JSP & Servlet #6
– 웹 애플리케이션 구축

목 차

Readings

Readings:

- ***Chapter 3:***

  *초 간단 미니 **MVC** 튜토리얼 : 초 간단 **MVC***

- ***Chapter 5:***

  웹 애플리케이션이 되어 보자 : 속성과 리스너

- ***Chapter 8:***

  *스크립트가 없는 페이지 : 스크립트가 없는 **JSP***

Objective

- MVC 패턴을 사용하여(서블릿이 프로세스를 담당하고, JSP가 화면을 담당) 웹 애플리케이션을 개발할 수 있다.
- 변환 또는 요청 시간을 기준으로 JSP 페이지 안에 파일을 포함할 수 있다.
- web.xml 파일에 초기화 파라미터를 설정하여 서블릿에서 파라미터 값을 사용할 수 있다.
- Model 1 아키텍처와 Model 2 아키텍처(MVC 패턴)의 차이를 설명할 수 있다.
- MVC 패턴(Model-View-Controller)을 따르는 애플리케이션 안에서 Controller, View, Model, Data store의 역할과 동작 방식을 설명할 수 있다.

1.Email List 애플리케이션

1. 서블릿 코드

❑ EmailServlet.java 소스

```
package email6;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import business.User;
import data.UserIO;

public class EmailServlet extends HttpServlet{

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws IOException, ServletException{

        String firstName = request.getParameter("firstName");
        String lastName = request.getParameter("lastName");
        String emailAddress = request.getParameter("emailAddress");
```

1. 서블릿 코드

❑ EmailServlet.java 소스 (계속)

```
            User user = new User(firstName, lastName, emailAddress);
            UserIO.addRecord(user, /UserEmail.txt);

            RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(
                                           "/email6/show_email_entry.jsp");
            dispatcher.forward(request, response);
        }

        public void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
            doGet(request, response);
        }
    }
```

2. JSP 코드

❑ show_email_entry.jsp 소스

```
<html>

<head>
  <title>JSP&Servlet 6 - Email List application</title>
</head>

<body>
<%
    String firstName = request.getParameter("firstName");
    String lastName = request.getParameter("lastName");
    String emailAddress = request.getParameter("emailAddress");
%>

<h1>메일 리스트에 가입되었습니다.</h1>
<p>입력한 정보 내역입니다. :</p>
```

