

JAVA SERVER FACES MANUAL

2004. 6

김승규

<http://www.java-inside.co.kr>

e-mail : zzzccc90@yahoo.co.kr

JAVA SERVER FACES MANUAL	1
1. JAVA SERVER FACES개요	7
2.1 Tomcat의 환경설정	9
2.2 톰캣에서의 한글	10
2.3 JSF를 위한 환경 구성	11
2.4 Ant	12
2.5 JSF에서의 한글	15
3. JSF로 구현한 간단한 예제	16
3.1 간단한 예제	16
3.2 rendered 속성	21
3.3 컴포넌트의 바인딩	23
3.4 이벤트	25
4장. JSF의 구성요소	29
4.1 컴포넌트	30
4.3 managed bean	35
4.4 렌더러	42
4.5 이벤트	43
4.6 validator & converter	45
4.6.1 Validator	45
4.6.2 Converter	46

4.7 navigation	47
5. JSF LIFE CYCLE	51
5.1 JSF Life Cycle	51
5.2 immediate 속성	53
6. JSF 컴포넌트	63
6.1 Html 컴포넌트	63
6.2 HtmlCommandButton	63
6.3 HtmlCommandLink	63
6.4 HtmlForm	65
6.5 HtmlGraphicImage	65
6.6 HtmlInputHidden	65
6.7 HtmlInputSecret	66
6.8 HtmlInputTextarea	66
6.9 HtmlInputText	67
6.10 HtmlMessage	67
6.11 HtmlMessages	70
6.12 HtmlOutputFormat	71
6.13 HtmlOutputLabel	72
6.14 HtmlOutputLink	73
6.15 HtmlOutputText	74
6.16 HtmlPanelGrid	74
6.17 HtmlPanelGroup	77

6.18 HtmlDataTable 컴포넌트	79
7. CORE 컴포넌트	90
7.1 <f:actionListener>	90
7.2 <f:attribute>	91
7.3 <f:convertDateTime>	94
7.4 <f:convertNumber>	98
7.5 <f:converter>	100
7.6 <f:facet>	101
7.7 <f:loadBundle>	101
7.8 <f:param>	101
7.9 <f:selectItem> , <f:selectItems>	102
7.10 <f:subview>	112
7.11 <f:validateDoubleRange> , <f:validateLength> , <f:validateLongRange>	113
7.12 <f:validator>	114
7.13 <f:valueChangeListener>	114
7.14 <f:view>	115
8. VALIDATOR, CONVERTER를 만들어 보자.	116
8.1 Validator 만들기	116
8.2 Converter	124
9. CUSTOM COMPONENT 구현 -1	130
9.1 태그 클래스 작성	131

9.2 TLD 파일의 작성	133
9.3 컴포넌트 클래스	134
9.4 렌더러	134
9.5 bean	136
9.6 faces-config.xml	139
9.7 테스트	140
10. CUSTOM COMPONENT 2	142
10.1 ImageListComponent의 구현	143
10.1.1 태그 클래스	143
10.1.2 TLD파일의 작성	147
10.1.3 컴포넌트 클래스	148
10.1.4 bean클래스 작성	152
10.2 ImageDisplayComponent	154
10.2.1 태그 클래스	154
10.2.2 TLD 파일의 작성	155
10.2.3 컴포넌트 클래스	156
10.3 faces-config.xml	159
10.4 테스트	160

* 상업적인 용도로 이용하실 수 없습니다.

1. Java Server Faces개요

JSF를 굳이 정의 하자면 “컴포넌트를 이용한 웹 어플리케이션의 개발”이라고 할 수 있겠다. 웹 어플리케이션의 개발에 필요한 여러가지 html요소들을 컴포넌트화 하고 바인딩을 통해 서버사이드에서 만들어진 컴포넌트들을 웹 브라우저에 표시함으로써, 기존의 웹 페이지에서만 가능했던 클라이언트 환경의 제어를 서버측에서 가능하게 한다. 물론 기존에 사용하던 자바스크립트도 사용이 가능하다. 또, JSF는 렌더러라는 개념을 도입하여 변화하는 클라이언트의 환경에 대응할 수 있도록 하고 있다. 무슨 뜻인가 하면, 클라이언트의 환경에 웹 브라우저가 아니라 XML, WML, SVG 혹은 핸드폰을 위해 사용할 수도 있다는 뜻이 되겠다. 만들어진 컴포넌트의 렌더러만 변경함으로써, 컴포넌트를 클라이언트의 환경에 맞는 작업을 수행하도록 하는 것이 가능하다.

JSF는 다음의 기술에 기반을 두고 있다.

- 커스텀 태그
- JSP2.0의 EL(Expression Language)
- jakarta commons project

JSF는 컴포넌트 기반 개발을 위해 다음의 기능들을 지원한다.

- 이벤트처리
- Validator & Converter
- Localization
- Navigation

JSF는 클라이언트에 의해 submit이 발생하는 경우, 입력되는 값이 변경되었을 경우 서버사이드에서 지정된 이벤트를 처리할수 있다.

또, 값의 유효함을 검사하기 위한 Validator, 값을 적절한 타입으로 변경하기 위한 Converter, 페이지의 이동을 관리하는 navigation기능들을 지원한다.

JSF는 jakarta struts와 유사한 면이 많다. 아마도 jakarta struts를 개발한 Craig McClanahan이 JSF의 개발에 참여하고 있기 때문일 것이다.

컴포넌트를 이용한 웹 어플리케이션의 개발이 가능해짐으로서 이미 JSF를 이용하여 비주요한 환경에서 웹 어플리케이션을 개발할 수 있는 툴이 나와 있다. 또, 많은 개발자 사이트에서 JSF를 이용하여 유용한 컴포넌트들이 개발되고 있다. 마지막 장에서 Open Source로 개발되고 있는 MyFaces를 살펴볼까 한다.

2. JSF를 공부하기 위한 환경

이 책의 예제들을 테스트 하기 위해서는 다음의 환경이 필요하다.

- JDK1.4 이상
- Jakarta Tomcat 5.0 - Servlet2.3/JSP1.2이상의 환경이 요구된다.
- JSF-1.0 SCSL
- Jakarta-ant 1.5
- MySql 4.0

JSF는 jwsdp(Java Web Service Developer Pack)에 기본적으로 포함되어 있다. 그러나, 이 책을 쓰는 현재 Sun에서 제공하는 jwsdp에는 JSF1.0EA4가 들어있다. JSF1.0EA4는 이책에서 사용하는 예제가 작동하지 않으므로 주의하기 바란다. EA4 까지의 JSF와 이후의 JSF는 태그의 이름이 다르고, 몇몇 클래스도 변경이 되었기 때문이다.

JSF의 최신버전은 <http://www.sun.com/software/communitysource/jsf/index.html>에서 받을 수 있다. 다운받은 소스를 직접 build하여 사용하거나, 혹은 이 책의 예제파일을 다운받아 WEB-INF/lib에 들어있는 jar파일을 이용하기 바란다.

JSF의 최신버전을 다운받아서 압축을 풀면 다음과 같은 4개의 폴더가 생긴다.

- jsf-api
- jsf-demo
- jsf-ri
- jsf-tools

각각의 폴더안에 있는 build.xml을 수행하면 각 폴더에 build폴더가 생기고, 그 안에 API문서와 jar파일이 생길 것이다. 각 폴더의 build.xml파일은 jwsdp의 jar파일들을 참조하도록 설정되어 있어서 build를 할 때 어려움이 있다. 제일 편한 방법은 jwsdp의 최신버전을 다운받고, 각 폴더의 build.properties 파일의 tomcat.home 값을 각자의 환경에 맞는 jwsdp경로명을 지정해주면 build가 보다 쉽게 된다.

```
...  
tomcat.home= jwsdp 폴더경로  
...
```

다음의 jsr 파일들이 이 책의 예제를 테스트 하는데 필요하다.

- commons-beanutils.jar
- commons-collections.jar
- commons-digester.jar
- commons-logging.jar
- jsf-api.jar
- jsf-impl.jar
- jstl.jar
- standard.jar
- mysql-connector-java-3.0.11-stable-bin.jar

jakarta-commons 프로젝트의 라이브러리와 jstl(Java Standard Tag Library)을 기본으로 사용하고, jsf에서 구현한 jsf-api.jar와 jsf-impl.jar가 들어 있다.

또, 데이터베이스로는 mySql을 사용하였으므로, mySql의 JDBC를 구하도록 한다.

2.1 Tomcat의 환경설정

테스트를 위한 웹서버로 Jakarta-tomcat-5.0.19를 이용한다.

Ant를 이용한 빌드를 용이하게 하기 위해 톰캣의 manager를 이용한다.

톰캣의 manager는 웹 어플리케이션의 배치를 위해 톰캣을 종료할 필요가 없이 동적인 업데이트를 가능하게 한다. 아직 불완전하지만, 테스트를 위해서는 쓸만한 환경을 제공한다. 우선, TOMCAT_HOME/conf/tomcat-users.xml파일을 열어서 아래의 내용을 추가한다.

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

위와 같이 manager를 위한 role을 추가하면 된다. reload를 계속하다보면 가끔 OutOfMemoryError를 일으키는 경우가 발생한다. 그때는 톰캣을 shutdown하고 다시 구동하도록 한다. 테스트하는데 많은 지장을 주는 부분은 아니므로 톰캣만 재시작 하면 문제가 없다.

2.2 톰캣에서의 한글

언제나 새로이 테스트 환경을 구축할 때 직면하는 문제가 한글을 표현하는 부분일 것이다. 다행히 최근에는 톰캣에서 제공하는 filter기능을 이용하여 한글처리문제를 해결할 수 있다.

TOMCAT_HOME/conf/web.xml의 파일을 아래와 같이 수정한다.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <filter>
        <filter-name>Set Character Encoding</filter-name>
        <filter-class>jsf.proj.filter.SetCharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>EUC_KR</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>Set Character Encoding</filter-name>
        <url-pattern>/faces/*</url-pattern>
    </filter-mapping>
</web-app>
```

SetCharacterEncodingFilter 클래스는 톰캣의 다음 경로에 위치한다.

TOMCAT_HOME/webapps/servlet-examples/WEB-INF/classes/filters

여기에 소스파일과 컴파일된 클래스가 같이 있으므로 골라서 사용하면 되겠다. 이 책에서는 패키지명만 바꾸어 사용하였다.

2.3 JSF를 위한 환경 구성

JSF를 이용하기 위해 필요한 설정을 web.xml 파일에 추가해보자.

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

웹 어플리케이션을 위한 FacesServlet 클래스를 설정한다. 그리고, /faces가 포함된 url을 처리할 때 FacesServlet이 구동될 수 있도록 한다.

```
<context-param>
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
    <param-value>client</param-value>
</context-param>
```

context를 위해 파라미터로 주어지는 STATE_SAVING_METHOD는 웹 페이지를 구성하는 JSF 컴포넌트 클래스들의 직렬화된 정보를 어디에 저장할 지는 결정하는 값이다. 위의 예에서처럼 클라이언트를 선택하면, 웹 페이지에 hidden 타입으로 직렬화된 값들이 저장된다. 서버를 선택하면, 서버측에서 정보를 관리한다. 디폴트 값은 server이다.

```
<context-param>
    <param-name>com.sun.faces.validateXml</param-name>
    <param-value>>true</param-value>
</context-param>
```

faces-config.xml의 xml 형식이 유효한가를 검사하는 옵션이다.

```
<context-param>
    <param-name>javax.faces.application.CONFIG_FILES</param-name>
```

```
<param-value>/WEB-INF/faces-config.xml</param-value>
</context-param>
```

웹 어플리케이션을 위한 환경설정 파일인 faces-config.xml의 위치를 지정한다.

생략하면, /WEB-INF/faces-config.xml을 찾게 될 것이다.

컴포넌트를 만들어 배포하는 경우, jar파일에 생기는 /META-INF에 faces-config.xml 파일을 두면, 컴포넌트를 위한 설정파일로 사용될 수 있다.

2.4 Ant

ant를 이용해서 웹 어플리케이션의 자바코드 컴파일 및 WAR파일의 배치를 편리하게 수행할 수 있다. 이 책의 예제를 테스트 해보기 위해 다음과 같은 build.xml을 이용한다.

```
<?xml version="1.0" encoding="euc-kr"?>

<project name="JSF_TEST" basedir="." default="reload">

    <property name="project-name" value="jsf-proj"/>
    <property name="src" value="."/>
    <property name="jsp" value="./jsp"/>
    <property name="lib" value="./lib"/>
    <property name="dest" value="c:/temp/dest"/>
    <property name="TOMCAT_HOME" value="c:/jakarta-tomcat-5.0.19"/>
    <property name="class-dest" value="${dest}/WEB-INF/classes"/>
    <property name="jsp-dest" value="${dest}/jsp"/>
    <property name="img-dest" value="${dest}/jsp/images"/>
    <property name="lib-dest" value="${dest}/WEB-INF/lib"/>
    <property name="config-dest" value="${dest}/WEB-INF"/>
    <property name="deploy-war" value="${TOMCAT_HOME}/webapps"/>
    <property name="lib-common" value="${TOMCAT_HOME}/common/lib"/>
    <property name="lib-server" value="${TOMCAT_HOME}/server/lib"/>
    <property name="manager-url" value="http://localhost:8080/manager"/>
    <property name="manager-id" value="zzzccc"/>
    <property name="manager-pass" value="zzzccc"/>
```

```

<property name="webapp-path" value="/${project-name}"/>

<path id="classpath">
  <fileset dir="${lib-common}">
    <include name="**/*.jar"/>
  </fileset>
  <fileset dir="${lib-server}">
    <include name="**/*.jar"/>
  </fileset>
  <fileset dir="${lib}">
    <include name="**/*.jar"/>
  </fileset>
</path>

<!-- DEFINE RELOAD TASK ON TOMCAT -->
<!-- ===== -->
<taskdef name="reload" classname="org.apache.catalina.ant.ReloadTask"
classpathref="classpath"/>

<!-- COMPILE PROJECT -->
<!-- ===== -->
<target name="compile">
  <mkdir dir="${class-dest}"/>
  <mkdir dir="${lib-dest}"/>
  <mkdir dir="${jsp-dest}"/>
  <mkdir dir="${img-dest}"/>
  <javac source="1.4" destdir="${class-dest}">
    <classpath refid="classpath"/>
    <src path="${src}"/>
  </javac>

  <native2ascii encoding="EUC-KR" src="${src}" dest="${class-dest}"
includes="**/*.properties" ext="_ko.properties"/>
</target>

```

```

<!-- COPY APP -->

<target name="copy-src" depends="compile">
    <copy todir="${jsp-dest}">
        <fileset dir="${jsp}">
            <include name="**/*.*/>
        </fileset>
    </copy>

    <copy todir="${lib-dest}">
        <fileset dir="${lib}">
            <include name="**/*.*/>
        </fileset>
    </copy>

    <copy todir="${config-dest}">
        <fileset dir=".">
            <include name="*.xml"/>
            <include name="*.tld"/>
        </fileset>
    </copy>
</target>

<target name="makewar" depends="copy-src">
    <jar jarfile="${project-name}.war" basedir="${dest}">
    </jar>
</target>
<!-- COPY -->
<!-- ===== -->
<target name="copy-war" depends="makewar">
    <copy todir="${deploy-war}">
        <fileset dir="${src}">
            <include name="*.war"/>
        </fileset>
    </copy>

```

```
        <delete dir="${dest}"/>
    </target>

    <!-- RELOAD PROJECT -->
    <!-- ===== -->
    <target name="reload" depends="copy-war">
        <reload url="${manager-url}" username="${manager-id}"
password="${manager-pass}" path="${webapp-path}"/>
    </target>

    <target name="clean">
        <delete dir="${TOMCAT_HOME}/webapps/${project-name}"/>
    </target>

</project>
```

2.5 JSF에서의 한글

톰캣의 filter기능을 이용하여, 한글이 처리되도록 하였지만, JSF의 properties 파일을 읽어오는 경우 여전히 한글이 깨지는 문제가 발생한다. 이런경우 한글을 유니코드로 인코딩하여 이용한다.

ant의 다음과 같은 작업으로, 확장자가 properties로 끝나는 모든 파일은 유니코드로 인코딩되도록 하였다.

```
<native2ascii encoding="EUC-KR" src="${src}" dest="${class-dest}"
includes="**/*.properties" ext="_ko.properties"/>
```

파일명의 끝에 _ko를 붙이도록 한 이유는 JSF가 클라이언트의 언어환경에 따라 자동적으로 파일을 참고하기 때문이다. 우리는 앞으로의 예제를 통해 Messages.properties 라는 파일과 Resources.properties 라는 property 파일을 이용하게 될것이다. 사용할 브라우저의 환경에 한글이면, JSF는 Messages_ko.propertie 와 Resources_ko.properties 를 참조할 것이다.

3. JSF로 구현한 간단한 예제

간단한 예제를 통해서 개략적이거나 JSF가 어떻게 움직이는지를 알아보도록 하자. 여기서 테스트해볼 프로그램은 간단하다. 단순히 입력을 받은 문자열을 다음 페이지에서 출력하도록 한다.

3.1 간단한 예제

프로그램을 테스트해보기 위해 아래의 과정을 따라 해보자.

- 데이터를 저장할 빈의 작성
- jsf페이지 작성
- 빈의 설정 및 페이지 이동을 위한 faces-config.xml의 작성

데이터를 저장할 bean의 작성

아래의 bean은 문자열 변수와 변수의 setter/getter메소드를 가지는 클래스이다.

```
package jsf.proj.example;

public class Example1{

    private String msg="Hello World !!";

    public void setValue(String msg){
        this.msg=msg;
    }
    public String getValue(){
        return msg;
    }
}
```

위에서 작성한 Example1 빈을 이용하기위해 간단한 jsf페이지를 아래와 같이 작성한다.

```
<!--
```

```
    /jsp/example/example1.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="myForm">
        <h:inputText id="in1" value="#{Example1.value}"/><br>
        <h:commandButton value="SUBMIT버튼" action="success"/>
      </h:form>
    </f:view>
  </body>
</html>
```

jsf의 태그라이브러리를 이용하기 위해서는 위와 같이 두개의 태그 라이브러리를 이용한다.

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

첫번째 태그 라이브러리는 html컴포넌트를 사용하기 위한 태그를 정의한 태그 라이브러리이다. 두번째는 jsf의 core컴포넌트를 나타내는 태그 라이브러리이다. html컴포넌트는 웹 브라우저에 나타날 수 있는 컴포넌트들, 예를들면 버튼이나 입력상자, 라디오버튼, 체크박스 등등의 태그를 사용하기 위한 것이고, core컴포넌트는 주로 html컴포넌트 내부에 쓰이거나, 브라우저에 나타나지 않는 처리작업들, 예를 들면 입력된 값을 검사하는 validator나 값을 변환하는 converter등을 사용하기 위해 core컴포넌트가 필요하다. 나중에 자세히 설명이 되므로 지금은 여기의 예제를 살펴보도록 하자.

view태그내에 사용될 컴포넌트들이 들어가야 한다. 요청된 페이지의 컴포넌트들은 jsf에 의해 내부적으로 컴포넌트 트리로 구성된다. 위에서 예들든 jsf페이지의 form태그는 html form엘리먼트로, inputText태그는 html input엘리먼트로 commandButton은 html button엘리먼트로 각각 해석될 것이다.

jsf는 값을 표현하는 방법으로 EL(Expression Language)를 이용한다. #{ }로 둘러싸인 표현식은 단순한 문자열로 해석되지 않고, jsf에 의해 이 문자열이 의미하는 값을 매핑한다. 위의 inputText태그의 value값인 #{Example1.value}는 Example1이라는 클래스의 getValue() 메소드를 의미하게 되는 것이다. 여기서 Example1이라는 클래스 이름은 faces-config.xml 파일에 등록된 jsf.proj.example.Example1클래스의 id이다.

commandButton의 action속성은 버튼이 클릭되었을 경우 어디로 이동할지를 나타내는 역할을 한다. 이 버튼을 클릭했을 경우 입력한 결과를 표시하는 다음의 페이지로 이동한다.

```
<!--
    /jsp/example/example1_result.jsp
-->

<%@page contentType="text/html;charset=euc-kr"%>

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="resultForm">
        <h:outputText value="#{Example1.value}"/>
      </h:form>
    </f:view>
  </body>
</html>
```

이번에는 위에서 작업한 예제의 bean과 페이지 이동을 위해 faces-config.xml파일을 설정해 보자.

```
<?xml version="1.0" encoding="euc-kr"?>

<!DOCTYPE faces-config PUBLIC
```

```

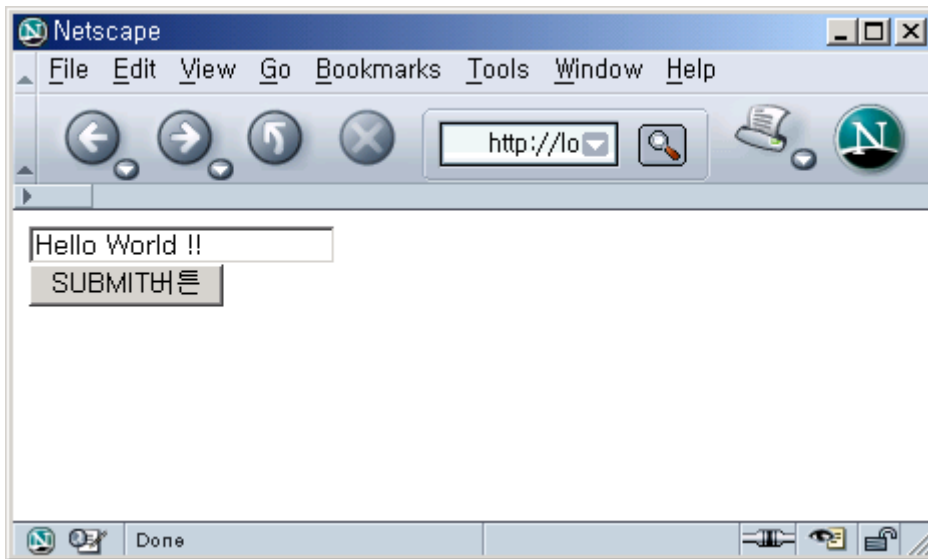
"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
"http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
  <managed-bean>
    <managed-bean-name>Example1</managed-bean-name>
    <managed-bean-class>jsf.proj.example.Example1</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>

  <navigation-rule>
    <from-view-id>/jsp/example/example1.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/jsp/example/example1_result.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>

```

위에서 만든 bean은 faces-config.xml에서 <managed-bean>...</managed-bean>내부에 위와 같이 사용하게 될 이름과 패키지 명을 포함한 클래스 전체이름, 그리고 이 bean이 사용될 scope를 기술하면 되겠다. 페이지의 이동은 <navigation-rule>...</navigation-rule>내부에 설정된다. 위의 예를 보면, /jsp/example/example1.jsp 페이지로부터 action 문자열이 success이면 /jsp/example/example1_result.jsp 페이지로 forward된다.



example1.jsp파일의 결과가 위의 화면에 나타나 있다. 브라우저의 소스보기를 이용해서 jsf가 html로 해석된 결과를 보도록 하자.

```
<!--
      example1.jsp
-->
<html>
<body>

<form id="myForm" method="post" action="/jsf-proj/faces/jsp/example/example1.jsp"
enctype="application/x-www-form-urlencoded">

<input id="myForm:in1" type="text" name="myForm:in1" value="Hello World !!" /><br>
<input type="submit" name="myForm:_id0" value="SUBMIT&#48260;&#53948;" />
<input type="hidden" name="myForm" value="myForm" /></form>
</body>
</html>
```

위의 내용과 비슷한 결과가 나올것이다. 위의 html소스를 보면 id를 붙이지 않은 컴포넌트는 jsf에 의해 id가 부여되어 있으며, html에서 각 엘리먼트들의 name속성이 formName:id명 과 같이 되어 있는 것을 알수 있다. 이렇게 폼이름:컴포넌트id 와 같은 형식을 취하므로, 자바스크립트를 쓰는 경우 이와 같이 이름이 부여 되는것에 유의해서 작성할 필요가 있다.

3.2 rendered 속성

rendered속성은 id와 마찬가지로 모든 컴포넌트가 가지는 속성이다. rendered속성이 true이면 컴포넌트는 처리되어 브라우저에 디스플레이 되고, false이면 해당 컴포넌트는 처리되지 않는다. rendered속성은 화면에 표시되는 컴포넌트를 제어할 때 요긴하게 이용할 수 있다.

위의 예제를 이용하여 다음과 같은 작업을 추가해보자.

우선 Example1.java를 다음과 같이 수정한다.

```
package jsf.proj.example;

public class Example1{

    private String msg="Hello World !!";

    private boolean display=false;

    public void setValue(String msg){
        this.msg=msg;
    }

    public String getValue(){
        return msg;
    }

    public boolean getDisplay() {
        return display;
    }

    public void setDisplay(boolean display) {
        this.display = display;
    }

}
```

display라는 boolean값과 이 boolean값에 대한 setter/getter메소드가 추가 되었다.

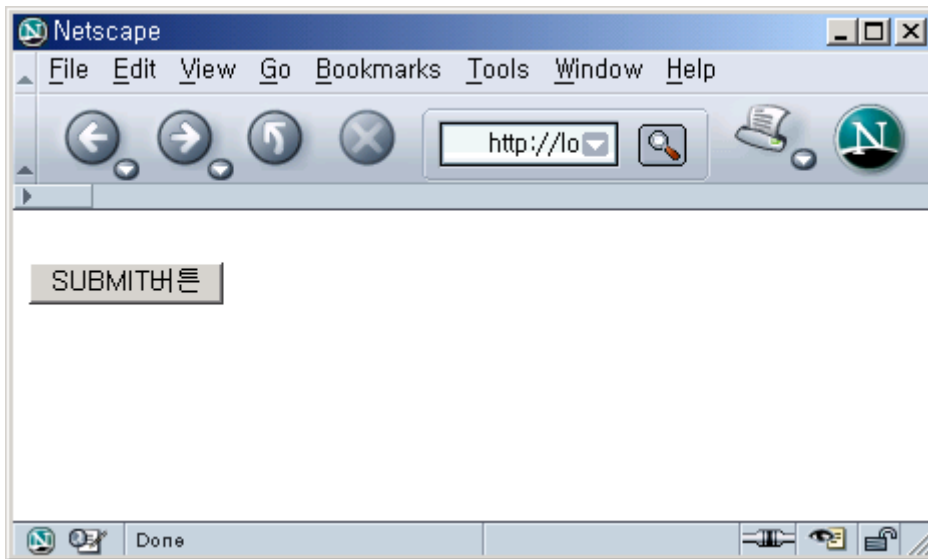
example1.jsp파일은 아래와 같이 수정한다.

```
<!--
    /jsp/example/example1.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="myForm">
        <h:inputText id="in1" value="#{Example1.value}" rendered="#{Example1.display}"
/><br>
        <h:commandButton value="SUBMIT버튼" action="success"
rendered="#{Example1.display==false}"/>
      </h:form>
    </f:view>
  </body>
</html>
```

컴포넌트에 rendered속성이 추가 되었다. 입력박스의 rendered속성값은 getDisplay()메소드가 반환하는 값이 될 것이고, 버튼의 rendered속성은 getDisplay()메소드가 반환하는 값이 false이면 표현식에 의해 true가 될 것이고, getDisplay()메소드가 true를 반환하면 false가 될 것이다. EL은 이보다 더 다양한 표현식을 사용 가능하게 한다. 더 다양한 표현식에 대해서는 차차 공부하기로 하자. Example1.java의 display속성값이 false이므로 아래와 같이 입력박스는 나타나지 않고, 버튼만 나타난다.



rendered속성은 모든 컴포넌트가 가지는 속성중 하나이다. rendered속성을 잘 이용하면 화면처리를 보다 더 유연하게 할 수 있다.

3.3 컴포넌트의 바인딩

id,rendered속성 이외에 모든 컴포넌트가 가지는 공통된 속성하나가 binding속성이다. binding속성은 bean에서 생성된 컴포넌트를 웹 페이지에 binding하는 기능을 제공한다.

또 다시 Example1.java 소스를 수정해보자.

```
package jsf.proj.example;

import javax.faces.component.html.*;

public class Example1{
...
    private HtmlInputText input;

    public Example1(){
        input=new HtmlInputText();
        input.setValue("컴포넌트 바인딩");
    }
}
```

```

    public HtmlInputText getInput() {
        return input;
    }

    public void setInput(HtmlInputText input) {
        this.input = input;
    }
...
}

```

HtmlInputText 컴포넌트는 html의 입력박스를 나타내는 컴포넌트이다. Exmample1.java에서 생성된 이 컴포넌트는 아래와 같이 웹 페이지에 바인딩된다.

```

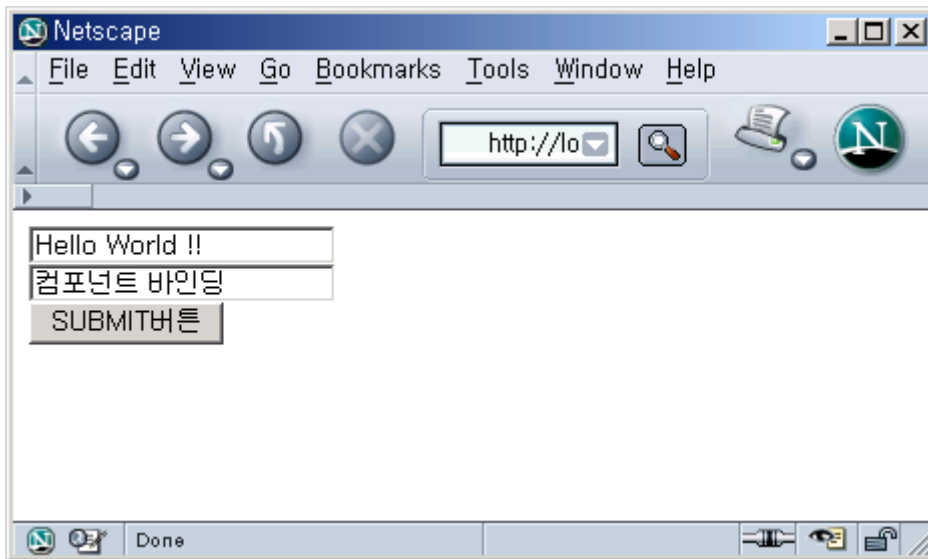
<!--
    /jsp/example/example1.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="myForm">
        <h:inputText id="in1" value="#{Example1.value}" /><br>
        <h:inputText binding="#{Example1.input}" /><br>
        <h:commandButton value="SUBMIT버튼" action="success"/>
      </h:form>
    </f:view>
  </body>
</html>

```

기존의 웹 페이지에서의 작업 대부분을 서버사이드에서 처리하고, binding속성을 통해 웹 페이지에 표현하도록 할수 있다. 위의 웹페이지는 다음과 같이 나타난다.



bean에서 생성한 HtmlInputText컴포넌트를 바인딩을 이용해서 웹브라우저에 표시된 결과다. 위의 예를 통해서 알수 있듯이 JSF는 html을 처리하기 위한 모든 요소들을 서버사이드에서 처리하고, 결과를 브라우저에 나타낼 수 있다. 물론 바인딩을 사용하지 않고, JSF페이지에서 처리하는 것도 가능하다. Sun Java Studio Creator 같은 비주얼한 개발툴은 바인딩을 통해 작성한 웹 페이지의 컴포넌트들을 처리한다.

3.4 이벤트

컴포넌트에 이벤트를 이용해보자. JSF에서 개발자들이 이용하게 될 이벤트는 `ActionEvent`와 `ValueChangeEvent`일 것이다. `ActionEvent`는 action이 일어날 수 있는 버튼과 링크 컴포넌트에 이용하고, `ValueChangeEvent`는 값의 변경이 일어날수 있는 입력상자, 라디오버튼, 메뉴, 체크박스 등등에 사용할 수 있다.

이벤트를 추가할 수 있는 방법은 두가지가 있는데 여기서 는 컴포넌트의 속성을 이용해서 메소드로 정의하는 방법을 테스트 해보도록 한다.

위에서 사용한 `example1.jsp`를 다음과 같이 수정한다.

