

## 4장. 액션 태그



*ACTION TAG*



# 액션 태그

## 액션 태그

- JSP 에서 객체 생성과 공유, 페이지 이동과 전달등의 기능을 제공하는 태그.
- `<% ... %>`와 같은 스크립트 태그 형식이 아닌 **XML 형식** `<jsp: .../>`을 사용함

액션 태그	형식	설 명
forward	<code>&lt;jsp:forward ... /&gt;</code>	다른 페이지로의 이동과 같은 페이지 흐름 제어
include	<code>&lt;jsp:include ... /&gt;</code>	외부 페이지의 내용을 포함하거나 페이지를 모듈화
useBean	<code>&lt;jsp:useBean ... /&gt;</code>	JSP 페이지에 자바빈즈를 설정함
setProperty	<code>&lt;jsp:setProperty ... /&gt;</code>	자바빈즈의 프로퍼티값을 설정함
getProperty	<code>&lt;jsp:getProperty ... /&gt;</code>	자바빈즈의 프로퍼티값을 얻어옴



# 액션 태그

## forward 액션 태그

- 현재 JSP page에서 다른 page로 이동하는 태그

```
<jsp:forward page="파일명" />
```

```
<head>
  <title>Action Tag</title>
</head>
<body>
  <h2>forward 액션 태그</h2>

  <jsp:forward page="forward_date.jsp" />

  <p>-----
</body>
</html>
```

action/forward.jsp



# 액션 태그

## forward 액션 태그

```
<title>날짜/시간</title>
</head>
<body>
  <h2>오늘의 날짜 및 시간</h2>
  <p><%=LocalDateTime.now() %></p>
</body>
</html>
```

action/forward\_date.jsp

url은 현재 페이지이지만,  
컨텐츠는 이동페이지임

localhost:8080/jweb01/action/forward.jsp

## 오늘의 날짜 및 시간

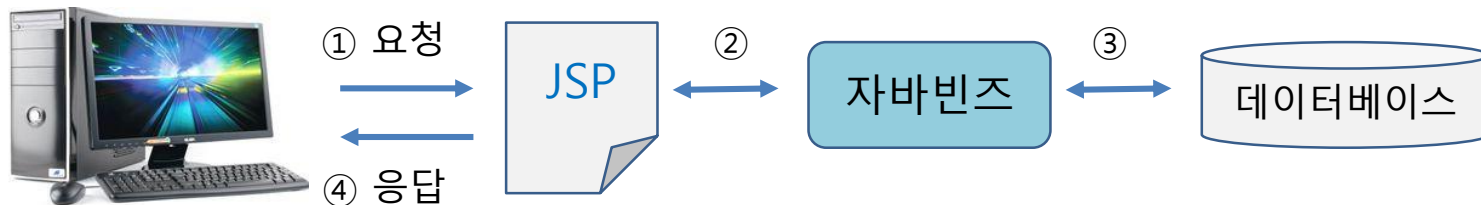
2025-10-03T07:00:16.979004100



# 자바빈즈 액션 태그

## 자바 빈즈(beans)

- 자바 코드를 사용하여 **자바 클래스**로 로직을 작성하는 방법이다.
- 자바코드와 HTML 코드를 함께 작성하면 기능을 확장하거나 코드를 재사용하는데 어려움이 있다.
- JSP 페이지에서 화면을 표현하기 위한 **계산식이나 자료의 처리를 담당하는 자바 코드**를 따로 분리하여 작성한다.



- ① 웹 브라우저가 JSP 페이지에 요청한다.
- ② JSP 페이지는 자바 빈즈와 통신한다.
- ③ 자바 빈즈가 데이터베이스에 연결된다.
- ④ JSP 페이지가 브라우저에 응답한다.