2. JSP 코드

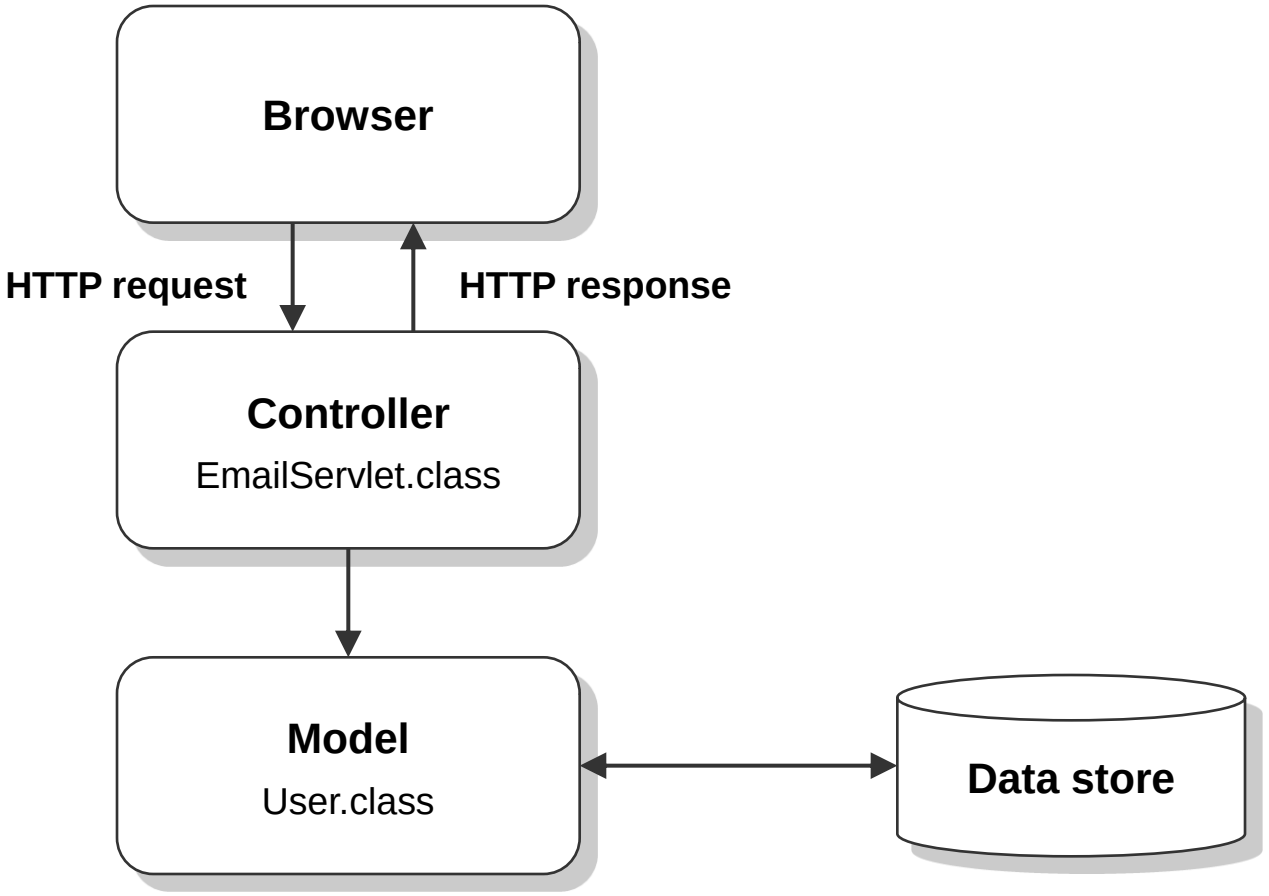
❑ show_email_entry.jsp 소스 (계속)

```
    <table cellspacing="5" cellpadding="5" border="1">
      <tr>
        <td align="right">First name:</td>
        <td><%= firstName %></td>
      </tr>
      <tr>
        <td align="right">Last name:</td>
        <td><%= lastName %></td>
      </tr>
      <tr>
        <td align="right">Email address:</td>
        <td><%= emailAddress %></td>
      </tr>
    </table>
  <form action="/stc/email6/join_email_list.html‟ method="post">
    <input type="submit" value="Return">
  </form>
  </body>
  </html>
```

