 볼만하다.

5.6 <fmt:formatNumber/>

다음은 숫자 형식을 표현하는 <fmt:formatNumber/> 태그이고 형식은 다음과 같다.

```
Syntax 1: body 없는 경우
<fmt:formatNumber value="numericValue"
        [type="{number|currency|percent}"]
        [pattern="customPattern"]
        [currencyCode="currencyCode"]
        [currencySymbol="currencySymbol"]
        [groupingUsed="{true|false}"]
        [maxIntegerDigits="maxIntegerDigits"]
        [minIntegerDigits="minIntegerDigits"]
        [maxFractionDigits="maxFractionDigits"]
        [minFractionDigits="minFractionDigits"]
        [var="varName"]
        [scope="{page|request|session|application}"]/>

Syntax 2: 형식에 맞출 수치가 body에 있는 경우
<fmt:formatNumber [type="{number|currency|percent}"]
        [pattern="customPattern"]
        [currencyCode="currencyCode"]
        [currencySymbol="currencySymbol"]
        [groupingUsed="{true|false}"]
        [maxIntegerDigits="maxIntegerDigits"]
        [minIntegerDigits="minIntegerDigits"]
        [maxFractionDigits="maxFractionDigits"]
        [minFractionDigits="minFractionDigits"]
```

```
        [var="varName"]
        [scope="{page|request|session|application}"]>
    형식화될 수치
</fmt:formatNumber>
```

각 속성을 정리한 표는 다음과 같다.

속성	동적값	Type	설명
value	true	String 또는 Number	형식화될 수치
type	true	String	숫자, 통화, 퍼센트 중 어느 것으로 표시할지 지정 {number|currency|percent}
pattern	true	String	사용자가 지정한 형식 패턴.
currencyCode	true	String	ISO 4217 통화 코드. 통화 형식일 때만 적용 (type="currency")
currencySymbol	true	String	통화 기호. 통화 형식일 때만 적용 (type="currency")
groupingUsed	true	boolean	형식 출력에 그룹 분리기호를 포함할지 여부
maxIntegerDigits	true	int	형식 출력에서 integer 최대 자리수
minIntegerDigits	true	int	형식 출력에서 integer 최소 자리수
maxFractionDigits	true	int	형식 출력에서 소수점 이하 최대 자리수.
minFractionDigits	true	int	형식 출력에서 소수점 이하 최소 자리수.
var	false	String	형식 출력 결과 문자열을 담는 scope에 해당하는 변수명
scope	false	String	var 의 scope

5.7 <fmt:parseNumber/>

반대로 정해진 패턴을 문자열에서 수치를 파싱해내는 태그는 <fmt:parseNumber/>이며 형식은 다음과 같다.

```
Syntax 1: body가 없는 경우
<fmt:parseNumber value="numericValue"
        [type="{number|currency|percent}"]
        [pattern="customPattern"]
        [parseLocale="parseLocale"]
        [integerOnly="{true|false}"]
        [var="varName"]
```

```
        [scope="{page|request|session|application}"]/>

Syntax 2: 파싱할 수치를 body 에 갖고 있는 경우
<fmt:parseNumber [type="{number|currency|percent}"]
        [pattern="customPattern"]
        [parseLocale="parseLocale"]
        [integerOnly="{true|false}"]
        [var="varName"]
        [scope="{page|request|session|application}"]>
    파싱할 수치
</fmt:parseNumber>
```

각 속성을 정리한 도표는 다음과 같다.

속성	동적값	Type	설명
value	true	String 또는 Number	파싱할 수치
type	true	String	숫자, 통화, 퍼센트 중 어느 것으로 표시할 지 지정 {number|currency|percent}
pattern	true	String	사용자가 지정한 형식 패턴.
parseLocale	true	String 또는 java.util.Locale	파싱 작업의 기본 형식 패턴(숫자, 통화, 퍼센트 각각)을 제공하는 Locale
integerOnly	true	boolean	주어진 값에서 integer 부분만 파싱할지 여부를 지정
var	false	String	파싱 결과 (java.lang.Number 타입)를 담는 scope에 해당하는 변수명
scope	false	String	var 의 scope

5.8 <fmt:formatDate/>

다음은 날짜 형식을 표현하는 <fmt:formatDate/> 태그이고 형식은 다음과 같다.

```
Syntax
<fmt:formatDate value="date"
        [type="{time|date|both}"]
        [dateStyle="{default|short|medium|long|full}"]
        [timeStyle="{default|short|medium|long|full}"]
        [pattern="customPattern"]
```

```
        [timeZone="timeZone"]
        [var="varName"]
        [scope="{page|request|session|application}"]/>
```

각 속성을 정리한 도표는 다음과 같다.

