09: JSTL (Custom Tag)

학습 목표

- JSTL에 대해 알아보고 JSTL 태그 사용법에 대해 이해한다.
- 커스텀 태그를 만들어 본다
- JSTL 설정을 알아본다

Chapter 09 JSTL (Custom Tag)

1. JSTL 정의

2. JSTL 태그

3. 커스텀 태그

4. JSTL 설정

JSTL >> JSTL (자바 표준 태그 라이브러리)

- EL 이나 표준 액션으로 처리하기 힘든 부분 담당
- 연산, 조건 처리
- 개발자가 따로 구현 가능

JSTL >> JSTL 태그

<c:forEach> : 루핑(for 문) 처리

JSP를 호출할 서블릿 코드

```
...
String[] movieList = {"Amelie", "Return of the King", "Mean Girls"};
request.setAttribute("movieList", movieList);
...
```

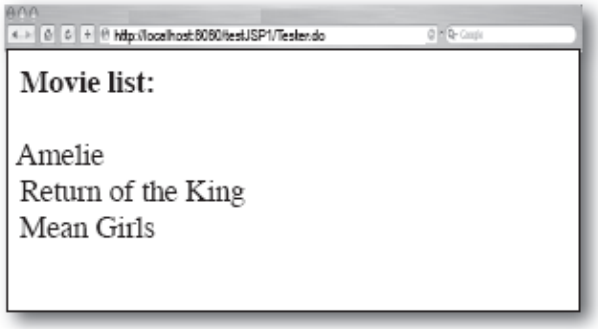
영화 제목을 집어넣기 위해 String[]을 사용합니다. 다음 배열을 request 속성으로 묶어두죠.

(taglib 지시자에 대해선 이 장 후반부에서 알아보겠습니다.)

JSP 코드

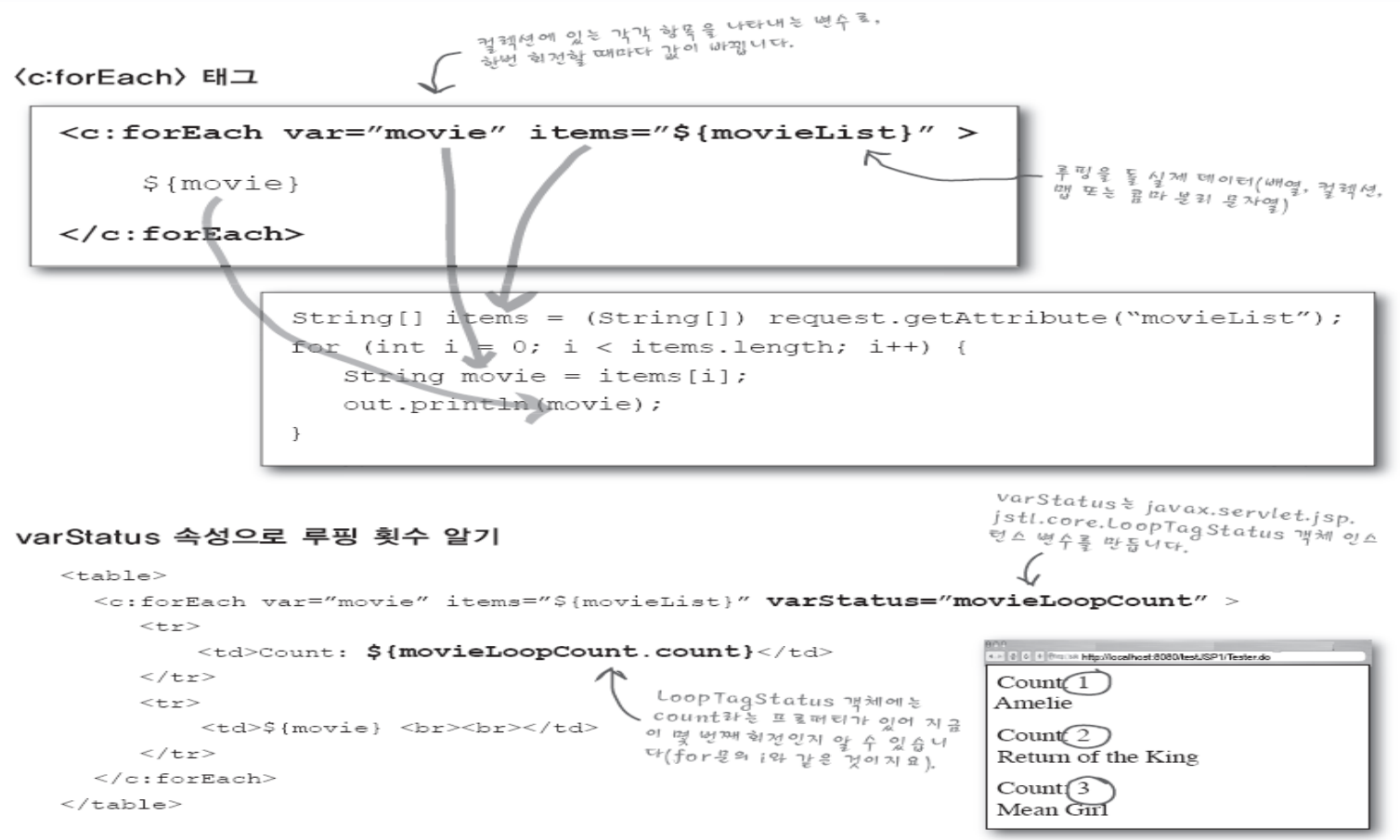
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><body>
<strong> Movie list:</strong>
<br><br>