2.서블릿 & JSP 연동

1. Model 1 아키텍처

❑ Model 1 아키텍처 구성



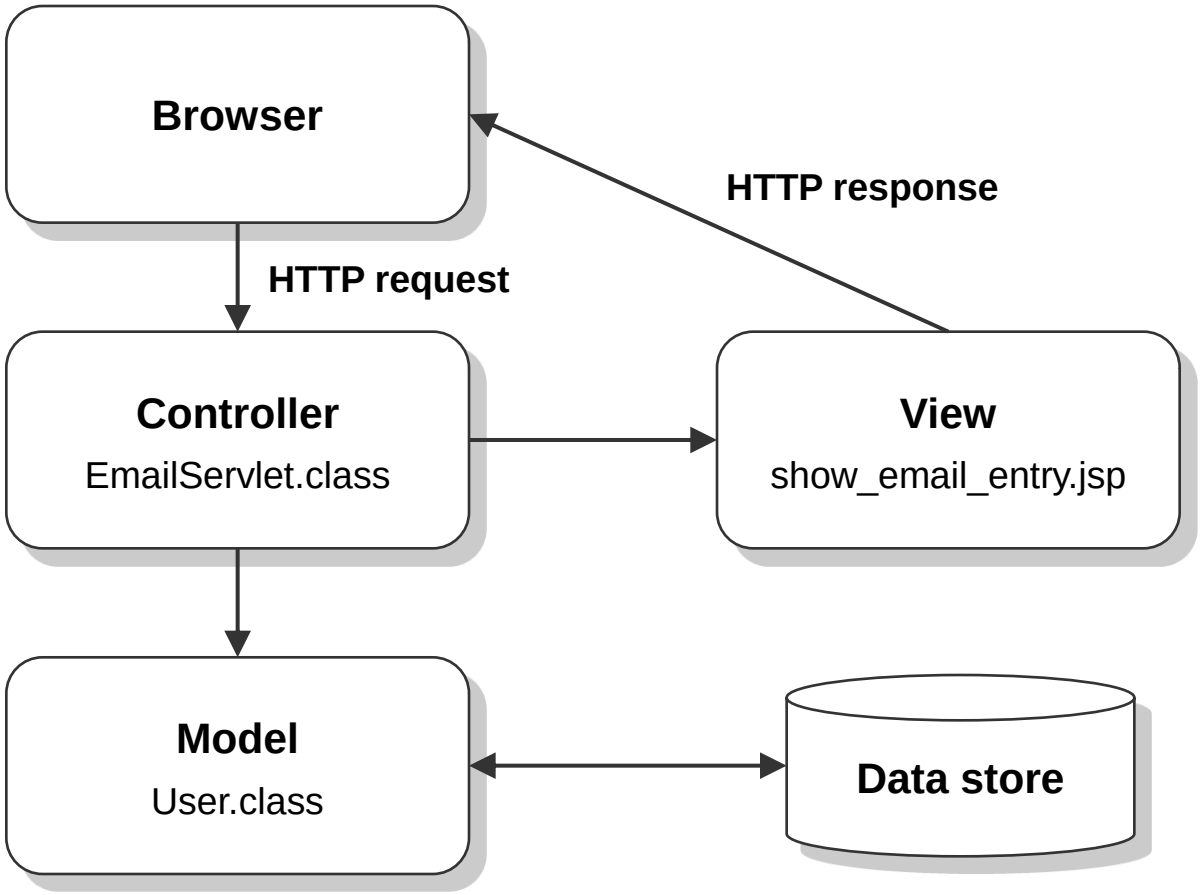
1. Model 1 아키텍처

❑ Model 1 아키텍처

- 제한된 요구사항의 웹 애플리케이션을 개발할 때 Model 1 아키텍처를 주로 사용한다.
- Model 1 아키텍처에서의 비즈니스 클래스는 *자바빈즈(JavaBeans)*로 작성된다.
  비즈니스 클래스는 애플리케이션의 비즈니스 프로세스와 데이터를 표현한다.
- 데이터베이스나 파일 시스템을 *데이터 저장소(data store)*로 활용한다. 애플리케이션이 종료된 후에도 데이터는 유지되어야 하기 때문에 *영구적인 데이터 저장소(persistent data storage)*라고도 불린다.
- UserIO 클래스와 같은 데이터 클래스의 메소드는 데이터 저장소의 데이터를 조회하거나 저장하는데 사용된다.

2. Model 2 아키텍처

❑ Model 2 아키텍처 구성 (Model–View–Control 패턴)



2. Model 2 아키텍처

❑ Model 2 아키텍처

- 주로 복잡한 요구사항을 처리하는 웹 애플리케이션에서 *MVC(Model-View-Controller pattern*) 패턴을 사용한다. MVC 패턴은 코딩과 유지보수를 쉽게 하고, Model 2 아키텍처라고도 불린다.
- MVC 패턴에서의 모델(model)은 자바빈즈로 구성되고, 뷰(view)는 HTML과 JSP, 그리고 컨트롤러(controller)는 서블릿으로 구성된다.
- UserIO 클래스와 같은 데이터 클래스의 메소드는 데이터 저장소의 데이터를 조회하거나 저장하는데 사용된다.
- MVC 패턴을 사용할 때는 가능한 한 각 레이어를 독립적으로 구성해야 한다. 그래야만 하나의 레이어를 수정하더라도 다른 레이어에 미치는 영향을 최소화 할 수 있다.

3. 요청 디스패치 vs 리다이렉트

❑ 요청 디스패치

– 문법

```
RequestDispatcher objectName =
    getServletContext().getRequestDispatcher("/url");
objectName.forward(request, response);
```

– 사용 예

```
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(
        "/email6/get_missing_fields.jsp");
dispatcher.forward(request, response);

RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher(
        "/email6/join_email_list.html");
dispatcher.forward(request, response);
```

3. 요청 디스패치 vs 리다이렉트

❑ 요청 리다이렉트

- 문법

```
response.sendRedirect("url");
```

- 사용 예

```
response.sendRedirect("http://www.naver.com");
response.sendRedirect("/stc/email5/show_email_entry.jsp");
```