```
<!--
    /jsp/example/example1.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>

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

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="myForm">
        <h:inputText id="in1" value="#{Example1.value}"
valueChangeListener="#{Example1.valueChanged}"/><br>

        <h:inputText binding="#{Example1.input}"/><br>

        <h:commandButton value="SUBMIT버튼" action="success"
actionListener="#{Example1.actionOccurred}"/>
      </h:form>
    </f:view>
  </body>
</html>

```

처음 입력상자에 valueChangeListener 속성을 이용하여 메소드를 바인딩한다.
 그리고, 명령버튼에 actionListener 속성을 이용하여, 역시 메소드를 바인딩한다.
 여기서, 지정된 메소드를 구현하면 Example1.java의 소스가 다음과 같이 변경된다.

```

package jsf.proj.example;

import javax.faces.component.html.*;
import javax.faces.component.*;
import javax.faces.event.*;

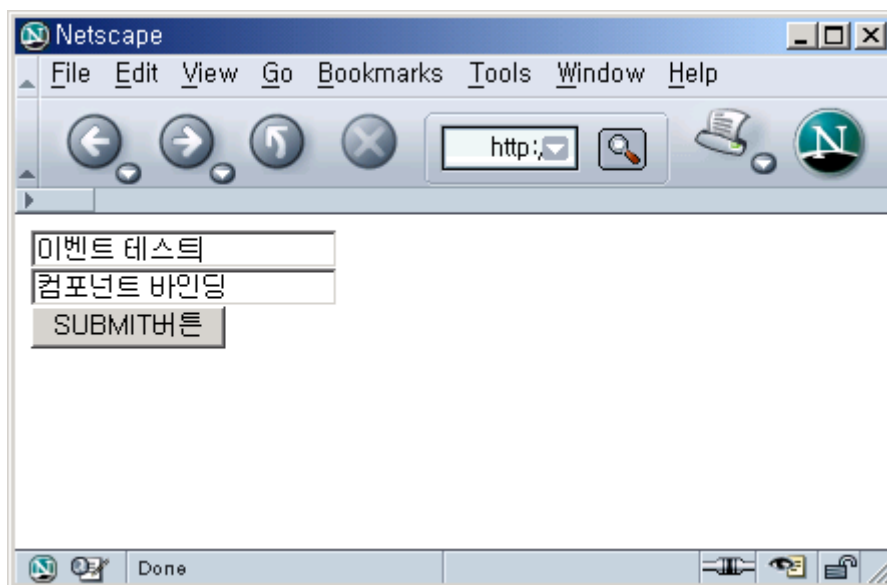
public class Example1{
  ...
  public void valueChanged(ValueChangeEvent evt){
    UIInput input=(UIInput)evt.getComponent();
    Object old=evt.getOldValue();
    Object newValue=evt.getNewValue();
    msg="컴포넌트 ID : "+input.getId()+"이전값 : "+old+" , 변경된 값 :
"+newValue;

```

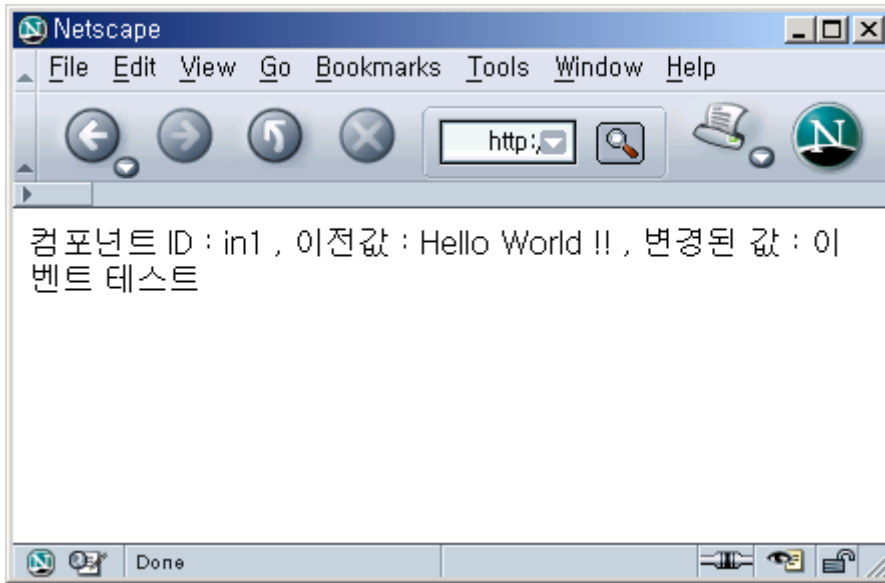
```
        input.setValue(msg);
    }
    public void actionPerformed(ActionEvent evt){
        System.out.println("ActionEvent가 발생되었습니다.");
    }
    ...
}
```

JSF의 스펙에 따라, 이벤트를 처리할 메소드는 public 이고, 리턴타입이 void이어야 한다. 그리고, 파라미터로 각각 ValueChangeEvent 개체와(ActionEvent) 개체를 설정한다. 메소드 이름은 상관없다. 각 이벤트 개체는 이벤트를 발생한 컴포넌트를 getComponent() 메소드를 이용하여 얻어낼 수 있다. 그리고, ValueChangeEvent는 값이 변경되기 전과 후의 값들을 얻어낼수 있는 메소드를 위와 같이 제공하고 있다.

다음과 같이 입력을 하고 버튼을 클릭해보자.



다음과 같은 내용이 출력된다.



주의 할 것은 `valueChanged()`메소드에서 `ValueChangeEvent` 의 `getNewValue()`메소드가 반환하는 값이 이 컴포넌트의 값이 된다. 따라서, `valueChanged()`메소드의 마지막에 `setValue()`메소드를 이용해 값을 설정하지 않으면 `getNewValue()`메소드가 반환하는 값이 여기 화면에 나타날 것이다. `valueChanged()`메소드의 맨 마지막라인에 있는 `input.setValue(msg)`메소드를 지우고 결과를 한번 보기 바란다.

이렇게 메소드를 바인딩하여 이벤트를 처리하는 경우, 이 메소드를 다른 곳에서 사용하기에는 불편하다. 여기서는 이 정도로 설명을 마치고 6장에서는 이벤트를 이용하는 다른 방법을 설명하도록 하겠다.

이번장에서는 간단하게 jsf의 몇가지 특징을 살펴보았다. 위에서 살펴본 내용들이 JSF의 가장 큰 특징이라 할 수 있겠다. 앞으로 살펴볼 내용들은 위의 예제에서 이용해보았던 컴포넌트들에 대한 더 자세한 내용과, 컴포넌트를 이용한 프로그래밍을 지원하는 이벤트 처리, `Validator`, `Converter` 그리고 직접 컴포넌트를 만들어보고 간단한 웹 어플리케이션을 제작해보도록 한다.

4장. JSF의 구성요소

JSF의 구성요소를 이해하기 위해 우선 JSF가 제공하는 API의 주요 패키지를 살펴보면 다음과 같다.

패키지 이름	설명
javax.faces.application	JSF로 개발된 웹 어플리케이션의 전체적인 기능들을 관리하기 위해 만들어진 패키지이다.
javax.faces.component	컴포넌트들의 기본적인 특징을 구현한 패키지.
javax.faces.component.html	javax.faces.component에서 구현된 컴포넌트들을 html환경에서 이용가능하도록 구현한 패키지.
javax.faces.context	request/response를 처리하기 위한 환경에 관련된 패키지.
javax.faces.convert	JSF의 기본 converter와 converter 인터페이스가 있는 패키지.
javax.faces.el	ValueBinding과 MethodBinding을 수행하기 위한 패키지.
javax.faces.lifecycle	request/response 하나의 사이클을 관리하는 패키지.
javax.faces.model	JSF가 지원하는 Data Model 패키지.
javax.faces.render	렌더러를 위한 패키지.
javax.faces.validator	JSF의 기본 validator와 validator인터페이스가 있는 패키지.
javax.faces.webapp	JSF의 컴포넌트를 이용하기 위한 관련 커스텀 태그가 있는 패키지.

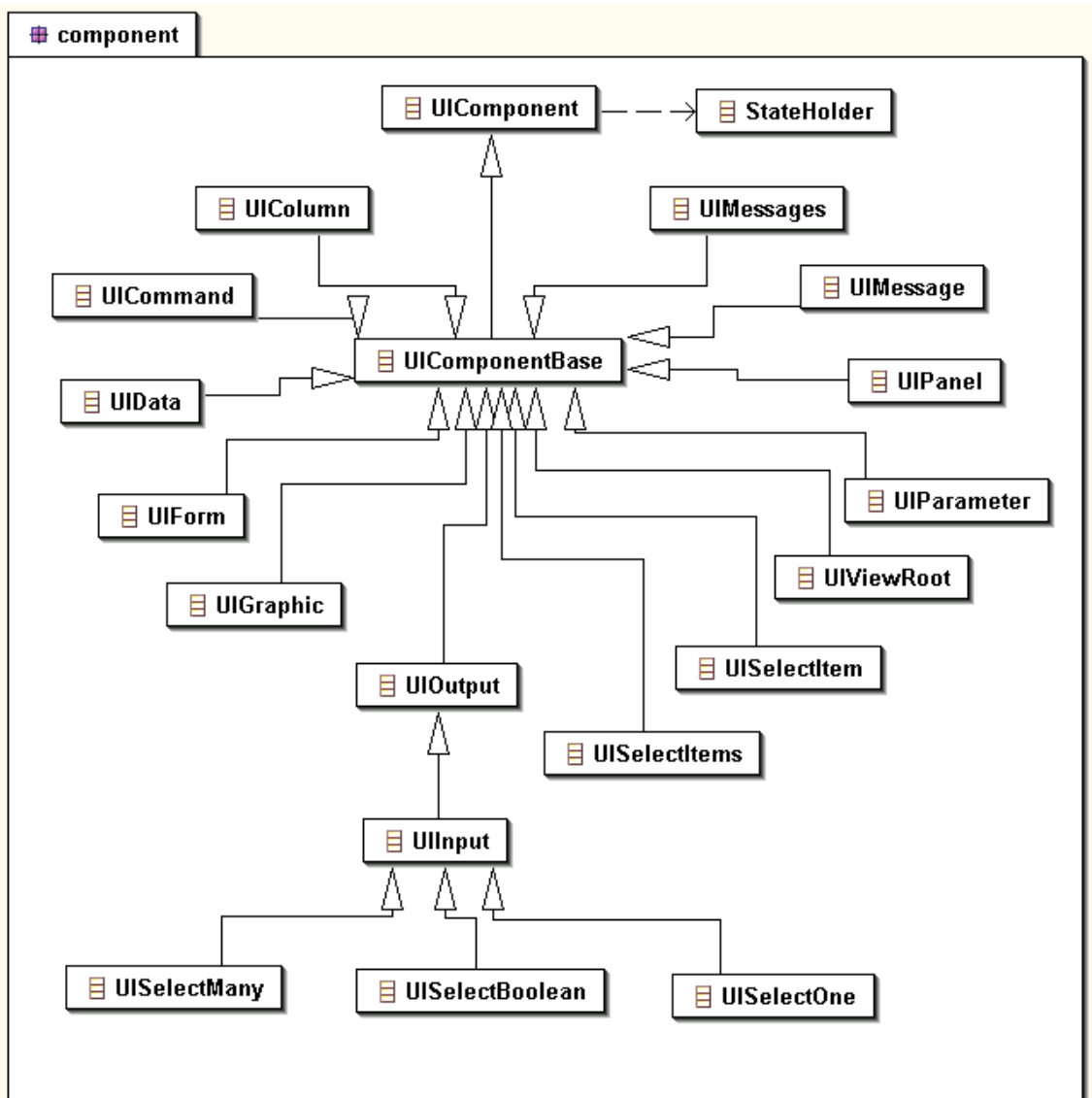
위의 패키지의 내용을 보면 JSF는 크게 다음과 같이 구성된다.

- JSF를 이용한 웹 어플리케이션의 전체 환경을 지원하는 패키지
- 컴포넌트 패키지
- 컴포넌트를 지원하는 이벤트, Validator, Converter, EL, 렌더러
- JSF에서 사용되는 데이터 집합을 처리하기 위한 데이터 모델

이 4가지 항목이 JSF의 전체적인 모습이 되겠다. 가능하면 많은 예제를 통해 위의 4가지 항목을 설명하도록 하겠다.

4.1 컴포넌트

아래의 그림은 JSF가 제공하는 `javax.faces.component` 패키지의 클래스 다이어그램이다.

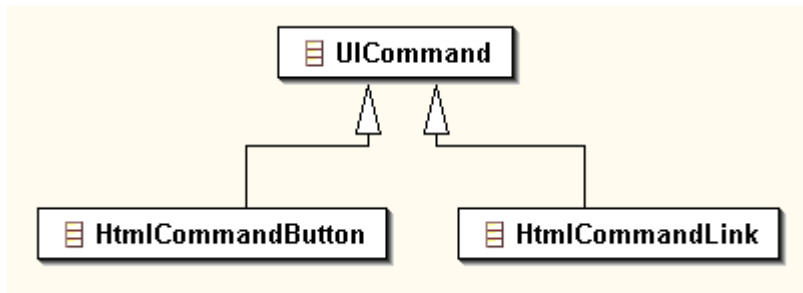


`javax.faces.component` 패키지에서 구현된 클래스들은 JSF에서 사용되는 컴포넌트들의 기본적인 특징들을 모두 구현해 놓았다.

이 패키지의 클래스들의 이름을 보면 알겠지만, 클래스 이름의 첫 두글자 UI만 빼면 html과

유사한 것들을 발견할 수 있을 것이다. 이 패키지에서 구현된 클래스들은 대응되는 html 요소들의 중요한 특징들을 모두 구현하고 있다. 이 패키지의 클래스들을 상속받아서 `javax.faces.component.html` 패키지의 클래스들이 구성된다.

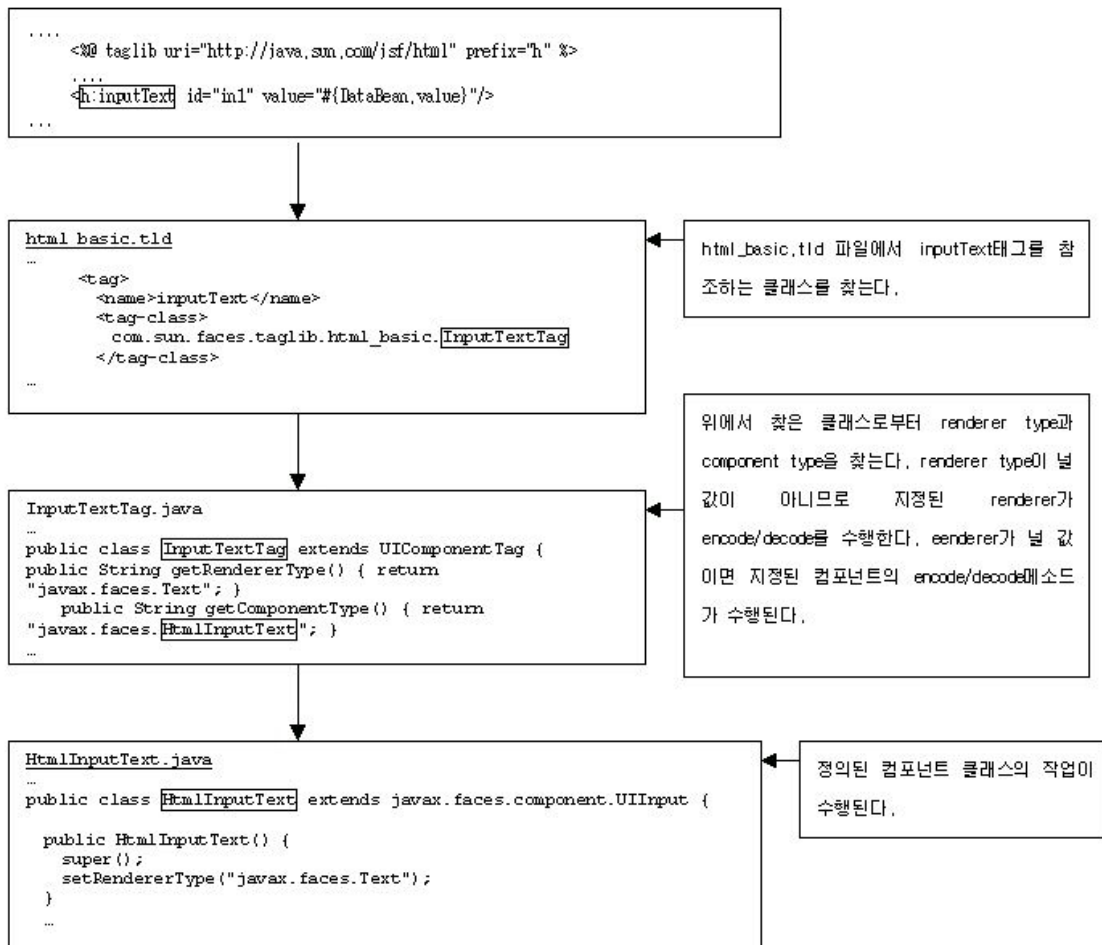
예를 들어, html버튼, html링크를 나타내는 `javax.faces.component.html` 패키지의 클래스 `HtmlCommandButton`과 `HtmlCommandLink`의 클래스 다이어그램을 보자.



html에서 버튼과 링크는 그 모양만 다를뿐, 특징이 유사하다. 이 특징을 `UICommand` 클래스가 구현하고 있다. `HtmlCommandButton`과 `HtmlCommandLink` 클래스는 자바스크립트를 위한 속성값, html스펙을 만족하기 위한 속성 그리고 CSS를 사용하기 위한 속성들을 위한 `getter/setter` 메소드만을 가지고 있을 뿐이다.

`javax.faces.component.html` 패키지의 클래스들이 모두 이러한 구조를 가지고 있다.

웹 페이지에서 컴포넌트를 사용하기 위해서 어떤 과정을 거치게 되는지를 보자.



위의 그림은 입력상자가 웹 브라우저에 나타나는 과정을 그린것이다. 다음의 순서에 따라 하나의 컴포넌트가 처리된다.

1. 접두어 'h'는 사용될 태그라이브러리의 위치를 표시한다.
2. 태그라이브러리로부터 inputText라는 태그를 찾는다.
3. inputText태그가 지시하는 태그클래스 InputTextTag를 얻는다.
4. InputTextTag 클래스는 렌더러를 지정하는 문자열과 컴포넌트는 지정하는 문자열을 얻고 있다. 렌더러가 있으므로 encode/decode메소드는 렌더러에 위임된다.
5. HtmlInputText 컴포넌트가 생성되고 처리된다. 렌더러가 있기 때문에 렌더러의 encode/decode 메소드가 호출된다.
6. 웹 브라우저에 <input type= ...>와 같이 나타난다.

이 과정사이에 속성값을 설정하는 등의 여러가지 작업이 있지만, 컴포넌트가 html로 해석되는 과정에 중점을 두어 이해해 주기 바란다. 또, 위에서 설명한 처리과정이 나중에 우리가 직접 컴포넌트를 만드는 과정과 동일하다. 직접 만들게 될 컴포넌트들도 jsf가 제공하는 컴

포넌트들과 동일한 방식으로 처리될 것이기 때문이다. 위의 예에서 알 수 있듯이 컴포넌트를 만들게 되면 컴포넌트를 나타낼 태그 클래스, 컴포넌트 클래스, 렌더러 클래스, 태그 정의의 파일이 필요하게 된다. 여기서, 렌더러 클래스는 선택적이다.

UIComponent 클래스는 모든 컴포넌트의 기본이 되는 클래스이다. 이 클래스는 request간의 상태를 저장하는 StateHolder 인터페이스를 구현하는 추상메소드로 이루어져 있다. UIComponent클래스를 사용하기 쉽도록 UIComponentBase클래스를 상속하여 컴포넌트들이 구성된다. 컴포넌트의 기본이 되는 UIComponentBase클래스의 주요 메소드를 살펴보자.

다음은 모든 컴포넌트들이 가지는 주요 메소드이다.

```
public void decode(javax.faces.context.FacesContext context)
```

이 컴포넌트의 이벤트가 발생할 때 호출되는 메소드이다.

```
public void encodeBegin(FacesContext context)
```

컴포넌트를 화면에 표시하기 위한 작업을 시작한다.

```
public void encodeChildren(FacesContext context)
```

이 컴포넌트의 자식컴포넌트, 즉 jsf에서 이 컴포넌트의 태그에 둘러싸여 있는 포함된 컴포넌트를 처리할 작업을 이 메소드에서 구현한다.

단, 아래에 나오는 getRendersChildren()메소드가 true값을 반환할 때이다.

```
public void encodeEnd(FacesContext context)
```

이 컴포넌트를 화면에 표시하는 작업을 종료한다.

```
public UIComponent findComponent(String id)
```

이 컴포넌트의 하위컴포넌트들 중에 지정된 id를 가지는 컴포넌트를 찾아 그 컴포넌트를 반환한다.

```
public Map getAttributes()
```

이 컴포넌트의 속성들을 java.util.Map으로 반환한다. 그러나, java.util.Map에서 제공하는 containsKey()나 remove()와 같은 메소드를 사용 할 수 없다.

```
public int getChildCount()
```

만약 getRendersChildren()메소드가 true를 반환하면, 이 컴포넌트의 자식 컴포넌트의

개수를 반환한다.

```
public List getChildren()
```

자식 컴포넌트를 java.util.List로 반환한다.

```
protected FacesContext getFacesContext()
```

현재 웹 어플리케이션과 클라이언트의 요청에 의해 생성된 FacesContext개체를 반환한다.

```
public UIComponent getFacet(String name)
```

Facet개체를 Facet개체의 속성인 name을 가지고 얻을 수 있다.

Facet는 Facet을 포함하고 있는 컴포넌트와 부모-자식 간의 관계를 가지지 않는 독립적인 컴포넌트이다.

```
public Map getFacets()
```

이 컴포넌트가 가지고 있는 모든 Facet을 java.util.Map으로 반환한다.

```
public Iterator getFacetsAndChildren()
```

이 컴포넌트가 가지고 있는 Facet와 자식 컴포넌트를 반환한다.

```
public String getId()
```

이 컴포넌트의 id를 반환한다. id는 모든 컴포넌트에서 유일하며, id를 부여받지 않은 컴포넌트는 jsf가 내부적으로 부여한다.

```
public UIComponent getParent()
```

이 컴포넌트의 부모컴포넌트를 반환한다.

```
public boolean getRendersChildren()
```

처리해야 할 컴포넌트가 내부에 다른 컴포넌트를 포함하는 경우 포함된 컴포넌트의 처리를 어느 컴포넌트가 하는지를 결정하는 메소드이다. 각각의 컴포넌트는 기본적으로 자기자신의 encode/decode메소드를 호출하여 작업을 처리한다. 그러나, 컴포넌트를 포함하고 있는 컴포넌트에서 이 메소드가 true를 반환하면, 이 컴포넌트가 포함하는 자식 컴포넌트의 encode/decode메소드는 사용되지 않는다. 이 컴포넌트의 encode/decode메소드가 자식 컴포넌트가 해야 할 작업을 구현하여 처리한다는 뜻이 되겠다.

jsf컴포넌트들은 기본적으로 false를 반환하도록 되어있으며, 개발자가 컴포넌트를 개발

할 때 오버라이딩해서 이용하면 된다.

이 컴포넌트가 포함하고 있는 자식컴포넌트의 화면표시(`encodeBegin/encodeEnd`)와 같은 처리를 어디서 하는지를 결정하는 메소드이다.

기본적으로 각 컴포넌트는 자신이 가지고 있는 `encodeBegin/encodeEnd`메소드를 이용해서 자기자신을 클라이언트의 화면에 표시하는 작업을 수행한다.

그러나, 이 메소드가 `true`를 반환하면, 자식 컴포넌트들의 `encodeBegin/encodeEnd`는 실행되지 않는다.

각 컴포넌트는 자신의 `encodeBegin/encodeEnd`메소드를 수행하도록 되어있다. 다시 말하면 각 컴포넌트의 `getRenderersChildren()`메소드는 디폴트 값으로 `false`를 반환한다.

또한, `false`를 반환하는 경우 자식 컴포넌트의 정보를 알수 없다. 예를 들어 위의 `getChildCount()`같은 메소드는 0을 반환할 것이다.

```
public boolean isRendered()
```

이 컴포넌트가 화면에 표시되는지 여부를 결정한다. 컴포넌트의 `rendered`속성으로도 지정할 수 있다.

```
public void setRendererType(String rendererType)
```

이 컴포넌트를 화면에 표시하는 작업(`encodeBegin/encodeEnd` 메소드)을 하게된 `renderer`를 지정한다. 이 값이 `null`이면 이 컴포넌트 자신의 (`encodeBegin/encodeEnd`) 메소드를 수행한다.

```
public void setValueBinding(String name, ValueBinding binding)
```

이 컴포넌트의 속성(`name`)에 바인딩되는 개체를 지정한다.

`jsf`페이지의 아래와 같은 작업을 예로 들면,

```
<h:inputText id="in1" value="#{DataBean.value}"/>
```

`UIInput`컴포넌트의 `value`속성이 가지는 값은 문자열 `'#{DataBean.value}'`이 아니라, `DataBean`클래스 인스턴스의 `getValue()`메소드가 반환하는 값이 될것이다.

즉, `ValueBinding`클래스는 입력된 표현식을 적절한 값으로 반환해주는 작업을 하는 역할을 한다.

4.3 managed bean

`faces-config.xml` 에 `<managed-bean>` 태그를 이용하여 `bean`을 등록할 수 있다. 여기에

등록된 bean들은 웹 페이지에 의해 처음 사용되는 경우 생성되며, scope속성에 따라 JSF가 알아서 내부적으로 관리하게 된다. 물론 jsp에서 처럼 <jsp:useBean...>과 같이 사용해도 무방하다. faces-config.xml에 등록되는 bean들은 bean의 속성값들을 설정하는 것도 가능하다.

테스트를 위해 bean을 하나 만들고 faces-config.xml파일의 설정을 보도록 하자.

```
package jsf.proj.example;
import java.util.*;
public class TestBean{
    private Map mapData;
    private List listData;
    private Integer number;
    private String name;
    public List getListData() {
        return listData;
    }
    public void setListData(List listData) {
        this.listData = listData;
    }
    public Map getMapData() {
        return mapData;
    }
    public void setMapData(Map mapData) {
        this.mapData = mapData;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Integer getNumber() {
        return number;
    }
    public void setNumber(Integer number) {
```

```
        this.number = number;
    }
}
```

테스트를 위해 위와 같은 멤버변수값들을 faces-config.xml에 다음과 같이 초기값을 설정할 수 있다. 물론, 위의 bean 클래스내에서 설정하는 것도 가능하다.

```
<?xml version="1.0" encoding="euc-kr"?>

<!DOCTYPE faces-config PUBLIC
    "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
...
    <managed-bean>

        <managed-bean-name>TestBean</managed-bean-name>
        <managed-bean-class>jsf.proj.example.TestBean</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>

        <managed-property>
            <property-name>listData</property-name>
            <list-entries>
                <value-class>java.lang.String</value-class>
                <value>홍길동</value>
                <value>이순신</value>
                <value>세종대왕</value>
                <value>강감찬</value>
            </list-entries>
        </managed-property>

        <managed-property>
            <property-name>mapData</property-name>
            <map-entries>
                <key-class>java.lang.String</key-class>
```

```
        <map-entry>
            <key>서울시</key>
            <value>강남구</value>
        </map-entry>
        <map-entry>
            <key>경기도</key>
            <value>이천시</value>
        </map-entry>
        <map-entry>
            <key>경상남도</key>
            <value>마산시</value>
        </map-entry>
    </map-entries>
</managed-property>

<managed-property>
    <property-name>number</property-name>
    <value>3</value>
</managed-property>

<managed-property>
    <property-name>name</property-name>
    <null-value/>
</managed-property>
</managed-bean>
</faces-config>
```

<managed-property> 태그를 이용해서 위와 같이 bean의 초기값을 설정하면 된다. listData, mapData 및 number 변수의 값을 설정하고, name 변수는 널값으로 설정하였다. 웹 페이지에서 이들의 결과값을 출력해보자.

출력을 위해 다음과 같은 페이지를 하나 만든다.

```
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsp/core" prefix="f" %>
```

```

<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
<body>
  <f:view >

    <h:form id="testForm">

      listData의 세번째 값 :
      <h:outputText value="#{TestBean.listData[2]}" /><br>

      mapData의 키 '서울시'의 값 :
      <h:outputText value="#{TestBean.mapData['서울시']}" /><br>

      mapData의 키 '경상남도'의 값 :
      <h:outputText value="#{TestBean.mapData[W"경상남도W"]}" /><br>

      number의 값 :
      <h:outputText value="#{TestBean.number}" /><br>

      number의 값이 3이면 true를 반환한다. :
      <h:outputText value="#{TestBean.number==3}" /><br>

      name의 값 :
      <h:outputText value="#{TestBean.name}" /><br>

    </h:form>
  </f:view>
</body>
</html>

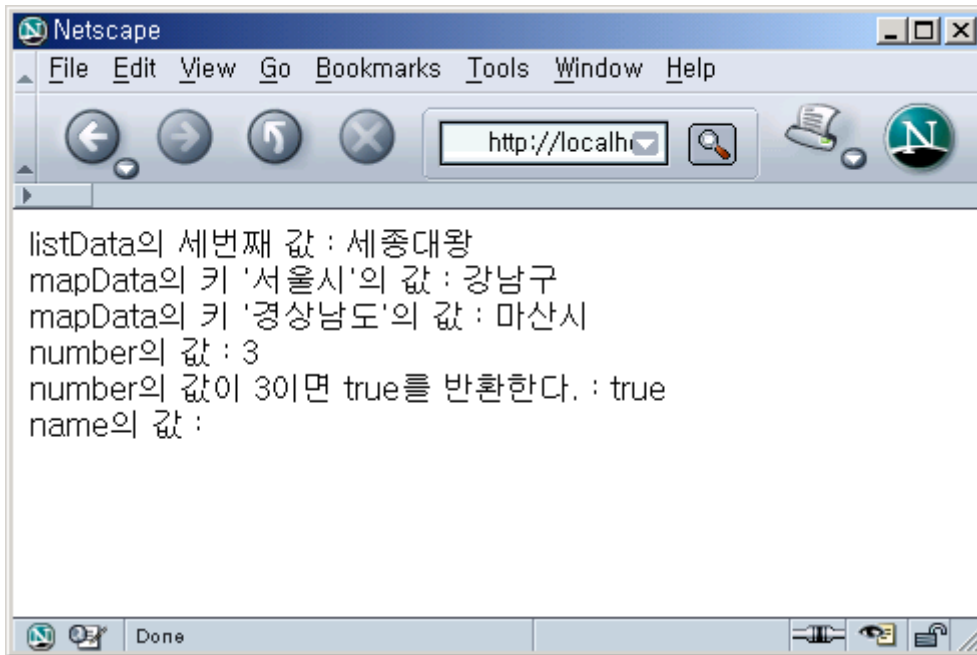
```

값이 바인딩되는 표현들을 몇가지 볼 수 있다. 데이터 집합의 요소들을 이용하기 위해 위와 같은 표현들이 자주 이용될 것이다. 키값을 사용하는 경우 “는 중복되므로 위와 같이 W”를 이용해야 한다. 또, "#{TestBean.number==3}"과 같은 논리값을 검사하는 것도 가능하다.

가능한 표현식은 다음과 같다.

- +-*/ 연산
- <, <=, >=, ==, !=, &&, ||, ! 연산
- ?: 연산

위의 결과는 브라우저에 다음과 같이 나타난다.



이러한 표현식을 처리하는 클래스가 javax.faces.el.ValueBinding 클래스이다. ValueBinding 클래스를 이용하여 bean에서도 위와 같이 #{ }로 둘러싸인 표현식을 이용하는 것도 가능하다. 위와 같은 표현식이 어떻게 처리되는지 ValueBinding 클래스를 이용해서 다음과 같은 테스트를 해보자. 테스트를 위해 TestBean 클래스를 다시 한번 이용한다.