<table>
  <c:forEach var="movie" items="${movieList}" >
    <tr>
      <td>${movie}</td>
    </tr>
  </c:forEach>
</table>

</body></html>
```

전체 배열(속성 "movieList")을 루핑 돌린 다음, 한 행에 하나씩 출력합니다(출력되는 테이블은 1행 1열 입니다).



JSTL >> JSTL 태그

<c:forEach>



JSTL >> JSTL 태그

<c:forEach> 안에 <c:forEach> 사용

서블릿 코드

```
String[] movies1 = {"Matrix Revolutions", "Kill Bill", "Boondock Saints"};
String[] movies2 = {"Amelie", "Return of the King", "Mean Girls"};
java.util.List movieList = new java.util.ArrayList();
movieList.add(movies1);
movieList.add(movies2);
request.setAttribute("movies", movieList);
```

JSP 코드

```
<table>
  <c:forEach var="listElement" items="${movies}" >
    <c:forEach var="movie" items="${listElement}" >
      <tr>
        <td>${movie}</td>
      </tr>
    </c:forEach>
  </c:forEach>
</table>
```

ArrayList 타입의 request 속성

바깥 루프

안쪽 루프

바깥 루프 "var" 속성에 할당된 String 배열 하나.

첫 번째 String[] 값들

두 번째 String[] 값들

JSTL >> JSTL 태그

■ <c:if> : 조건문(if 문) 처리

JSP 코드

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html><body>
<strong>Member Comments</strong> <br>
<hr>${commentList}<hr>

<c:if test="${userType eq 'member' }" >
   <jsp:include page="inputComments.jsp"/>
</c:if>
</body></html>
```

사용자 로그인 정보로부터 userType이라는 속성을 JSP를 호출할 서블릿에서 설정했다고 가정합시다.

'member' 주위에 단일 인용부호(')를 사용하였습니다. 사실 태그나 EL에선 단일 인용 부호든 이중 인용부호(")든 둘 다 가능합니다.

포함할 페이지("inputComments.jsp")

```
<form action="commentsProcess.jsp" method="post">
Add your comment: <br>
<textarea name="input" cols="40" rows="10"></textarea> <br>
<input name="commentSubmit" type="button" value="Add Comment">
</form>
```

JSTL >> JSTL 태그

■ <c:choose>, <c:when>, <c:others> : 조건문(if else 문) 처리

<c:choose> 태그와 그의 친구 <c:when>, <c:others>를 소개합니다

적어도 여기에 있는 다개 몸체(<c:others> 포함해서) 중 하나는 반드시 실행합니다.

(switch문과 차이점은 중간에 빠져 나가지 못한다는 겁니다)

```
<c:choose>
    <c:when test="${userPref == 'performance'}">
        Now you can stop even if you <em>do</em> drive insanely fast.
    </c:when>

    <c:when test="${userPref == 'safety'}">
        Our brakes will never lock up, no matter how bad a driver you are.
    </c:when>

    <c:when test="${userPref == 'maintenance'}">
        Lost your tech job? No problem--you won't have to service these brakes
for at least three years.
    </c:when>

    <c:otherwise>
        Our brakes are the best.
    </c:otherwise>
</c:choose>
<!-- 페이지 나머지 부분이 밑에 나와야겠죠. -->
```

위에 있는 <c:when> 중 어느 하나에도 해당하지 않을 때, <c:others>가 디폴트로 실행됩니다.