3. 요청 디스패치 vs 리다이렉트

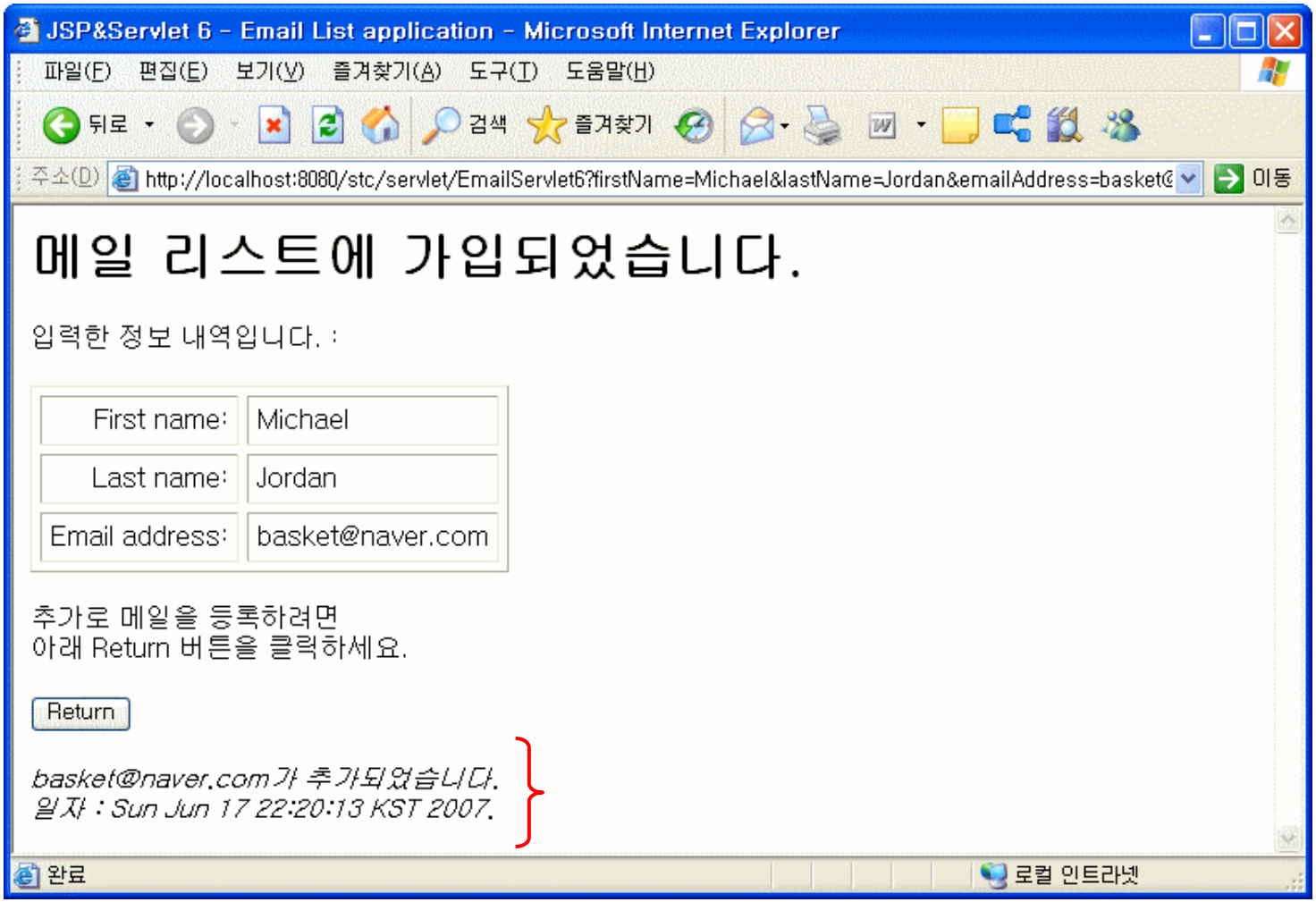
❑ 요청 디스패치 vs 리다이렉트

- RequestDispatcher 객체의 forward 메소드는 동일 서버의 다른 자원(JSP, 서블릿 등)에게 제어를 넘긴다. 동일한 서버에서 요청이 처리되고, 지정된 자원에서 request 객체와 response 객체에 접근한다.
- ServletContext 객체의 getReqestDispatcher 메소드를 사용해서 RequestDispatcher 객체를 얻는다.
- ServletContext 객체를 얻기 위해서는 HttpServlet 클래스의 getReqestDispatcher 메소드를 사용한다.
- 다른 서버의 자원에 제어를 넘길 때는 response 객체의 sendRedirect 메소드를 사용한다.
- sendRedirect 메소드를 사용하면 지정된 자원에서 request 객체와 response 객체에 접근할 수 없다.