# 자바빈즈 액션 태그

## useBean 액션 태그로 자바빈즈 사용하기

- 자바빈즈를 사용하기 위해 실제 자바 클래스를 선언하고 초기화하는 태그.
- 설정된 id, scope 속성으로 자바빈즈의 **객체**를 검색하고, 객체가 발견되지 않으면 빈 객체를 생성한다.

```
<jsp:useBean id="식별 이름" class="자바빈즈 이름" scope="범위"/>
```

```
<!-- member는 Member 클래스의 인스턴스(객체) -->  
<jsp:useBean id="member" class="bean.Member" scope="request" />
```



# 자바빈즈 액션 태그

## useBean 액션 태그로 자바빈즈 사용하기

### ➤ useBean 액션 태그의 속성

속성	설 명
id	자바빈즈를 식별하기 위한 이름이다.
class	패키지 이름을 포함한 자바빈즈의 이름이다.
scope	page(기본값), <b>request</b> , <b>session</b> , application 중 하나의 값을 사용

scope는 생명주기 범위를 의미하며, 만약 session으로 설정하면 session의 유효 범위 동안 해당 클래스 인스턴스의 상태가 유지된다.

# 자바빈즈 액션 태그

## ✓ Member 클래스 생성

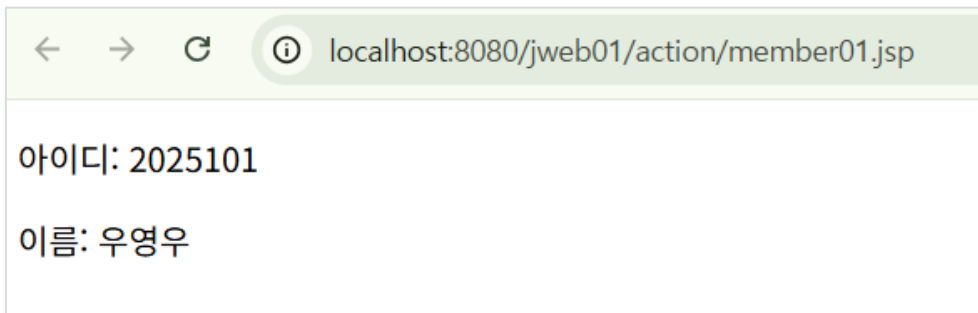
```
public class Member {  
  
    private int id = 2025101;  
    private String name = "우영우";  
  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```





# 자바빈즈 액션 태그

- ✓ useBean 액션 태그로 아이디와 이름 출력



action/member01.jsp

```
<!-- member는 Member 클래스의 인스턴스(객체) -->  
<jsp:useBean id="member" class="bean.Member" scope="request" />  
  
<p>아이디: <%=member.getId() %></p>  
<p>이름: <%=member.getName() %>
```

request 객체는 저장  
기능이 있다.



# 자바빈즈 액션 태그

- ✓ `setProperty` 액션 태그로 **아이디와 이름 설정**하여 출력

← → ↻ ⓘ localhost:8080/jwbook/ch03/member/member3.jsp

아이디: 2022102

이름: 안영이

action/member02.jsp

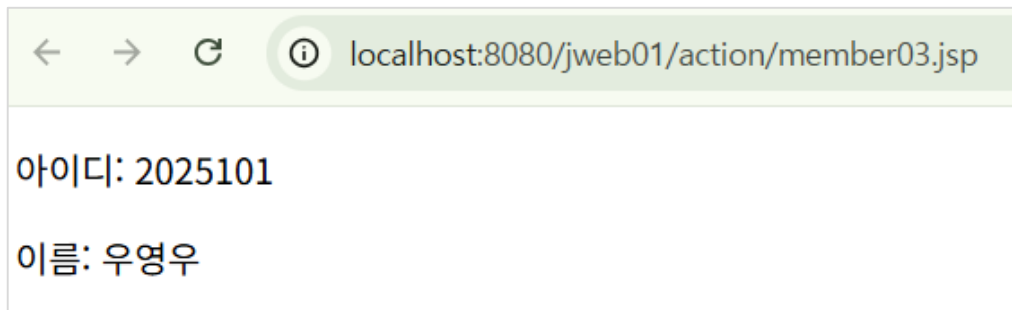
```
<jsp:useBean id="member" class="bean.MemberBean" scope="request"/>
<jsp:setProperty property="id" name="member" value="2021102"/>
<jsp:setProperty property="name" name="member" value="안영이"/>

<p>아이디 : <jsp:getProperty property="id" name="member"/>
<p>이름 : <jsp:getProperty property="name" name="member"/>
```



# 자바빈즈 액션 태그

✓ 디렉티브 page로 import 하여 아이디와 이름 설정하여 출력