```

package jsf.proj.example;

...

import javax.faces.el.*;
import javax.faces.context.*;
import javax.faces.application.*;

public class TestBean{
...

```

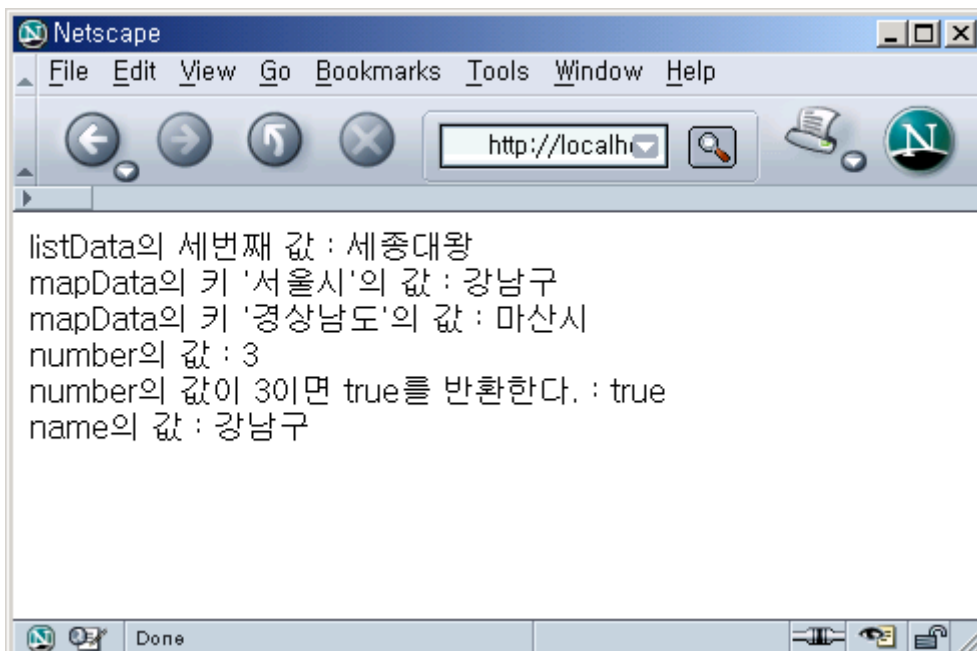
```

    public String getName() {
        FacesContext fc=FacesContext.getCurrentInstance();
        Application app=fc.getApplication();
        ValueBinding
vb=app.createValueBinding("#{TestBean.mapData['서울시']}");
        name=(String)vb.getValue(fc);
        return name;
    }
...
}

```

테스트를 위해 간단히 name변수의 값에 #{TestBean.mapData['서울시']} 식의 결과값을 설정하였다. 이러한 표현식을 처리하고 결과를 얻어내는 클래스가 ValueBinding클래스이다. FacesContext는 현재 request를 관리하는 context개체이고, 이 개체로부터 동적인 ValueBinding을 수행하기 위해 Application개체를 얻어와서 ValueBinding을 수행한다.

결과로 '서울시'를 키값으로 가지는 개체가 name변수의 값으로 설정되어 아래와 같이 출력된다.



JSF는 이렇게 표현식문자열을 처리하는 ValueBinding 뿐만 아니라, 앞장의 예제에서 본 것처럼 컴포넌트를 처리할 수도 있고, 또, 이벤트를 위해 메소드를 바인딩 할수도 있다. 메소

드를 바인딩하는 부분은 이벤트를 처리하는 부분에서 다루기로 하자.

4.4 렌더러

jsf는 다른 프레임 워크와 달리 렌더링을 분리했다. 렌더링이란 특별한 개념이 아니고, 컴포넌트들이 클라이언트의 화면에 어떻게 디스플레이가 되는지를 지정한다는 것이다. 예를 들에 UICommand 컴포넌트는 브라우저에 버튼이나 링크를 나타내는 HtmlCommandButton 컴포넌트나 HtmlCommandLink 컴포넌트의 부모클래스이다. 이들 컴포넌트 클래스는 클라이언트의 화면에 어떻게 표시되는지에 대한 정보가 없다. 컴포넌트를 처리한 결과가 웹브라우저에 표시되는지, xml파일로 저장되는지 컴포넌트는 알 수 없다. 지정된 렌더러가 이 결과를 처리한다. 아래의 표는 jsf가 제공하는 컴포넌트들과 컴포넌트의 상위 클래스 그리고, 각 컴포넌트에 지정된 렌더러를 표로 나타낸것이다.

컴포넌트 (javax.faces.component.html 패키지)	상위클래스 (javax.faces.component)	렌더러 타입(단순한 문자열이다.)
HtmlCommandButton	UICommand	javax.faces.Button
HtmlCommandLink	UICommand	javax.faces.Link
HtmlDataTable	UIData	javax.faces.Table
HtmlForm	UIForm	javax.faces.Form
HtmlGraphicImage	UIGraphic	javax.faces.Image
HtmlInputHidden	UIInput	javax.faces.Hidden
HtmlInputSecret	UIInput	javax.faces.Secret
HtmlInputText	UIInput	javax.faces.Text
HtmlInputTextarea	UIInput	javax.faces.TextArea
HtmlMessage	UIMessage	javax.faces.Message
HtmlMessages	UIMessages	javax.faces.Messages
HtmlOutputFormat	UIOutput	javax.faces.Format
HtmlOutputLabel	UIOutput	javax.faces.Label
HtmlOutputLink	UIOutput	javax.faces.Link
HtmlOutputText	UIOutput	javax.faces.Text
HtmlPanelGrid	UIPanel	javax.faces.Grid
HtmlPanelGroup	UIPanel	javax.faces.Group
HtmlSelectBooleanCheckbox	UISeleceBoolean	javax.faces.CheckBox
HtmlSelectManyCheckbox	UISelectMany	javax.faces.CheckBox
HtmlSelectManyListbox	UISelectMany	javax.faces.Menu

HtmlSelectManyMenu	UISelectMany	javax.faces.ListBox
HtmlSelectOneListbox	UISelectOne	javax.faces.ListBox
HtmlSelectOneMenu	UISelectOne	javax.faces.Menu
HtmlSelectOneRadio	UISelectOne	javax.faces.Radio

렌더링을 분리함으로써, jsf로 개발된 컴포넌트들이 브라우저가 아닌 다른 클라이언트의 환경에 맞도록 적절히 변화될 수 있다.

4.5 이벤트

JSF가 제공하는 이벤트는 javax.faces.event패키지에 현재 다음의 3가지가 있다.

- ActionEvent
- ValueChangeEvent
- PhaseEvent

PhaseEvent는 JSF의 각 단계별 life cycle에 따라 발생할 수 있는 이벤트이다.

웹 어플리케이션을 개발하는 경우 ActionEvent와 ValueChangeEvent를 사용하게 될 것이다. ActionEvent는 submit이 발생할 수 있는 컴포넌트 즉, 명령버튼이나 링크에 사용한다. 그리고, ValueChangeEvent는 값의 변경이 가능한 컴포넌트들에 이용된다.

이벤트는 html form으로부터 submit이 일어나야만 발생한다는 사실에 유의하자. 자바스크립트처럼 클라이언트 환경에서 작동하는 것이 아니기 때문이다. 자바스크립트에 의한 submit이나 버튼에 의한 submit이 발생하는 경우 JSF는 지정된 ActionEvent가 있다면, 이벤트를 처리하고, 값이 변경될 수 있는 라디오버튼이나, 체크박스, 입력상자 등의 값이 변경되는 경우 지정된 ValueChangeEvent를 실행할 수 있다.

ActionEvent를 이용할 수 있는 컴포넌트는 javax.faces.component 패키지의 UICommand 클래스를 상속한 클래스들이다.

ValueChangeEvent를 이용할 수 있는 컴포넌트는 javax.faces.component 패키지의 UIInput 클래스를 상속한 HtmlInputHidden, HtmlInputSecret, HtmlInputText, HtmlInputTextarea, UISelectBoolean, UISelectMany, UISelectOne 들이다.

JSF는 이벤트를 처리하는 다음과 같은 두가지 방법을 제공한다.

하나는, 컴포넌트의 속성을 이용해서 메소드를 바인딩하는 것이다.

```
<h:commandButton value="SUBMIT버튼" action="success"
actionListener="#{Example1.actionOccurred}"/>
```

위의 명령버튼에 지정된 `actionListener` 속성은 버튼이나 링크가 가지는 속성이다. 여기에 바인딩된 문자열의 의미는 `Example1` 클래스의 `actionOccurred` 라는 메소드를 실행한다는 것이다. JSF의 스펙에 따라, `actionOccurred` 메소드는 `public` 이고, 리턴타입이 `void` 이며, 파라미터로 `ActionEvent`를 가져야 한다. 다음과 같이 될 것이다.

```
public void actionOccurred(ActionEvent evt){
...
}
```

두번째 방법은, `Listener` 인터페이스를 구현하면 된다.

`javax.faces.event.ActionListener` 인터페이스를 구현하면 된다.

```
package jsf.proj.event;

import javax.faces.event.*;

public class MyActionListener implements ActionListener{
...
public void processAction(ActionEvent evt){
...
}
...
}
```

이 때에는 태그를 사용하는 방법이 조금 달라진다.

위의 예에서 사용한 명령버튼에 `MyActionListener` 를 이용하기 위해서는 다음과 같이 변경한다.

```
<h:commandButton value="SUBMIT버튼" action="success">
<f:actionListener type="jsf.proj.event.MyActionListener"/>
</h:commandButton>
```

위와 같이 리스너를 구현한 클래스를 직접 만들어 이용할 수도 있다. 이 때 type속성으로 패키지명을 포함한 클래스이름을 지정해야 한다. 그러면 해당 인터페이스의 지정된 메소드가 수행될 것이다.

이와 같은 방식은 ValueChangeEvent를 처리할 때에도 동일하다. 또한, 여기서 설명한 두가지 방식은 Validator나 Converter를 사용할 때에도 위의 두가지 동일한 방법을 사용할 수 있다.

4.6 validator & converter

4.6.1 Validator

JSF가 기본적으로 제공하는 validator는 다음 세가지이다.

- DoubleRangeValidator
- LengthValidator
- LongRangeValidator

DoubleRangeValidator 와 LongRangeValidator는 값의 유효한 범위를 지정할 수 있는 방법을 제공하고, LengthValidator는 문자열의 길이의 유효한 범위를 지정한다. validation에서 에러가 발생하는 경우 ValidatorException이 발생한다.

위에서 설명한 이벤트의 경우처럼 validator를 이용하는 방법도 두가지가 있다.

하나는 validator 속성을 이용하여 validator 메소드를 바인딩하는 것이고, 두번째는 javax.faces.validator패키지의 Validator 인터페이스를 구현하는 것이다. Validator 인터페이스를 구현하여 validator를 사용하는 경우 faces-config.xml에 정의된 validator에 대한 설정이 필요하다.

validator 메소드 역시 JSF의 스펙에 따라 public 이고, 리턴타입은 void이고, 파라미터로는 FacesContext, UIComponent, Object를 가져야 한다.

다음과 같은 형태가 될것이다. Validator 인터페이스의 파라미터도 역시 동일하다.

```
public void validateTest(FacesContext context, UIComponent comp, Object value){  
...  
}
```

현재 request가 가지는 FacesContext와 이 메소드를 호출하는 컴포넌트 그리고, 평가될 값이 Object 형태로 넘어올 것이다.

4.6.2 Converter

Converter는 웹 페이지에서 처리되는 문자열을 적절한 타입의 값으로 변환하는 역할을 한다. Converter 역시 converter 속성을 이용해서 다음은 JSF가 기본적으로 제공하는 converter 이다.

- BigDecimalConverter
- BigIntegerConverter
- BooleanConverter
- ByteConverter
- CharacterConverter
- DateTimeConverter
- DoubleConverter
- FloatConverter
- IntegerConverter
- LongConverter
- NumberConverter
- ShortConverter

converter는 사용하는 방법이 조금 틀리다. 메소드를 바인딩하는 방법은 사용되지 않고, javax.faces.convert 패키지의 Converter 인터페이스를 구현하여 이용한다. Converter 인터페이스는 다음과 같은 두 가지 메소드가 있다.

```
public Object getAsObject(FacesContext context,
                        UIComponent component,
                        String value)
public String getAsString(FacesContext context,
                        UIComponent component,
                        Object value)
```

getAsObject()메소드는 파라미터로 넘어오는 문자열 value 값이 설정되는 컴포넌트의 속성의 적절한 타입에 맞도록 문자열을 변환한다. 또, getAsString() 메소드는 컴포넌트의 값을 문자열로 변환한다. 적절한 타입으로 변환할 수 없을 경우, ConvertException이 발생한다.

4.7 navigation

이번에는 JSF에서의 웹 페이지의 이동에 대해서 알아보자. 어느 페이지에서 어느 페이지로 이동할지를 결정하는 모든 설정은 faces-config.xml에서 이루어진다. 웹 페이지에 submit 이 발생하는 경우 navigation handler에 action속성에 지정된 문자열 값이 건네어지고, 이 문자열 값으로 다음에 나타날 페이지가 결정된다. 만약, 주어진 action 속성의 문자열 값과 일치하는 navigation을 찾지 못하면, 현재 페이지를 다시 표시한다.

faces-config.xml에 페이지의 이동을 위한 navigation은 다음의 형식을 따라 설정된다.

```
<navigation-rule>
  <from-view-id>/from.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/to.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/from.jsp</from-view-id>
  <navigation-case>
    <from-outcome>fail</from-outcome>
    <to-view-id>/to_fail.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

경로를 표시하는 문자열은 / 로 시작해야 한다는것에 유의하자.

from.jsp 라는 페이지로부터 action 속성의 문자열 success가 반환된다면, from.jsp 페이지로부터 to.jsp라는 페이지로 request는 forward 된다. fail 이라는 문자열이 얻어진다면, to_fail.jsp 페이지로 forward 될 것이다.

from.jsp 페이지의 다음과 같은 버튼에 의해 action이 발생한다고 하면

```
...
<h:commandButton value="OK" action="#{Login.confirm}"/>
...
```

Login클래스의 confirm 메소드를 구현하자.

```

...
public class Login{
public String confirm(){
if (login ok) return "success";// 적절한 로직을 구현하자 !!!
else return "fail";
}
}
...

```

confirm 메소드가 반환하는 문자열에 따라 forward되는 페이지가 달라질 것이다.

위의 예에서, from-view-id 태그가 없을 수도 있다. 그러면 어떠한 페이지에서건, action 속성이 문자열 success 나 fail 을 반환하면 to-view-id에 지정된 url로 forward 될 것이다. from-view-id 태그에 와일드 카드 *를 사용하여도 결과는 마찬가지이다.

```

<navigation-rule>
    <from-view-id>*</from-view-id>
...
</navigation-rule>

```

다음과 같이 하나의 navigation-rule에 동일한 두 문자열이 얻어지는 경우를 보자.

from.jsp 페이지에 다음과 같이 버튼이 두개 있다고 하자.

```

...
<h:commandButton value="OK" action="#{Login.confirm}"/>
<h:commandButton value="OK" action="#{Login.check}"/>
...

```

각각의 메소드가 다음과 같다고 하자.

```

...
public class Login{
public String confirm(){
if (login ok) return "success";
else return "fail";
}
}

```

```

public String check(){
if (ok) return "success";
else return "chkFail";
}
...

```

navigation-rule에서는 confirm() 메소드에서 얻어지는 success 인지 check() 메소드에서 얻어지는 문자열인지 상관없이 to.jsp로 forward 될 것이다. 개발자가 check() 메소드가 반환하는 문자열이 success 이면 check.jsp로의 이동을 원한다면, 다음과 같이 from-action 태그를 이용하면 된다.

```

<navigation-rule>
    <from-view-id>/from.jsp</from-view-id>
    <navigation-case>
    <from-action>#{Example1.confirm}</from-action>
        <from-outcome>success</from-outcome>
        <to-view-id>/to.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
    <from-action>#{Example1.check}</from-action>
        <from-outcome>success</from-outcome>
        <to-view-id>/check.jsp</to-view-id>
    </navigation-case>
</navigation-rule>

```

from-action 태그에 지정된 문자열값은 클래스그이름과 메소드 이름을 나타낸다. 그러나 이 문자열이 메소드를 호출하는 것은 아니다. 단순히 success라는 문자열을 구분하는 키값 정도에 불과하다. 이렇게 함으로써, 각 버튼에 의해 발생하는 action을 이용해서 서로다른 페이지로 보내도록 할 수도 있다.

마지막으로, JSF는 기본적으로 forward로 페이지 이동을 한다.

redirect를 원한다면, redirect 태그를 지정하면 된다. 위의 예에서 맨 마지막의 경우 check.jsp 페이지로 redirect를 원한다면, 다음과 같이 수정한다.

```

<navigation-rule>

```

...

<to-view-id>/check.jsp</to-view-id>

<redirect/>

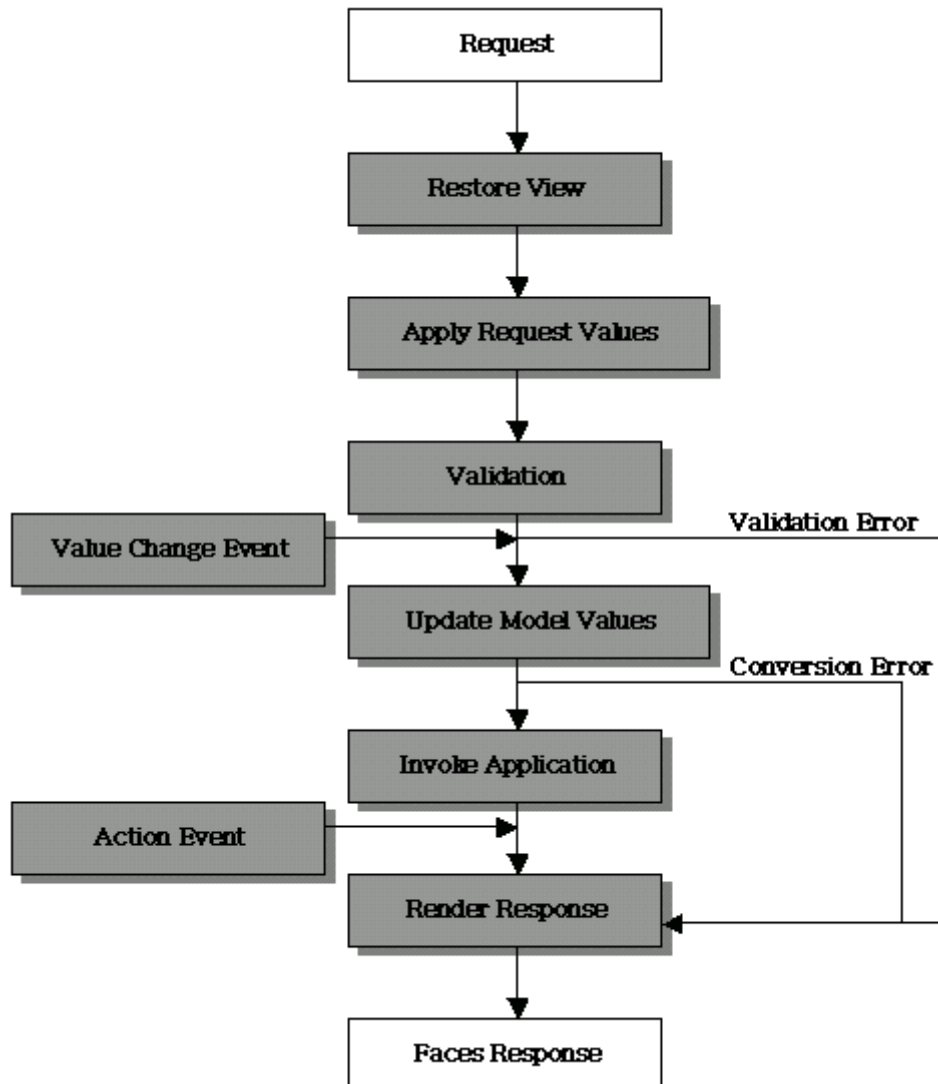
...

5. JSF Life Cycle

5.1 JSF Life Cycle

클라이언트의 request를 처리하고, response를 생성하기 위해 JSF는 내부적으로 많은 일들을 처리한다. request에 포함된 컴포넌트 트리를 구성하고, 파라미터로 넘어온 값들을 지정된 converter 나 validator 를 이용해서 유효한 값인가를 검사하고, 각 단계에서 필요한 이벤트를 처리한다. 아주 짧은 시간에 일어나는 이러한 작업단계들을 JSF는 아래와 같이 구분하고 있다. 이번장에서는 이러한 각 단계(Phase)에서 어떠한 작업들이 처리되는지와 이러한 처리작업 단계를 변경할 수 있는 immediate속성에 대해 알아보도록 한다.

가장 일반적인 JSF의 request처리는 다음의 단계를 거친다.



각각의 단계에 대해 알아보자.

1. Restore View

request를 처리하기 위한 FacesContext객체가 생성된다.

request에 포함된 JSF컴포넌트들의 컴포넌트 트리를 구성한다. 컴포넌트 트리는 UIViewRoot라는 클래스 인스턴스를 최고상위 노드로 하는 트리이다.

2. Apply Request Values

컴포넌트의 변경되는 값들을 임시로 관리한다. validation, conversion과정을 거치지 않았기 때문에 변경될 값들은 아직 컴포넌트에 적용되지 않았다.

3. Validation

각 컴포넌트마다 지정된 validation이 있는 경우 validation을 수행한다. validation에 실패하면 validation에러가 발생한다. 값의 변경이 발생하는 경우를 위해 컴포넌트가 Value Change Event를 지정했다면 Validation이 끝난후 이벤트가 수행된다. 물론, validation이 성공적으로 수행된 경우이다.

4. Update Model Values

모든 값들이 유효하므로 이 단계에서 컴포넌트의 값들을 변경한다. 이제 컴포넌트들이 가지는 이전값들은 없어지고, 새로운 값들이 설정된다. 이 새로운 값들을 컴포넌트의 값으로 적절히 변환될 수 없는 경우 conversion에러가 발생한다.

5. Invoke Application

에러가 발생하지 않고 이 단계까지 오면, 모든 컴포넌트의 값들이 에러없이 성공적으로 갱신되었다는 뜻이 되겠다. 이제 버튼이나 링크의 지정된 ActionEvent 있으면, 이 이벤트가 수행된다.

6. Render Response

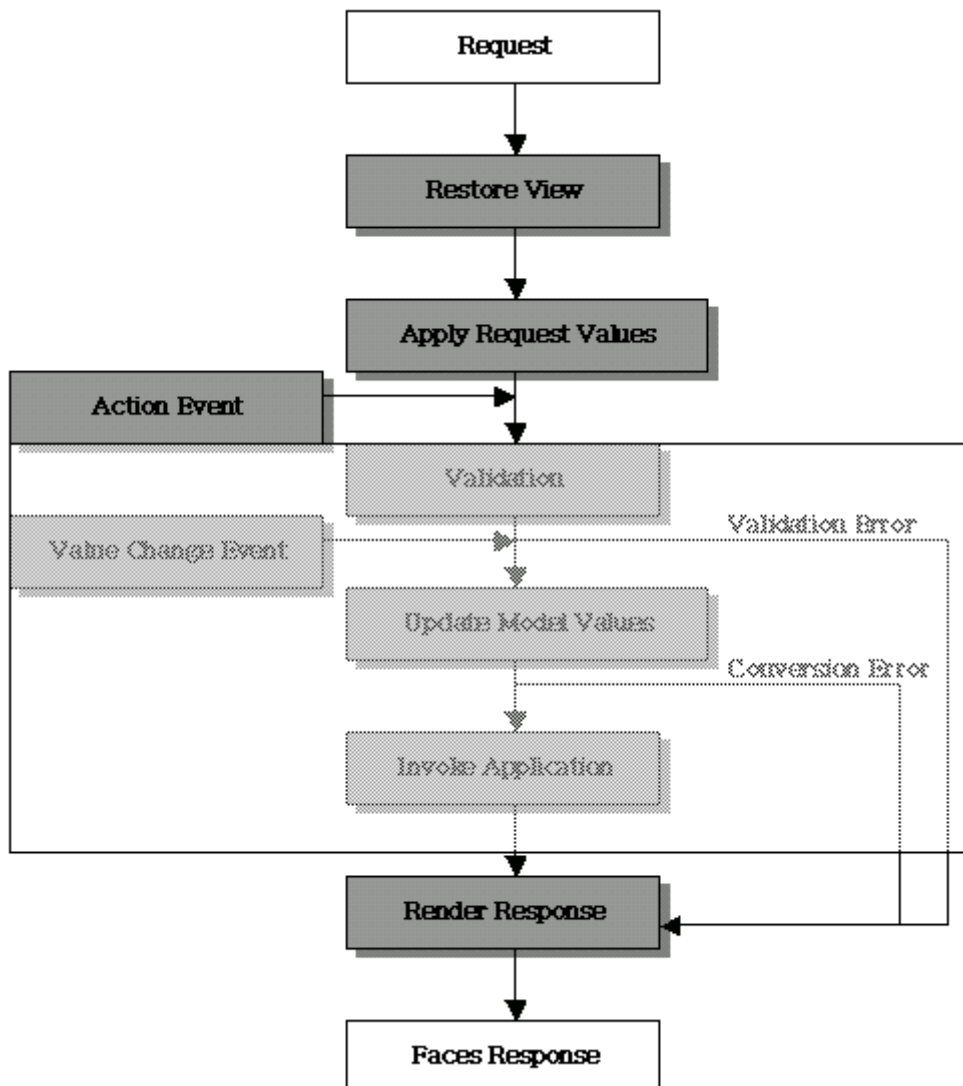
현재 상태를 저장하고, 이동할 페이지로 request가 forward된다. JSF는 기본적으로 모든

submit을 forward한다. forward가 필요치 않은 경우 faces-config.xml의 navigation에 redirect를 지정하면 된다.

5.2 immediate 속성

immediate 속성은 버튼이나 링크 컴포넌트 그리고 값이 변경될 수 있는 입력상자, 라디오 버튼, 체크박스, 메뉴 등의 컴포넌트들만 가지고 있다. immediate속성은 true/false 를 값으로 가질 수 있으며, 이 속성이 지정되지 않는 경우는 false이다. immediate속성값이 false인 경우 JSF의 life cycle은 위의 그림과 동일한 과정을 따른다. 그러나, 해당 컴포넌트에 immediate속성값이 true로 설정되면 조금 복잡하다.

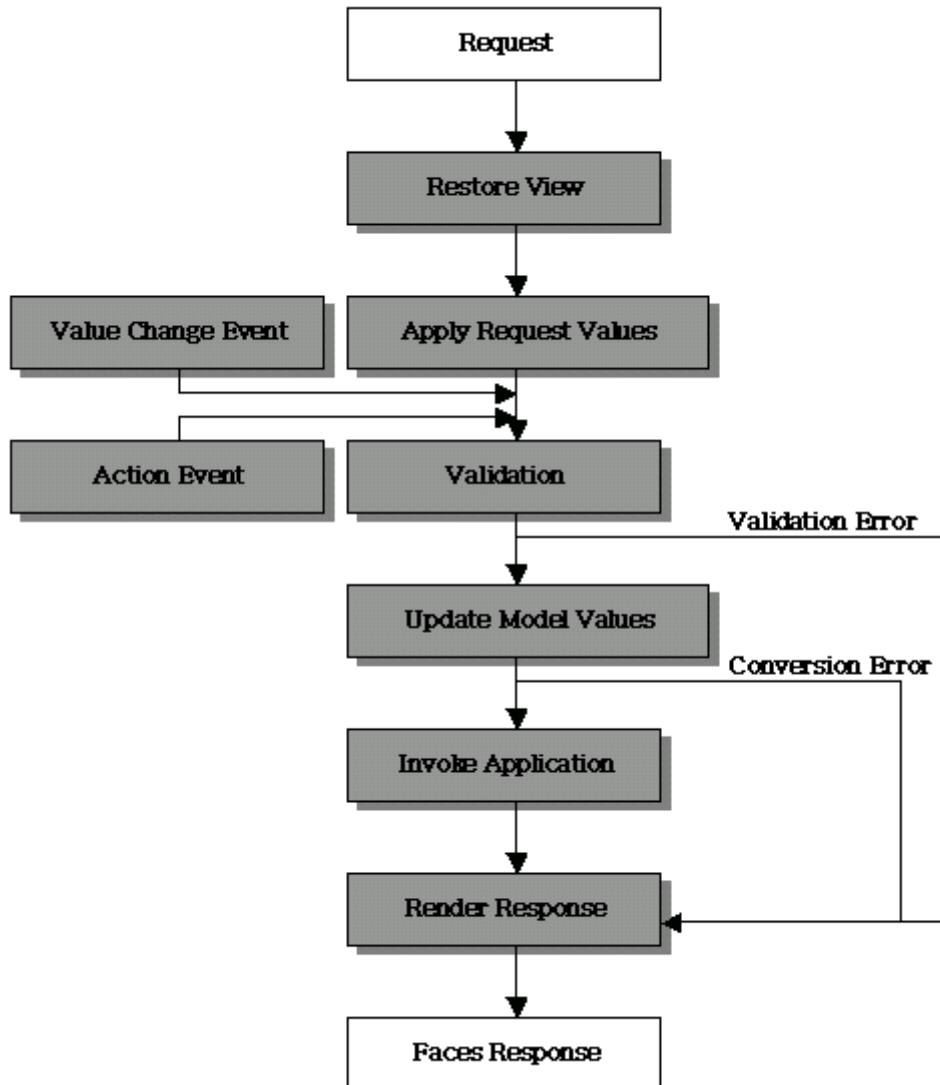
예를 들어 어떤 웹페이지에 입력상자와 명령버튼이 있다고 하자. 입력상자의 immediate속성은 false이고, 명령버튼의 immediate속성이 true이면 아래와 같은 과정을 거치게 될 것이다.



ActionEvent란 쉽게 말하면, 웹 브라우저에서 버튼이나 링크를 클릭해서 submit이 발생하는 경우이다. 따라서, ActionEvent가 발생하면, 진행과정을 모두 멈추고, 결과를 브라우저에 출력하기 위해 html을 생성하는 render response상태로 넘어가게 된다. 물론 입력박스의 지정된 validation이나 Value Change 와 같은 이벤트는 발생하지 않는다. 위의 불투명하게 터리된 과정들을 건너 뛰기 때문이다. 그러나, Action Event가 발생하지 않는다면 위의 불투명하게 처리된 과정들을 모두 수행하게 된다.

이번에는 값이 변경될 수 있는 컴포넌트인 입력상자의 immediate속성값이 true라고 가정해 보자.

그러면 다음의 그림과 같이 처리된다.