속성	동적값	Type	설명
value	true	java.util.Date	형식화될 Date 와 time
type	true	String	형식화할 데이터가 시간, 날짜, 모두 인지 셋 중 하나를 지정한다.
dateStyle	true	String	미리 정의된 날짜 형식. Java.text.DateFormat 클래스에 정의된 문법을 따른다. type="date", type="body", type속성이 생략된 경우 사용.
timeStyle	true	String	미리 정의된 날짜 형식. Java.text.DateFormat 클래스에 정의된 문법을 따른다. type="time", type="body" 의 경우 사용.
pattern	true	String	사용자 지정 형식 스타일
timeZone	true	String 또는 java.util.TimeZone	형식화 시간에 나타날 타임존
var	false	String	형식 출력 결과 문자열을 담는 scope에 해당하는 변수명
scope	false	String	var 의 scope

5.9 <fmt:parseDate/>

정해진 패턴의 문자열에서 날짜를 파싱해내는 태그는 <fmt:parseDate/>이며 형식은 다음과 같다.

```
Syntax 1: body 없는 경우
<fmt:parseDate value="dateString"
        [type="{time|date|both}"]
        [dateStyle="{default|short|medium|long|full}"]
        [timeStyle="{default|short|medium|long|full}"]
        [pattern="customPattern"]
        [timeZone="timeZone"]
        [parseLocale="parseLocale"]
        [var="varName"]
        [scope="{page|request|session|application}"]/>
```

```
Syntax 2: 파싱한 값이 body 에 있는 경우
<fmt:parseDate [type="{time|date|both}"]
        [dateStyle="{default|short|medium|long|full}"]
        [timeStyle="{default|short|medium|long|full}"]
        [pattern="customPattern"]
        [timeZone="timeZone"]
        [parseLocale="parseLocale"]
        [var="varName"]
        [scope="{page|request|session|application}"]>
    파싱할 Date 와 time
</fmt:parseDate>
```

각 속성을 정리한 도표는 다음과 같다.

속성	동적값	Type	설명
value	true	java.util.Date	파싱할 Date 와 time
type	true	String	파싱할 데이터가 시간, 날짜, 모두 인지 셋 중 하나를 지정한다.
dateStyle	true	String	미리 정의된 날짜 형식. Java.text.DateFormat 클래스에 정의된 문법을 따른다. type="date", type="body", type속성이 생략된 경우 사용.
timeStyle	true	String	미리 정의된 날짜 형식. Java.text.DateFormat 클래스에 정의된 문법을 따른다. type="time", type="body" 의 경우 사용.
pattern	true	String	사용자 지정 형식 스타일
timeZone	true	String 또는 java.util.TimeZone	형식화 시간에 나타날 타임존
parseLocale	true	String 또는 java.util.Locale	파싱하는 동안 적용될 미리 정의된 형식 스타일의 Locale
var	false	String	파싱 결과(java.util.Date)를 담는 scope에 해당하는 변수명
scope	false	String	var 의 scope

다음은 숫자, 날짜에 대한 태그 사용 예제이다.

예제 17. jstlfmt05.jsp

```
<%@ page pageEncoding="MS949" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt" %>
<pre><fmt:setLocale value="ko_KR"/>
number  : <fmt:formatNumber value="9876543.61" type="number"/>
currency: <fmt:formatNumber value="9876543.61" type="currency"/>
percent : <fmt:formatNumber type="percent">9876543.61</fmt:formatNumber>

pattern=".000"    :<fmt:formatNumber value="9876543.61" pattern=".000" />
pattern="#,#00.0#":<fmt:formatNumber value="9876543.612345" pattern="#,#00.0#"/>

<jsp:useBean id="now" class="java.util.Date"/>
<c:out value="${now}"/>
date: <fmt:formatDate value="${now}" type="date"/>
time: <fmt:formatDate value="${now}" type="time"/>
both: <fmt:formatDate value="${now}" type="both"/>

default:<fmt:formatDate value="${now}"
            type="both" dateStyle="default" timeStyle="default"/>
short  :<fmt:formatDate value="${now}"
            type="both" dateStyle="short"   timeStyle="short"  />
medium :<fmt:formatDate value="${now}"
            type="both" dateStyle="medium"  timeStyle="medium" />
long   :<fmt:formatDate value="${now}"
            type="both" dateStyle="long"    timeStyle="long"   />
full   :<fmt:formatDate value="${now}"
            type="both" dateStyle="full"    timeStyle="full"   />

pattern="yyyy년MM월dd일 HH시mm분ss초"
         <fmt:formatDate value="${now}" type="both"
            pattern="yyyy년MM월dd일 HH시mm분ss초"/>
</pre>
```