3.JSP에서 파일 포함

1. JSP에서 파일 포함

❑ JSP 페이지 (파일을 포함한 결과)



1. JSP에서 파일 포함

❑ header.htm 소스

```
<head>
  <title>JSP&Servlet 6 - Email List application</title>
</head>
```

❑ footer.jsp 소스

```
<%@ page import="java.util.Date" %>
<p><i><%= emailAddress %>가 추가되었습니다. <br>
일자 : <%= new Date() %>.</p>
```

❑ show_email_entry.jsp 소스

```
<html>
<jsp:include page="/includes/header.htm" flush="true" />
<body>
    // missing code that displays the body of the page
    <%@ include file="/includes/footer.jsp" %>
</body>
</html>
```

2. 파일 포함하는 방법

❑ 변환 시점에 파일 포함 (지시자)

- 문법

```
<%@ include file="fileLocationAndName" %>
```

- 사용 예

```
<%@ include file="/includes/header.htm" %>
<%@ include file="/includes/footer.jsp" %>
```

❑ 요청 시점에 파일 포함 (표준 액션)

- 문법

```
<jsp:include page="fileLocationAndName" flush="true" />
```

- 사용 예

```
<jsp:include page="/includes/header.htm" flush="true" />
<jsp:include page="/includes/footer2.jsp" flush="true" />
```

2. 파일 포함하는 방법

❑ JSP에서 파일을 포함하는 방법

- 컴파일 또는 변환 시점에 JSP에서 파일을 포함하려면 include 지시자(directive)를 사용한다.
- include 지시자를 사용하면 포함되는 파일에 있는 코드는 변환되는 서블릿의 일부분이 된다. 결과적으로, 포함되는 파일이 수정되더라도 JSP 페이지가 다시 컴파일 되기 전까지는 반영이 안 된다.
- 실행 또는 요청 시점에 JSP에서 파일을 포함하려면 include 액션(action)을 사용한다.
- include 액션을 사용하면 포함되는 코드는 변환되는 서블릿의 일부분이 되지 않는다. 따라서 포함되는 파일이 수정되면 JSP가 요청되는 시점에 수정된 내용이 출력된다.
- 변환 시점에 파일을 포함하면 포함되는 파일은 JSP에 정의되어있는 변수와 메소드에 접근할 수 있다. 요청 시점에 파일을 포함하면 포함되는 파일은 이러한 변수들에 접근할 수 없다.

4.web.xml

1. web.xml 개요

❑ web.xml 파일

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.// DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <context-param>
    <param-name>dbName</param-name>
    <param-value>stcdb</param-value>
  </context-param>
  <servlet>
    <servlet-name>Email6 list</servlet-name>
    <servlet-class>email6.EmailServlet</servlet-class>
    <init-param>
      <param-name>filename</param-name>
      <param-value>
        /UserEmail.txt
      </param-value>
    </init-param>
  </servlet>
```

1. web.xml 개요

❑ web.xml 파일 (계속)

```
    <servlet-mapping>
      <servlet-name>Email6 list</servlet-name>
      <url-pattern>/servlet/EmailServlet6</url-pattern>
    </servlet-mapping>

    <!-- 에러를 별도로 처리하려면 주석을 지운다. -->
    <!--
    <error-page>
      <exception-type>java.lang.Throwable</exception-type>
      <location>/email6/error.htm</location>
    </error-page>

    <error-page>
      <error-code>404</error-code>
      <location>/email6/show_error_page.jsp</location>
    </error-page>
    -->
  </web-app>
```

1. web.xml 개요

❑ web.xml 개요

- web.xml 파일은 애플리케이션의 WEB-INF 디렉터리에 위치하고, 서블릿 엔진이 구동하면서 web.xml 을 읽어 들인다.
- XML 태그를 사용하여 XML 요소(element)를 정의한다. XML 요소(element)는 여러 개의 자식 요소(child element)을 가질 수 있다.
- web.xml 파일에서 특정 부분을 주석으로 정의하려면 XML 주석 태그를 사용하면 된다. XML 주석 태그는 HTML 주석 태그와 동일한 형식이다. <!-- 주석 문장 -->
- web.xml에서 요소들의 순서가 틀리면 web.xml 파일을 읽어 들일 때 톰캣이 에러 메시지를 표시한다.
- web.xml 파일을 수정하고 나면 톰캣을 재구동 해야만 변경된 내용이 반영된다.

2. 초기화 파라미터 설정

- 초기화 파라미터를 설정하는 XML 코드