노트: <c:choose> 태그에 <c:others> 태그가 반드시 있어야 하는 것은 아닙니다.

JSTL >> JSTL 태그

<c:set> : 값 설정

〈c:set〉 속성 var 설정하기

① 몸체가 없는 버전

session 생존범위에 "userLevel"이란 이름의 속성이 없으면, 〈c:set〉 태그는 하나를 새로 만듭니다. (value 속성값이 널(null)이 아니란 가정 하에)

```
<c:set var="userLevel" scope="session" value="Cowboy" />
```

var 속성은 필수항목이지만, scope 속성은 옵션입니다. 값(value)은 명기되어야 하지만 방식은 value 속성에 해도 되고, 태그 몸체에 기술해도 문제없습니다(아래 예제처럼).

value가 꼭 문자열일 필요는 없습니다.

```
<c:set var="Fido" value="${person.dog}" />
```

${person.dog}의 Dog 객체를 Fido 변수에 설정합니다. Fido는 Dog 타입 변수가 되는 거죠.

② 몸체가 있는 버전

몸체가 있기 때문에 /가 없습니다.

```
<c:set var="userLevel" scope="session" >
     Sheriff, Bartender, Cowgirl
</c:set>
```

몸체에 있는 값이 변수값이 됩니다.

JSTL >> JSTL 태그

■ <c:set> : 빈과 맵에 사용

〈c:set〉 태그의 target, property, value 설정하기

① 몸체가 없는 경우

```
<c:set target="${PetMap}" property="dogName" value="Clover" />
```

target이 빈인 경우, "dogName"은 프로퍼티 이름이 됩니다.

target은 널이어서는 안 됩니다!!

target이 빈인 경우, "dogName"은 키 이름이 됩니다.

② 몸체가 있는 경우

```
<c:set target="${person}" property="name" >
  ${foo.name}
</c:set>
```

target 속성에 "id" 이름을 기입하면 안됩니다.

/는 없습니다. 시험에 잘 나옵니다.

몸체 안에는 문자열 또는 표현식이 들어갈 수 있습니다.

JSTL >> JSTL 태그

<c:set> : 정리

- <c:set>태그에 var 와 target을 동시에 사용할 수 없다.
- scope는 옵션. 업는 경우 page 생존범위가 기본
- value가 널이면, var에 있는 속성은 제거
- var 이름으로 속성이 없으면, 자동으로 생성. 단, value가 널이 아닌 경우
- target 표현식이 널이면, 컨테이너는 Exception 을 던진다.
- target에는 실제 객체를 표현하는 표현식이 들어가야 한다. target은 빈이나 맵의 속성명이 아니라 속성 객체가 들어가야 한다.
- target 표현식이 빈이나 맵이 아니면 컨테이너는 Exception을 던진다.