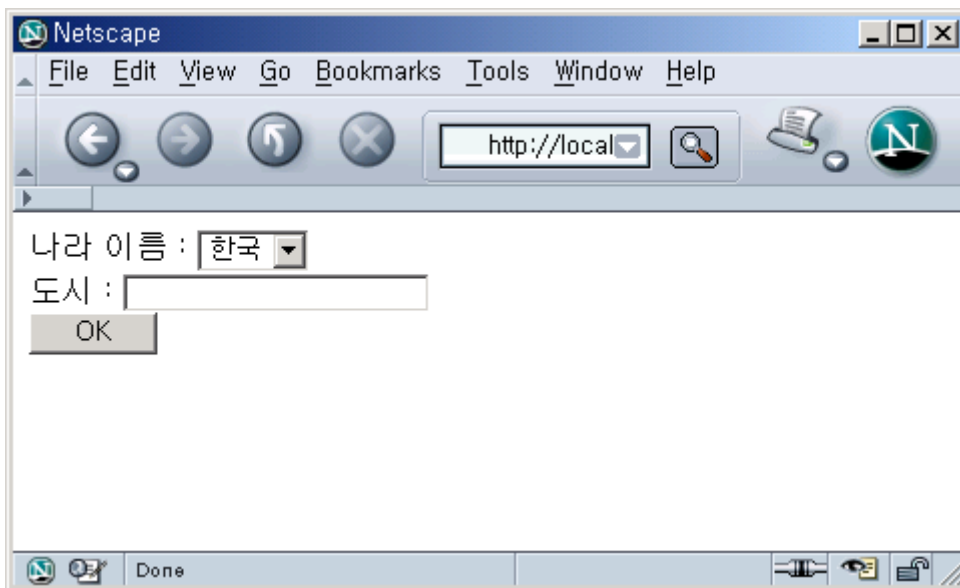


위의 그림은 입력상자와 버튼으로 구성된 웹 페이지의 두 컴포넌트가 모두 immediate 속성을 true로 가질 때 JSF가 request를 처리하는 과정을 나타내는 것이다.

immediate속성은 지정된 컴포넌트의 이벤트 처리시점을 옮기게 함으로써 JSF의 기본적인 lifecycle을 개발자들이 요구하는데로 조정될 수 있는 유연성을 제공한다고 볼 수 있다.

그러면, 어떤 경우에 immediate 속성을 써야 할까 ?

예를 들어 다음과 같이 국가를 선택하고, 도시명을 적어야 하는 페이지가 있다고 가정하자.



우리는 이 페이지에서 국가를 선택하고, 도시명을 적도록 한다. 이 페이지의 소스는 다음과 같다.

```
<!--
    /jsp/example/cycle_test.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="testForm">
        나라 이름 :
        <h:selectOneMenu value="KO" onchange="submit()"
          valueChangeListener="#{LifeCycleTest.valueChanged}">
          <f:selectItem itemValue="KO" itemLabel="한국" />
          <f:selectItem itemValue="JP" itemLabel="일본"/>
          <f:selectItem itemValue="USA" itemLabel="미국"/>
        </h:selectOneMenu>
        도시 : 
        <h:commandButton value="OK" />
      </h:form>
    </f:view>
  </body>
</html>
```

```

    </h:selectOneMenu>
    <br>
    도시 : <h:inputText id="in1" value="#{LifeCycleTest.city}" required="true"/><br>
    <h:commandButton value="OK" action="success"/>
    <br>
    <h:message for="in1"/>
  </h:form>
</f:view>
</body>
</html>

```

selectOneMenu 태그는 메뉴를 만들어내는 태그이다. 메뉴에서 국가이름을 선택하면 valueChanged라는 메소드가 호출되도록 되어 있고, submit이 발생한다. immediate 속성이 어떤 역할을 하는지를 살펴보는 보기 위해 아직 사용해보지 않은 컴포넌트인 메뉴박스를 사용하였다. 각 컴포넌트의 사용법은 다음장에서 자세히 설명하도록 한다.

메뉴에서 국가이름을 바꿀때마다, LifeCycleTest 클래스의 valueChanged 메소드가 호출되도록 되어 있다. 또, 도시이름을 입력받는 입력상자는 반드시 입력을 하도록 하는 required 속성이 true로 되어 있다. 아무값을 입력하지 않으면, message 컴포넌트에 의해 에러메시지가 브라우저에 나타난다. message 컴포넌트의 for 속성이 가리키는 값은 컴포넌트의 id이다.

위의 페이지를 위해 사용될 LifeCycleTest.java 소스는 다음과 같다.

```

package jsf.proj.example;

import javax.faces.event.*;
import javax.faces.context.*;

public class LifeCycleTest{

    private String city;

    public void valueChanged(ValueChangeEvent evt){
        System.out.println("value change event가 발생되었습니다.");
    }
}

```

```
public String getCity() {
    return city;
}
public void setCity(String city) {
    this.city = city;
}
}
```

위의 bean을 보면, 도시명을 변수로 가지고, 메뉴 선택시 발생하는 이벤트를 처리하는 valueChanged메소드를 가지고 있다.

ok버튼을 클릭하면 도시명을 출력하는 다음의 페이지로 이동한다.

```
<!--
    /jsp/example/cyce_test_dest.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="testForm">
        도시이름 : <h:outputText value="#{LifeCycleTest.city}"/><br>
      </h:form>
    </f:view>
  </body>
</html>
```

이제 마지막으로 navigation과 bean을 faces-config.xml에 등록하자.

```
<?xml version="1.0" encoding="euc-kr"?>

<!DOCTYPE faces-config PUBLIC
```

```

"-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
"http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
...

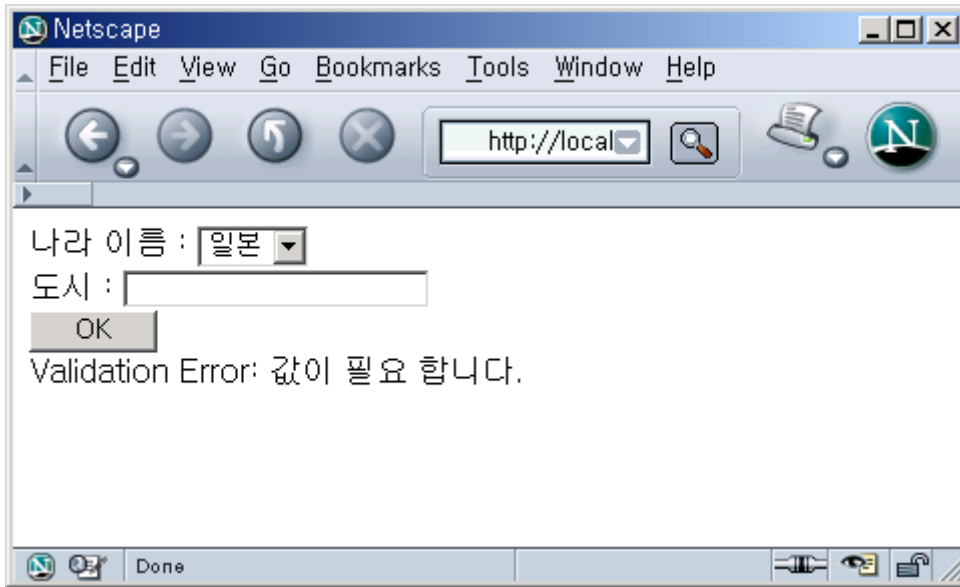
    <managed-bean>
        <managed-bean-name>LifeCycleTest</managed-bean-name>
        <managed-bean-class>jsf.proj.example.LifeCycleTest</managed-bean-
class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>

    <navigation-rule>
        <from-view-id>/jsp/example/cycle_test.jsp</from-view-id>
        <navigation-case>
            <from-outcome>success</from-outcome>
            <to-view-id>/jsp/example/cycle_test_dest.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>
...
</faces-config>

```

이제 ok버튼을 클릭하면 cycle_test_dest.jsp페이지로 이동을 하고 도시이름을 나타낼 것이다.

이제 cycle_test.jsp페이지를 불러서 국가이름을 변경하면 다음과 같은 에러메시지가 발생한다.



이 페이지의 국가 이름을 변경하는 경우를 보면 다음과 같은 과정을 거칠 것이다.

- 메뉴에 지정된 `valueChanged` 메소드가 호출되고 처리된다.
- `submit`이 발생한다. 현재 페이지를 갱신할 것이다.
- 도시이름을 입력받는 컴포넌트에 값이 없으므로, `message`태그에 의해 `context`내의 에러메시지가 출력된다.

`message` 태그는 `FacesContext`내에서 발생한 에러를 출력하는 역할을 할 뿐이다. `message` 태그를 없애고, 에러가 화면에 나타나지 않는다고 해서 에러가 발생하지 않은 것은 아니라는 얘기다. 또, 도시이름을 입력하지 않고, `ok`버튼을 클릭하면 페이지는 이동하지 않는다. 여전히 `validation error`가 발생하고 있기 때문이다.

JSF를 이용해서 웹 어플리케이션의 개발시, 위와 같은 유사한 상황들이 많이 발생할 것이다.

이러한 경우를 해결하기 위해 `immediate`속성이 필요하다.

`cycle_test.jsp` 메뉴부분을 다음과 같이 변경한다. 위치는 상관없다.

```

...
<h:selectOneMenu value="KO"
onchange="submit()" valueChangeListener="#{LifeCycleTest.valueChanged}"
immediate="true"
>

```

...

메뉴의 `immediate` 속성을 `true`로 설정한다. 그러나 위와 같이 해도 여전히 JSF의 life cycle은 진행되어서, 입력상자의 validation error는 계속 발생한다. 우리는 단지 `immediate` 속성을 이용해서 이벤트가 처리되는 시점을 앞당겼을 뿐이다. 입력상자를 나타내는 컴포넌트는 JSF의 life cycle을 수행하게 되므로 여전히 validation error가 발생할 것이기 때문이다.

따라서, `LifeCycleTest` bean의 `valueChanged`메소드가 실행되는 경우, 강제로 `render response`상태로 넘어가도록 하여 validation을 건너뛰도록 한다.

`LifeCycleTest` bean을 다음과 같이 수정한다.

```
...
public void valueChanged(ValueChangeEvent evt){
    System.out.println("value change event가 발생되었습니다.");
    FacesContext fc=FacesContext.getCurrentInstance();
    fc.renderResponse();
}
...
```

`FacesContext`의 `renderResponse`메소드는 호출되는 순간 바로 현재 페이지의 나머지 단계들을 생략하고 `render response` 상태로 넘긴다. 따라서 validation 에러를 피할 수 있다.

명령버튼이나 링크에 의한 액션은 이러한 메소드 호출이 필요없다. 액션이 발생하면 바로 `render response`상태로 넘어간다.

JSF에서 `immediate` 속성을 이해하는 것이 조금 까다로운 부분일 것이다. 나름대로 여러가지 테스트를 해보면서 감을 잡기 바란다.

이 장에서의 설명을 요약하면 다음과 같다.

- `immediate` 속성이 JSF의 life cycle 에서 이벤트가 발생하는 시점을 조절한다.
- `immediate` 속성이 `true`인 명령버튼이나 링크가 클릭되는 경우 해당 페이지의 나머지 모든 과정이 생략되고, 화면 표시를 위한 `render response`상태로 넘어가게 된다.
- Value Change Event가 발생할수 있는 컴포넌트들, 예를들면 메뉴, 입력상자, 라디오버튼등의 `immediate` 속성이 `true`이면 이벤트가 발생하는 시점이 조절되고,

render response로 넘어가기 위해서는 FacesContext의 renderResponse()메소드를 호출해야 한다.

6. JSF 컴포넌트

6.1 Html 컴포넌트

이번장에서는 JSF에서 제공되는 기본 컴포넌트에 대한 설명을 하도록한다. 모든 컴포넌트는 다음 세가지 속성을 각각 가지고 있다는 것을 명심하자.

- id
- rendered
- binding

6.2 HtmlCommandButton

HTML input 요소를 생성하는 컴포넌트이다. 다음의 속성들을 이용할 수 있다.

구분	속성
JSF	id, rendered, binding, action, actionListener, immediate, value
html	accesskey, alt, dir, disabled, image, lang, readonly, tabindex, title, type
javascript	onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect
CSS	style, styleClass

이 컴포넌트는 앞의 예제에서 많이 이용하였으므로, 중요한 속성에 대해서만 설명을 하도록 하겠다. type 속성은 submit 이거나 reset 값이다. type 속성을 생략하면, 기본적으로 submit이 된다. action 속성이 반환하는 문자열은 JSF의 navigation handler에 의해 faces-config.xml 설정파일의 navigation-rule 태그의 from-outcome 값이 될 것이다.

actionListener 속성에 지정되는 메소드는 public 이고, 리턴타입이 void 이며, javax.faces.event.ActionEvent 개체를 파라미터로 가져야 한다.(ActionEvent 개체는 이벤트를 발생한 컴포넌트를 GetComponent() 메소드를 이용하여 얻을 수 있다.

6.3 HtmlCommandLink

HTML “a” 앵커를 생성한다.

구분	속성
----	----

JSF	id, rendered, binding, action, actionListener, immediate, value
html	accesskey, charset, coords, dir, hreflang, rel, rev, shape, tabindex, target, title, type
javascript	onblur, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup
CSS	style, styleClass

HtmlCommandLink컴포넌트와 위에서 설명한 HtmlCommandButton컴포넌트는 똑같이 UICommand클래스를 상속한다. HtmlCommandLink컴포넌트역시 HtmlCommandButton과 같은 방법으로 이벤트를 처리할 수 있도록 action 속성과 action Listener 속성을 지원한다.

이 컴포넌트가 생성하는 html을 살펴보도록 하자. 다음의 예를 보자.

```

...
<h:form id="testForm">
<h:commandLink id="link1" action="success">
<h:outputText value="CLICK !!"/>
<f:param name="name" value="kimsk"/>
</h:commandLink>
</h:form>
...

```

위의 commandLink태그는 다음과 같은 자바스크립트를 생성하도록 되어있다. 위에 사용된 param 태그는 파라미터를 설정하는 core 컴포넌트이다. 브라우저의 소스보기를 선택하여 소스를 보면, 다음과 같은 내용이 생성되어 있음을 알수 있다.

파라미터 값이 자바스크립트의 onclick이 설정된다.

```

<a id="testForm:link1 "
  href="#"
  onclick="document.forms['testForm']['testForm:link1'].value='testForm:link1';
  document.forms['testForm']['name'].value='kimsk';
  document.forms['testForm'].submit(); return false;">CLICK !!</a>

```

HtmlCommandLink 는 내부적으로 위와 같은 자바스크립트를 생성하여, submit을 처리한다.

6.4 HtmlForm

HTML form 요소를 생성하는 컴포넌트이다.

구분	속성
JSF	id, rendered, binding
html	accept, acceptcharset, dir, enctype, lang, target, title
javascript	onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onreset, onsubmit
CSS	style, styleClass

form의 method 속성은 POST로 설정된다.

6.5 HtmlGraphicImage

html의 img태그를 생성한다.

구분	속성
JSF	id, rendered, binding, url, value
html	alt, dir, height, width, ismap, lang, longdesc, title, usemap
javascript	onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup
CSS	style, styleClass

url 속성을 이용하여 상대경로나 절대경로로 이미지파일명을 지정하면 된다. 다음과 같이 사용할 수 있다.

```
<h:graphicImage url="images/img1.jpg" alt="이미지 예제입니다." />
```

6.6 HtmlInputHidden

HTML input 태그의 타입이 hidden인 요소를 만든다.

구분	속성
JSF	converter, id, rendered, binding, immediate, required, validator, value, valueChangeListener
html	없음
javascript	없음

CSS	없음
-----	----

html 의 input type이 hidden인 html을 생성한다.

```
<h:inputHidden id="hid1" value="test"/>
```

위의 내용은 아래와 같이 html로 번역된다.

```
<input id="myForm:hid1" type="hidden" name="myForm:hid1" value="test" />
```

6.7 HtmlInputSecret

html 의 input type이 password인 html을 생성한다.

구분	속성
JSF	converter, id, rendered, binding, immediate, required, validator, value, valueChangeListener
html	accesskey, alt, dir, disabled, lang, maxlength, readonly, redisplay, size, tabindex, title
javascript	onblur, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect
CSS	style, styleClass

html 의 input type이 password인 html을 생성한다.

```
<h:inputSecret />
```

위의 소스는 아래와 같이 번역된다.

```
<input type="password" name="myForm:_id1" value="" />
```

6.8 HtmlInputTextarea

HTML textarea 요소를 생성한다.

구분	속성
JSF	converter, id, rendered, binding, immediate, required, validator, value, valueChangeListener
html	accesskey, cols, dir, disabled, lang, readonly, rows, tabindex, title
javascript	onblur, onchange, onclick, ondblclick, onfocus, onkeydown,

	onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect
CSS	style, styleClass

```
<h:inputTextarea id="ta1" cols="10" rows="10" />
```

위의 소스는 아래와 같이 번역된다.

```
<textarea id="myForm:ta1" name="myForm:ta1" cols="10" rows="10"></textarea>
```

6.9 HtmlInputText

구분	속성
JSF	converter, id, rendered, binding, immediate, required, validator, value, valueChangeListener
html	accesskey, alt, dir, disabled, lang, maxlength, readonly, size, tabindex, title
javascript	onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onselect
CSS	style, styleClass

HtmlInputHidden, HtmlInputSecret, HtmlInputTextarea와 여기서 설명하는 HtmlInputText 컴포넌트는 모두 UInput 클래스를 상속 받았다. 따라서 여기서 설명하는 컴포넌트의 주요속성은 위의 4개의 컴포넌트에 모두 적용된다.

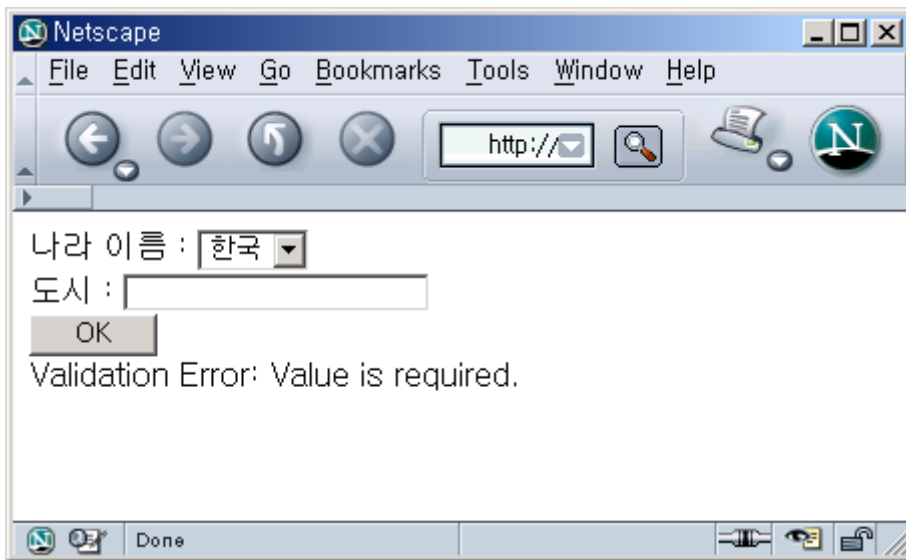
UInput 클래스가 가지는 주요속성은 다음과 같다.

- required 속성은 이 컴포넌트의 value 값을 반드시 입력받도록 한다.
- submit 이 발생하는 경우 이 컴포넌트의 값이 변경되었을 경우 ValueChangeEvent를 발생 하도록 할 수 있다.
- validator 를 이용하여 유효한 값이 입력되었는지 체크 할 수 있다.
- converter 를 이용하여, 값을 적절한 타입으로 변경할수 있다.

6.10 HtmlMessage

JSF가 제공하는 에러메시지를 출력하는 컴포넌트이다.

구분	속성
JSF	id, rendered, for, binding, showDetail, showSummary, title, tooltip,
html	없음
javascript	없음
CSS	errorClass, errorStyle, fatalClass, fatalStyle, infoClass, infoStyle, style, styleClass, warnClass, warnStyle



위의 그림에서 도시이름을 입력하지 않고, ok버튼을 클릭하는 경우 Validation error가 발생한다. 이 에러를 브라우저에 표시하는 역할을 하는 것이 HtmlMessage 컴포넌트가 수행한다.

```

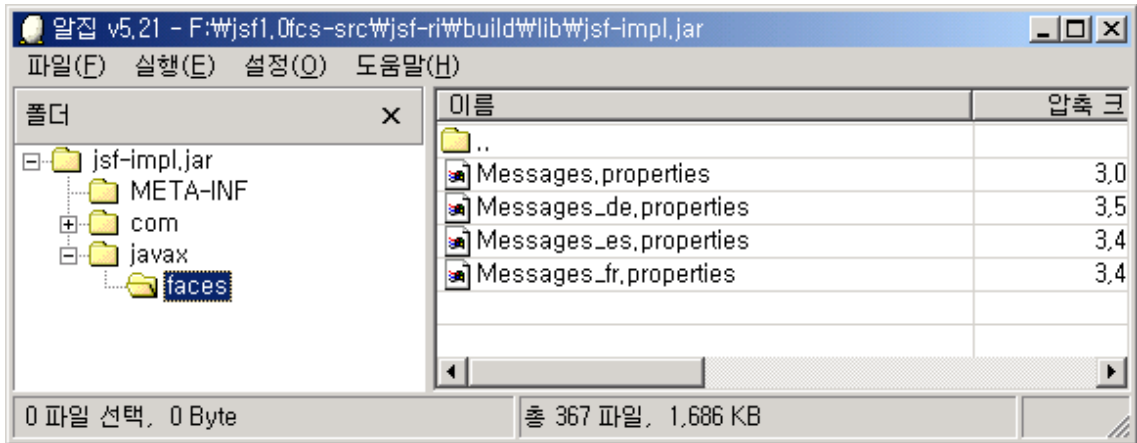
...
도시 : <h:inputText id="in1" value="#{LifeCycleTest.city}" required="true"/><br>
<h:commandButton value="OK" action="success"/><br>
<h:message for="in1"/>
...

```

message 태그가 가지는 속성 for는 컴포넌트의 id를 나타낸다. 어느 컴포넌트에서 발생한 에러를 표시할것인지를 나타내는 것이다. 따라서, message 태그를 이용하는 경우 for 속성은 반드시 지정한다.

이러한 메시지들은 Messages.properties 파일에 있다.

기본적으로 JSF가 가지는 Messages.properties 파일은 jsf-impl.jar 파일에 있다.



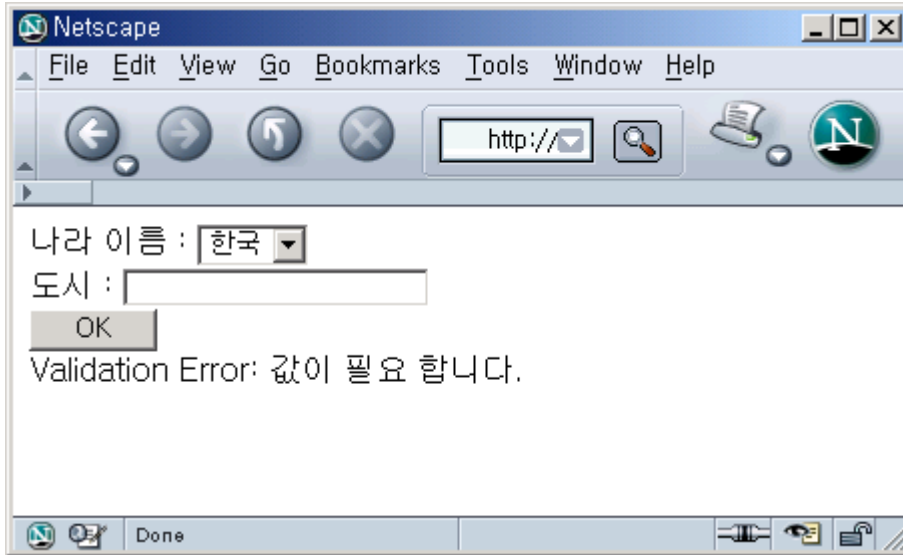
위의 그림과 같이 언어환경에 따라, 영어, 독일어, 스페인어, 프랑스로 된 메시지를 나타낼 수 있다. 한글로 된 메시지를 출력하고 싶으면, Messages_ko.properties 를 작성하고, 위치를 faces-config.xml에 작성하면, 사용가능하다. 이 때 한글은 자바 유틸리티 native2ascii를 이용하여 유니코드로 인코딩하는 작업이 필요하다. 이 책의 예제를 위해 build.xml에 native2ascii task를 작성해 놓았으므로 참고하도록 하자.

한글로 된 메시지를 이용하기 위해서 faces-config.xml에 다음과 같이 설정하도록 한다.

```
...
<faces-config>
  <application>
    <message-bundle>jsf.proj.resource.Messages</message-bundle>
    <locale-config>
      <default-locale>ko</default-locale>
      <supported-locale>de</supported-locale>
      <supported-locale>fr</supported-locale>
      <supported-locale>es</supported-locale>
    </locale-config>
  </application>
  ...
</faces-config>
```

다국어 환경의 웹 어플리케이션이 아니라면, <locale-config>는 생략해도 된다.

ant에 의해 Messages.properties 파일이 Messages_ko.properties 파일로 이름이 변경되고, 유니코드로 인코딩되어 톰캣에 배치된다. 위와 같이 설정을 하면, 한글로 된 에러메시지를 볼수 있다.



이 책의 예제소스에 Messages.properties 파일의 메시지들이 한글로 처리되어 있으므로 살펴보도록 하자.

6.11 HtmlMessages

구분	속성
JSF	globalOnly, id, rendered, binding, showDetail, showSummary, title, tooltip
html	없음
javascript	없음
CSS	errorClass, errorStyle, fatalClass, fatalStyle, infoClass, infoStyle, style, styleClass, warnClass, warnStyle

HtmlMessage와 동일한 역할을 하지만, 현재 JSF의 context에 포함되어 있는 모든 에러메시지가 나타난다. HtmlMessage 처럼 하나의 컴포넌트에 대한 에러메시지가 아니다. 에러들이 즉 나열되므로 디버깅시에는 사용할 만한 컴포넌트 인 것 같다.

6.12 HtmlOutputFormat

파라미터를 처리한 문자열을 출력한다.

구분	속성
JSF	converter, id, binding, rendered, value, escape, title
html	없음
javascript	없음
CSS	style, styleClass

다음의 예제를 살펴보자

```
<!--
    /jsp/example/example2.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
  <body>
    <f:loadBundle basename="jsf.proj.resource.Resources" var="bundle"/>
    <f:view >
      <h:form id="myForm" >
        <h:outputFormat value="#{bundle.testMessage}">
          <f:param value="자바"/>
          <f:param value="듀크"/>
        </h:outputFormat>
      </h:form>
    </f:view>
  </body>
</html>
```

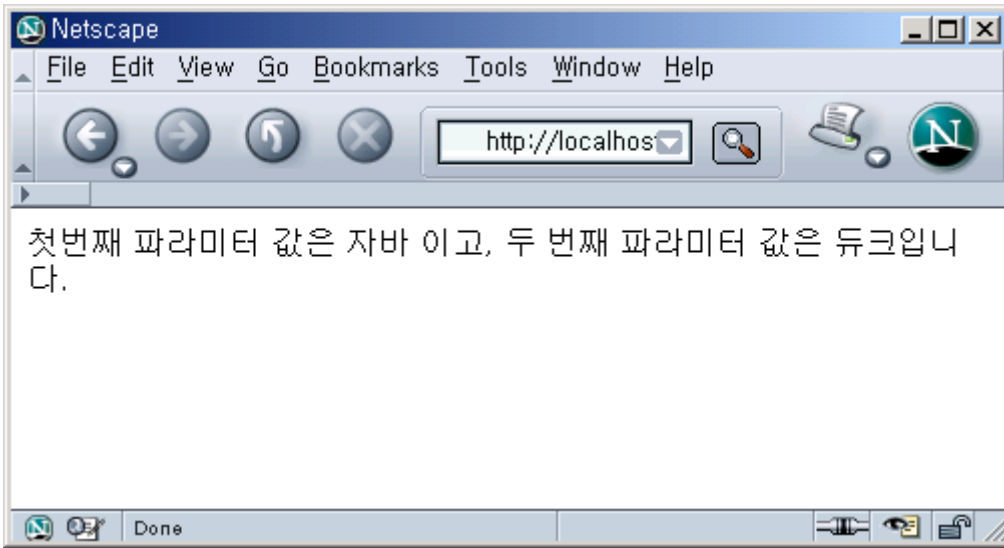
<f:loadBundle.../>은 웹 어플리케이션에 사용될 메시지나 기타 내용들을 모아놓은 properties 파일을 읽어 들이는데 이용하는 core 컴포넌트이다.

Messages.properties 파일과 마찬가지로, 위에서 지정된 Resources.properties 파일은 ant 에 의해 Resources_ko.properties 파일로 변경되에 배치된다. 한글역시 유니코드로 인코딩

되어야 브라우저에 제대로 나타난다.

파라미터로 넘겨질 값 '자바' 와 '듀크' 라는 문자열은 Resources.properties 파일의 다음과 같은 메시지의 {0} 과 {1} 에 각각 매핑되어 브라우저에 나타난다.

```
...
testMessage=첫번째 파라미터 값은 {0} 이고, 두 번째 파라미터 값은 {1}입니다.
..
```



다음과 같이 사용해도 결과는 동일하다. 꼭 Resources.properties를 사용할 필요는 없다.

```
...
<h:outputFormat value 첫번째 파라미터 값은 {0} 이고, 두 번째 파라미터 값은
{1}입니다.">
<f:param value="자바"/>
<f:param value="듀크"/>
..
```

6.13 HtmlOutputLabel

구분	속성
----	----

JSF	converter, id, binding, for, rendered, value, escape
html	accesskey, dir, lang, tabindex, title
javascript	onblur, onclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup
CSS	style, styleClass

HTML label 요소를 만들어 낸다. label 요소는 어느 컴포넌트를 위한 label인지를 지정해야 하는 for 속성을 가지고 있으므로 for 속성을 지정해야 한다. HTML스펙에 따라, 여기서 지정된 label문자열을 클릭하면 해당 컴포넌트로 포커스가 맞추어질 것이다.

6.14 HtmlOutputLink

구분	속성
JSF	converter, id, binding, rendered, value
html	accesskey, charset, coords, hreflang, rel, rev, shape, tabindex, target, title, type
javascript	onblur, onclick, onfocus, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup
CSS	style, styleClass

HTML 앵커를 만들어낸다.

```

<!--
    /jsp/example/example3.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
  <body>
    <f:view>
      <h:form id="myForm-result">
        <h:outputLink value="mailto:zzzccc90@yahoo.co.kr" >
        <h:outputText value="mail me"/>
      </h:form>
    </f:view>
  </body>
</html>

```

```

</h:outputLink><br>

<h:outputLink value="example2.jsp">
<h:outputText value="이전페이지로"/>
  <f:param name="id" value="zzzccc"/>
  <f:param name="name" value="kimsk"/>
</h:outputLink>
</h:form>
</f:view>
</body>
</html>

```

파라미터를 넘기고 싶은 경우 위와 같이 param 태그를 이용하여, 작성하면 된다.

6.15 HtmlOutputText

구분	속성
JSF	converter, id, binding, rendered, value,escape
html	title
javascript	없음
CSS	style, styleClass

value속성으로 지정된 문자열을 화면에 출력한다.