5.10 <fmt:setTimeZone/>, <fmt:timeZone/>

특정 스코프의 타임존을 설정하는 <fmt:setTimeZone/> 태그의 형식은 다음과 같다.

```
Syntax
<fmt:setTimeZone value="timeZone"
        [var="varName"]
        [scope="{page|request|session|application}"]/>
```

타임존을 부분 적용하는 <fmt:timeZone/> 태그는 다음과 같은 형식이다.

```
Syntax
<fmt:timeZone value="timeZone">
    body content
</fmt:timeZone>
```

타임존을 적용한 시간 표시 예제이다.

예제 18. jstlfmt06.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<pre><fmt:setLocale value="ko_KR"/>
<jsp:useBean id="now" class="java.util.Date"/>

default: <c:out value="${now}"/>
Korea   KST  : <fmt:formatDate value="${now}" type="both" dateStyle="full"
                    timeStyle="full"/>
<fmt:timeZone value="GMT">
Swiss   GMT  : <fmt:formatDate value="${now}" type="both" dateStyle="full"
                    timeStyle="full"/>
</fmt:timeZone>
<fmt:timeZone value="GMT-8">
NewYork GMT-8: <fmt:formatDate value="${now}" type="both" dateStyle="full"
                    timeStyle="full"/>
</fmt:timeZone>
</pre>
```

이상으로 태그를 이용해서 숫자, 날짜, 시간 등을 표현하는 방법을 알아보았다. 태그의 속성들이 직관적이기 때문에 사용에 그리 불편하지 않을 것이다.

6 JSTL SQL 태그

DataSource 를 이용해서 SQL을 처리하는 sql 태그는 다음과 같은 것들이 있다.

기능	태그	prefix
DataSource 설정	SetDataSource	sql
SQL	query (dateParam, param) , update (dateParam, param) , transaction	

sql 태그를 사용하기 위해서 페이지 상단에 다음과 같이 선언되어야 된다.

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

6.1 <sql:setDataSource/>

DataSource 를 지정하는 방식은 <sql:setDataSource/> 태그의 사용법은 다음과 같다.

```
Syntax
<sql:setDataSource
        {dataSource="dataSource" |
            url="jdbcUrl"
            [driver="driverClassName"]
            [user="userName"]
            [password="password"]}
        [var="varName"]
        [scope="{page|request|session|application}"]/>
```

오라클에서 사용을 한다면 다음과 같이 DataSource 를 설정할 수 있다.

```
<sql:setDataSource
            url="jdbc:oracle:thin:@localhost:1521:ora81"
            driver="oracle.jdbc.driver.OracleDriver"
            user="scott"
            password="tiger"
        var="okjspDS"
        scope="application" />
```

이미 컨텍스트에 JNDI 설정이 되어있다면 다음과 같이 바로 불러서 사용하거나 <sql:query/> 에서 바로 사용할 수 있다.

기존의 dataSource 를 불러와 사용하는 경우

```
<sql:setDataSource
        dataSource="jdbc/myora81"
```

```
        var="okjspDS"
        scope="application" />

<sql:query/> 에서 바로 사용하는 경우
<sql:query var="emp"
        dataSource="jdbc/myora81">
```

6.2 <sql:query/>

java 와는 달리 sql 문장을 문자열로 연결하지 않아도 가독성을 높여서 작성할 수 있다. <sql:query/>태그의 형식은 다음과 같다.

```
Syntax 1: body 없는 경우
<sql:query sql="sqlQuery"
        var="varName" [scope="{page|request|session|application}"]
        [dataSource="dataSource"]
        [maxRows="maxRows"]
        [startRow="startRow"]/>

Syntax 2: body 에 쿼리의 파라메터가 있는 경우
<sql:query sql="sqlQuery"
        var="varName" [scope="{page|request|session|application}"]
        [dataSource="dataSource"]
        [maxRows="maxRows"]
        [startRow="startRow"]>
    <sql:param> 액션들
</sql:query>

Syntax 3: 쿼리와 파라메터들이 body 에 있는 경우
<sql:query var="varName"
        [scope="{page|request|session|application}"]
        [dataSource="dataSource"]
        [maxRows="maxRows"]
        [startRow="startRow"]>
    sqlQuery
    선택적 <sql:param> 액션들
</sql:query>
```

6.3 <sql:dateParam/> , <sql:param/>

파라메터에는 두 가지가 있는데, 날짜 형식의 <sql:dateParam/> 와 일반적인 <sql:param/> 태그가 있으며 형식은 다음과 같다.