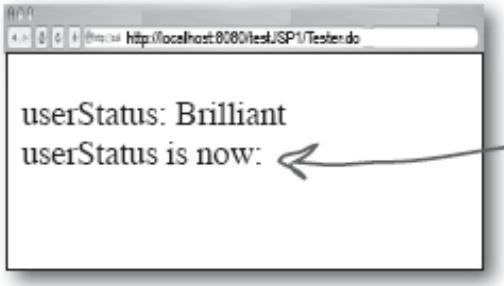
JSTL >> JSTL 태그

<c:remove> : 삭제

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

   <c:set var="userStatus" scope="request" value="Brilliant" />

   userStatus: ${userStatus} <br>

   <c:remove var="userStatus" scope="request" />

   userStatus is now: ${userStatus}

</body></html>
```

var 속성에는 문자열이 들어갑니다. 여기다가 표현식을 넣으면 안됩니다.

scope은 옵션 사항입니다. 언제나처럼 page가 기본값이죠.



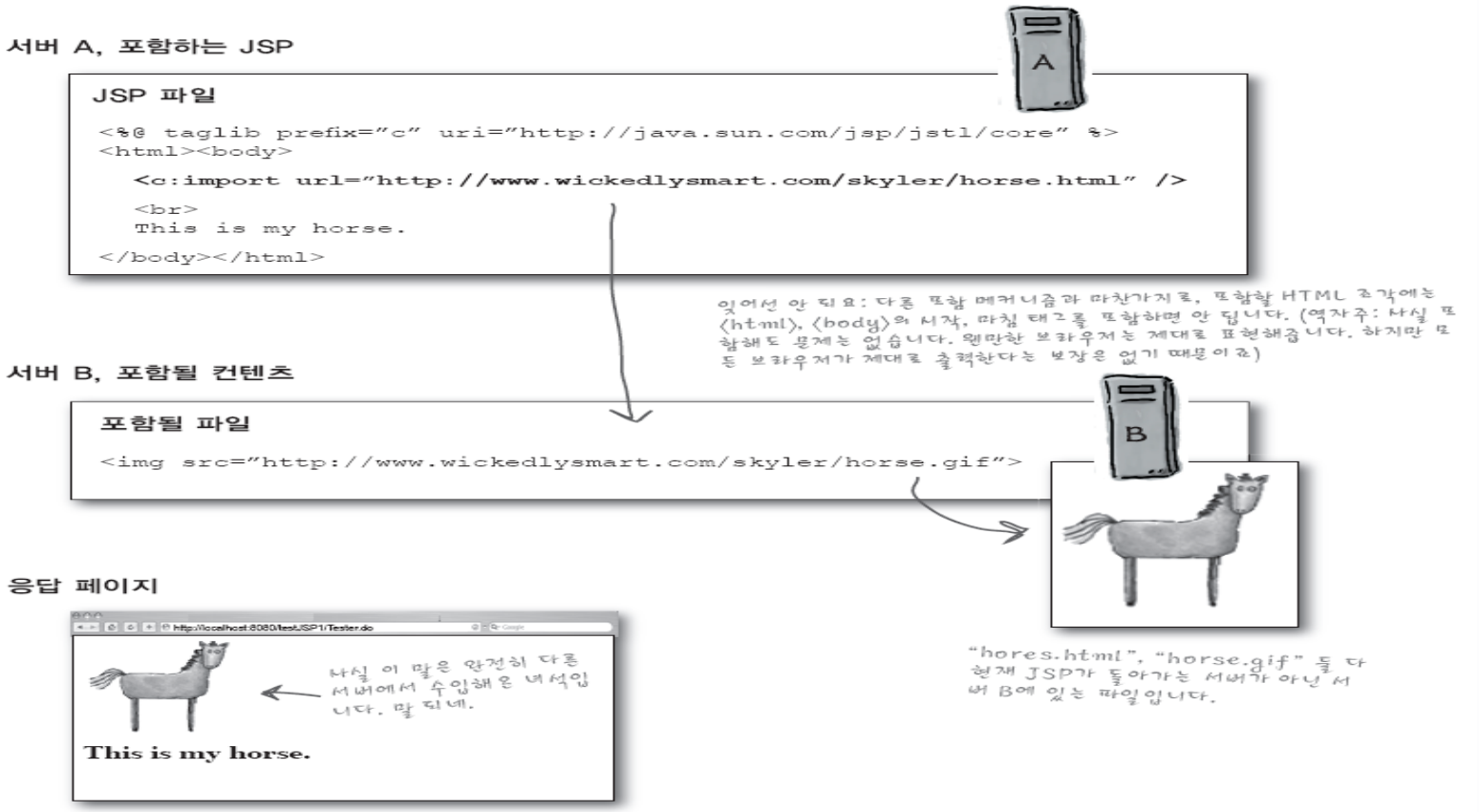
userStatus 값*이 제거되었습니다. (c:remove) 태그를 사용한 다음에 나오는 EL 표현식은 아무 것도 출력하지 않습니다.

*역자주: 여기선 값이라고 했는데 값이 아닌 변수 자체가 제거됩니다. 스펙을 보면 PageContext.removeAttribute(varName) 메소드를 호출하여 제거한다고 되어 있습니다.

JSTL >> JSTL 태그

<c:import> : 다른 컨텐츠 포함

- inclue 지시자와 include 표준 액션과 달리 웹 컨테이너 외부 자원도 포함가능



JSTL >> JSTL 태그

■ <c:import> : 파라미터 전달

① 〈jsp:include〉 태그가 있는 JSP

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

<c:import url="Header.jsp" >

   <c:param name="subTitle" value="We take the sting out of SOAP." />

</c:import>

<br>
<em>Welcome to our Web Services Support Group.</em> <br><br>

Contact us at: ${initParam.mainEmail}

</body></html>
```

/가 없습니다. 왜냐고요 아래에 몸체가 있으니까...

② 포함될 파일("Header.jsp")

```
<img src="images/Web-Services.jpg" > <br>

<em><strong>${param.subTitle}</strong></em>
<br>
```

이 페이지는 바짝게 없습니다. 파라미터를 소개목을 받기만 하면 되니까, 자신을 어떻게 호출하든지 관심이 없죠.