```
<%@ page import="bean.Member"%>
```

```
<%  
    //Member 클래스 - import  
    Member member = new Member();  
%>
```

action/member03.jsp

```
<p>아이디: <%=member.getId() %></p>  
<p>이름: <%=member.getName() %>
```



# 사용자 로그인 구현

## ● 사용자 로그인 구현

### 프로그램 소스 목록

파일 이름	역 할
loginForm.html	사용자 로그인을 위해 아이디와 비밀번호 입력받는 화면
loginProcess.jsp	입력받은 아이디 정보를 빈즈 클래스를 이용 확인 처리한 화면
LoginBean.java	사용자가 입력한 계정 정보를 매핑하는 빈즈 클래스로써, 미리 저장된 계정 값과 비교해 로그인 성공 여부를 반환하는 메서드를 포함

# 사용자 로그인 구현

- 사용자 로그인 구현

**로그인**

아이디 :

비밀번호 :



로그인 성공!!

---

사용자 아이디: myuser

비밀번호: 0000

# 사용자 로그인 구현

- 사용자 로그인 구현

loginBean.java

```
package bean;

public class LoginBean {
    private String userid;
    private String passwd;

    final String _userid = "myuser";
    final String _passwd = "0000";

    //로그인 체크
    public boolean checkUser() {
        if(userid.equals(_userid) && passwd.equals(_passwd)) {
            return true;
        }else {
            return false;
        }
    }
}
```

```
public String getUserid() {
    return userid;
}

public void setUserid(String userid) {
    this.userid = userid;
}

public String getPasswd() {
    return passwd;
}

public void setPasswd(String passwd) {
    this.passwd = passwd;
}
```



# 사용자 로그인 구현

## ● 사용자 로그인 구현

```
<h2>로그인</h2>
<form action="LoginProcess.jsp" method="post">
  <ul>
    <li>
      <label for="uid">아이디 : </label>
      <input type="text" id="userid" name="userid">
    </li>
    <li>
      <label for="passwd">비밀번호 : </label>
      <input type="password" id="passwd" name="passwd">
    </li>
    <li>
      <!-- <input type="submit" value="로그인"> -->
      <button type="submit">로그인</button>
    </li>
  </ul>
</form>
```

loginForm.jsp

# JSP 빈즈 프로그래밍

## ● 사용자 로그인 구현

loginProcess.jsp