```
Syntax
<sql:dateParam value="value" type="[timestamp|time|date]"/>

Syntax 1: value 속성에 파라메터 값이 지정된 경우
<sql:param value="value"/>

Syntax 2: body 내용에 파라메터 값이 지정된 경우
<sql:param>
    parameter value
</sql:param>
```

<sql:dateParam/>은 java.sql.PreparedStatement.setTimestamp() 역할을 하고,
<sql:param/> 은 java.sql.PreparedStatement.setString() 의 역할을 한다. 바인드변수의 순서에 따라서 써주면 된다.

7장의 jdbc_resultset.jsp 파일을 JSTL로 바꾸어서 변경한 것이다.

예제 19. jstlsql01.jsp

```
<%@ page pageEncoding="MS949" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<fmt:setLocale value="ko" />
<sql:query var="emp"
        dataSource="jdbc/myora81">
SELECT  EMPNO AS 사원번호, ENAME AS 이름,
            SAL AS 월급여, HIREDATE AS 입사일
    FROM EMP
</sql:query>

<table border="1">
  <tr>
<%-- 필드의 정보를 출력한다.            --%>
```

```
  <c:forEach var="columnName" items="${emp.columnNames}">
    <th><c:out value="${columnName}"/></th>
  </c:forEach>

<%-- 데이터를 한 줄씩 출력한다.          --%>
  <c:forEach var="row" items="${emp.rowsByIndex}">
    <tr>
<%-- 필드의 길이만큼 반복한다.           --%>
      <c:forEach var="column" items="${row}" varStatus="i">
        <c:choose>
          <c:when test="${i.index==3}">
      <td><fmt:formatDate value="${column}" pattern="yyyy/MM/dd"/></td>
          </c:when>
          <c:otherwise>
      <td><c:out value="${column}"/></td>
          </c:otherwise>
        </c:choose>
      </c:forEach>
  </c:forEach>
</table>
<hr>
<table border="1">
  <c:forEach var="row" items="${emp.rows}">
  <tr>
    <td>번호: <c:out value="${row['사원번호']}"/></td>
    <td>이름: <c:out value="${row['이름']}"/></td>
  </tr>
  </c:forEach>
</table>
```

dataSource="jdbc/myora81" 의 JNDI는 7장에서 설명한 것처럼 컨텍스트에 설정된 DataSource 명이다.

SQL문은 보기 좋게 정렬을 해도 문자열로 덧붙일 필요가 없다. body 에 있는 sql문을 실행한 결과는 emp 라는 변수에 담겨서 이후에 사용이 된다. 이 변수는 ResultSet 과 같은데, JSTL에서 확장한 ResultSet 이고, javax.servlet.jsp.jstl.sql public interface Result 클래스로 내부적으로 정의된다. 지원하는 메소드는 다음과 같다.

```
javax.servlet.jsp.jstl.sql
public interface Result
    public java.util.SortedMap[] getRows()
    public Object[][] getRowsByIndex()
    public String[] getColumnNames()
    public int getRowCount()
    public boolean isLimitedByMaxRows()
```

위와 같은 메소드들이 내부적으로 정의되어있기 때문에 각각의 getter 메소드들을 사용해서 변수 emp를 활용할 수 있다.
<c:forEach/> 의 items 에 있는 ${emp.columnNames}는 getColumnNames() 메소드를 불렀다는 것을 알 수 있다. 이전의 7장에서 ResultSetMetaData 를 활용해서 뽑아낸 정보와 같은 효과를 볼 수 있다.
테이블 내용을 반환하는 ${emp.rowsByIndex} 도 유념할 만하다. 한 행을 row라는 변수에 넣은 뒤에 다음의 <c:forEach/> 에서 이 row변수의 컬럼별로 내용을 출력한다.
column 이라는 변수에 넣은 뒤에 column의 index가 3일 경우 날짜형식을 출력하기 위해서 <fmt:formatDate/> 태그를 사용했고, 그 외의 경우는 바로 출력하게 했다.
다음 <c:forEach/> 에서는 결과를 SortedMap 배열에 넣은 뒤에 한 줄씩 빼서 컬럼이름으로 빼내는 방식이다.

6.4 <sql:update/>

java.sql.Statement.executeUpdate() 메소드에 해당하는 <sql:update/> 태그의 형식은 다음과 같다.