JSTL >> JSTL 태그

<c:url> : URL 재작성

JSP에서 URL 재작성

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

This is a hyperlink with URL rewriting enabled.


<a href="<c:url value='/inputComments.jsp' />">Click here</a>


</body></html>
```

"value"에 들어있는 상대 경로 뒤에 jsessionid를 추가합니다(쿠키를 사용 못하는 경우).

JSTL >> JSTL 태그

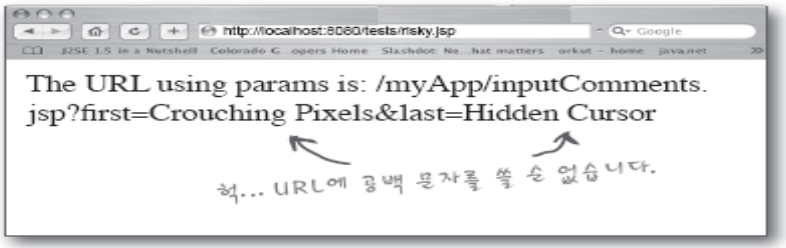
<c:url>

질의어를 가진 〈c:url〉 사용하기

〈c:url〉 태그는 URL 재작성은 하지만, URL 인코딩을 하지 않는다는 사실 명심하세요.

```
<c:set var="last" value="Hidden Cursor" />
<c:set var="first" value="Crouching Pixels"/>

<c:url value="/inputComments.jsp?first=${first}&last=${last}" var="inputURL" />

The URL using params is: ${inputURL} <br>
```

옵션 속성인 "var"는 나중에 이 URL을 참조하기 위해 씁니다.



짭~ 질의어로 넘어가는 파라미터는 인코딩되어야만 한다는군요. 공백 문자는 "+"로 변환한 다음 넘겨야 된다나 뭐라나...

〈c:url〉 몸체에 〈c:param〉 사용하기

이제 됐습니다. URL 재작성도 하고 URL 인코딩도 됩니다.

```
<c:url value="/inputComments.jsp" var="inputURL">
   <c:param name="firstName" value="${first}" />
   <c:param name="lastName" value="${last}" />
</c:url>
```

/ 없음

이제 URL이 이렇게 바뀌었습니다:

```
/myApp/inputComments.jsp?firstName=Crouching+Pixels&lastName=Hidden+Cursor
```

이제 안심해도 됩니다. 〈c:param〉이 알아서 인코딩 해주니까요.

JSTL >> JSTL 태그

■ 오류 페이지 처리 : web.xml

모두 다 적용되는(디폴트) 오류 페이지 선언

아래와 같이 선언하면 JSP 뿐만 아니라 웹 애플리케이션에 있는 모든 페이지에 적용됩니다. 개별 JSP에서 page 지시자 errorPage 속성을 명시하면 이 값을 재정의할 수 있습니다.

```
<error-page>
   <exception-type>java.lang.Throwable</exception-type>
   <location>/errorPage.jsp</location>
</error-page>
```

특정 예외사항에 대한 오류 페이지 선언

아래 설정은 ArithmetricException이 발생하는 경우에만 해당 오류 페이지를 보여주도록 설정하는 부분입니다. 바로 위 선언과 이 선언이 동시에 있는 경우, ArithmetricException 이외의 예외사항이 발생하면 errorPage.jsp가 호출됩니다.

```
<error-page>
   <exception-type>java.lang.ArithmeticException</exception-type>
   <location>/arithmeticError.jsp</location>
</error-page>
```

HTTP 상태 코드에 따라 다르게 오류 페이지 선언

아래 설정은 응답 상태 코드가 "404" (File Not Found)일 경우에만 지정된 오류 페이지가 호출되도록 한 것입니다.

```
<error-page>
   <error-code>404</error-code>
   <location>/notFoundError.jsp</location>
</error-page>
```

<location> 태그의 경로는 웹 애플리케이션 루트/컨텍스트에서부터 시작해야 합니다. 즉 반드시 처음이 /로 시작해야 한다는 말입니다(오류 페이지가 <error-code>이든 <exception-type>이든 관계없이 /로 시작해야 합니다).

JSTL >> JSTL 태그

■ 오류 페이지 처리 : exception 객체 사용

명시적으로 오류 페이지임을 나타낸 페이지("errorPage.jsp")