6.16 HtmlPanelGrid

구분	속성
JSF	id, binding, rendered, value, columns, frame, rules
html	bgcolor, cellpadding, cellspacing, dir, lang, title, width, summary
javascript	onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup,
CSS	columnClass, footerClass, headerClass, rowClasses, style, styleClass

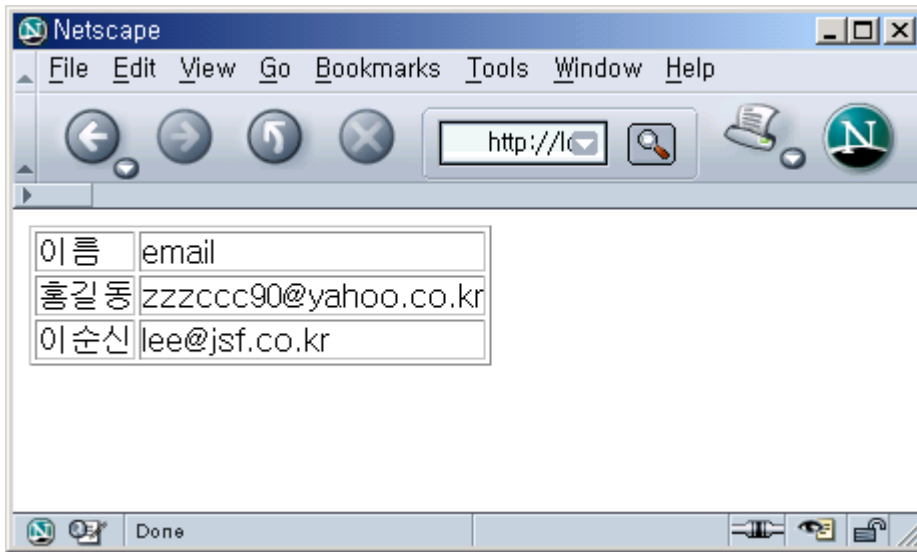
HTML 테이블을 생성하는 컴포넌트이다.

```

<!--
    /jsp/example/example4.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
  <body>
    <f:view>
      <h:form id="myForm" >
        <h:panelGrid id="pg1" title="panel grid test" columns="2" border="1"
cellpadding="0">
          <h:outputText value="이름"/>
          <h:outputText value="email"/>
          <h:outputText value="홍길동"/>
          <h:outputText value="zzzccc90@yahoo.co.kr"/>
          <h:outputText value="이순신"/>
          <h:outputText value="lee@jsf.co.kr"/>
        </h:panelGrid>
      </h:form>
    </f:view>
  </body>
</html>

```

columns속성에 의해 컬럼의 수가 결정된다. 컬럼의 수만큼 반복되면서 값이 매핑되고 다음과 같이 테이블로 만들어진다.



다음장에서 다음 core컴포넌트의 facet을 이용해서 테이블의 머리말과 꼬리말을 설정할 수도 있다.

```

<!--
        /jsp/example/example4.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
  <body>
    <f:view>
      <h:form id="myForm" >
        <h:panelGrid id="pg1" title="panel grid test" columns="2" border="1"
cellpadding="0">

          <f:facet          name="header">
            <h:outputText  value="머리말입니다."/>
          </f:facet>

          <f:facet          name="footer">
            <h:outputText  value="꼬리말입니다."/>
          </f:facet>

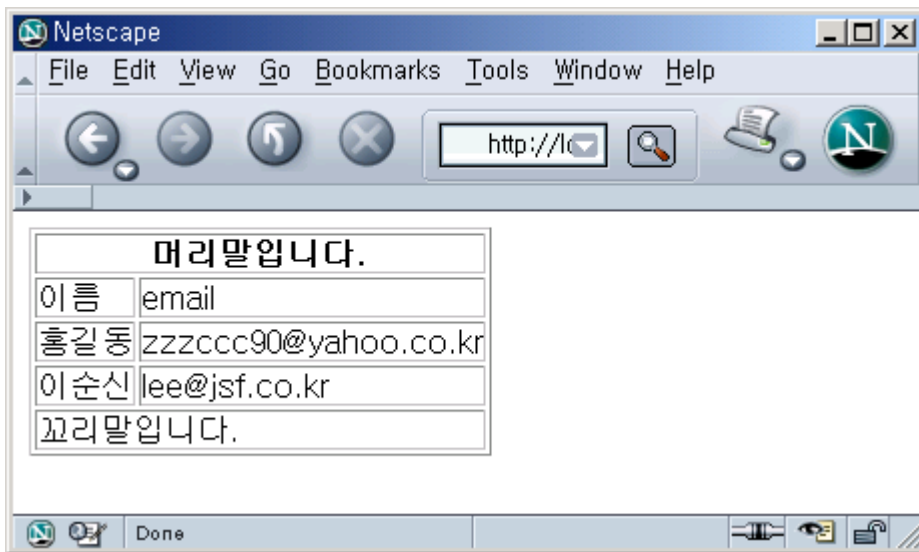
```

```

    <h:outputText value="이름"/>
    <h:outputText value="email"/>
    <h:outputText value="홍길동"/>
    <h:outputText value="zzzccc90@yahoo.co.kr"/>
    <h:outputText value="이순신"/>
    <h:outputText value="lee@jsf.co.kr"/>
  </h:panelGrid>
</h:form>
</f:view>
</body>
</html>

```

facet태그의 이름을 header , footer로 지정해서 테이블에 다음과 같은 작업이 가능하다. facet 태그는 facet 태그를 포함한 컴포넌트에서 보통 이러한 작업들을 처리한다.



6.17 HtmlPanelGroup

구분	속성
JSF	id, binding, rendered
html	없음
javascript	없음
CSS	style, styleClass

panelGroup으로 묶여진 컴포넌트들은 하나의 컬럼으로 처리된다.

```
<!--
    /jsp/example/example4.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
  <body>
    <f:view>
      <h:form id="myForm" >
        <h:panelGrid id="pg1" title="panel grid test" columns="2" border="1"
cellpadding="0">

          <f:facet          name="header">
            <h:outputText  value="머리말입니다."/>
          </f:facet>

          <f:facet          name="footer">
            <h:outputText  value="꼬리말입니다."/>
          </f:facet>

          <h:outputText value="이름"/>
          <h:outputText value="email"/>
          <h:outputText value="홍길동"/>
          <h:outputText value="zzzccc90@yahoo.co.kr"/>
          <h:outputText value="이순신"/>
          <h:outputText value="lee@jsf.co.kr"/>

          <h:panelGroup>
            <h:outputText value="세종대왕"/>
            <h:outputText value="king@jsf.co.kr"/>
          </h:panelGroup>

          <h:panelGroup>
            <h:outputText value="장보고"/>
          </h:panelGroup>
        </h:panelGrid>
      </h:form>
    </f:view>
  </body>
</html>
```

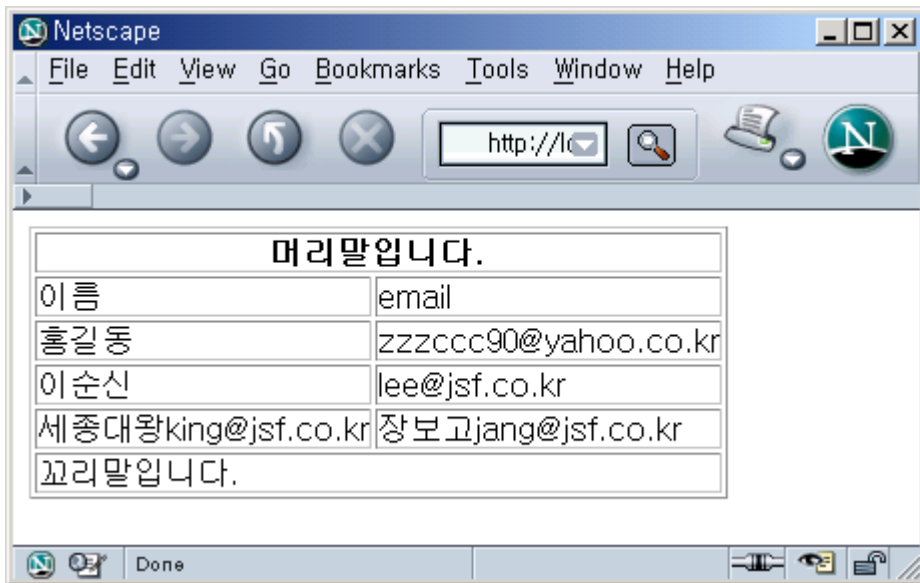
```

    <h:outputText value="jang@jsf.co.kr" />
  </h:panelGroup>

  </h:panelGrid>
</h:form>
</f:view>
</body>
</html>

```

html 소스를 보면 테이블을 처리하는 방법이 조금 복잡해 보일 것이다. panelGrid 컴포넌트나 panelGroup 컴포넌트의 테이블 생성기능을 사용하는 경우 다음에 나오는 DataTable 컴포넌트를 이용하는 경우가 더 자주 있을 것이라고 생각한다. 다음에 나오는 DataTable 컴포넌트는 데이터를 불러들여 동적인 테이블을 생성하는 강력한 컴포넌트이다. 데이터의 집합을 표현하기 위해 개발자는 루프를 돌릴 필요가 없다. JSF가 내부적으로 데이터 집합을 표현하기 위해 루프를 처리할 것이다.



6.18 HtmlDataTable 컴포넌트

구분	속성
JSF	first, id, binding, rendered, rows, value, var, frame, rules
html	bgcolor, border, cellpadding, cellspacing, dir, lang, summary, title, width

javascript	onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup
CSS	columnClasses, footerClass, headerClass, rowClasses, style, styleClass

DataTable 컴포넌트는 웹 어플리케이션에서 아마도 한번 이상은 기본적으로 이용되지 않을까 한다. 데이터베이스로부터 쿼리한 결과를 테이블로 표현하는 경우가 가장 보편적일 것이다. 지금까지 이러한 처리를 위해 사용된 루프문을 IDDataTable 컴포넌트에서는 JSF가 대신한다. 개발자는 JSF가 지원하는 데이터 집합의 타입에 맞는 데이터만 넘겨주면 JSF가 데이터 집합을 테이블 형태로 처리한다. 많은 사이트에서 이 DtaTable 컴포넌트를 확장하여, 보다 사용하기 좋은 형태로 만들어진 컴포넌트들을 많이 개발하고 있다. 예를 들어 각 컬럼의 링크를 클릭하면 그 컬럼의 값들을 sort 할 수 있도록 하는 기능등이 첨부되는 것이다. 여기서 설명하는 JSF의 DataTable 컴포넌트는 그런 기능을 아직 가지고 있지는 않다. 이 DataTable 컴포넌트를 이용하면 쉽게 개발할 수 있으리라 본다.

우선, 이 컴포넌트의 몇가지 속성을 살펴보자.

value 속성은 데이터 집합을 가진다. 이 컴포넌트에서 표현될 데이터는 JSF가 지원하는 것 이라야 한다. 그러면 어떤 데이터형들을 DataTable컴포넌트가 지원하는가 알아보자.

클래스	설 명
ArrayDataModel	java.lang.Object[] 타입의 데이터
ListDataModel	java.util.List 타입의 데이터 예) ArrayList, LinkdedList, Vector
ResultDataModel	javax.servlet.jsp.jstl.sql.Result타입의 데이터
ResultSetDataModel	java.sql.ResultSet 타입의 데이터
ScalarDataModel	Object타입의 데이터
SelectItem	core컴포넌트의 selectItem컴포넌트의 DataModel
SelectItemGroup	core컴포넌트의 selectItem컴포넌트의 DataModel

위의 표를 보면, 자바에서 흔히 사용하는 데이터 집합의 대부분을 지원한다. 위의 타입들이 value값으로 넘어오면 DataTable 컴포넌트는 내부적으로 루프를 돌려 데이터들을 표현할 수 있다. 여기서는 SelectItem과 SelectItemGroup을 제외한 나머지 DataModel에 대해 알아보고, SelectItem과 SelectItemGroup은 다음장의 core컴포넌트에서 설명하도록 한다.

우선, 간단하게 위에서 설명한 데이터 모델들을 테스트 해보기 위해 간단한 데이터 집합을 만들어보자.

```
package jsf.proj.example;

import java.sql.*;
import java.util.*;

public class DataTableExample{

    private Connection conn=null;
    private Statement stmt=null;
    private ResultSet rs=null;
    private List list=null;

    public DataTableExample(){
    }

    //String 배열을 생성하고 반환한다.
    public String[] getDataAsString(){
        String[] str=new String[5];
        str[0]="Korea";
        str[1]="America";
        str[2]="India";
        str[3]="Brazil";
        str[4]="Britain";
        return str;
    }

    // java.sql.ResultSet 을 생성하고 반환한다.
    public ResultSet getDataAsResultSet(){
        try{
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            Connection
conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/test?user=id&password=pa
ss");
```

```

        Statement stmt=conn.createStatement();
        rs=stmt.executeQuery("select * from test2");

        }catch(Exception e){e.printStackTrace();}
        return rs;
    }
//java.util.List 형의 데이터를 생성하고 반환한다.
// List내부에는 DataTableBean이라는 테스트용 클래스가 저장된다.
    public List getDataAsList(){
        list=new ArrayList();
        list.add(new DataTableBean("hong","HongGilDong"));
        list.add(new DataTableBean("kim","KimGabDol"));
        list.add(new DataTableBean("lee","LeeSunShin"));
        list.add(new DataTableBean("jang","JangGilSan"));
        list.add(new DataTableBean("robert","Robert Junior III"));
        return list;
    }
}

```

여기서 사용되는 데이터베이스의 테이블 test2를 위해 다음의 스크립트를 수행한다.

```

create table test2( email varchar(30), name varchar(20));
insert into test2 values('zzzccc90@yahoo.co.kr' , 'LeeSunShin');
insert into test2 values('lim@time.co.kr' , 'LimGukJung');
insert into test2 values('hong@hong.co.kr' , 'HoneGilDong');
insert into test2 values('jun@cool.co.kr' , 'PARK');
insert into test2 values('who@yahoo.co.kr' , 'WHO');

```

DataTableExample 클래스는 jsf페이지에 직접 사용되므로 faces-config.xml 에 다음과 같이 설정을 할 필요가 있다.

```

...
<managed-bean>
    <managed-bean-name>DataTableExample</managed-bean-name>
    <managed-bean-class>jsf.proj.example.DataTableExample</managed-bean-

```

```
bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
...
```

위의 소스에서 List타입의 데이터를 반환하는 세번째 메소드에 사용되는 DataTableBean클래스는 다음과 같다.

```
package jsf.proj.example;

public class DataTableBean{
    private String id;
    private String name;

    public DataTableBean(){
    }

    public DataTableBean(String id,String name){
        this.id=id;
        this.name=name;
    }
    public String getId(){
        return id;
    }
    public void setId(String id){
        this.id=id;
    }
    public String getName(){
        return name;
    }
    public void setName(String name){
        this.name=name;
    }
}
}
```

위의 클래스를 DataTable컴포넌트를 어떻게 사용하는지 살펴보자.

```
<!--
    /jsp/example/datatable_example.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="testForm">
        String배열을 읽어들인다.
        <br>태그에 의해 2번째 데이터부터 5번째 데이터까지만 읽는다.
        <h:dataTable value="#{DataTableExample.dataAsString}" var="data" first="2"
rows="5">
          <h:column>
            <h:outputText value="#{data}"/>
          </h:column>
        </h:dataTable>
      <hr>
      <br>
      <h:dataTable value="#{DataTableExample.dataAsResultSet}" var="data"
border="1">
        <f:facet name="header">
          <h:outputText value="데이터베이스를 통해 ResultSet을 넘겨받는다."/>
        </f:facet>

        <f:facet name="footer">
          <h:outputText value="데이터베이스를 통해 ResultSet을 넘겨받는다."/>
        </f:facet>

        <h:column>
          <f:facet name="header">
```

```
<h:outputText value="ID"/>
</f:facet>
<h:outputText value="#{data.email}"/>
</h:column>

<h:column>
<f:facet name="header">
<h:outputText value="NAME"/>
</f:facet>

<h:outputText value="#{data.name}"/>
</h:column>
</h:dataTable>
<hr>
java.util.List형의 데이터를 읽어들인다.
<h:dataTable value="#{DataTableExample.dataAsList}" var="data">
<h:column>
<h:outputText value="#{data.id}"/>
</h:column>
<h:column>
<h:outputText value="#{data.name}"/>
</h:column>
</h:dataTable>
</h:form>
</f:view>
</body>
</html>
```

제일 처음에 DataTable 예제를 보자.

```
...
<h:dataTable value="#{DataTableExample.dataAsString}" var="data" first="2"
rows="5">
<h:column>
<h:outputText value="#{data}"/>
</h:column>
```

```
</h:dataTable>
```

```
...
```

DataTable 컴포넌트는 DataTableExample 클래스의 getDataAsString() 메소드가 반환하는 문자배열을 값으로 가진다.

var이라는 속성은 dataTable태그내에서 value로 반환받은 값들을 사용할 때 쓰게될 일종의 별칭(alias)이다. 데이터 집합의 하나의 행에 포함된 값에 접근하고자 할 때 이용된다. first 속성은 제일 처음 표시하고자 하는 데이터 인덱스(번호)를 0을 기준으로 지정한다. 위의 예에서 first="2"라고 하였으므로 0을 기준으로 3번째 데이터부터 표시될 것이다.

rows는 first속성이 지정한 값을 기준으로 해서 몇 개의 열을 표시할 것인지를 지정한다. 위의 예에서는 5개이상이라도 5개의 열을 표시한다.

따라서, 위의 경우 3번째 데이터부터 5개까지의 데이터 행을 표시한다.

만약, 위의 결과 배열의 3번째 배열 요소만 표현하고 싶다면, outputText의 value값을 다음과 같이 표현하면 된다.

```
<h:outputText value="#{DataTableExample.dataAsString[2]}/>
```

이렇게 하면, 결과는 first와 rows에 의해 정해진 횟수만큼 루프를 돌면서, 배열의 3번째 요소만(0부터 시작하므로) 루프의 횟수만큼 표현될 것이다.

column컴포넌트는 DataTable컴포넌트가 루프를 돌면서 생성하는 테이블의 하나의 컬럼을 나타낸다.

```
...
<h:dataTable value="#{DataTableExample.dataAsResultSet}" var="data" border="1">
  <f:facet name="header">
    <h:outputText value="데이터베이스를 통해 ResultSet을 넘겨받는다."/>
  </f:facet>

  <f:facet name="footer">
    <h:outputText value="데이터베이스를 통해 ResultSet을 넘겨받는다."/>
  </f:facet>

  <h:column>
    <f:facet name="header">
      <h:outputText value="ID"/>
    </f:facet>
    <h:outputText value="#{data.email}"/>
  </h:column>
</h:dataTable>
```

```
</h:column>

<h:column>
<f:facet          name="header">
<h:outputText    value="NAME"/>
</f:facet>

<h:outputText value="#{data.name}"/>
</h:column>
</h:dataTable>

...
```

위의 예는 두번째 DataTable컴포넌트 이다. <f:facet name="header">와 <f:facet name="footer">는 각각 생성되는 테이블의 머리말과 꼬리말을 표시한다.

또, 컬럼 태그안에 들어있는 <f:facet...>는 각각 그 컬럼의 제목을 표시하고 있다. 이 DataTable의 value값은 ResultSet이 반환되는데,

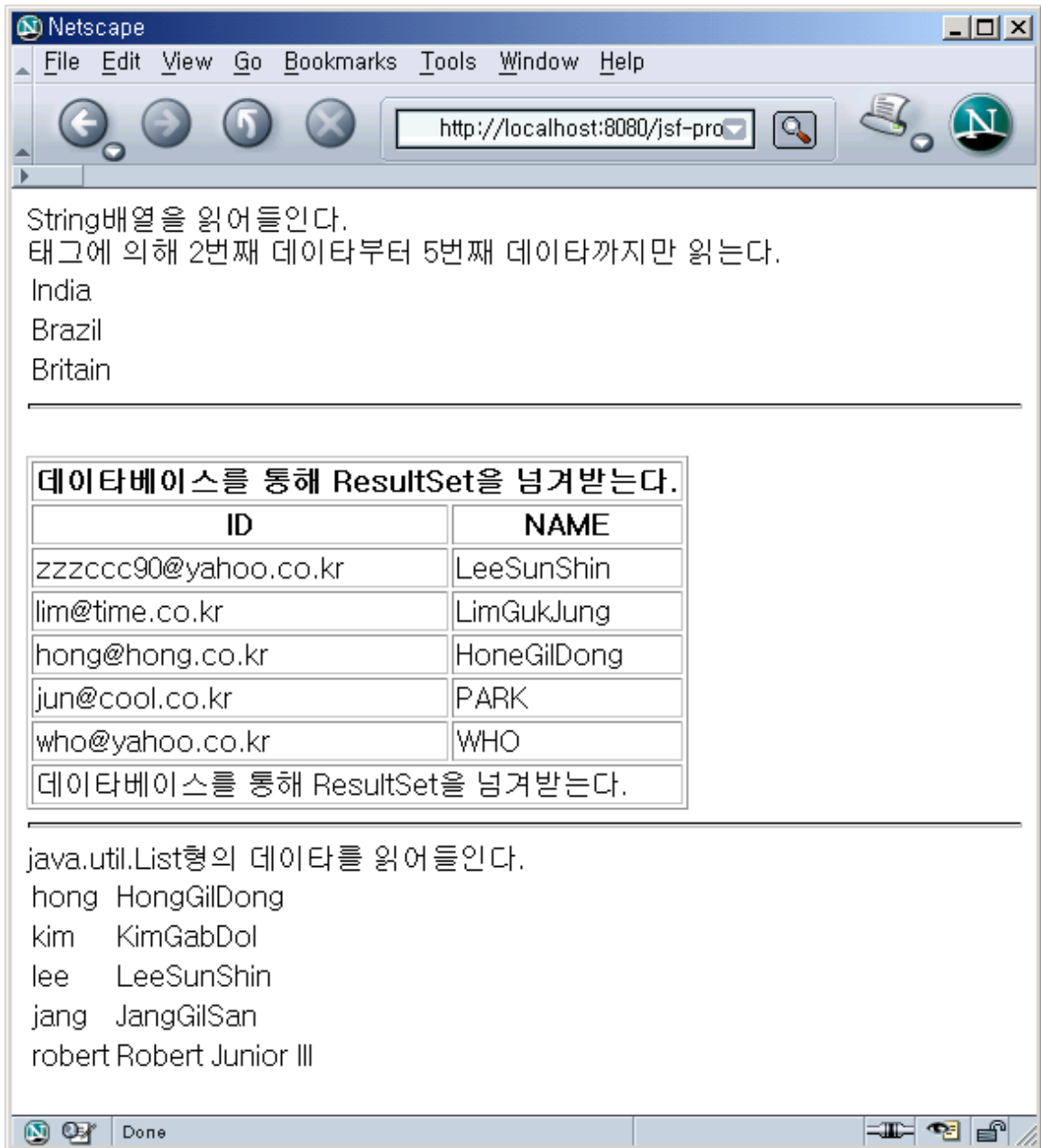
ResultSet의 email컬럼의 값과 name컬럼의 값을 각각 표시하고 있다. 여기서 id와 name은 데이터 베이스로부터 얻은 ResultSet에 포함된 컬럼의 이름이다.

```
...
<h:dataTable value="#{DataTableExample.dataAsList}" var="data">
<h:column>
<h:outputText value="#{data.id}"/>
</h:column>
<h:column>
<h:outputText value="#{data.name}"/>
</h:column>
</h:dataTable>
...
```

세번째 예는 List타입의 데이터를 반환하는데, List타입의 데이터 내에 DataTableBean이라는 클래스가 포함되어 있다.

<h:outputText value="#{data.id}"/>를 보면, List타입의 데이터내에 DataTableBean이라는 클래스의 getId()라는 메소드를 호출한 값이 표현될 것이다.

위의 작업결과가 브라우저에 다음과 같이 출력된다.



DataTable컴포넌트의 특징은 무엇보다도, 정해진 데이터집합을 이용해 내부적으로 루프를 사용해서 데이터를 표현해 낼수 있다는 것이다. 루프문이 내부적으로 처리됨으로 좀더 깔끔한 코드의 작성이 가능하다. 나중에 다루게 될 custom component부분에서도 이러한 데이터 집합을 처리하는 부분에 대해 자세히 설명한다.

7. Core 컴포넌트

core컴포넌트는 렌더링이 필요없는 컴포넌트들이다. 대부분 html 컴포넌트를 지원하는 컴포넌트로 이용되거나, core 컴포넌트 홀로 이용 된다. JSF페이지에 작성해야 하는 다음의 문자열은 core컴포넌트의 태그정의 파일을 가리킨다.

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

7.1 <f:actionListener>

앞장에서 UICommand로 되변되는 HtmlCommandButton , HtmlCommandLink 컴포넌트에서액션이 발생하는 경우 이벤트를 처리하는 방법을 공부했다. Core컴포넌트를 이용하여 이벤트를 처리할 수도 있는데 이때 사용되는 것이 actionListener 이다. actionListener태그의 type속성에 javax.faces.event.ActionListener인터페이스를 구현한 클래스를 파라미터로 넘기면 ActionListener인터페이스에 지정된 processAction메소드가 실행된다. 이 클래스를 faces-config.xml에 등록할 필요는 없다.

```
package jsf.proj.event;
import javax.faces.component.*;
public class MyActionOccurred implements ActionListener{
    // ActionListener인터페이스의 구현 해야 할 메소드
    public void processAction(javax.faces.event.ActionEvent event){
        UIComponent comp=event.getComponent();
        System.out.println("컴포넌트 id = "+comp.getId()+"에서 이벤트가 발생되었습니다.");
    }
}
```

core 컴포넌트의 actionListener는 다음과 같이 사용된다.

```
<!--
    /jsp/example/action_example.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
```

```

<body>
<f:loadBundle basename="javax.faces.Resources" var="bundle"/>
<f:view>

<h:form id="myForm">

<h:commandButton id="myAction" value="MyAction" action="success">
    <f:actionListener type="jsf.proj.event.MyActionOccurred"/>
</h:commandButton>

</h:form>

</f:view>

</body>
</html>

```

버튼을 클릭하면, MyActionOccurred 클래스의 processAction 메소드가 호출된다. type 속성에 패키지명을 포함한 클래스 이름만 적어주면 된다.

7.2 <f:attribute>

컴포넌트의 속성과 속성값을 넘길 수 있다. 다른 컴포넌트 태그 내부에서 이용해야 한다. 이 태그를 포함하는 컴포넌트는 getAttributes() 메소드를 통해 java.util.Map 형으로 속성값 전체를 얻어 필요한 값을 얻어낸다.

다음의 예를 보자.

```

<!--
    /jsp/example/attr_example.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
<body>
<f:view>
<h:form id="myForm">

```

```

<h:inputText id="in1" value="#{AttrExample.name}">

<f:attribute name="attr1" value="alpha"/>
<f:attribute name="attr2" value="this is message"/>
</h:inputText>

<h:commandButton action="success" value="SUBMIT" />

</h:form>
</f:view>
</body>
</html>

```

컴포넌트의 `getAttribute()` 메소드를 이용해서 반환된 `java.util.Map` 클래스로부터 `put()` 메소드를 이용하여 값을 변경할 수 있다. 단, 변경이 가능한 값에 한해서 이다. 변경이 불가능한 값에 대해 `put()` 메소드를 사용하면 `IllegalArgumentException`이 발생한다. 이해를 돕기 위해 여기서 이 페이지의 컴포넌트들에 대한 정보를 bean에서 얻어내어 이용하는 방법에 대해 알아보자.

위의 예를 테스트 해보기 위해 `AttrExample.java` 파일을 다음과 같이 만든다. 물론 이 클래스는 `faces-config.xml`에 설정이 되어야 한다.

```

package jsf.proj.example;

public class AttrExample{
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

이 클래스에서 jsf페이지의 컴포넌트 id가 in1 인 입력상자의 컴포넌트에 대한정보를 어떻게 얻어 올수 있는지를 알아보자. 위의 예에서 입력상자의 속성값 attr1 과 value

값을 어떻게 얻어내는지 알아보자. JSF의 컴포넌트들이 트리구조를 이룬다는 사실과 트리의 최고상위 노드가 UIViewRoot 클래스라는 것을 이미 앞서 설명했었다.

현재의 웹 페이지를 위해 이러한 정보를 담고 있는 개체가 FacesContext 이다. 따라서, 우리는 FacesContext 개체를 얻고, context 개체로부터 UIViewRoot 개체를 얻어서, UIViewRoot 개체의 findComponent() 메소드를 이용하여 우리가 원하는 컴포넌트를 찾아서 조작할 수 있다. 이런 작업을 수행하는 findComp()메소드를 위의 AttrExample.java클래스에 다음과 같이 추가한다.

```
package jsf.proj.example;

import javax.faces.context.*;
import javax.faces.component.*;
import java.util.*;

public class AttrExample{
...
    public AttrExample(){
        findComp();
    }
    public void findComp(){
        FacesContext fc=FacesContext.getCurrentInstance();
        UIViewRoot root=fc.getViewRoot();
        UIInput comp=(UIInput)root.findComponent("myForm:in1");

        Map attrs=comp.getAttributes();
        Iterator keys=attrs.keySet().iterator();
        while(keys.hasNext()){
            Object key=keys.next();
            System.out.println("key : "+key+" value : "+attrs.get(key));
        }
    }
...
}
```

이 메소드를 보면, 현재 context 정보를 담고 있는 FacesContext 개체로부터 UIViewRoot 개체를 얻고, findComponent() 메소드를 사용하고 있다. findComponent() 메소드를 이용하

여 컴포넌트의 id가 in1 인 컴포넌트를 찾는데 myForm:in1 과 같이 form의 id를 사용하고 있다는 것에 유의하자. UIViewRoot의 하위 노드에 form 컴포넌트가 여러 개 있을수 있기 때문이다. form 컴포넌트로부터 컴포넌트 id를 이용하여 findComponent() 메소드를 수행하는 경우는 from id를 사용할 필요가 없을 것이다. 위의 메소드에 의해 컴포넌트 in1에 지정된 속성값들을 다음과 같이 얻을 수 있다.

```
2004. 6. 6. 오후 8:44:15 org.apache.catalina.core.StandardContext reload
정보: Reloading this Context has started
key : attr1 value : alpha
key : attr2 value : this is message
```

JSF를 이용하여 웹 어플리케이션을 개발하는 경우, 위의 예처럼 서버사이드에서 컴포넌트를 찾아서 작업을 해야하는 경우가 생길 수 있다. 서버사이드에서 컴포넌트를 생성하여 바인딩하는 경우는 바로 컴포넌트에 접근할 수 있으므로 문제될 것이 없을 것이다.

7.3 <f:convertDateTime>

컴포넌트의 지정된 값을 날짜와 시간에 관련된 값으로 변경해준다.

먼저, 날짜를 생성할 빈을 만들자. 이 클래스를 DateTimeExample 이라는 이름으로 faces-config.xml에 등록한다.

```
package jsf.proj.example;

import java.util.Date;

public class DateTimeExample{

    private Date currentDate;

    public DateTimeExample(){
        currentDate=new Date();
    }

    public Date getCurrentDate() {
        return currentDate;
    }

    public void setCurrentDate(Date currentDate) {
        this.currentDate = currentDate;
    }

}
```

```
}
```

```
<!--  
    /jsp/example/datetime_example.jsp  
-->  
<%@page contentType="text/html;charset=euc-kr"%>  
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>  
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>  
  
<html>  
  <body>  
    <f:view>  
      <h:form id="myForm" >  
        시간과 날짜를 변경하는 converter<br>  
  
        날짜패턴(yyyy-MM-dd-hh-mm) ::  
        <h:outputText value="#{DateTimeExample.currentDate}">  
          <f:convertDateTime pattern="yyyy-MM-dd-hh-mm" />  
        </h:outputText><br>  
  
        날짜패턴(G yyyy년 MM월 dd일 a hh시 mm분 ss초 ) ::  
        <h:outputText value="#{DateTimeExample.currentDate}">  
          <f:convertDateTime pattern="G yyyy년 MM월 dd일 E요일 a hh시 mm분  
ss초 " />  
        </h:outputText><br>  
  
        날짜패턴(yyyy/MM/dd) ::  
        <h:outputText value="#{DateTimeExample.currentDate}">  
          <f:convertDateTime pattern="yyyy/MM/dd" />  
        </h:outputText><br>  
  
        날짜패턴 : 없음 ::  
        <h:outputText value="#{DateTimeExample.currentDate}">  
          <f:convertDateTime />
```

```
</h:outputText><br>

날짜패턴 : 없음 , type=date ::
<h:outputText value="#{DateTimeExample.currentDate}">
  <f:convertDateTime type="date"/>
</h:outputText><br>

날짜패턴 : 없음 , type=time ::
<h:outputText value="#{DateTimeExample.currentDate}">
  <f:convertDateTime type="time"/>
</h:outputText><br>

날짜패턴 : 없음 , type=both ::
<h:outputText value="#{DateTimeExample.currentDate}">
  <f:convertDateTime type="both"/>
</h:outputText><br>

</h:form>
</f:view>
</body>
</html>
```

위의 작업의 결과는 아래와 같다. 위의 입력과 비교해보자.



아래는 convertDateTime컴포넌트를 이용할 때 사용되는 주요 속성이다.

속성이름	기본값	유효한 값	참고
timeStyle	default	full long short medium default	Type속성값이 time이거나 both 일때
dateStyle	default	full long short medium default	Type속성값이 date이거나 both 일때
type	date	date both time	

pattern	없음	G : AD or BC y : 년도 M : 월 d : 일 E : 요일 a : 오전 / 오후 h : 시 m : 분 s : 초
---------	----	--

7.4 <f:convertNumber>

입력되는 숫자의 포맷을 여러가지 형식으로 지정한다. 여기에 사용되는 숫자는 원시타입이 아니다. 다음과 같은 클래스를 만들어 테스트 해보자.