```
<web-app>
  <context-param>
    <param-name>dbName</param-name>
    <param-value>stcdb</param-value>
  </context-param>

  <servlet>
    <servlet-name>Email6 list</servlet-name>
    <servlet-class>email6.EmailServlet</servlet-class>
    <init-param>
      <param-name>filename</param-name>
      <param-value>
         /UserEmail.txt
      </param-value>
    </init-param>
  </servlet>

</web-app>
```

2. 초기화 파라미터 설정

❑ 초기화 파라미터를 를 위한 XML 요소

요소(element)	설 명
<web-app>	web.xml의 루트 요소(root element)이다. 모든 다른 요소들은 루트 요소에 포함되어야 한다.
<context-param>	모든 서블릿에서 사용 가능한 파라미터를 정의한다.
<servlet>	애플리케이션의 특정 서블릿을 식별한다.
<servlet-name>	web.xml 파일에서 사용되는 서블릿의 이름을 정의한다.
<servlet-class>	서블릿 클래스의 실제 패키지와 클래스 명을 나타낸다.
<init-param>	서블릿 초기화 파라미터의 이름/값 쌍을 정의 한다.
<param-name>	파라미터의 이름을 나타낸다.
<param-value>	파라미터의 값을 나타낸다.

2. 초기화 파라미터 설정

❑ 초기화 파라미터 설정

- 모든 서블릿에서 사용 가능한 초기화 파라미터(컨텍스트 초기화 파라미터, context initialization parameter)를 만들기 위해서는 <context-param> 요소 내에 <param-name>,<param-value> 요소를 작성해야 한다.
- 특정 서블릿에서 사용 가능한 초기화 파라미터(서블릿 초기화 파라미터, servlet initialization parameter)를 만들기 위해서는 <init-param> 요소 내에 <param-name>,<param-value> 요소를 작성해야 한다.
- <servlet>, <servlet-name>, <servlet-class> 요소를 통해서 서블릿을 정의한 후에 서블릿 초기화 파라미터를 작성해야 한다.

3. 초기화 파라미터 사용

❑ 서블릿 초기화 파라미터

특정 서블릿에 대한
초기화 파라미터 값을 읽는다.

```
public class EmailServlet extends HttpServlet{
    private String file;
    public void init() throws ServletException{
        ServletConfig config = getServletConfig();
        file = config.getInitParameter("filename");
    }
```

❑ 컨텍스트 초기화 파라미터

모든 서블릿에서 접근 가능한
초기화 파라미터 값을 읽는다.

```
public class EmailServlet extends HttpServlet{
    private String dbName;
    public void init() throws ServletException{
        ServletConfig config = getServletConfig();
        ServletContext context = config.getServletContext();
        dbName = context.getInitParameter("dbName");
    }
```

4. 서블릿 매핑

- 서블릿 매핑을 추가하는 XML 태그

```
<servlet>
  <servlet-name>Email6 list</servlet-name>
  <servlet-class>email6.EmailServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>Email6 list</servlet-name>
  <url-pattern>/servlet/EmailServlet6</url-pattern>
</servlet-mapping>
```

배포자가 만든 내부적인 이름

실제 서블릿 파일명

클라이언트가 아는 URL 이름

- 서블릿을 요청하는 Form 태그

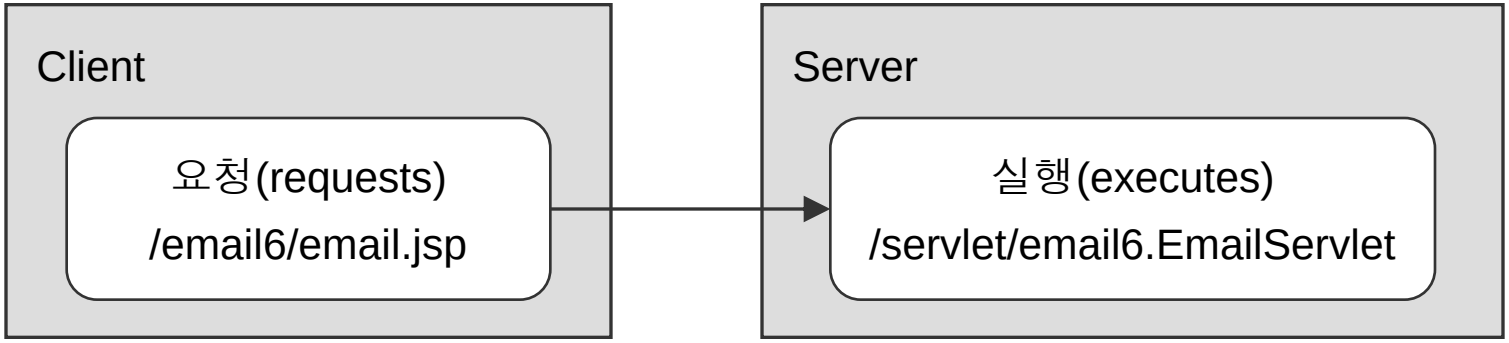
```
<form action="/stc/servlet/EmailServlet6" method="post">
```

- 주소창에 보여지는 URL