```
<%@ page  isErrorPage="true" %>

<html><body>
<strong>Bummer.</strong><br>

You caused a ${pageContext.exception} on the server.<br>

<img src="images/bummerGuy.jpg">
</body></html>
```

내장 객체 exception은 page 지시자에 명시적으로 오류 페이지라고 표기한 아래 문구가 있을 때만 사용할 수 있습니다.

<% page isErrorPage=" true" %>

DD 파일에서 오류 페이지라고 설정했다고 해서, 컨테이너가 자동으로 이 페이지가 exception 객체를 사용할 수 있는 능력을 주지는 않는다는 말이지요.

JSTL >> JSTL 태그

<c:catch> : try/catch 문 처리

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

About to do a risky thing: <br>

<c:catch>

    <% int x = 10/0; %>

</c:catch>

If you see this, we survived.

</body></html>
```

이 스크립틀릿은 100% 예외사항을 던집니다. 하지만 오류 페이지가 아닌 이 페이지에서 이 예외을 처리합니다.

이 문구가 출력된다면, 예외사항을 성공적으로 잡은 것이겠죠. 즉 오류 페이지로 빠지지 않았다는 걸 의미하지요.

JSTL >> JSTL 태그

<c:catch> : Exception을 속성으로 사용

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

About to do a risky thing: <br>

<c:catch var="myException">

    Inside the catch...
    <% int x = 10/0; %>
</c:catch>

<c:if test="${myException != null}">
   There was an exception:  ${myException.message} <br>
</c:if>

We survived.
</body></html>
```

이 태그는 "myException"이란 이름의 page 생존범위 속성을 하나 만듭니다. 예외가 발생하면 여기에 exception 객체를 설정하겠죠.

이제 여기엔 myException이란 객체가 있습니다. 당연히 타입이 Throwable이니 "message" 프로퍼티가 있을 겁니다 (Throwable에는 getMessage() 메소드가 있기 때문이죠).

JSTL >> JSTL 태그

■ 그 외 태그

코어(Core) 라이브러리

일반적인 것

<c:out>
<c:set>
<c:remove>
<c:catch>

조건

<c:if>
<c:choose>
<c:when>
<c:otherwise>

URL 관련

<c:import>
<c:url>
<c:redirect>
<c:param>

반복

<c:forEach>
<c:forEachToken>

포맷팅 라이브러리

국제화

<fmt:message>
<fmt:setLocale>
<fmt:bundle>
<fmt:setBundle>
<fmt:param>
<fmt:requestEncoding>

포맷팅

<fmt:timeZone>
<fmt:setTimeZone>
<fmt:formatNumber>
<fmt:parseNumber>
<fmt:parseDate>

SQL 라이브러리

데이터베이스 접근

<sql:query>
<sql:update>
<sql:setDataSource>
<sql:param>
<sql:dateParam>

XML 라이브러리

코어 XML 액션

<x:parse>
<x:out>
<x:set>

XML 흐름 제어

<x:if>
<x:choose>
<x:when>
<x:otherwise>
<x:forEach>

변환 액션

<x:transform>
<x:param>

JSTL >> 사용자 JSTL (Custom JSTL)

TLD

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
version="2.0">
    <tlib-version>0.9</tlib-version>
    <short-name>RandomTags</short-name>
    <function>
        <name>rollIt</name>
        <function-class>foo.DiceRoller</function-class>
        <function-signature>int rollDice()</function-signature>
    </function>
    <uri>randomThings</uri>
    <tag>
        <description>random advice</description>
        <name>advice</name>
        <tag-class>foo.AdvisorTagHandler</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>user</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

JSP 2.0에서 쓸 XML 스키마 버전을 의미합니다. 외울 필요는 없습니다. 필요할 때 복사해서 쓰면 됩니다.

필수 항목(태그가 필수란 뜻, 값이 아니라) – 태그 라이브러리 버전을 선언하는 곳

필수 항목: 가장 중요하게 사용할 문구입니다.

앞 장에서 사용했던 EL 함수.

taglib 지시자에서 사용할 자신을 나타내는 유일한 이름!

옵션 항목, 하지만 이렇게 써 놓아야 여럿에게 도움이 된다.

태그 안에서 쓸 이름(예: <my:advice>)

필수 항목: JSP에서 태그를 사용한 경우 컨테이너가 호출할 클래스.

필수 항목: 이 뜻은 이 태그는 몸체가 없다는 말.

태그 속성이 있다면 속성을 정의하는 곳. 속성이 하나면 <attribute> 항목도 하나.

태그 안에 "user" 속성을 반드시 명기해야 한다는 말.

무슨 말이냐 하면 "user" 속성에는 런타임 표현식 값(run time expression value)이 들어간다는 말(문자열이 아니라).

JSTL >> 커스텀 태그

■ 커스텀 태그 작성

TLD 에서 advice 태그 부분