```

package jsf.proj.example;

public class ConvertExample{

    private Integer num1=new Integer(3500);
    private Float num2=new Float(3.141592);
    private Float num3=new Float(0.4321);

    public Integer getNum1() {
        return num1;
    }
    public void setNum1(Integer num1) {
        this.num1 = num1;
    }
    public Float getNum2() {
        return num2;
    }
    public void setNum2(Float num2) {
        this.num2 = num2;
    }
    public Float getNum3() {

```

```

        return num3;
    }
    public void setNum3(Float num3) {
        this.num3 = num3;
    }
}

```

위의 클래스의 값을 이용하는 웹 페이지를 만들었다.

```

<!--
    /jsp/example/converter_example.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="myForm" >
        <h:inputText value="#{ConvertExample.num2}">
          <f:convertNumber type="number" integerOnly="false"
            maxFractionDigits="2" maxIntegerDigits="5" />
        </h:inputText><br>

        <h:inputText value="#{ConvertExample.num3}">
          <f:convertNumber type="percent"/>
        </h:inputText><br>

        <h:inputText value="#{ConvertExample.num3}">
          <f:convertNumber type="percent" pattern="###.##"/>
        </h:inputText><br>

        <h:outputText value="#{ConvertExample.num1}">
          <f:convertNumber type="currency" />
        </h:outputText><br>

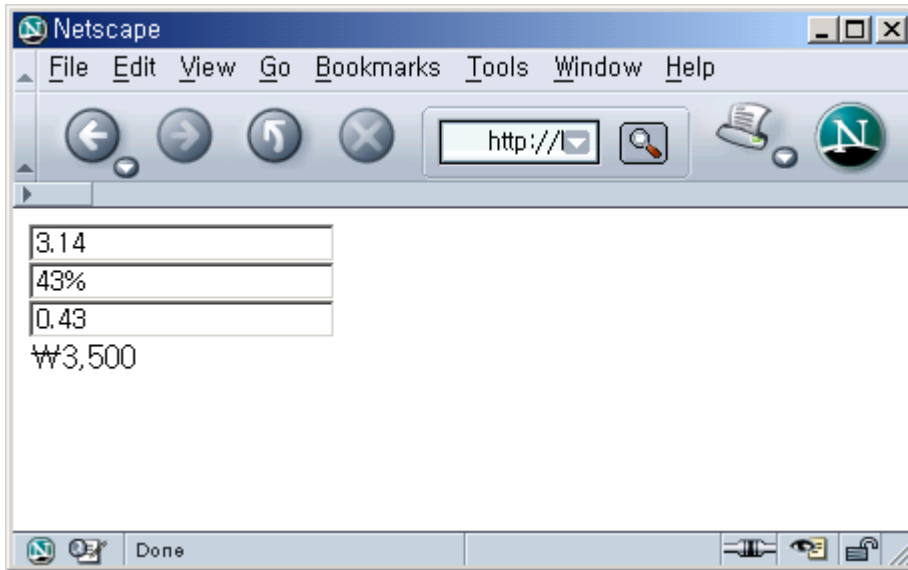
```

```

    </h:form>
  </f:view>
</body>
</html>

```

위의 예제의 결과는 브라우저에 다음과 같이 나타난다.



다음의 표는 convertNumber 에 사용되는 주요 속성들이다.

속성이름	기본값	유효한 값	참고
integerOnly	false	True/false	
maxFractionDigits	없음	숫자	최대 소수 자리수
maxIntegerDigits	없음	숫자	최대 정수 자리수
minFractionDigits	없음	숫자	최소 소수 자리수
minIntegerDigits	없음	숫자	최소 정수 자리수
pattern	없음	###	예) ###.##
type	number	number currency percent	

7.5 <f:converter>

개발자가 직접 작성한 converter 를 사용하도록 하는 태그이다.

이 태그는 converterId 라는 하나의 속성만을 가진다. 여기에 직접 작성한 converter 클래스

래스를 패키지명을 포함하여 기술하면 된다.

converter 클래스는 javax.faces.convert.Converter 인터페이스를 구현해야 한다.

7.6 <f:facet>

facet은 위의 예에서도 보았듯이, facet을 포함한 컴포넌트와 특별한 관계를 가지는 컴포넌트를 표시하고자 할때 이용한다. HtmlPanelGrid 컴포넌트나 HtmlDataTable컴포넌트내에서 테이블의 제목이나 컬럼의 제목을 나타낼 때 유용하게 이용된다. facet을 포함한 컴포넌트에서 getFacets() 메소드를 이용하여 java.util.Map 형으로 facet을 얻어 내어 이용할 수 있다.

7.7 <f:loadBundle>

지정된 로케일에 따른 Resouecrs.properties파일을 로드한다.

var 속성은 Resources.properties 파일을 참조하는 별명이다. 브라우저의 언어환경이 한글이라면, Resources_ko.properties 파일을 찾고, 이 파일이 없다면 Resources.properties 파일을 참조한다.

다음과 같은 형식으로 이용하면 된다.

```
...  
  
<HTML>  
  
<BODY>  
  <f:loadBundle basename="javax.faces.Resources" var="bundle"/>  
  <f:view>  
    <h:form id="myForm">  
      ...  
      <h:outputText value="#{bundle.msg}"/>  
    </h:form>  
  </f:view>  
  ...  
</BODY>  
</HTML>  
...
```

7.8 <f:param>

파라미터 값을 ReousrcBundle이나 컴포넌트에 전달하거나, request에 포함될 파라미터 값들을 지정할 때에 이용된다. 아래와 같이 이용하는 경우 {0}과 {1}에 파라미터가 각각 매핑되어 출력될 것이다.

```
<%@page contentType="text/html;charset=euc-kr"%>
```

```

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>

<BODY>
  <f:view>
    <h:form id="myForm">
      ...
      <h:outputFormat value="{0}, {1}">
        <f:param value="홍길동"/>
        <f:param value="이순신"/>
      </h:outputFormat>
    ...
  </f:view>
</BODY>
</html>

```

7.9 <f:selectItem> , <f:selectItems>

core 컴포넌트 중에서 조금 복잡한 것들이 이 두 컴포넌트 인 것 같다. JSF 컴포넌트중 메뉴,리스트박스,라디오버튼,체크박스에 대한 컴포넌트들을 살펴보자.

컴포넌트 (javax.faces.component.html 패키지)	상위클래스 (javax.faces.component)	렌더러 타입
HtmlSelectBooleanCheckbox	UISeleceBoolean	javax.faces.CheckBox
HtmlSelectManyCheckbox	UISelectMany	javax.faces.CheckBox
HtmlSelectManyListbox	UISelectMany	javax.faces.Menu
HtmlSelectManyMenu	UISelectMany	javax.faces.ListBox
HtmlSelectOneListbox	UISelectOne	javax.faces.ListBox
HtmlSelectOneMenu	UISelectOne	javax.faces.Menu
HtmlSelectOneRadio	UISelectOne	javax.faces.Radio

위의 표를 보면 하나 이상의 값을 선택하는 경우에 이용될 컴포넌트들은 UISelectMany 클래스를 상속하고, 하나의 값만을 선택가능한 컴포넌트들은 UISelectOne 클래스를 상속하고 있음을 알 수 있다. boolean 체크박스는 따로 구현되어 있다. 여기서 알 수 있는 것은 HtmlSelectOneRadio 로 이용되던 컴포넌트들은 쉽게

HtmlSelectOneMenu로 변경할 수 있다. 내부적인 작동방식은 USelectOne을 따르고 있기 때문이다. 클라이언트 환경 즉, 브라우저에 표현되는 방법을 지정하는 렌더러만 위와같이 변경되면 된다.

먼저 아래와 같은 html을 살펴보자. 하나만 선택가능한 메뉴를 만드는 html이다.

```
<select id="myForm:menu1" name="myForm:menu1" size="1" title="MENU">
  <option value="BRAZIL">브라질</option>
  <option value="CHINA">중국</option>
  <option value="VIETNAM">베트남</option>
  <option value="JAPAN">일본</option>
</select>
```

위와 동일한 결과를 만들수 있는 웹 페이지를 jsf를 이용하여 구성해보자.

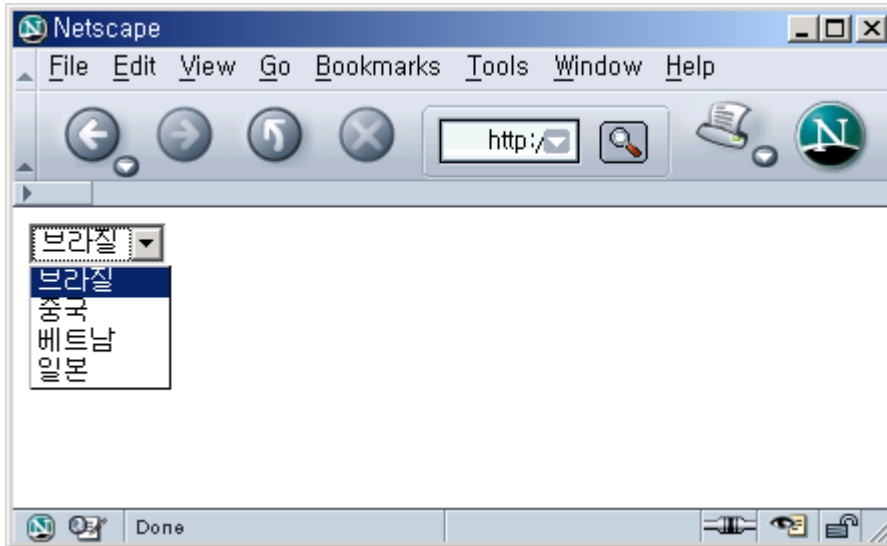
```
<!--
    /jsp/example/select_example.jsp
-->
<%@page contentType="text/html; charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>

    <f:view>
      <h:form id="myForm">

        <h:selectOneMenu>
          <f:selectItem itemValue="BRAZIL" itemLabel="브라질" />
          <f:selectItem itemValue="CHINA" itemLabel="중국"/>
          <f:selectItem itemValue="VIETNAM" itemLabel="베트남"/>
          <f:selectItem itemValue="JAPAN" itemLabel="일본"/>
        </h:selectOneMenu>
      </h:form>
    </f:view>
  </body>
```

```
</html>
```



위의 두 결과는 동일하다. 위의 예처럼 사용하게 된다면, html을 작성하는것과 별반 다르지 않다.

이번에는 <f:selectItems...> 태그를 사용해 보자.

```
<!--
    /jsp/example/select_example.jsp
-->
<%@page contentType="text/html; charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
  <body>
    <f:view>
      <h:form id="myForm">
        <b>javax.faces.model.SelectItem , javax.faces.model.SelectItemGroup를
이용한 바인딩</b><br>
        라디오버튼 :
        <h:selectOneRadio id="radio2" title="RADIO1">
          <f:selectItems value="#{SelectExample.selectData}"/>
        </h:selectOneRadio>
```

```
<br>

메뉴 :
<h:selectOneMenu id="menu2" title="MENU1">
    <f:selectItems value="#{SelectExample.selectData}"/>
</h:selectOneMenu>
<br>

리스트 박스 :
<h:selectOneListbox id="list2" title="LIST1">
    <f:selectItems value="#{SelectExample.selectData}"/>
</h:selectOneListbox>
</h:form>
</f:view>
</body>
</html>
```

selectItems 속성에 이용되는 value값은 SelectItem 클래스의 인스턴스가 저장된 List 나 Array 타입이어야 한다. 위의 selectItems를 이용하기 위해 다음과 같은 클래스가 필요하다.

```
package jsf.proj.example;

import java.util.*;
import javax.faces.component.*;
import javax.faces.model.*;

public class SelectExample{
    private ArrayList data;

    private SelectItem options1[] = {
        new SelectItem("SEOUL", "서울"),
        new SelectItem("DAEJUN", "대전"),
        new SelectItem("TAEGU", "대구", "대전시"),
// 이값은 비활성화 될것이다.
        new SelectItem("PUSAN", "부산", "부산시", true)
    }
}
```

```

};
public ArrayList getSelectData(){

    data=new ArrayList();
    SelectItemGroup group1 = new SelectItemGroup("우리나라의 대도시 : ", null,
true, options1);
    data.add(group1);

    return data;
}
public void setSelectData(Collection data) {
    this.data = new ArrayList(data);
}
}

```

SelectItem의 생성자는 다음과 같다.

```

SelectItem()
SelectItem(Object value)
SelectItem(Object value, String label)
SelectItem(Object value, String label, String description)
SelectItem(Object value, String label, String description, boolean disabled)

```

각각 순서대로 값, 라벨, 설명, 그리고 활성화 비활성화를 나타내는 파라미터로 구성된다. 위의 예처럼 boolean 값이 true 이면 비활성화 될것이다.

SelectItemGroup의 생성자는 다음과 같다.

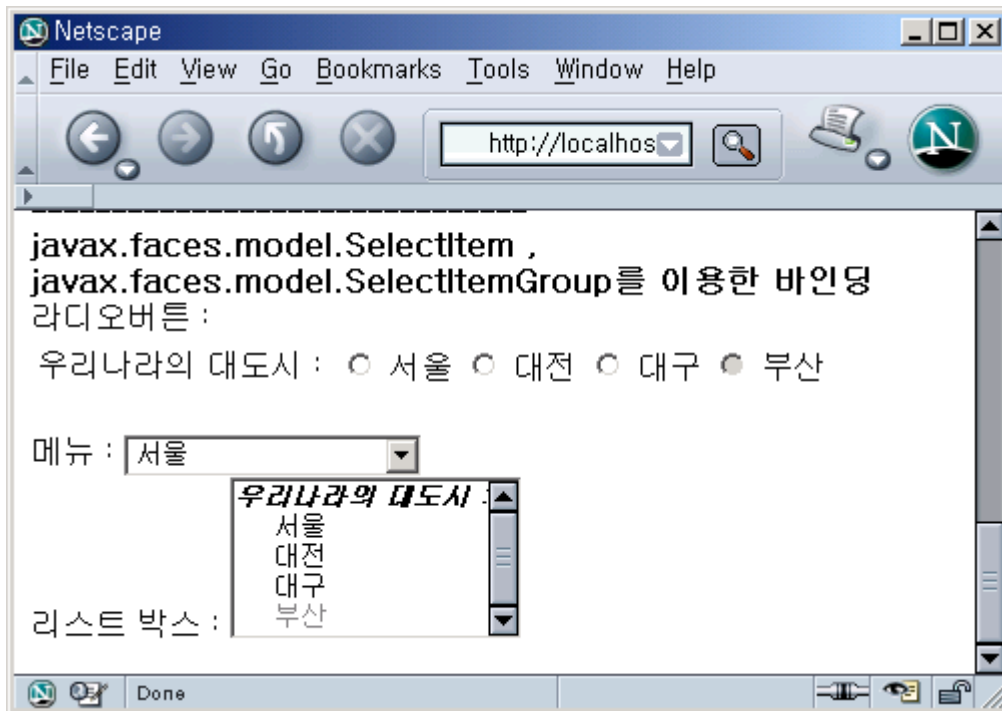
```

SelectItemGroup()
SelectItemGroup(String label)
SelectItemGroup(String label, String description, boolean disabled, SelectItem[]
selectItems)

```

위의 페이지의 결과를 보면 라디오 버튼, 메뉴, 리스트 박스를 만들거나 변경하는 것이 별로 어렵지 않다는 것이다. 비슷한 성격의 컴포넌트들을 렌더러만 달리하여 표현될 수

있도록 하여 지금처럼 유연한 처리를 할수 있도록 하고 있다.



이번에는 위와 같은 작업을 UISelectOne컴포넌트를 이용한다.

다음의 예제를 작성해보자.

```
<!--
    /jsp/example/select_example.jsp
-->
<%@page contentType="text/html; charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<html>
<body>
<f:view>
<h:form id="myForm">
<b>javax.faces.component.UISelectOne를 이용한 바인딩</b><br>
라디오버튼 : <h:selectOneRadio id="radio1" title="RADIO"
binding="#{SelectExample.createSelectOne}" /><br>
메뉴 : <h:selectOneMenu id="menu1" title="MENU"
```

```

        binding="#{SelectExample.createSelectOne}"
        valueChangeListener="#{SelectExample.selectOneClicked}" /><br>
        리스트 박스 : <h:selectOneListbox id="list1" title="LIST"
        binding="#{SelectExample.createSelectOne}" />
    </h:form>
</f:view>
</body>
</html>

```

위의 소스를 살펴보면, 라디오버튼과 메뉴와 리스트가 모두 동일하게 `getCreateSelectOne()` 메소드가 반환하는 값을 바인딩 하고 있다. 그러나 브라우저에 나타나는 모양은 틀리다. 각각의 태그에 의해 렌더러가 변경될 것이기 때문이다.

값이 변경되는 경우 이벤트를 처리하는 것을 보이기 위해 메뉴상자에 `valueChangeListener` 를 간단히 추가하였다. 위의 예제에 사용되는 `SelectExample` 클래스를 보면 다음과 같다.

```

package jsf.proj.example;

import java.util.*;

import javax.faces.application.*;
import javax.faces.component.*;
import javax.faces.context.*;
import javax.faces.event.*;
import javax.faces.model.SelectItem;
import javax.faces.model.SelectItemGroup;

public class SelectExample{
    UISelectOne selectOne;//select one컴포넌트
    public SelectExample(){
        public UISelectOne getCreateSelectOne(){
//FacesContext fc=FacesContext.getCurrentInstance();
//Application app=fc.getApplication();
            //selectOne=(UISelectOne)app.createComponent("javax.faces.SelectOne");
            selectOne=new UISelectOne();
            //UISelectItem
            item1=(UISelectItem)app.createComponent("javax.faces.SelectItem");

```

```

        UISelectItem item1=new UISelectItem();
        UISelectItem item2=new UISelectItem();
        UISelectItem item3=new UISelectItem();
        UISelectItem item4=new UISelectItem();

        item1.setItemLabel("브라질");item1.setItemValue("BRAZIL");
        item2.setItemLabel("중국");item2.setItemValue("CHINA");
        item3.setItemLabel("베트남");item3.setItemValue("VIETNAM");
        item4.setItemLabel("일본");item4.setItemValue("JAPAN");

        selectOne.getChildren().add(item1);
        selectOne.getChildren().add(item2);
        selectOne.getChildren().add(item3);
        selectOne.getChildren().add(item4);

        return selectOne;
    }

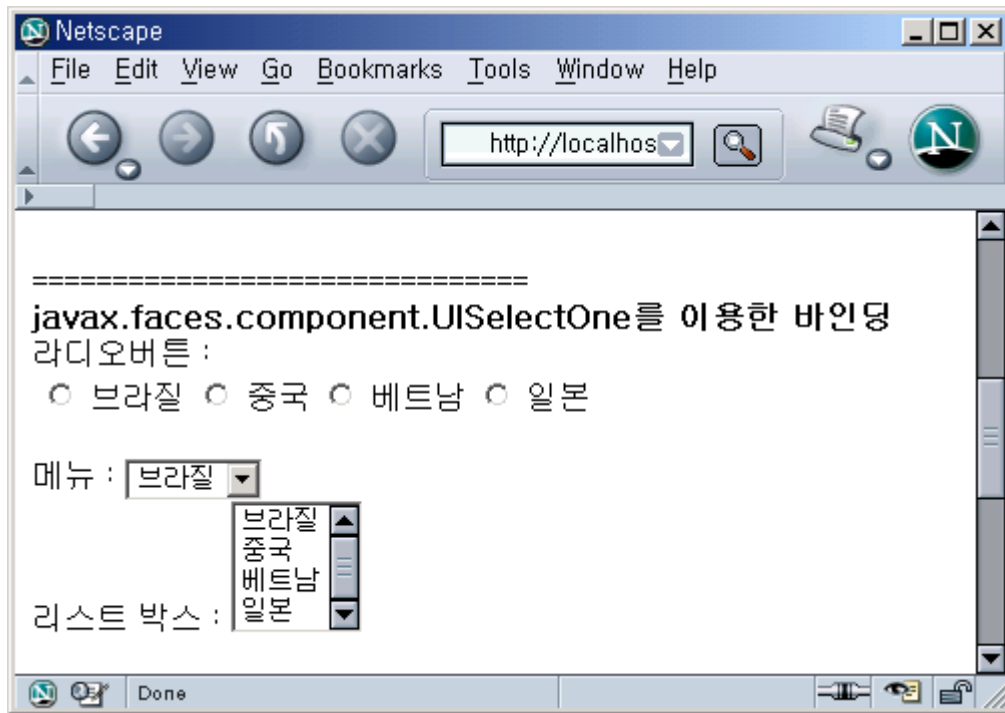
    public void setCreateSelectOne(UISelectOne comp){
        this.selectOne=comp;
    }

    public void selectOneClicked(ValueChangeEvent valueChange){
        System.out.println("ValueChangeEvent 발생...");
        UIComponent comp=valueChange.getComponent();
        Object oldValue=valueChange.getOldValue();
        Object newValue=valueChange.getNewValue();
        System.out.println("selectOneClicked메소드에서 component id =
        "+comp.getId()+"의 select one 값이 "+oldValue+" 에서 "+newValue+" 로
        바뀌었습니다.");
    }
}

```

바인딩되는 UISelectOne 클래스의 인스턴스가 가지는 getter/setter 메소드를 구현하고 있다. get 메소드 호출시 UISelectOne 인스턴스와 SelectItem 인스턴스를 생성하고, 반환한다. 위의 클래스 인스턴스를 생성할 때 주석처리한 부분과 같이 코딩을 하여도 된다. 참고하기 바란다. UISelectOne 인스턴스의 add() 메소드로 UISelectItem을 담으면 된다. UISelectItem 인스턴스가 버튼이나 메뉴의 요소로 화면에 처리되어 다음과 같

이 나타난다.



라디오 버튼이나 메뉴, 리스트 박스의 한가지 아이템만을 선택가능하게 하는 컴포넌트이다.

UISelectMany 컴포넌트도 위와 동일한 과정을 따르게 된다.

```
<!--  
    /jsp/example/select_example.jsp  
-->  
<%@page contentType="text/html;charset=euc-kr"%>  
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>  
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>  
  
<html>  
  <body>  
    <f:view>  
      <h:form id="myForm">  
        =====  
      <br>
```

```

<b>javax.faces.component.UISelectMany를 이용한 바인딩</b>
<br>
리스트 박스 : <h:selectManyListbox title="select boolean checkbox"
binding="#{SelectExample.createSelectMany}" /><br>
메뉴 : <h:selectManyMenu title="select boolean checkbox"
binding="#{SelectExample.createSelectMany}" /><br>
체크박스 : <h:selectManyCheckbox title="select boolean checkbox"
binding="#{SelectExample.createSelectMany}" />
</h:form>
</f:view>
</body>
</html>

```

UISelectOne 과 동일하므로, 설명은 생략하도록 한다. 여기서 사용될 클래스의 소스는 다음과 같다.

```

package jsf.proj.example;

import java.util.*;
import javax.faces.component.*;
import javax.faces.model.*;

public class SelectExample{
    UISelectMany selectMany;//select many컴포넌트
    public UISelectMany getCreateSelectMany(){
//selectMany=(UISelectMany)app.createComponent("javax.faces.SelectMany");
        selectMany=new UISelectMany();

        UISelectItem item1=new UISelectItem();
        UISelectItem item2=new UISelectItem();
        UISelectItem item3=new UISelectItem();
        UISelectItem item4=new
UISelectItem();//(UISelectItem)app.createComponent("javax.faces.SelectItem");

        item1.setItemLabel("브라질");item1.setItemValue("BRAZIL");
        item2.setItemLabel("중국");item2.setItemValue("CHINA");

```

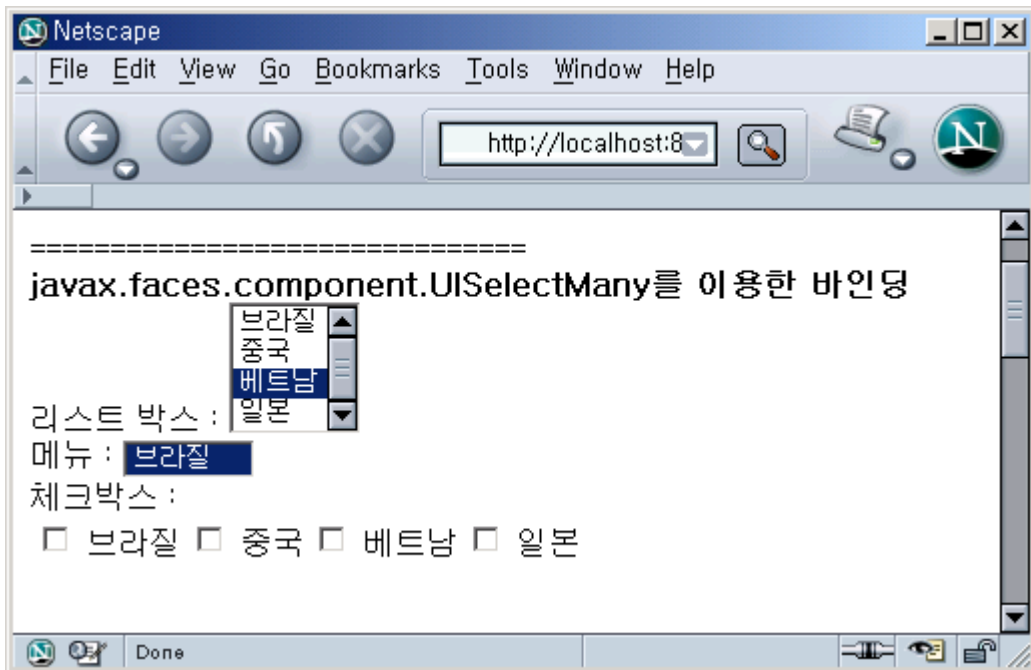
```

        item3.setItemLabel("베트남");item3.setItemValue("VIETNAM");
        item4.setItemLabel("일본");item4.setItemValue("JAPAN");

        selectMany.getChildren().add(item1);
        selectMany.getChildren().add(item2);
        selectMany.getChildren().add(item3);
        selectMany.getChildren().add(item4);
        return selectMany;
    }
    public void setCreateSelectMany(UISelectMany comp){
        this.selectMany=comp;
    }
}

```

결과화면을 보자.



7.10 <f:subview>

다른 jsf페이지에 include되어 사용될 수 있는 jsf페이지를 작성할 때 사용되는 최상위 태그이다. JSF의 모든 기능을 사용할 수 있으며, subview내부에 view태그가 있으면 안 된다.

```

<!--
    /jsp/example/subpage_example.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<HTML>
  <body>
    <f:subview id="subview1">
      <h:form id="subpageform1">
        <h:outputText value="서브 페이지 1" />
      </h:form>
    </f:subview>
  </body>
</html>

```

JSP에서 이 페이지를 불러들이는 경우 jsp에서 처럼 사용한다.

```
<jsp:include page="subpage1.jsp"/>
```

7.11 <f:validateDoubleRange> , <f:validateLength> , <f:validateLongRange>

각각 Double형 데이터와 문자열의 길이 Long형 데이터가 범위에 있는지를 결정한다.
이 컴포넌트들은 속성값으로 각각 maximum과 minimum을 가지고 있다.

```

<!--
    /jsp/example/validate_example.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="/WEB-INF/validation.tld" prefix="ph" %>
<html>
  <body>
    <f:view>

```

```

<h:form id="Testform">
  <h:inputText id="in1" required="true" >
    <f:validateLength maximum="10000.34" minimum="3.14"/>
  </h:inputText>
  <h:message for="in1"/><br>
  <h:inputText id="in2" required="true" >
    <f:validateDoubleRange maximum="10000.34" />
  </h:inputText>
  <h:message for="in2"/><br>
  <h:inputText id="in3" required="true">
    <f:validateLength minimum="3"/>
  </h:inputText>
  <h:message for="in3"/><br>

  <h:commandButton id="cmd1" value="submit" action="success"/><br>
</h:form>
</f:view>
</body>
</html>

```

message태그는 for속성으로 지정된 컴포넌트에 에러가 발생할 경우 메시지를 출력한다.

7.12 <f:validator>

다음장의 validator 부분에서 직접정의한 validator를 구현해보도록 한다.

7.13 <f:valueChangeListener>

값이 변경되는 컴포넌트들의 이벤트를 처리한다.

javax.faces.event.ValueChangeListener 인터페이스를 구현한 클래스를 속성으로 지정하면 된다.

ValueChangeListener 인터페이스를 구현하는 클래스는 다음과 같이 processValueChange() 메소드를 정의하여야 한다.

```
package jsf.proj.event;
```

```

import javax.faces.event.*;
import javax.faces.component.*;

public class MyValueChanged implements ValueChangeListener{

    public void processValueChange(ValueChangeEvent event) {
        UIComponent comp=event.getComponent();
        Object oldValue=event.getOldValue();
        Object newValue=event.getNewValue();
        System.out.println("MyValueChanged 클래스 component id = "+comp.getId()+"의
select one 값이 "+oldValue+" 에서 "+newValue+" 로 바뀌었습니다.");
    }
}

```

```

...
<h:inputText value="test"
<f:valueChangeListener type="javax.faces.event.MyValueChanged"/>
</h:inputText>
...

```

type 속성으로 패키지 명을 포함한 클래스이름만 적어주면 된다.

7.14 <f:view>

JSF의 모든 html컴포넌트와 core컴포넌트는 이 태그의 내부에 있어야 한다.
속성값으로 로케일을 설정 할 수 있다.