```
<style>
    body{margin: 30px;}
</style>
<jsp:useBean id="login" class="bean.LoginBean" scope="request" />
<jsp:setProperty property="userid" name="login"/>
<jsp:setProperty property="passwd" name="login"/>
<body>
    <%
        if(login.checkUser()){
            out.println("로그인 성공!!");
        }else{
            out.println("로그인 실패!!");
        }
    %>
    <hr>
    <p>사용자 아이디: <jsp:getProperty property="userid" name="login"/>
    <p>비밀번호: <jsp:getProperty property="passwd" name="login"/>
</body>
```





# 서블릿 포워드 기능

## 포워드(forward) 기능

- 하나의 서블릿에서 다른 서블릿이나 JSP와 연동하는 방법

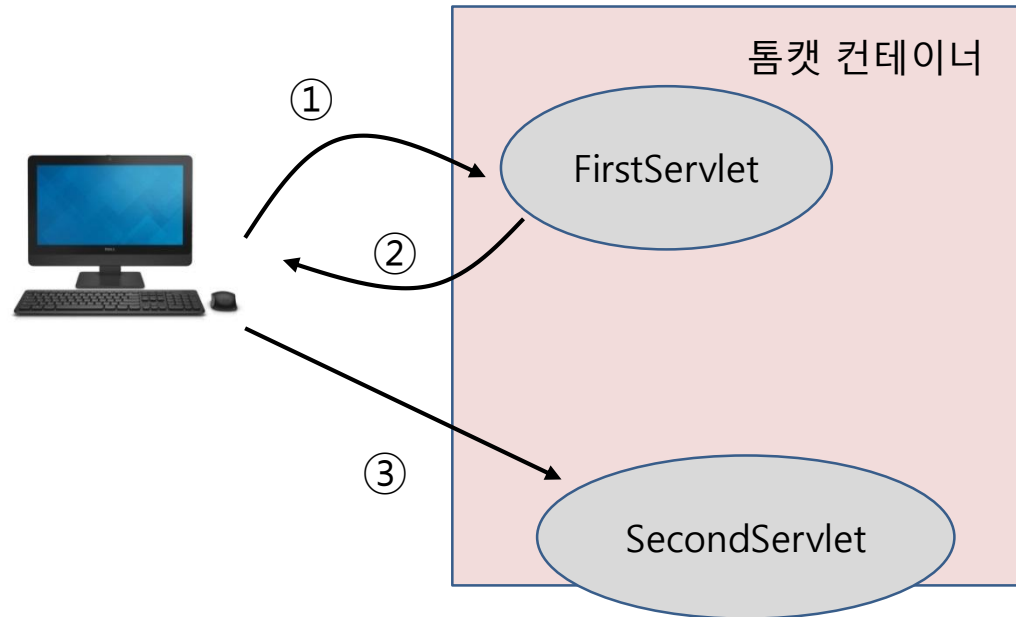
## 포워드 방법

- **redirect 방법** – HttpServletResponse 객체의 sendRedirect() 메서드 이용  
**리디렉션**: 서버가 클라이언트에게 새로운 페이지로 다시 접속하도록 응답을 보내고, 응답을 받은 클라이언트가 다시 새로운 페이지로 접속하는 방식
- **dspatch 방법** – RequestDispatcher 클래스의 forward() 메서드 이용  
**forward 액션**: 클라이언트가 새롭게 접속하는 것이 아니라 서버에서 내부적으로 새로운 페이지로 이동하고, 그 페이지의 내용을 클라이언트에게 응답으로 전달하는 방식



# 서블릿 포워드 기능

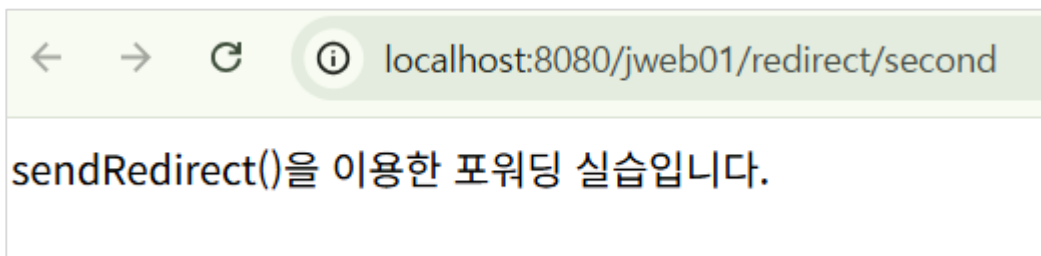
## ➤ redirect를 이용한 포워딩 과정



- ① 클라이언트의 웹 브라우저에서 첫번째 서블릿에 요청합니다.
- ② 첫번째 서블릿은 `sendRedirect()` 메서드를 이용해 웹 브라우저에게 다시 요청하게 함.
- ③ 웹 브라우저는 두 번째 서블릿으로 다시 요청 (**URL은 두번째 서블릿 경로로 이동**)

# 서블릿 포워드 기능

## ➤ redirect 방법 실습



forward.redirect/FirstServlet.java

```
@WebServlet("/redirect/first")
public class FirstServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse
        response.setContentType("text/html, charset=utf-8");

        response.sendRedirect("second"); //second 경로로 이동
    }
}
```



# 서블릿 포워드 기능

## ➤ redirect 방법 실습

forward.redirect/SecondServlet.java

```
@WebServlet("/redirect/second")
public class SecondServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse
        response.setContentType("text/html; charset=utf-8");