```
<taglib ...>
...
<uri>randomThings</uri>
<tag>
    <description>random advice</description>
    <name>advice</name>
    <tag-class>foo.AdvisorTagHandler</tag-class>
    <body-content>empty</body-content>

   <attribute>
      <name>user</name>
      <required>true</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>

</tag>
</taglib ...>
```

앞 페이지에서 본 것과 동일한 내용입니다. 주석만 빼고.

태그를 사용하는 JSP

```
<html><body>

<%@ taglib prefix="mine" uri="randomThings"%>

Advisor Page<br>

<mine:advice user="${userName}" />

</body></html>
```

여기 있는 uri하고 TLD에 있는 <uri>하고 맞아야 합니다.

여기서 EL 쓰는 것 맞습니다. TLD에서 user 속성의 <rtexprevalue> 태그가 참(true)이었기 때문에 EL 쓰는 것 맞습니다. ("userName"은 앞에서 정의되었다고 하고 넘어갑시다)

여기 있는 /의 의미. TLD에서 몸체가 없는 태그라고 정의했기 때문이죠. empty라는 부분 보이죠.

JSTL >> 커스텀 태그

■ 커스텀 태그 핸들러

```
package foo;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;
import java.io.IOException;

public class AdvisorTagHandler extends SimpleTagSupport {

    private String user;

    public void doTag() throws JspException, IOException {
       getJspContext().getOut().write( "Hello " + user + " <br>" );
       getJspContext().getOut().write( "Your advice is: " + getAdvice() );
    }

    public void setUser(String user) {
       this.user=user;
    }

    String getAdvice() {
       String[] adviceStrings = {"That color's not working for you.",
              "You should call in sick.", "You might want to rethink that haircut."};
       int random = (int) (Math.random() * adviceStrings.length);
       return adviceStrings[random];
    }
}
```

SimpleTagSupport 클래스에 커스텀 태그에 필요한 모든 것들이 이미 구현되어 있습니다.

JSP가 태그를 실행하면, 컨테이너는 doTag() 메소드를 호출합니다.

태그 속성값을 설정하기 위하여 컨테이너가 호출하는 메소드입니다. "user" 속성은 setUser() 메소드를 호출하는 것을 보면 알겠지만, 자바 빈 프로퍼티 명명 규칙을 사용합니다.

내부 필요에 의해 사용하는 메소드

JSTL >> 커스텀 태그

<rtexprvalue> : 속성 값이 런타임 시 또는 실행 시 실행 여부 표시

아래처럼 정의되어 있다면:

```
<attribute>
    <name>rate</name>
    <required>true</required>
    <rtexprvalue>false</rtexprvalue>
</attribute>
```

아니면:

```
<attribute>
    <name>rate</name>
    <required>true</required>
</attribute>
```

(rtexprevalue) 태그가 없으면 디폴트로 'false' 입니다.

그러면 아래 코드는 작동하지 않습니다.

```
<html><body>
  <%@ taglib prefix="my" uri="myTags"%>

  <my:handleIt rate="${currentRate}" />
</body></html>
```

NO! 여기에 표현식을 쓸 수 없습니다. 문자열(String)만 가능합니다.

JSTL >> 커스텀 태그

■ 태그 몸체

- 하나의 시작 태그와 끝 태그 사이에 들어갈 수 있는 것
- 태그 몸체에 들어갈 수 있는 종류