```

<f:view locale="ko">
    ...
</f:view>

```

8. Validator, Converter를 만들어 보자.

8.1 Validator 만들기

이번에는 직접 JSF의 validation을 구현해 보도록 하자. 여기서 만들어볼 validator 는 유저가 입력하는 핸드폰 번호가 유효한 번호인지 아닌지를 구별하도록 만들어 본다.

이 validator는 다음의 조건이면 유효한 번호로 간주한다.

- 010,016,017,018,019로 시작한다.
- 국번호는 3자리 혹은 4자리 숫자이다.
- 번호는 4자리 숫자이다.
- 번호사이에 '-'이 올수 있으며, '-'없이 번호전체를 입력해도 된다.

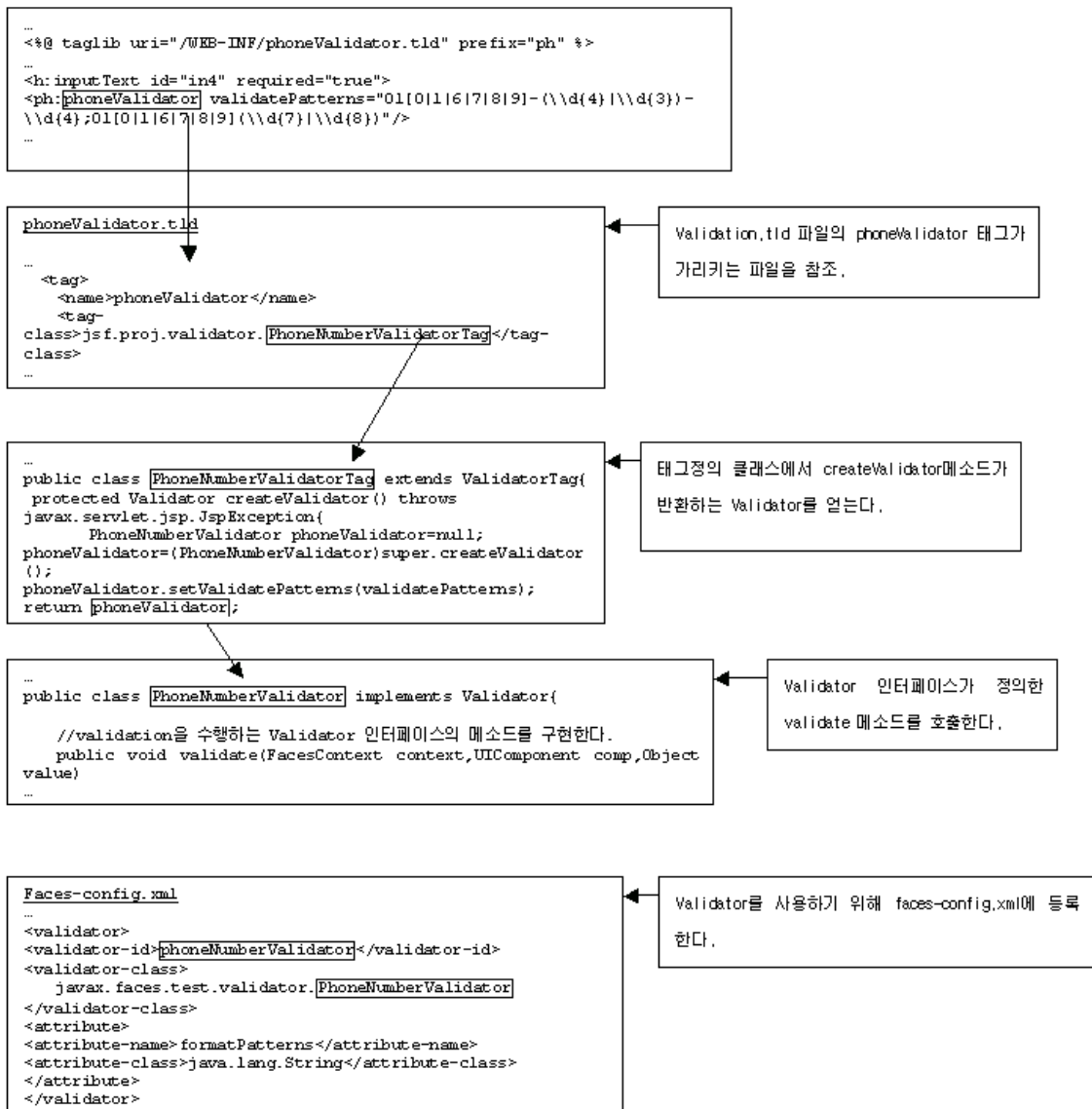
우리가 만들어 볼 validator 에서 위의 조건을 다만족하는 입력값이면, 입력값은 유효한것으로 처리한다.

validator를 구현하기 위해서는 다음과 같은 파일이 필요하다.

- 태그 정의 TLD파일
- validator를 사용하기 위한 태그 클래스
- Validator 인터페이스를 구현한 validator 클래스

그리고, 구현한 validator 클래스를 faces-config.xml에 설정하면 된다.

validator가 처리되는 과정을 간단히 도식화 하였다.



validator가 처리되는 과정을 보면, 웹 페이지로부터, 태그에 대한 정보를 TLD 파일로부터 얻고, 해당 태그를 TLD 파일에서 찾는다. 그리고 나서, 태그를 구현한 태그 클래스 PhoneNumberValidatorTag 로부터 PhoneNumberValidator 인스턴스를 얻는다. 이렇게 얻어진 PhoneNumberValidator 클래스 인스턴스의 validate() 메소드를 수행한다. 이와 같은 과정을 따라 validator 를 구현해 보자.

우선 웹 페이지에서 다음과 같은 패턴을 가지는 validator를 지정했다고 가정하고 과정을 설명한다.

```

<!--

```

```
    /jsp/example/custom_validator.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="/WEB-INF/phoneValidator.tld" prefix="ph" %>

<html>
<body>
<f:view>

<h:form id="add-form">

<h:inputText id="in1" required="true">
<ph:phoneValidator validatePatterns="01[0|1|6|7|8|9]-(\Ww{4}|\Ww{3})-
\Ww{4};01[0|1|6|7|8|9](\Ww{7}|\Ww{8})"/>
</h:inputText>
<h:message for="in1"/><br>
<h:commandButton id="cmd1" value=" VALIDATE"    action="success"/><br>

</h:form>
</f:view>
</body>
</html>
```

입력상자에는 값을 반드시 입력하도록 required 속성을 설정하고, validatePatterns의 정규식을 만족하면, 유효한 전화번호로 처리하게 한다. 정규식은 맨위에서 설명한 유효한 전화번호의 조건을 나타낸다. 만약, 이 조건을 만족하지 않으면, 웹 브라우저에 에러메시지가 나타날 것이다. 정규식은 ;로 구분되어 있으며, validator 클래스에서 각각의 정규식을 입력한 값과 비교하여 유효한 번호인지를 검사하게 된다.

TLD 파일

TLD 파일을 정의한다. 태그 이름과 태그 클래스, 그리고 태그의 속성을 정의하면 된다.

```
<?xml version="1.0" encoding="euc-kr" ?>
```

```

<!DOCTYPE taglib
  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
  <tlib-version>0.03</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>JavaServerFaces Framework Tag Library</short-name>
  <description>
</description>

  <!-- ===== PhoneNumberValidator tags ===== -->
  <tag>
    <name>phoneValidator</name>
    <tag-class>jsf.proj.validator.PhoneNumberValidatorTag</tag-class>
    <description>
</description>
    <attribute>
      <name>validatePatterns</name>
      <required>true</required>
      <rtexprvalue>>false</rtexprvalue>
      <description>
</description>
    </attribute>
  </tag>
</taglib>

```

태그의 속성 validatePatterns는 전화번호의 형식 패턴을 가질 것이다.

태그 클래스

```

package jsf.proj.validator;

import javax.faces.validator.Validator;
import javax.faces.webapp.ValidatorTag;

```

```

public class PhoneNumberValidatorTag extends ValidatorTag{

    private String validatePatterns;

    public PhoneNumberValidatorTag(){
        super();
        super.setValidatorId("phoneNumberValidator");
    }

    protected Validator createValidator() throws javax.servlet.jsp.JspException{
        PhoneNumberValidator phoneValidator=null;
        phoneValidator=(PhoneNumberValidator)super.createValidator();
        phoneValidator.setValidatePatterns(validatePatterns);
        return phoneValidator;
    }

    public void release(){
        super.release();
        validatePatterns=null;
    }

    public String getValidatePatterns() {
        return validatePatterns;
    }

    public void setValidatePatterns(String validatePatterns) {
        this.validatePatterns = validatePatterns;
    }
}

```

TLD 파일에 지정된 태그 클래스는 ValidatorTag 인터페이스를 구현해야 한다. createValidator 메소드가 validate를 수행할 클래스 인스턴스를 반환하고, release() 메소드는 사용한 자원을 반환하는 메소드이다. 그리고, 속성값 validatePatterns를 위한 getter/setter메소드를 정의한다.

Validator 클래스

```
package jsf.proj.validator;
```

```

import java.util.*;
import java.util.regex.*;
import javax.faces.application.*;
import javax.faces.component.*;
import javax.faces.context.*;
import javax.faces.validator.*;

public class PhoneNumberValidator implements Validator{
//jsf페이지에서 넘어오는 정규식을 포함하는 문자열변수다.
    private String validatePatterns;
    //정규식이 하나 이상이므로 ;를 구분자로 배열로 저장한다.
    private ArrayList patternList;
    public PhoneNumberValidator(){

        //validation을 수행하는 Validator 인터페이스의 메소드를 구현한다.
        public void validate(FacesContext context,UIComponent comp,Object value){
            boolean valid=isCorrect(value);
            if(!valid){
                FacesMessage fm=new FacesMessage("핸드폰 번호가
아닙니다.");
                //jsf페이지에 에러메시지를 출력한다.
                throw new ValidatorException(fm);
            }
        }
        public boolean isCorrect(Object value){
            boolean isValid=false;
            System.out.println("Value = "+value);
            if(patternList==null || patternList.size()==0 || value.toString()=="") return false;
            String str=value.toString();
            Object[] patterns=patternList.toArray();
            for(int i=0;i<patterns.length;i++){

                // 정규식으로 문자를 포함하는지 찾는다.
                Pattern p = Pattern.compile(patterns[i].toString());
                Matcher m = p.matcher(str);

```

```
        if (m.matches()){
            isValid=true;
            break;//매칭이 되면 루프를 종료한다.
        }else{
            isValid=false;
        }
    }
    System.out.println("input = "+str+" , Validation = "+isValid);
    return isValid;
}
public void parsePatterns(){
    if (validatePatterns == null || validatePatterns.length() == 0 ) return;
    else{
        patternList=new ArrayList();
    }
    StringTokenizer st=new StringTokenizer(validatePatterns,";");
    while(st.hasMoreTokens())
    {
        String token=st.nextToken();
        patternList.add(token);
        System.out.println("토큰 : "+token);
    }
}

public String getValidatePatterns() {
    return validatePatterns;
}
public void setValidatePatterns(String validatePatterns) {
    this.validatePatterns = validatePatterns;
    parsePatterns();
}
}
```

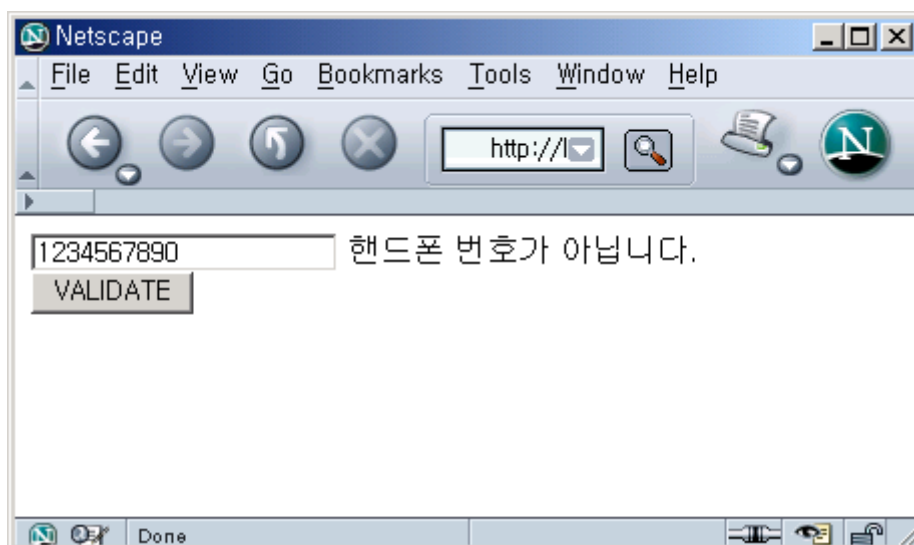
validatePattern 값을 얻으면, validator 클래스는 패턴을 StringTokenizer를 이용해서 ;를 기준으로 패턴을 구분하여 저장한다. 이 메소드가 parsePatterns()이다. 이렇게 지정된 패턴을 저장한 후에 validate() 메소드가 호출하는 isCorrect() 메소드가 정규식을 이용하여, 패턴과

입력된 값을 비교한다. 입력된 값이 패턴과 일치하면 true를, 아니면 ValidatorException을 발생한다. 이렇게 발생한 예외는 웹 페이지에 message 컴포넌트에 의해 브라우저에 출력된다.

이제 마지막으로 validator를 faces-config.xml에 다음과 같은 형식으로 설정을 해야 한다.

```
...  
<faces-config>  
...  
    <validator>  
        <validator-id>phoneNumberValidator</validator-id>  
        <validator-class>jsf.proj.validator.PhoneNumberValidator</validator-  
class>  
        <attribute>  
            <attribute-name>formatPatterns</attribute-name>  
            <attribute-class>java.lang.String</attribute-class>  
        </attribute>  
    </validator>  
...
```

전화번호를 입력하고, VALIDATE 버튼을 클릭해보기 바란다. 번호가 잘못 입력되면 다음과 같이 메시지가 나타난다.



번호를 제대로 입력하는 경우 에러메시지는 없어지고, 현재 페이지를 다시 표시한다. 왜냐하면, 이동할 페이지에 대한 설정을 하지 않았기 때문이다.

validator를 만드는 과정이 좀 복잡하다고 느낄지도 모르겠다. 컴포넌트를 만드는 과정도 이것을 만드는 과정과 동일하다. 조금만 연습해보면 금방 익숙해 질 것이다.

8.2 Converter

브라우저에서 입력되는 파라미터들을 bean을 위한 적절한 타입으로 변경하기 위해 사용되는 기능이다. JSF가 만들어 놓은 converter에 대해서는 core 컴포넌트 부분에서 설명이 있었고, 이번에는 개발자가 직접 작성한 converter를 등록하고, 이용해 보도록 한다.

Converter 는 validator에 비해 좀더 간단히 만들어진다. converter를 정의하는 클래스는 javax.faces.convert패키지의 Converter 인터페이스를 구현하면 된다.

이 Converter인터페이스의 getObject(), getString() 메소드를 구현한다.

웹 페이지의 값이 bean에 설정되는 경우 getObject() 메소드가 이용되고, bean의 값이 웹 페이지에 출력될 경우 getString() 메소드가 이용된다.

```
getString(FacesContext context, UIComponent component, Object value)
getObject(FacesContext context, UIComponent component, String value)
```

이 두 메소드는 파라미터로 현재 웹 페이지의 FacesContext, 이 converter가 속해있는 컴포넌트, 그리고 값을 파라미터로 가진다.

테스트를 위해 다음과 같은 웹 페이지를 만들자.

```
<!--
    /jsp/example/custom_converter.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
<body>
```

```

<f:view>
<h:form id="converter-example-form">

Converter Example<br>
<h:inputText id="in1" required="true" value="#{ConverterBean.value}" >
  <f:converter converterId="ConverterExample"/>
</h:inputText>
<h:message for="in1"/><br>
<h:commandButton id="cmd1" value="SUBMIT" action="success"/><br>

</h:form>
</f:view>
</body>
</html>

```

다음 페이지는 submit이 발생하면, 단순히 변경된 값을 브라우저에 출력한다.

```

<!--
      /jsp/example/custom_converter_result.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<html>
<body>
<f:view>

<h:form id="result-form">

ConvertBean.value =
<h:outputText value="#{ConverterBean.value}"/><br>

</h:form>
</f:view>
</body>
</html>

```

converter되는 값을 저장하기 위해 간단한 bean을 하나 만든다.

```
package jsf.proj.converter;

public class ConverterBean {
    private Integer value;
    public Integer getValue() {
        return value;
    }
    public void setValue(Integer value) {
        this.value = value;
    }
}
```

이제 converter를 만들어보자. 여기서 만드는 converter는 간단하다. 입력된 값에서 숫자만을 얻어내어, Integer 형으로 저장한다. 여기서도 입력된 값으로부터 숫자를 얻어내기 위해 정규식을 이용한다.

```
package jsf.proj.converter;

import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;

public class ConverterExample implements Converter {

    public ConverterExample(){

    }

    public Object getAsObject(FacesContext context, UIComponent component,
        String oldValue) throws ConverterException{
        //정규식을 이용해서, 숫자가 아닌 문자를 없앤다.
        String newValue=oldValue.replaceAll("WWD", "");
        if (newValue.length()==0) throw new ConverterException();
    }
}
```

```

        System.out.println("getAsObject oldValue = "+oldValue+", newValue =
"+newValue);
        return new Integer(newValue);
    }
    public String getAsString(FacesContext context,UIComponent component,
        Object oldValue) throws ConverterException{
        return oldValue.toString();
    }
}
}

```

converter에서 값을 변경할 수 없는 경우는 ConverterException이 발생한다. getAsString() 메소드는 bean의 값을 문자로 반환하는 것이므로 여기서는 특별히 하는일이 없다.

faces-config.xml에 다음과 같이 converter 와 bean 그리고 navigation을 설정한다.

```

...
<faces-config>
...

    <converter>
        <description> Converter TEST </description>
        <converter-id>ConverterExample</converter-id>
        <converter-class>jsf.proj.converter.ConverterExample</converter-class>
    </converter>

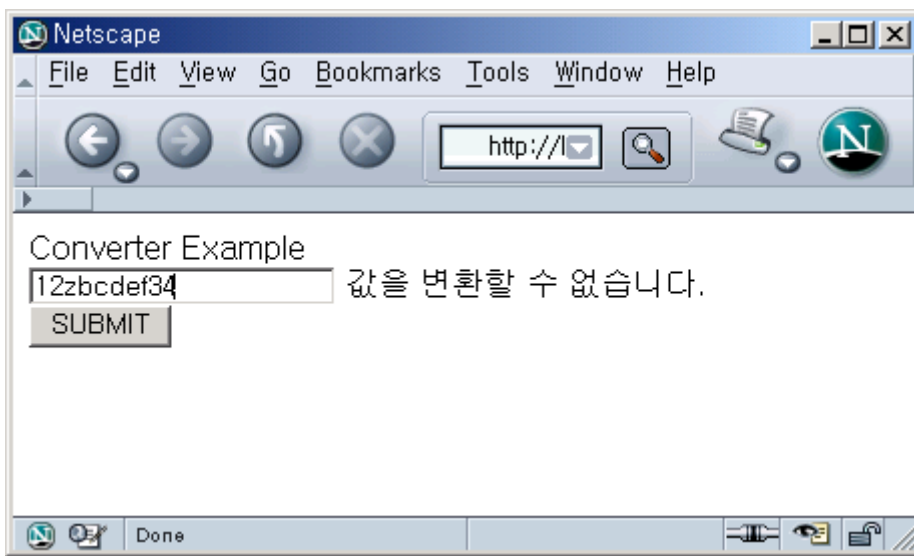
    <managed-bean>
        <managed-bean-name>ConverterBean</managed-bean-name>
        <managed-bean-class>jsf.proj.converter.ConverterBean</managed-bean-
class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
...

    <navigation-rule>
        <from-view-id>/jsp/example/custom_converter.jsp</from-view-id>
        <navigation-case>

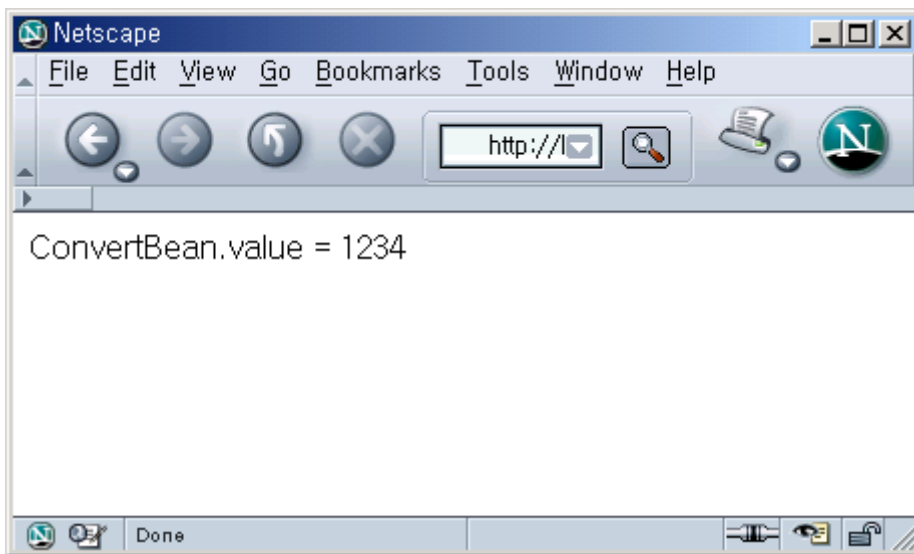
```

```
<from-outcome>success</from-outcome>
<to-view-id>/jsp/example/custom_converter_result.jsp</to-view-id>
</navigation-case>
</navigation-rule>
</faces-config>
```

다음과 같은 값을 입력해보자.



그러면, 다음과 같이 ConverterBean의 value 값이 지정되어 출력된다.



9. Custom Component 구현 -1

JSF를 이용하여 재사용가능한 컴포넌트를 만들어보자. JSF를 이용하는 궁극적인 목적은 가능한한 재사용이 쉬운 컴포넌트로 웹 어플리케이션을 개발하는 것이다.

이번장에서 만들어볼 컴포넌트는 쿼리만 주어진다면 DB로부터 데이터를 읽어서 브라우저에 출력하게끔 하는 컴포넌트이다. 쿼리된 컬럼이름과 결과를 Vector클래스에 저장하고, 반환한다. 컴포넌트를 만드는 방법은 앞서 설명한 Validator와 거의 동일한 과정을 따른다.

컴포넌트를 만들기 위해 필요한 작업은 다음과 같다.

작업	설명
태그	태그로 사용될 클래스를 작성한다. 태그로 사용될 클래스가 하는일은 컴포넌트의 이름과 렌더러의 이름(있다면)을 반환하고, 입력되는 파라미터의 ValueBinding을 수행한다. 그리고, 사용된 리소스를 해제한다.
TLD	태그 클래스 정의 TLD파일
컴포넌트	컴포넌트로 사용될 클래스이다. 이 예제에서는 javax.faces.component.UIData클래스를 바로 이용하였다.
렌더러	컴포넌트는 렌더러에 encode/decode작업을 위임할 수 있다. 위임된 encode/decode작업을 처리한다. 위임하지 않고, 컴포넌트 자체의 encode/decode 메소드를 이용해도 된다.
bean	DB로부터 얻어진 쿼리결과를 처리할 bean이다.

먼저 DB관련 환경설정을 Resources.properties 파일에 다음과 같이 하도록 한다.

```

DATABASE_DRIVER=com.mysql.jdbc.Driver
CONNECTION_URL=jdbc:mysql://localhost:3306/test?user=zzzccc&password=zzzccc&characterEncoding=euc-kr&useUnicode=true
QUERY=select * from test1
    
```

db에 test1테이블에 다음과 같이 임시 데이터를 넣어준다.

```

create table test1( id varchar(20), name varchar(20));
insert into test1 values('zzzccc','kimsk');
insert into test1 values('lee','LeeSunShin');
insert into test1 values('hong','HongGilDong');
insert into test1 values('park','ParkCH');
    
```

```
insert into test1 values('lim','LimGJ');
```

이 QUERY키 값으로 주어지는 쿼리를 수행하고, 결과를 브라우저에 나타내게 될 것이다.

9.1 태그 클래스 작성

태그 클래스는 다음과 같다.

```
package jsf.proj.component1;

import javax.faces.webapp.UIComponentTag;
import javax.faces.context.*;
import javax.faces.component.*;
import javax.faces.el.*;

public class DataFromDBTag extends UIComponentTag{
    //이 tag가 가지는 속성값이다.
    private String value=null;

    public void setValue(String value){
        print("setValue..");
        this.value=value;
    }
    //리소스를 해제한다.
    public void release()    {
        super.release();
        print("release..");
        value=null;
    }
    //어떤 컴포넌트를 이용할 것인가 ?
    public String getComponentType(){
        print("getComponentType.."+UIData.COMPONENT_TYPE);
        // 컴포넌트로 UIData를 이용한다.
        //JSF가 내부적으로 UIData클래스를 정의하는 문자열이다.
        return UIData.COMPONENT_TYPE;
    }
}
```

```

    }
    //Renderer 가 rendering을 수행한다.
    //컴포넌트가 수행하게 하고 싶으면 null을 반환한다.
    public String getRendererType(){
        print("getRendererType..");
        return "DBRenderer";
    }
    //jsf페이지로부터 입력되는 속성값을 바인딩한다.
    public void setProperties(UIComponent component){
        print("setProperties.");
        super.setProperties(component);
        FacesContext context = FacesContext.getCurrentInstance();

        if (value !=null){
            if (isValueReference(value)){
                ValueBinding
vb=context.getApplication().createValueBinding(value);
                component.setValueBinding("value",vb);
            }else{
                component.getAttributes().put("value",value);
            }
        }
    }
    public void print(String msg){
        System.out.println("DataFromDBTag."+msg);
    }
}

```

태그 클래스는 javax.faces.webapp.UIComponentTag 클래스를 상속하여 구현한다. 태그 클래스를 구성하는 메소드에 대해 알아보자.

메소드이름	설명
setValue()	jsp 페이지로부터 주어질 value 속성을 저장한다.
release()	사용하고 있는 리소스를 해제한다.
getComponentType()	컴포넌트의 이름을 반환한다. 여기서는 UIData를 컴포넌트로 이용하므로 UIData의 이름을 반환하고 있다.

getRendererType()	렌더러의 이름을 반환한다. 여기서 반환하는 렌더러 이름 DBRenderer는 faces-config.xml에서 참조한다. 이 메소드가 null값을 반환하면 컴포넌트 클래스의 encode/decode 메소드가 수행된다. 렌더러가 있다면(null 이 아니면) 주어진 렌더러의 encode/decode 메소드를 수행한다.
setProperty()	이 메소드는 jsp페이지로부터 주어지는 value속성에 주어지는 값이 평가되어야 할 식인지를 알아보기위해 isValueRederende()메소드를 수행한다. 만약, 평가되어야 할 식이라면 단순한 문자열로 처리되지 않고, ValueBinding을 수행한다. 만약, 평가 되어야 할 식이 아니라면, 속성값으로 저장한다.

9.2 TLD 파일의 작성

```
<?xml version="1.0" encoding="euc-kr" ?>

<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>0.03</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>DataFromDB Component</short-name>
  <description>
  </description>

  <tag>
    <name>dataFromDB</name>
    <tag-class>jsf.proj.component1.DataFromDBTag</tag-class>
    <body-content>JSP</body-content>

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

```
<rtexprvalue>>false</rtexprvalue>
</attribute>
</tag>
</taglib>
```

태그로 사용될 이름과 태그클래스 그리고, 사용될 속성을 정의한 TLD 파일이다.

9.3 컴포넌트 클래스

이번 예제에서는 컴포넌트 클래스를 구현하지 않고, 이미 구현되어 있는 `javax.faces.UIData`를 컴포넌트로 이용하였다.

9.4 렌더러

렌더러의 소스는 다음과 같다.

```
package jsf.proj.component1;

import java.io.IOException;
import java.util.Iterator;
import java.util.List;
import javax.faces.component.*;
import javax.faces.context.*;
import javax.faces.render.Renderer;

public class DataFromDBRenderer extends Renderer {
    public DataFromDBRenderer(){
        super();
    }
    public void encodeBegin(FacesContext context, UIComponent component) throws
    IOException {
        super.encodeBegin(context, component);
        print("encode begin...");
        ResponseWriter writer = context.getResponseWriter();
        //현재 컴포넌트는 tag 클래스에서 정의한 컴포넌트 UIData이다.
        UIData data = (UIData) component;
```

```

        print("ROW AVAILABLE = "+data.isRowAvailable() +" , ROW INDEX =
"+data.getRowIndex() );
        //컴포넌트로부터 List를 얻는다.
        List list=(List)data.getValue();
        Iterator iter=list.iterator();
        writer.write("<table border='1'>");
        while(iter.hasNext()){
            writer.write("<tr>");
            String[] rs=(String[])iter.next();

            for(int i=0;i<rs.length;i++){
                if(rs[i].length()>0) writer.write("<td>"+rs[i]+"</td>");
                else writer.write("<td> * </td>");//컬럼값이 없으면 -을
출력한다.
            }
            writer.write("<tr>");
        }
        writer.write("</table>");
    }
    public void encodeEnd(FacesContext context, UIComponent component) throws
IOException {
        print("encode end...");
    }

    public void print(String msg){
        System.out.println("DataFromDBRenderer."+msg);
    }
}

```

렌더러는 현재 JSF의 context와 이 렌더러에 encode/decode메소드를 위임한 컴포넌트를 파라미터로 가진다. encodeBegin 메소드가 FacesContext객체로부터 ResponseWriter를 얻어 브라우저로의 출력을 위한 작업을 수행한다. 출력될 데이터를 가진 UIData컴포넌트로부터 쿼리된 결과를 List형으로 받아서 테이블을 생성한다. encodeEnd 메소드는 따로 작업할 내용이 없다.

9.5 bean

DataFromDBBean 클래스는 Resources.properties 에 있는 QUERY키 값으로 주어진 쿼리를 수행하고, 그 결과를 TableData클래스에 저장한다.

```
package jsf.proj.component1;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.List;
import java.util.ResourceBundle;

public class DataFromDBBean {
    // ResourceBundle파일이 있는 경로
    private String RESOURCE_BUNDLE_PATH="jsf.proj.resource.Resources";
    private String DRIVER_CLASS;
    private String CONNECTION_URL;
    private String QUERY;
    private Connection conn=null;
    private Statement stmt=null;
    private ResultSet rs=null;
    private ResultSetMetaData rsMetaData=null;
    private List list=null;

    public DataFromDBBean() {
        // 에러시 문자열을 출력하기 위해 Resource.properties에서 필요한 문자열을
        // 읽어온다.
        ResourceBundle rb=ResourceBundle.getBundle(RESOURCE_BUNDLE_PATH);
        DRIVER_CLASS=rb.getString("DATABASE_DRIVER");
        CONNECTION_URL=rb.getString("CONNECTION_URL");
        QUERY=rb.getString("QUERY");
    }

    public List getList() {
```

```

print("getList");
try{
    Class.forName(DRIVER_CLASS).newInstance();
    conn=DriverManager.getConnection(CONNECTION_URL);
    stmt=conn.createStatement();

    rs=stmt.executeQuery(QUERY);
    //컬럼이름을 얻기위해 메타데이터를 얻는다.
    rsMetaData=rs.getMetaData();
    // 컬럼이름을 vector에 넣는다.
    TableData tableData=new TableData(rsMetaData);
    while(rs.next())
        //ResultSet을 vector에 넣는다.
        tableData.setData(rs);
    list=tableData.getTableData();
    stmt.close();
    conn.close();
}catch(Exception e){
    System.out.println(e.toString());
    try{
        stmt.close();
        conn.close();
    }catch(Exception ee){}
}
return list;
}
public void print(String msg){
    System.out.println("DataFromDBBean."+msg);
}
}

```

TableData클래스는 ResultSet을 파라미터로 받아서, 배열의 맨처음에 컬럼이름을 저장하고, 그다음부터 각각의 레코드셋을 저장한다. ResultSetMetaData 클래스는 ResultSet에서 컬럼 이름을 얻기위한 메타데이터를 포함하고 있는 클래스이다.