```
Syntax 1: body 없는 경우
<sql:update sql="sqlUpdate"
        [dataSource="dataSource"]
        [var="varName"] [scope="{page|request|session|application}"]/>

Syntax 2: update 파라메터가 body에 있는 경우
<sql:update sql="sqlUpdate"
        [dataSource="dataSource"]
        [var="varName"] [scope="{page|request|session|application}"]>
    <sql:param> 액션들
</sql:update>
```

```
Syntax 3: update 문과 선택적 update 파라메터가 body에 있는 경우
<sql:update [dataSource="dataSource"]
        [var="varName"] [scope="{page|request|session|application}"]>
    sqlUpdate
    선택적 <sql:param> 액션들
</sql:update>
```

형식과 동작은 <sql:query/> 태그와 동일하다. 다른 점은 executeUpdate() 메소드를 수행하기 때문에 DB에 변경을 가할 수 있다는 것이다.

6.5 <sql:transaction/>

트랜잭션을 구현하는 <sql:transaction/> 태그의 형식은 다음과 같다.

```
Syntax
<sql:transaction [dataSource="dataSource"]
        [isolation=isolationLevel]>
    <sql:query> 과 <sql:update> 문들
</sql:transaction>

isolationLevel ::= "read_committed"
                | "read_uncommitted"
                | "repeatable_read"
                | "serializable"
```

격리 수준(isolationLevel)은 java.sql.Connection 의 setTransactionIsolation() 메소드를 사용한다. 정리해 놓은 도표는 7장 JDBC의 Transaction 을 참고하기 바란다.

7 JSTL XML 태그

7.1 xml 태그와 XPath

xml 태그를 사용하기 위해서는 XPath 를 먼저 이해할 필요가 있다. xml 소스 트리의 정확한 위치를 지정해주기 위한 경로지정 문법이며 XSLT와 XPointer 를 위해서 만들어진 것이다. xml 엘리먼트들을 노드(node) 로 접근한다. 파일 시스템과 유사하며 다음과 같은 특성이 있다.

- / 로 시작하면 절대경로처럼 root node 에서 시작된다.
- //로 시작할 경우는 모든 영역에서 해당 엘리먼트를 선택하게 된다.
- 표시는 이전 엘리먼트 아래의 모든 자식 엘리먼트를 나타낸다.
- 동일한 엘리먼트들이 있을 경우, [] 안에 포함된 숫자는 엘리먼트의 순번이다. 조건식이 올 경우 해당하는 것이 선택된다. last() 일 경우는 맨 마지막 엘리먼트를 표시한다.

- 속성은 @ 로 시작된다.
- normalize-space() 함수는 앞뒤 공백을 제거하는 trim() 역할을 한다.

이 장에 나오는 XPath의 기능은 이 정도로 소개하겠고, XPath에 관한 보다 자세한 내용은 이 장의 마지막에 소개한 인터넷 튜토리얼을 참고하기 바란다.

JSTL에서 XPath를 통해서 내장객체에 쉽게 접근할 수 있다.

표현	매핑
$foo	pageContext.findAttribute("foo")
$param:foo	request.getParameter("foo")
$header:foo	request.getHeader("foo")
$cookie:foo	maps to the cookie's value for name foo
$initParam:foo	application.getInitParameter("foo")
$pageScope:foo	pageContext.getAttribute("foo", PageContext.PAGE_SCOPE)
$requestScope:foo	pageContext.getAttribute("foo", PageContext.REQUEST_SCOPE)
$sessionScope:foo	pageContext.getAttribute("foo", PageContext.SESSION_SCOPE)
$applicationScope:foo	pageContext.getAttribute("foo", PageContext.APPLICATION_SCOPE)

예를 들어서 다음 문장은 parameter 로 받은 "name"의 값이 bar엘리먼트의 x속성의 값과 같은 것들을 선택하게 된다.

/foo/bar[@x=$param:name]

xml 태그는 다음과 같은 것들이 있다.

기능	태그	prefix
기본	out, parse, set	x
흐름 제어	choose (when, otherwise), forEach, if	
변환	transform (param)	

xml 태그를 사용하기 위해서 페이지 상단에 다음과 같이 선언되어야 된다.

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

7.2 <x:out/>

XPath에 지정한 패턴에 따라 xml내용을 출력하는 <x:out/> 태그의 형식은 다음과 같다.

Syntax

```
<x:out select="XPathExpression" [escapeXml="{true|false}"]/>
```

7.3 <x:parse/>

xml문서를 읽어서 파싱하는 <x:parse/> 태그는 다음과 같은 형식이다.

```
Syntax 1: String 또는 Reader 객체로 지정된 XML 문서
<x:parse xml="XMLDocument"
        {var="var" [scope="scopeName"]|varDom="var" [scopeDom="scopeName"]}
        [systemId="systemId"]
        [filter="filter"]/>

Syntax 2: body 내용으로 지정된 XML 문서
<x:parse
        {var="var" [scope="scopeName"]|varDom="var" [scopeDom="scopeName"]}
        [systemId="systemId"]
        [filter="filter"]>
    파싱할 XML 문서
</x:parse>
scopeName 은 {page|request|session|application} 중의 하나
```

7.4 <x:set/>

XPath에 따라 선택된 내용을 변수에 저장하는 <x:set/> 태그의 형식은 다음과 같다.