```
<body-content>empty</body-content>
<body-content>scriptless</body-content>
<body-content>tagdependent</body-content>
<body-content>JSP</body-content>
```

몸체를 가질 수 없습니다.

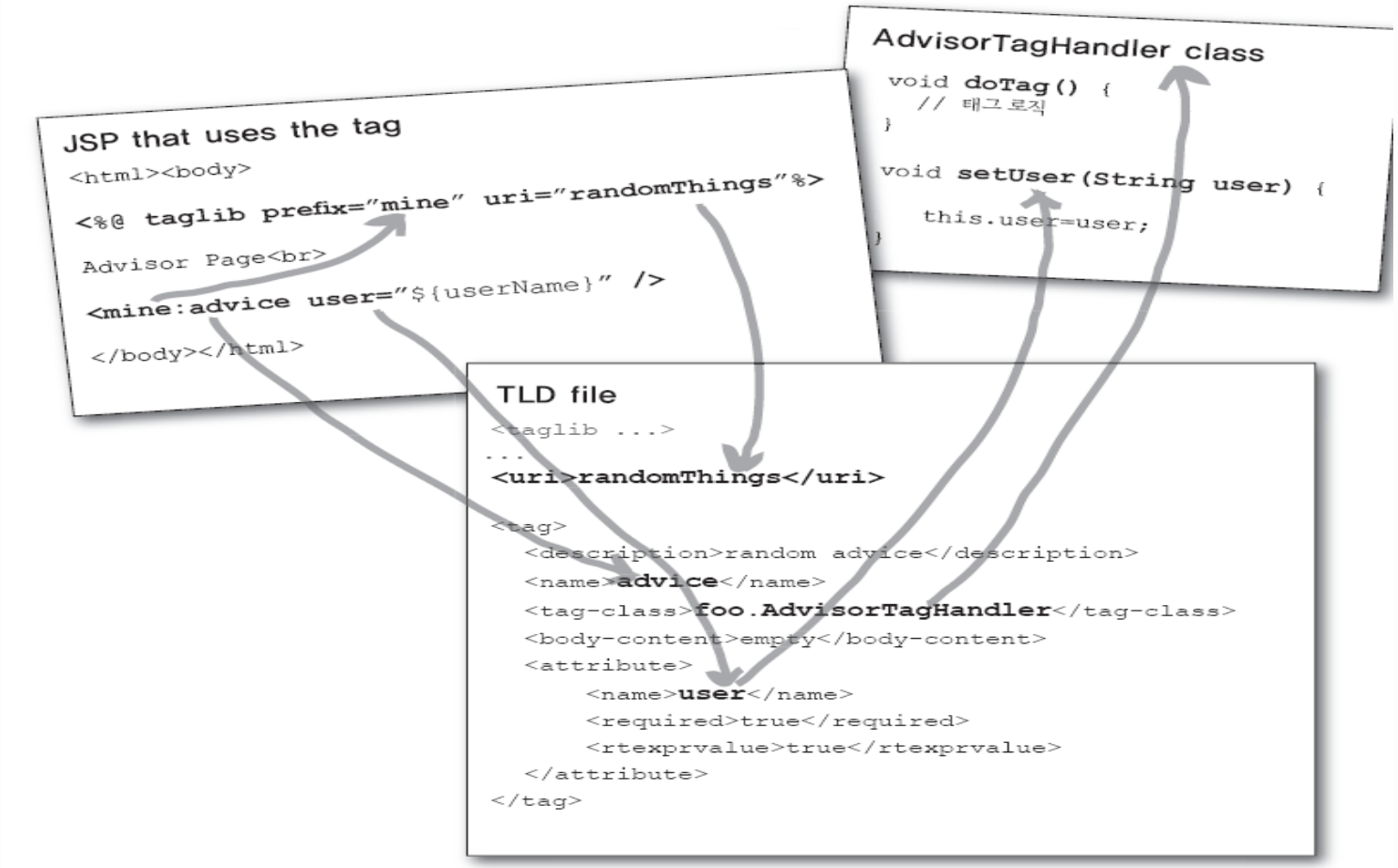
스크립팅(스크립틀릿, 스크립팅 표현식, 선언문)은 올 수 없다는 말입니다. 하지만 탬플릿 텍스트, EL, 커스텀 태그, 표준 액션은 가능하지요.

태그 몸체를 그냥 텍스트로 취급합니다. 따라서 EL을 실행한 값이 아닌 글자 그대로 취급하고, 태그/액션도 마찬가지로 텍스트로 취급합니다.

JSP 안에 들어 갈 수 있는 것이라면 무엇이든지 가능하다는 의미입니다.

JSTL >> 커스텀 태그

■ 태그 핸들러, TLD, JSP 의 관계



JSTL >> JSTL 설정

taglib 지시자와 TLD 파일 매핑

- **JSP 2.0 이전 : DD (web.xml)**에 매핑 정보 추가

```
<web-app>
...
  <jsp-config>
    <taglib>
      <taglib-uri>randomThings</taglib-uri>
      <taglib-location>/WEB-INF/myFunctions.tld</taglib-location>
    </taglib>
  </jsp-config>
</web-app>
```

DD에서 TLD에 있는 <uri>와 TLD 파일의 실제 경로를 매핑합니다.

- **JSP 2.0 : DD (web.xml)**에 매핑정보 필요없음. 컨테이너에서 자동 처리

JSTL >> JSTL 설정

■ TLD 위치