TableData클래스의 소스는 다음과 같다.

```
package jsf.proj.component1;

import java.sql.*;
import java.util.*;

public class TableData{

    private List table;
    private int columnCount;

    TableData(ResultSetMetaData rsMetaData) throws SQLException{
        table=new ArrayList();
        //컬럼이름을 저장한다.
        setColumnData(rsMetaData);
    }
    public List getTableData(){
        return table;
    }
    //ResultSetMetaData를 이용해서 컬럼이름을 얻는다.
    //컬럼이름은 vector의 firstElement로 저장된다.
    private void setColumnData(ResultSetMetaData rsMetaData) throws
SQLException{
        this.columnCount=rsMetaData.getColumnCount();
        String[] data=new String[columnCount];
        //컬럼의 인덱스는 1부터 시작한다.
        for (int i=1;i<=columnCount;i++)
            data[i-1]=rsMetaData.getColumnName(i);
        table.add(data);
    }
    // ResultSet의 결과를 Vector에 넣는다.
    public void setData(ResultSet rs) throws SQLException{
        String[] data=new String[columnCount];
        for(int i=1;i<=columnCount;i++)
            data[i-1]=rs.getString(i);
    }
}
```

```
        table.add(data);
    }
}
```

9.6 faces-config.xml

컴포넌트를 사용하기 위해 컴포넌트를 faces-config.xml에 등록하는 작업이 필요하다. 다음과 같이 설정한다. 이 예제에서 사용된 컴포넌트 UIData는 JSF가 이미 알고 있으므로 따로 등록할 필요는 없다.

```
<?xml version="1.0" encoding="euc-kr"?>
<!DOCTYPE faces-config PUBLIC
    "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
    "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>

    <application>
        <message-bundle>jsf.proj.resource.Messages</message-bundle>
        <locale-config>
            <default-locale>ko</default-locale>
            <supported-locale>de</supported-locale>
            <supported-locale>fr</supported-locale>
            <supported-locale>es</supported-locale>
        </locale-config>
    </application>
    ...
    <!-- DataFromDB Bean -->
    <managed-bean>
        <managed-bean-name>DataFromDBBean</managed-bean-name>
        <managed-bean-class>jsf.proj.component1.DataFromDBBean</managed-
bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    ...
```

```

        <render-kit>
            <renderer>
                <component-family>javax.faces.Data</component-family>
                <renderer-type>DBRenderer</renderer-type>
                <renderer-
class>jsf.proj.component1.DataFromDBRenderer</renderer-class>
            </renderer>
        </render-kit>
</faces-config>

```

9.7 테스트

이것으로 컴포넌트의 작성이 모두 끝났다. 테스트를 위해 다음의 jsp페이지를 만들어보자.

```

<!--
        /jsp/component1/datafromdb.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="/WEB-INF/datafromdb.tld" prefix="db" %>
<html>
<body>
<f:view>
<h:form id="data-db-form">
<db:dataFromDB value="#{DataFromDBBean.list}"/>
</h:form>
</f:view>
</body>
</html>

```

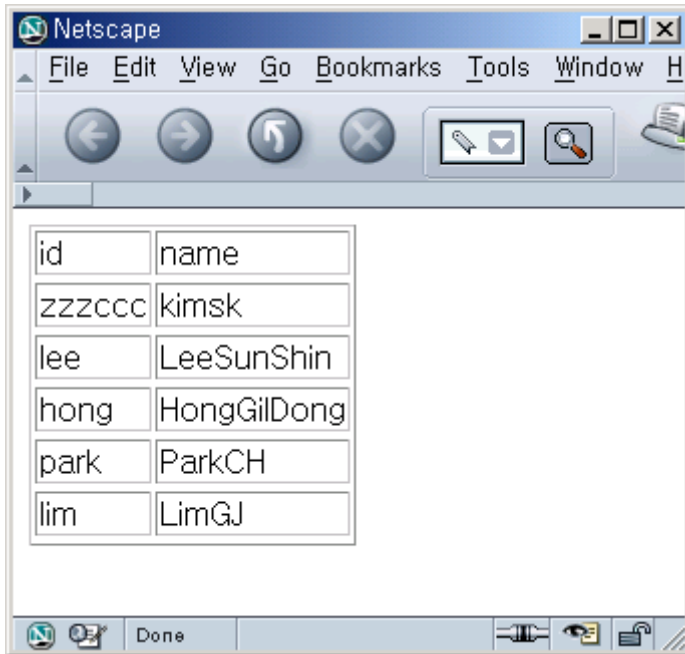
태그를 사용하기 위해 TLD파일의 위치와 별명을 다음과 같이 지정해야 한다.

```

<%@ taglib uri="/WEB-INF/datafromdb.tld" prefix="db" %>

```

http://localhost:8080/jsf-proj/faces/jsp/component1/datafromdb.jsp



위와 같은 결과가 브라우저에 나타날 것이다. CSS를 지원할 속성을 몇가지 추가하여, 보기 좋도록 꾸미면 될것이다.

처음 컴포넌트를 만들 때 복잡하다고 느낄수 있지만, 몇번 만들어 보면 의외로 간단하다. 서블릿과 커스텀 태그를 섞어 쓴다는 느낌으로 만들어 보면 될 것 같다.

다음장에서는 다른 컴포넌트를 하나더 구현해보도록 하자.

10. Custom Component 2

이번에 만들어볼 컴포넌트는 Resources.properties에 지정된 이미지를 나열하고, 선택된 이미지에 대한 설명과 이미지를 보여주는 간단한 이미지 게시판 정도로 보면 되겠다. 여기서 구현할 컴포넌트는 MethodBinding을 이용하여 이벤트를 처리하도록 하는 부분을 유의해서 보도록 하자. 나머지 부분은 앞서 설명한 컴포넌트와 별반 다를 것이 없다.

컴포넌트를 작성하는 과정역시 앞의 과정과 동일하므로, 바로 소스에 대한 설명을 하도록 한다.

먼저 Resources.properties에 다음의 내용을 추가하자.

```
# image-list.jsp파일에 나타날 제목
imgList_title=Image를 Display하는 custom component

# 화면에 표시할 이미지의 총 갯수를 표시한다.
imgList_total=9

# 이미지 파일의 경로
imgList_img0=/jsp/images/img1.jpg
imgList_img1=/jsp/images/img2.jpg
imgList_img2=/jsp/images/img3.jpg
imgList_img3=/jsp/images/img4.jpg
imgList_img4=/jsp/images/img5.jpg
imgList_img5=/jsp/images/img6.jpg
imgList_img6=/jsp/images/img7.jpg
imgList_img7=/jsp/images/img8.jpg
imgList_img8=/jsp/images/img9.jpg

# 각 이미지 파일에 대한 설명
imgList_desc_img0=그림 1에 대한 설명입니다.
imgList_desc_img1=그림 2에 대한 설명입니다.
imgList_desc_img2=그림 3에 대한 설명입니다.
imgList_desc_img3=그림 4에 대한 설명입니다.
imgList_desc_img4=그림 5에 대한 설명입니다.
imgList_desc_img5=그림 6에 대한 설명입니다.
```

```
imgList_desc_img6=그림 7에 대한 설명입니다.
```

```
imgList_desc_img7=그림 8에 대한 설명입니다.
```

```
imgList_desc_img8=그림 9에 대한 설명입니다.
```

우선 모든 이미지가 나열될 때 표시할 제목과, 이미지 파일의 개수, 각 이미지 파일의 경로 명 그리고, 각 이미지에 대한 간단한 설명을 추가한다. 위의 9개의 이미지가 image-list.jsp 파일에서 열리고, 이미지를 클릭하면 해당 이미지에 대한 설명과 앞뒤의 이미지를 볼수 있는 링크가 생성되도록 컴포넌트를 구성한다. 이러한 작업을 수행하도록 하기 위해 이미지를 나열하는 작업을 하는 컴포넌트, 그리고 하나의 이미지에 대한 표시와 이미지의 이동을 담당할 링크를 생성하는 작업을 하는 컴포넌트 이렇게 두개의 컴포넌트가 구현될 것이다.

먼저, Resources.properties를 읽어 모든 이미지를 표시할 컴포넌트를 만들어보자.

10.1 ImageListComponent의 구현

10.1.1 태그 클래스

ImageListComponent 클래스를 위한 태그 클래스는 다음과 같다.

```
package jsf.proj.component2;

import javax.faces.component.UIComponent;
import javax.faces.webapp.UIComponentTag;
import javax.faces.context.*;
import javax.faces.component.*;
import javax.faces.el.*;
import javax.faces.event.*;

public class ImageListTag extends UIComponentTag{
    private String actionListener;
    private String title;
    private String width;
    private String height;
    private String value;

    public void setActionListener(String actionListener) { this.actionListener =
```

```

actionListener; }

    public void setTitle(String title) { this.title= title; }
    public void setWidth(String width) { this.width=width; }
    public void setHeight(String height) { this.height=height; }
    public void setValue(String value) { this.value=value; }

    public String getComponentType(){
        return "ImageListComponent";
    }
    public String getRendererType(){
        return null;
    }
    public void release(){
        super.release();
        title=null;
        width=null;
        height=null;
        actionListener=null;
        value=null;
    }
    public void setProperties(UIComponent component){
        super.setProperties(component);
        ValueBinding vb=null;
        FacesContext context = FacesContext.getCurrentInstance();
        if (actionListener != null) {
            if (isValueReference(actionListener)) {
                Class args[] = { ActionEvent.class };
                MethodBinding mb =
FacesContext.getCurrentInstance().getApplication().createMethodBinding(actionListener,
args);

                ((ImageListComponent)component).setActionListener(mb);
                print(" actionListener");
            } else {
                Object params [] = {actionListener};
                throw new javax.faces.FacesException();
            }
        }
    }
}

```

```

    }
}
if (title !=null){
    if (isValueReference(title)){
        vb = context.getApplication().createValueBinding(title);
        component.setValueBinding("title",vb);
        print(" title");
    }else{
        component.getAttributes().put("title",title);
    }
}
if (width !=null){
    if (isValueReference(width)){
        vb = context.getApplication().createValueBinding(width);
        component.setValueBinding("width",vb);
        print(" width");
    }else{
        component.getAttributes().put("width",width);
    }
}
if (height !=null) {
    if (isValueReference(height)){
        vb =
context.getApplication().createValueBinding(height);
        component.setValueBinding("height",vb);
        print(" height");
    }else{
        component.getAttributes().put("height",height);
    }
}
if (value !=null){
    if (isValueReference(value)){
        vb = context.getApplication().createValueBinding(value);
        component.setValueBinding("value",vb);
        print(" value");
    }else{

```

```
        component.getAttributes().put("value",value);
    }
}
}
public void print(String msg){
    System.out.println("ImageListTag."+msg);
}
}
```

이 태그 클래스는 표시될 이미지의 크기를 결정할 속성 width, height와 이미지의 배열을 반환할 value 속성, 그리고, 지정된 이벤트를 처리할 actionListener속성을 가진다. 사실, actionListener속성에 지정된 메소드는 그냥 단순히 문자열만 출력하고 하는일은 없는 메소드이다. 여기에 actionListener 속성을 사용한 이유는 actionListener속성에 지정되는 문자열은 ValueBinding으로 처리되지 않고, MethodBinding을 이용한다는 것을 설명하기 위해서이다.

위의 소스중 다음부분을 살펴보자.

```
...
Class args[] = { ActionEvent.class };
        MethodBinding mb =
FacesContext.getCurrentInstance().getApplication().createMethodBinding(actionListener,
args);
((ImageListComponent)component).setActionListener(mb);
...

```

ValueBinding과 마찬가지로, MethodBinding역시 평가되어야 할 문자열을 받아서, 해당 문자열을 처리하여야 하는데, ValueBinding처럼 결과값이 있는 것이 아니다. 그리고, 파라미터로 처리될 클래스 역시 필요하다. 따라서, 파라미터로 처리될 클래스정보를 배열로 만들어 해당 클래스 타입을 args[]로 넘길 필요가 있다. 앞으로 작성하게 될 ImageListComponent는 UICommand를 상속받아 사용하므로, UICommand클래스의 메소드 setActionListener()에 평가된 MethodBinding을 설정한다. JSF가 제공하는 기본적인 컴포넌트들의 actionListener나 valueChangeListener역시 내부적으로 이와 동일한 과정을 처리하도록 되어 있다. 이 컴포넌트는 렌더러를 따로 구현하지 않으므로 getRendererType()메소드가 null을 반환한다. 따라서, 컴포넌트가 자체적으로 encode/decode 메소드를 구현할 것이다.

나머지 부분은 앞장의 태그 클래스와 동일하므로 설명을 생략하도록 한다.

10.1.2 TLD파일의 작성

ImageListComponent 태그에서 이용될 속성들을 지정한다.

```
<?xml version="1.0" encoding="euc-kr" ?>
<!DOCTYPE taglib
PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
  <tlib-version>0.03</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>ImageList component tag library</short-name>
  <tag>
    <name>imageList</name>
    <tag-class>jsf.proj.component2.ImageListTag</tag-class>
    <body-content>JSP</body-content>
    <description>Resorces.properties에서 읽은 이미지들을 표시한다.</description>
    <attribute>
      <name>title</name>
      <required>>false</required>
      <rtexprvalue>>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>width</name>
      <required>>false</required>
      <rtexprvalue>>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>height</name>
      <required>>false</required>
      <rtexprvalue>>false</rtexprvalue>
    </attribute>
    <attribute>
      <name>value</name>
      <required>>false</required>
      <rtexprvalue>>false</rtexprvalue>
  </tag>
</taglib>
```

```
</attribute>
<attribute>
  <name>actionListener</name>
  <required>>false</required>
  <rtexprvalue>>false</rtexprvalue>
</attribute>
</tag>
</taglib>
```

10.1.3 컴포넌트 클래스

아래에 컴포넌트 클래스의 소스를 나타내었다. 이 컴포넌트 클래스는 jsp페이지에서 읽은 속성값을 얻어내고, 자바스크립트 생성을 위해 form컴포넌트를 찾아서(form이 view태그내에 여러 개 있을수 있으므로) 해당 form컴포넌트의 id를 얻고, html을 생성한다.

```
package jsf.proj.component2;

import java.io.IOException;
import javax.faces.el.*;
import javax.faces.component.*;
import javax.faces.context.*;
import javax.faces.context.ResponseWriter;
import javax.faces.el.MethodBinding;
import javax.faces.event.*;
import java.util.*;

public class ImageListComponent extends UICommand{

    public ImageListComponent() {
        super();
    }

    public String getFamily(){
        return"ImageDisplayComponent";
    }
}
```

```

public void encodeBegin(FacesContext context) throws IOException {
    print("encode begin...");
}

public void encodeEnd(FacesContext context) throws IOException{

    print("encodeEnd...");

    ExternalContext ec=context.getExternalContext();
    ResponseWriter writer = context.getResponseWriter();
    //width,height는 표현식이 아니고 값으로 들어오므로, 속성값에서 얻을 수 있다.
    String width=(String)getAttributes().get("width");
    String height=(String)getAttributes().get("height");
    // 표현식으로부터 값을 얻어낸다.
    ValueBinding t = getValueBinding("title");
    String title=(String)t.getValue(context);
    ValueBinding vb = getValueBinding("value");
    Vector imgList=(Vector)vb.getValue(context);
    //context path를 얻는다.
    String path=ec.getRequestContextPath();
    //스크립트 생성을 위해 form의 id를 얻어낸다.
    String id=getFormId(context);
    //html을 생성한다.
    writer.write("<input type='hidden' name='currentIndex'>");
        writer.write("<h2 align='center'>" +title+ "</h2><p>");
        writer.startElement("table",null);
        writer.startElement("tr",null);
        for(int i=0;i<imgList.size();i++){
            ImageBean img=(ImageBean)imgList.elementAt(i);
            writer.startElement("td",null);
            writer.write("<a href='#"
onmousedown=W"document.forms[" +id+ "]" ['currentIndex'].value="+i+";document.forms
[" +id+ "].submit()W">");
            writer.write("<img src='"+path+(img.getImg())+"' width='"+width+"
height='"+height+"' >");

```

```
        writer.endElement("a");
        writer.endElement("td");
    }
    writer.endElement("tr");
    writer.endElement("table");
}

public void decode(FacesContext context){
    print("decode");
    setAction(new MethodBindingImpl("success"));
    queueEvent(new ActionEvent(this));
}

//form의 id를 찾는다.
protected String getFormId(FacesContext context){
    String formId=null;
    UIViewRoot root=context.getViewRoot();
    Iterator children=root.getChildren().iterator();
    while(children.hasNext()){
        UIComponent comp=(UIComponent)children.next();
        if(comp instanceof UIForm) {
            formId=comp.getId();
            break;
        }
    }
    return formId;
}

public void print(String msg){
    System.out.println("ImageListComponent."+msg);
}
}
```

위의 소스를 보면, 이미지의 크기를 나타내는 width, height는 getAttributes()메소드를 이용해 값을 얻고 있다. width, height속성이 jsp페이지로부터 #{ }로 둘러싸인 표현식이 아닌 단순히 값만 주어지기 때문이다. 또, <a href...>링크에 사용될 form의 id를 얻기 위해

getFormId()라는 메소드를 구현하였다.

decode()메소드를 살펴보자. decode()메소드는 이미지를 클릭하는 경우 MethodBinding개체를 생성하고, 이 MethodBinding개체를 컴포넌트에 설정한다. MethodBinding개체에 파라미터로 주어지는 success라는 문자열은 faces-config.xml의 <navigation-rule>의 <from-outcome>에 지정된 문자열이다. queueEvent() 메소드로 컴포넌트의(ActionEvent)를 저장한다. 아래의 MethodBinding인터페이스를 구현한 클래스의 invoke()메소드가 반환하는 문자열이(success 가 될것이다) NavigationHandler에 전달되어 페이지가 이동될 것이다.

다음은 MethodBindingImpl클래스의 소스이다.

```
package jsf.proj.component2;

import javax.faces.context.FacesContext;
import javax.faces.el.MethodBinding;

public class MethodBindingImpl extends MethodBinding {
    private String action = null;
    public MethodBindingImpl() {
        print("constructor");
    }
    public MethodBindingImpl(String action) {
        print("constructor action = "+action);
        this.action = action;
    }
    public Object invoke(FacesContext context, Object params[]) {
        print("invoke");
        return action;
    }
    public Class getType(FacesContext context) {
        return String.class;
    }
    public void print(String msg){
        System.out.println("MethodBindingImpl"+msg);
    }
}
```

10.1.4 bean클래스 작성

bean클래스는 Resources.properties로부터 이미지에 관련된 정보를 읽고 배열로 저장하고, 앞서 설명한 ActionListener에 지정된 메소드를 구현하는 클래스이다.

```
package jsf.proj.component2;
import javax.faces.component.*;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
import java.util.*;
public class ImageListBean{
    private Vector list=null;
    private String RESOURCE_BUNDLE_PATH="jsf.proj.resource.Resources";
    private String IMAGE_FILE="imgList_img";
    private String IMAGE_DESC="imgList_desc_img";
    private int IMAGE_TOTAL;
    private ResourceBundle rb;
    public ImageListBean() {
    }
    public Vector getList(){
        rb=ResourceBundle.getBundle(RESOURCE_BUNDLE_PATH);
        IMAGE_TOTAL=Integer.parseInt(rb.getString("imgList_total"));
        list=new Vector();
        for(int i=0;i<IMAGE_TOTAL;i++){
            String file=rb.getString(IMAGE_FILE+i);
            String desc=rb.getString(IMAGE_DESC+i);
            list.addElement(new ImageBean(file,desc));
        }
        //System.out.println("getList() 메소드가 호출되었습니다.");
        return list;
    }
    public void processClickEvent(ActionEvent event) {
        FacesContext context = FacesContext.getCurrentInstance();
        UIComponent component = event.getComponent();
        System.out.println("컴포넌트 id = "+component.getId()+"에서 이벤트가
```

```
발생했습니다.");  
    }  
}
```

Resources.properties에서 읽은 이미지파일 경로와 이미지에 대한 설명을 저장하는 ImageBean클래스는 다음과 같다.

```
package jsf.proj.component2;  
  
public class ImageBean {  
  
    private String img;//이미지 경로  
    private String desc;//이미지에 대한 설명  
  
    public ImageBean(){  
  
    }  
    public ImageBean(String img,String desc){  
        this.img=img;  
        this.desc=desc;  
    }  
    public String getDesc() {  
        return desc;  
    }  
    public void setDesc(String desc) {  
        this.desc = desc;  
    }  
    public String getImg() {  
        return img;  
    }  
    public void setImg(String img) {  
        this.img = img;  
    }  
}
```

10.2 ImageDisplayComponent

ImageDisplayComponent 컴포넌트는 이미지리스트에서 선택된 이미지와 이미지에 대한 설명을 표시하고, 이미지를 앞뒤로 이동하는 링크를 생성한다. 역시, 컴포넌트를 만드는 과정은 동일하다.

10.2.1 태그 클래스

ImageListComponent에서 반환하는 이미지를 포함하는 Vector클래스를 value속성으로 가진다. 소스는 다음과 같다.

```
package jsf.proj.component2;

import javax.faces.component.UIComponent;
import javax.faces.webapp.UIComponentTag;
import javax.faces.el.*;
import javax.faces.context.*;
import javax.faces.application.*;
import javax.faces.*;

public class ImageDisplayTag extends UIComponentTag{

    private String value;
    public void setValue(String value) { this.value=value; }

    public String getComponentType(){
        return "ImageDisplayComponent";
    }
    public String getRendererType(){
        return null;
    }
    public void release(){
        super.release();
        value=null;
    }
}
```

```
public void setProperties(UIComponent component){
    super.setProperties(component);
    ValueBinding vb=null;
    FacesContext context = FacesContext.getCurrentInstance();
    if (value !=null){
        if (isValueReference(value)){
            vb = context.getApplication().createValueBinding(value);

            ((ImageDisplayComponent)component).setValueBinding("value",vb);

        }else{
            component.getAttributes().put("value",value);
        }
    }
}
}
```

10.2.2 TLD 파일의 작성

10.1.2 에서 작성한 TLD파일에 다음을 추가한다.

```
...
<tag>
  <name>imageDisplay</name>
  <tag-class>javax.faces.componentIMG.ImageDisplayTag</tag-class>
  <body-content>JSP</body-content>
  <description> 하나의 이미지를 표시하는 컴포넌트 태그이다.
</description>
  <attribute>
    <name>value</name>
    <required>>false</required>
    <rtexprvalue>>false</rtexprvalue>
  </attribute>
</tag>
...
```

10.2.3 컴포넌트 클래스

선택된 이미지를 표시하고, prev, next 링크를 생성한다.

```
package jsf.proj.component2;

import java.io.IOException;
import java.util.Map;
import java.util.Vector;

import javax.faces.component.*;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.faces.context.ResponseWriter;
import javax.faces.el.*;

public class ImageDisplayComponent extends UIComponentBase {

    private Vector vec;

    public String getFamily(){
        print("getFamily called...");
        return "ImageDisplayComponent";
    }

    public ImageDisplayComponent() {
        super();
        print("constructor");
    }

    public void encodeBegin(FacesContext context) throws IOException {
        print("encode begin...");
        ValueBinding vb = getValueBinding("value");
```

```

Vector imgList=(Vector)vb.getValue(context);

//Vector imgList=getValue();
if(imgList==null || imgList.isEmpty()){
    ExternalContext ec=context.getExternalContext();
    ec.redirect("image-list.jsp");
}else{
    print("Vector size = "+imgList.size());
}
}

public void encodeEnd(FacesContext context) throws IOException{

    print("encodeEnd...");

    ExternalContext ec=context.getExternalContext();
    Map request=ec.getRequestParameterMap();
    String path=ec.getRequestContextPath();

    ValueBinding vb = getValueBinding("value");
    Vector imgList=(Vector)vb.getValue(context);
    int inx=0;
    try{
        inx=Integer.parseInt((String)request.get("currentIndex"));
    }catch(Exception e){
        ec.redirect("image-list.jsp");
    }
    ImageBean currentImgBean=(ImageBean)imgList.elementAt(inx);

    ResponseWriter writer = context.getResponseWriter();
    writer.write("<input type='hidden' name='currentIndex' value='"+inx+"'>");
    writer.write("<table border='1'>");
    writer.write("<tr align='center'><td colspan='3'>");
    writer.write(currentImgBean.getDesc());
    writer.write("</td></tr>");
    writer.write("<tr align='center'><td colspan='3'>");

```

```

writer.write("<img src='"+path+currentImgBean.getImg()+">");
writer.write("</td></tr>");
writer.write("<tr align='center'><td>");
writer.write(getPrev(inx,imgList));
writer.write("</td><td>");
writer.write("<a href='image-list.jsp'>HOME</a>");
writer.write("</td><td>");
writer.write(getNext(inx,imgList));
writer.write("</td></tr>");
writer.write("</table>");
writer.writeComment("이것은 연습입니다. ");
}

public void decode(FacesContext context){
    print("decode");
}

// previous 링크를 생성하는 문자열을 반환한다.
public String getPrev(int inx,Vector imgList){
    if (inx==0) return "PREV";
    String p="document.forms[0].currentIndex.value="+(--inx)+"";
    p+="document.forms[0].submit();";
    String prev="<a href='#' onmousedown=W"+p+W">PREV</a>";
    return prev;
}

// next 링크를 생성하는 문자열을 반환한다.
public String getNext(int inx,Vector imgList){
    if (inx==(imgList.size()-1)) return "NEXT";
    String p="document.forms[0].currentIndex.value="+(++inx)+"";
    p+="document.forms[0].submit();";
    String next="<a href='#' onmousedown=W"+p+W">NEXT</a>";
    return next;
}

public void print(String msg){
    System.out.println("ImageDisplayComponent."+msg);
}

```

```

    public Vector getVec() {
        return vec;
    }
    public void setVec(Vector vec) {
        this.vec = vec;
    }
}

```

10.3 faces-config.xml

드디어 설정이다.

다음의 내용을 faces-config.xml에 추가하자.

```

...
<faces-config>
...
    <!-- ImageListComponent 등록 -->
        <component>
            <component-type>ImageListComponent</component-type>
            <component-
class>jsf.proj.component2.ImageListComponent</component-class>
        </component>
        <component>
            <component-type>ImageDisplayComponent</component-type>
            <component-
class>jsf.proj.component2.ImageDisplayComponent</component-class>
        </component>
    <!-- ImageList Bean -->
        <managed-bean>
            <managed-bean-name>ImageListBean</managed-bean-name>
            <managed-bean-class>jsf.proj.component2.ImageListBean</managed-
bean-class>
            <managed-bean-scope>request</managed-bean-scope>
        </managed-bean>

```

```

...
    <navigation-rule>
      <from-view-id>/jsp/component2/image-list.jsp</from-view-id>
      <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/jsp/component2/image-list-result.jsp</to-view-id>
      </navigation-case>
    </navigation-rule>

    <navigation-rule>
      <from-view-id>/jsp/component2/image-result.jsp</from-view-id>
      <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/jsp/component2/image-list.jsp</to-view-id>
      </navigation-case>
    </navigation-rule>
...
</faces-config>

```

10.4 테스트

테스트는 간단하다. 다음의 두개의 jsp파일을 만들자.

이미지의 리스트를 나타낼 jsp페이지이다.

```

<!--
    /jsp/component2/image-list.jsp
-->
<%@ page contentType="text/html;charset=euc-kr" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="/WEB-INF/imageList.tld" prefix="imgList" %>

<html>

```

```

<f:loadBundle basename="jsf.proj.resource.Resources" var="resource" />

<body>
<f:view>

<h:form id="imgListForm">

<imgList:imageList title="#{resource.imgList_title}"
width="100"
height="100"
actionListener="#{ImageListBean.processClickEvent}"
value="#{ImageListBean.list}"/>

</h:form>
</f:view>
</body>
</html>

```

선택된 이미지와 설명을 나타낼 페이지이다.

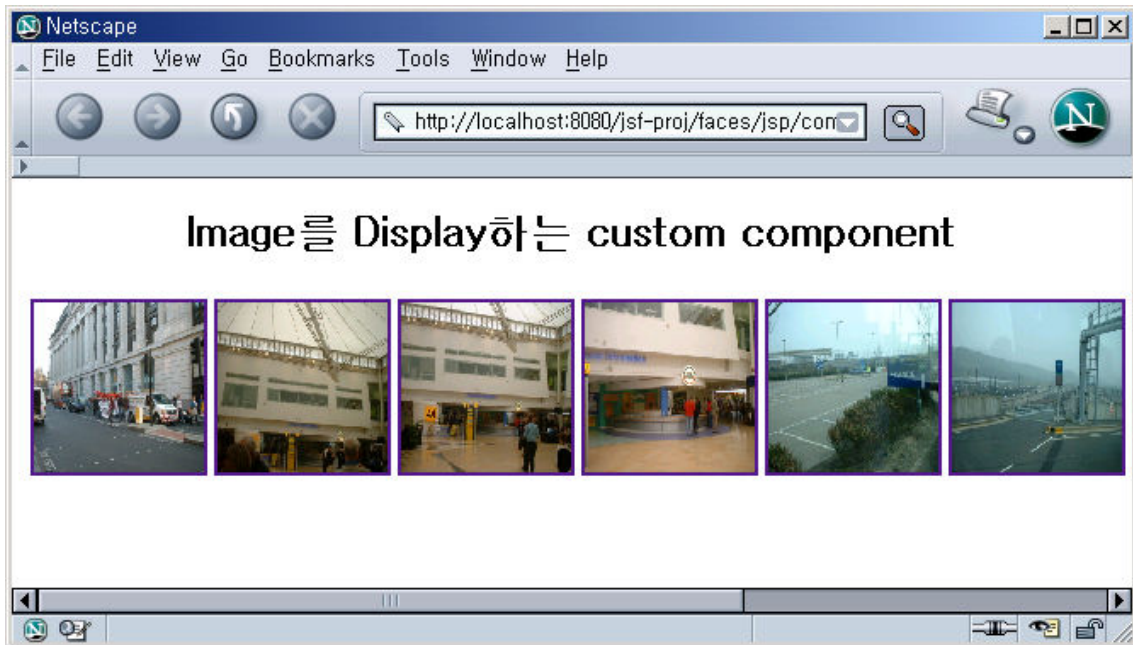
```

<!--
      /jsp/component2/image-list-result.jsp
-->
<%@page contentType="text/html;charset=euc-kr"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="/WEB-INF/imageList.tld" prefix="imgList" %>

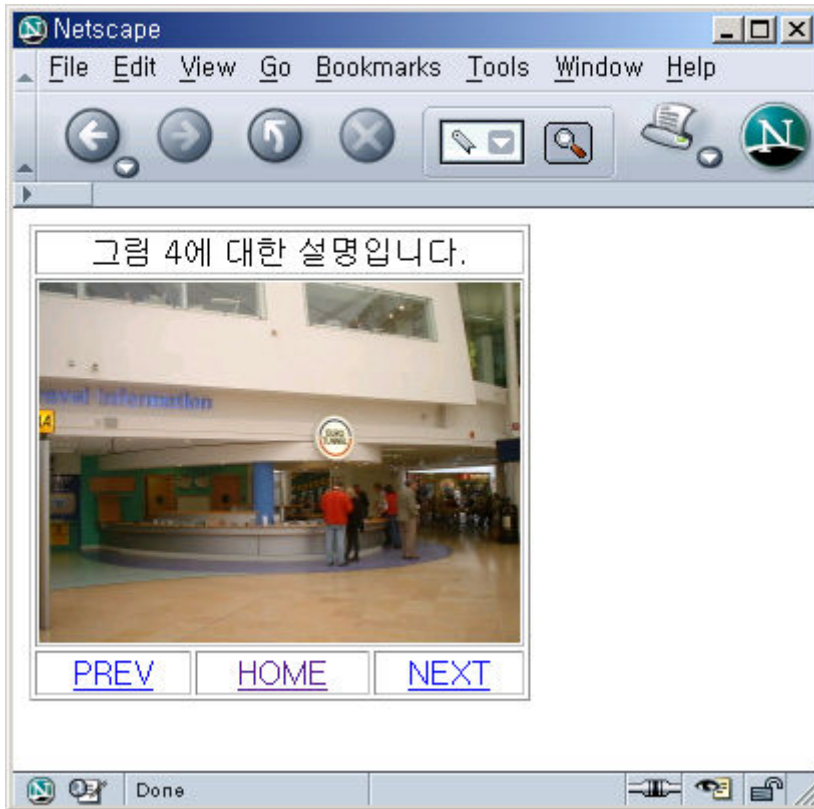
<html>
<body>
<f:view>
<h:form id="img-list-result-form">
  <imgList:imageDisplay value="#{ImageListBean.list}" />
</h:form>
</f:view>
</body>
</html>

```

다음과 같이 이미지들이 나타난다.



아무 이미지나 클릭해보자.



* 마이크로소프트의 인터넷 익스플로러는 잘 작동하는 반면에, 넷스케이프에서는 가끔 링크가 이상하게 나타나기도 한다. 자바스크립트의 문제인지 JSF의 문제인지 아직 확실치 않다. 이 컴포넌트는 인터넷 익스플로러로 테스트해보기 바란다.