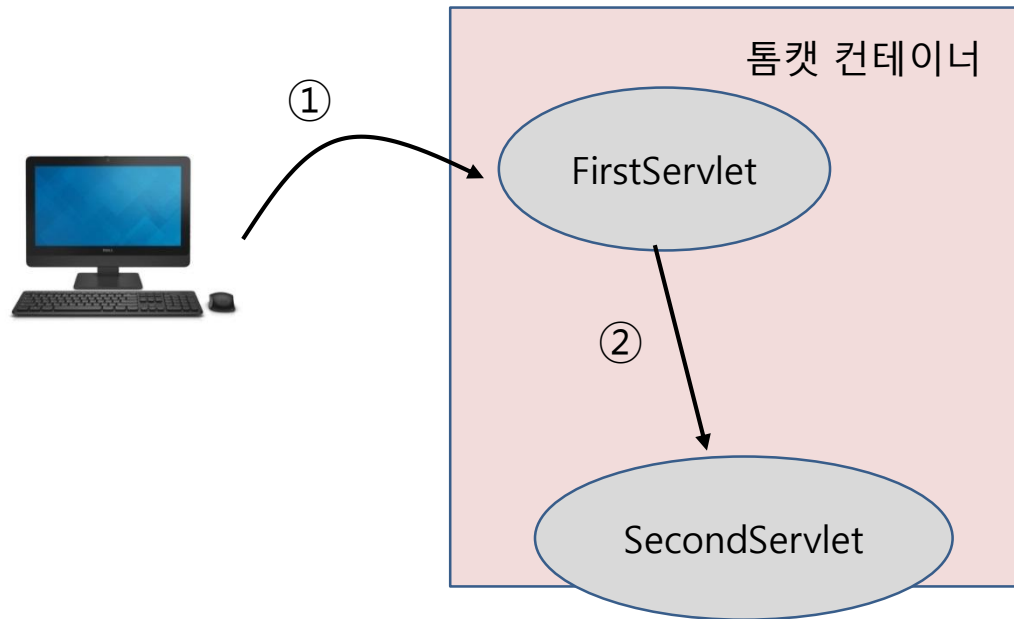
        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        out.println("sendRedirect()을 이용한 포워딩 실습입니다.");
        out.println("</body><html>");
    }
```



# 서블릿 포워드 기능

## ➤ dispatch를 이용한 포워딩 과정




- ① 클라이언트의 웹 브라우저에서 첫 번째 서블릿에 요청합니다.
- ② 첫 번째 서블릿은 RequestDispatcher를 이용해 두 번째 서블릿으로 포워드합니다.

**(URL 경로는 첫번째 서블릿임)**

# 서블릿 포워드 기능

## ✓ Java EE API-> 서블릿 -> RequestDispatcher 클래스



The screenshot shows the Java EE API documentation for the `RequestDispatcher` interface. On the left, a navigation pane lists various packages and classes, with `RequestDispatcher` highlighted. The main content area displays the `Interface RequestDispatcher` under the `javax.servlet` package. It includes the signature `public interface RequestDispatcher` and a description: "Defines an object that receives requests from the client and sends them to the server. The servlet container creates the `RequestDispatcher` object located at a particular path or given by a particular name." Below this, it states: "This interface is intended to wrap servlets, but a servlet container can also use it to wrap other resources."

클라이언트에서 요청을 수신하여 서버의 리소스(예: 서블릿, HTML 파일 또는 JSP 파일)로 보내는 객체를 정의합니다.

서블릿 컨테이너는 `RequestDispatcher` 객체를 생성하며, 이 객체는 특정 경로에 있거나 특정 이름에 의해 지정된 서버 리소스를 감싸는 래퍼로 사용됩니다.



# 서블릿 포워드 기능

## ➤ dispatch방법 실습

← → ↻ ⓘ localhost:8080/jweb01/dispatch/first

RequestDispatcher를 이용한 포워딩 실습입니다. 이름: 김기용

forward.dispatch/FirstServlet2.java

```
@WebServlet("/dispatch/first")
public class FirstServlet2 extends HttpServlet { //클래스 이름이 중복되지 않게 함
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        response.setContentType("text/html; charset=utf-8");

        //경로는 first이고 데이터(name)만 파라미터로 전달함
        RequestDispatcher rd =
            request.getRequestDispatcher("second?name=김기용");
        rd.forward(request, response); //URL은 first를 유지함
    }
```



# 서블릿 포워드 기능

## ➤ dispatch방법 실습

forward.dispatch/SecondServlet2.java

```
@WebServlet("/dispatch/second")
public class SecondServlet2 extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        //브라우저로 응답하기
        response.setContentType("text/html; charset=utf-8");

        //쿼리스트링 데이터 받기
        String name = request.getParameter("name");

        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("RequestDispatcher를 이용한 포워딩 실습입니다.");
        out.println("이름: " + name + "<br>");
        out.println("</body></html>");
    }
```