```
Syntax
<x:set select="XPathExpression"
        var="varName" [scope="{page|request|session|application}"]/>
```

7.5 <x:if/>

<c:if/> 태그와 마찬가지로 xml태그에도 <x:if/> 가 있고 형식은 <c:if/> 태그와 유사하다. <x:if/>의 형식은 다음과 같다.

```
Syntax 1: Body 없는 경우
<x:if select="XPathExpression"
        var="varName" [scope="{page|request|session|application}"]/>

Syntax 2: Body 있는 경우
<x:if select="XPathExpression"
        [var="varName"] [scope="{page|request|session|application}"]>
```

```
    body content
</x:if>
```

test 속성 대신에 select 속성으로 진위를 따지는데, 다음 3가지 기준을 유념할 필요가 있다. 이 세 가지 기준은 <x:choose/>의 <x:when/> 과 <x:forEach/> 에서도 동일하게 사용된다.

```
1. number 가 true 인 때는 + 또는 - 0 도 아니고, NaN(Not A Number) 도 아닐 경우
2. node-set 이 true 인 때는 empty 가 아닐 경우
3. string 이 true 인 때는 길이가 0 이 아닐 경우
```

7.6 <x:choose/>, <x:when/>, <x:otherwise/>

<c:choose/> 태그와 마찬가지로 xml태그에도 <x:choose/> 가 있고 형식은 <c:choose/> 태그와 유사하다.

```
Syntax
<x:choose>
    body content (<x:when> and <x:otherwise> 서브태그)
</x:choose>

Syntax
<x:when select="XPathExpression">
    body content
</x:when>

Syntax
<x:otherwise>
    conditional block
</x:otherwise>
```

7.7 <x:forEach/>

<x:forEach/> 태그는 XPath에 따라서 해당하는 엘리먼트 수만큼 반복하게 된다.

```
Syntax
<x:forEach [var="varName"] select="XPathExpression">
    body content
</x:forEach>
```

xml 태그를 활용한 예제이다.

예제 20. jstlxml01.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/xml" prefix="x" %>
<% response.setContentType("text/html;"); %>
<%-- 파라메터 받아서 출력 --%>
<c:if test="${!empty param.name}">
param: <x:out select="$param:name"/>
</c:if>
<form>
name: <input type="text" name="name">
<input type="submit">
</form>
<hr>
<%-- xml 데이터를 xdata 변수에 할당 --%>
<x:parse var="xdata">
<namecard>
    <person>
        <name>허광남</name>
        <id>남자</id>
        <email>kenu@email.com</email>
        <phone>111-2222-3333</phone>
    </person>
    <person>
        <name>노재춘</name>
        <id>남자</id>
        <email>suribada@email.com</email>
        <phone>222-3333-4444</phone>
    </person>
    <person>
        <name>이선재</name>
        <id>남자</id>
        <email>hsboy@email.com</email>
        <phone>333-4444-5555</phone>
    </person>
</namecard>
</x:parse>
```

```
<%-- XPath 를 이용해서 xdata에서 추출 --%>
<x:out select="$xdata//person[1]/name"/>
<x:out select="$xdata//person[last()]/name"/>
<hr>
<%-- person 으로 반복해서 email과 phone 출력 --%>
<table border="1">
<x:forEach select="$xdata//person">
<tr><td><x:out select="email" /></td>
<td><x:out select="phone" /></td></tr>
</x:forEach>
</table>
```

파라메터 name 의 접근은 EL에서는 ${param.name} 으로 사용하지만 XPath 에서는 $param:name 을 사용한다는 차이가 있다.
파라메터를 받아서 <x:out/> 으로 출력하는 부분이 제일 상단이고, 그 다음은 xml 데이터를 파싱하는 부분이다.
DTD 까지 쓰지 않아도 형식이 잘 갖춰지기만 하면 (well-formed) xml 데이터로 인식을 한다.
이것을 xdata라는 변수에 할당한 다음에 이후에 처리하게 된다.

xml 데이터는 파일로 따로 만든 후에 접근할 수 있다. 이때는 <c:import> 를 같이 사용하는데, 다음과 같이 쓸 수 있다.