```
http://localhost:8080/murach/email6/email.jsp
```

4. 서블릿 매핑

❑ 서블릿 매핑 동작



❑ 서블릿 매핑에 관련된 XML 요소

요소(element)	설 명
<servlet-mapping>	URL과 서블릿을 매핑을 가능하도록 한다.
<servlet-name>	서블릿의 이름을 나타낸다. <servlet> 요소에 정의한 서블릿 이름(<servlet-name>)과 동일한 이름을 적어준다.
<url-pattern>	<servlet-mapping>에 표현된 서블릿으로 요청을 보내기 위한 URL 명을 나타낸다. 충돌을 피하기기 위해서는 실제 존재하지 않는 디렉터리나 파일명으로 지정한다.

5. 커스텀 에러 처리

- 특정 **Exception** 타입에 대한 에러 처리

```
<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/email6/error.htm</location>
</error-page>
```

- 특정 에러 코드에 대한 에러 처리

```
<error-page>
  <error-code>404</error-code>
  <location>/email6/error_404.jsp</location>
</error-page>
```

5. 커스텀 에러 처리

❑ 에러 처리를 위한 XML 요소

요소(element)	설 명
<error-page>	애플리케이션에 에러 또는 특정 HTTP 상태 코드가 발생했을 때 표시하고자 하는 HTML 페이지나 JSP 페이지를 나타낸다.
<exception-type>	패키지 명을 포함한 Java 익셉션 클래스 명을 나타낸다.
<error-code>	HTTP 상태 코드 번호를 나타낸다.
<location>	표시될 HTML 페이지나 JSP페이지의 위치를 나타낸다.

❑ 커스텀 에러 처리

- 웹 애플리케이션에서 (1) 에러(uncaught exception), (2) 특정 HTTP 상태 코드가 발생했을 때 <error-page> 요소를 사용해서 특정 페이지가 출력되도록 할 수 있다.
- <error-page> 요소는 <servlet>과 <servlet-mapping> 요소 다음에 위치해야 한다.

Lab #1

❑ include와 web.xml 활용하기

1. 두 개의 파일을 포함하도록 **Email List** 애플리케이션의 **show_email_entry.jsp** 파일을 수정한다.
   - 현재의 시간 표시하는 **JSP** 페이지를 만들고, **show_email_entry.jsp**의 하단에 포함한다.
   - **<head>……</head>** 에 해당하는 부분을 **JSP** 페이지로 만들고 포함한다.
   - 먼저 변환 시점에 포함되도록 한 후에 확인하고, 요청 시점에 포함되도록 수정한다.
2. **UserIO** 클래스에 전달되는 텍스트 파일의 위치와 이름을 서블릿 초기화 파라미터를 이용한다. 초기화 파라미터 값을 읽지 못하는 경우에는 **null** 값으로 입력되기 때문에 서블릿 코드 안에서 적절히 처리하도록 한다.
3. **EmailServlet** 클래스에 대한 **URL**이 **servlet/email.jsp** 가 되도록 서블릿 매핑을 구현하라. 즉, 브라우저의 주소창에 **http://localhost:8080/stc/servlet/email.jsp** 라고 입력했을 때 **EmailServlet** 클래스가 호출되도록 수정한다.