예제 21. namecard.xml

```
<?xml version="1.0" encoding="euc-kr" ?>
<namecard>
    <person>
        <name>허광남</name>
        <id>남자</id>
        <email>kenu@email.com</email>
        <phone>111-2222-3333</phone>
    </person>
    <person>
        <name>노재춘</name>
        <id>남자</id>
        <email>suribada@email.com</email>
        <phone>222-3333-4444</phone>
    </person>
```

```
    <person>
        <name>이선재</name>
        <id>남자</id>
        <email>hsboy@email.com</email>
        <phone>333-4444-5555</phone>
    </person>
</namecard>
```

예제 22. jstlxml02.jsp

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jstl/xml" prefix="x"%>
<% response.setContentType("text/html;"); %>
<%-- namecard.xml 파일을 불러와 xdata 변수에 할당 --%>
<c:import url="namecard.xml" var="xmldata" />
<x:parse xml="${xmldata}" var="xdata"/>

<%-- XPath 를 이용해서 xdata에서 추출 --%>
<x:out select="$xdata//person[1]/name"/>
… 이하 생략 …
```

이 xdata 변수에 들어간 xml 에서 추출하는 방식은 XPath 를 사용한다고 했다. $select="$xdata//person[1]/name" 에서 person[1] 은 person 으로 사용하는 것과 같다. [] 안에는 순서가 들어가고 생략될 경우 첫 엘리먼트를 찾기 때문이다.
<x:forEache/> 태그로 <person> 엘리먼트 수만큼 반복하게 했다. 이때 자동으로 기준은 <person> 이 되기 때문에 그 이후에 <email> 이나 <phone> 엘리먼트들은 /person/email 과 /person/phone 을 바로 참조하게 된다.

7.8 <x:transform/>, <x:param/>

xml과 xslt 파일을 결합해서 새로운 형식의 문서를 생성해 내는 <x:transform/>의 형식은 다음과 같다.

```
Syntax 1: Body 없는 경우
<x:transform
        xml="XMLDocument" xslt="XSLTStylesheet"
        [xmlSystemId="XMLSystemId"] [xsltSystemId="XSLTSystemId"]
        [{var="varName" [scope="scopeName"]|result="resultObject"}]>
```

```
Syntax 2: 변환 파라메터를 body에서 지정하는 경우
<x:transform
        xml="XMLDocument" xslt="XSLTStylesheet"
        [xmlSystemId="XMLSystemId"] [xsltSystemId="XSLTSystemId"]
        [{var="varName" [scope="scopeName"]|result="resultObject"}]>
    <x:param> 액션들
</x:transform>

Syntax 3: XML문서와 선택적 변환 파라메터들이 body에 지정된 경우
<x:transform
        xslt="XSLTStylesheet"
        xmlSystemId="XMLSystemId" xsltSystemId="XSLTSystemId"
        [{var="varName" [scope="scopeName"]|result="resultObject"}]>
    XML Document
    optional <x:param> actions
</x:parse>
scopeName 은 {page|request|session|application} 중에 하나
```

var 속성에 지정된 결과와 result 속성에 지정된 결과의 차이점은 var 속성은 scope 지정해서 다른 곳에서도 사용할 수 있고, result 는 현재 페이지에서만 사용할 수 있다는 것이다. 또한 타입도 차이가 나는데, var는 org.w3c.dom.Document 로, result는 javax.xml.transform.Result 객체로 저장이 된다.

xml의 파라메터를 지정하는 <x:param/> 태그의 형식은 다음과 같다.

```
Syntax
Syntax 1: value 속성에 파라메터 값이 지정된 경우
<x:param name="name" value="value"/>

Syntax 2: body 내용에 파라메터 값이 지정된 경우
<x:param name="name">
    parameter value
</x:param>
```

여기서 잠깐

<x:transform/> 태그를 사용할 때 jdk1.4 내에 있는 xalan과 jstl이 충돌을 일으킨다. 이런 경우, 톰캣 실행을 중지하고, jstl의 WEB-INF/lib디렉토리에 있는 **xalan.jar** 파일과 **xercesImpl.jar** 파일 두 개를 **<CATALINA_HOME>/common/endorsed** 디렉토리에 복사한다. 이전 버전의 xercesImpl.jar 파일을 덮어씌운다. 만일 이 과정이 생략되면, 다음과 같은 예외를 만나게 된다.

org.apache.xml.utils.WrappedRuntimeException:
The output format must have a '{http://xml.apache.org/xslt}content-handler' property!

<x:transform/> 활용 예제를 보자.

예제 23. jstlxml03.jsp

```
<%@ page pageEncoding="MS949" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jstl/xml" %>

<html>
<head>
  <title>JSTL: XML Support -- Transform</title>
</head>
<body bgcolor="#FFFFFF">
<c:set var="xml">
    <?xml version="1.0" encoding="MS949"?>
    <namecard>
        <person>
            <name>허광남</name>
            <id>남자</id>
            <email>kenu@email.com</email>
            <phone>111-2222-3333</phone>
        </person>
        <person>
            <name>노재춘</name>
            <id>남자</id>
            <email>suribada@email.com</email>
            <phone>222-3333-4444</phone>
        </person>
        <person>
            <name>이선재</name>
            <id>남자</id>
```

```
            <email>hsboy@email.com</email>
            <phone>333-4444-5555</phone>
        </person>
    </namecard>
</c:set>

<c:set var="xsl">
    <?xml version="1.0"?>
    <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
        <xsl:template match="/">
        <table border="1">
            <tr>
                <th>이름  </th>
                <th>이메일</th>
                <th>연락처</th>
            </tr>
        <xsl:for-each select="namecard/person">
            <tr>
                <td><xsl:value-of select="name" /></td>
                <td><xsl:value-of select="email" /></td>
                <td><xsl:value-of select="phone" /></td>
            </tr>
        </xsl:for-each>
        </table>
        </xsl:template>

    </xsl:stylesheet>
</c:set>

<x:transform xml="${xml}" xslt="${xsl}"/>

</body>
</html>
```

jsp 소스 안에 있는 xml(예제 21. namecard.xml) 과 xsl을 따로 파일로 빼놓으면 아래와 같이 바뀐다.

예제 24. namecard.xsl

```
<?xml version="1.0" encoding="euc-kr" ?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:template match="/">
    <table border="1">
        <tr>
            <th>이름  </th>
            <th>이메일</th>
            <th>연락처</th>
        </tr>
    <xsl:for-each select="namecard/person">
        <tr>
            <td><xsl:value-of select="name" /></td>
            <td><xsl:value-of select="email" /></td>
            <td><xsl:value-of select="phone" /></td>
        </tr>
    </xsl:for-each>
    </table>
    </xsl:template>

</xsl:stylesheet>
```

xsl 에 관한 문법적인 내용은 여기에서 다루지 않겠다.

namecard.xsl 파일은 namecard.xml 에 html 을 입히는 역할을 하게 되며, 데이터를 제외한 html 틀을 갖고 있다고 볼 수 있다.

이 두 개의 파일을 하나로 합쳐서 html 코드를 만들어 내는 소스는 다음과 같다.

예제 25. jstlxml04.jsp

```
<%@ page pageEncoding="MS949" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jstl/xml" %>

<html>
<head>
```

```
  <title>JSTL: XML Support -- Transform</title>
</head>
<body bgcolor="#FFFFFF">
<h1>연락처</h1>
<c:import var="xml" url="namecard.xml" charEncoding="MS949"/>
<c:import var="xsl" url="namecard.xsl" charEncoding="MS949"/>

<x:transform xml="${xml}" xslt="${xsl}"/>

</body>
</html>
```

jstlxml03.jsp 에서 <x:set/> 부분은 <c:import/> 로 바뀌었고, 파일을 불러올 때 charEncoding 속성을 통해서 인코딩해 주었다.
xml 변수와 xsl 변수 두 개를 파싱하지 않고 바로 <x:transform/> 을 통해서, jsp 파일에 출력했다.
<x:transform/> 에 var 속성을 주면 출력되지 않고, 변수에 org.w3c.dom.Document 타입으로 저장된다.

이렇게 JSTL 의 4가지 표준 태그에 대해서 알아보았다. 프로그래밍의 기본인 조건, 반복, 연산이 가능하기 때문에 스크립틀릿을 거의 모두 제거할 수 있음을 알 수 있었다.

8 참고자료

JSTL 스펙
http://java.sun.com/products/jsp/jstl/
자카르타 taglibs
http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html
Hans Bergsten article
JSTL 1.0: Standardizing JSP, Part 1
http://www.onjava.com/pub/a/onjava/2002/08/14/jstl1.html
JSTL 1.0: What JSP Applications Need, Part 2
http://www.onjava.com/pub/a/onjava/2002/09/11/jstl2.html
JSTL 1.0: What JSP Applications Need, Part 3
http://www.onjava.com/pub/a/onjava/2002/10/30/jstl3.html
JSTL in Action by Alex Garrett

http://www.codercoop.com/Members/alex_garrett/jstlnewapproach

XPath Tutorial

http://www.zvon.org/xxl/XPathTutorial/General/examples.html

XSL Tutorial

http://www.w3schools.com/xsl/

XSLT Tutorial

http://www.zvon.org/xxl/XSLTutorial/Books/Book1/