**JSP & Servlet 정리**

**1. 개요**

*웹 프로그래밍이란?*

* 웹을 기반으로 실행되는 애플리케이션을 제작하는 과정
* 정적인 정보만 제공하는 HTML의 한계를 넘기 위함
* 다양한 정보를 데이터베이스에서 얻거나 저장
* 페이지를 요청하면 웹 서버는 웹 애플리케이션 서버(WAS)로 전송
* 웹 애플리케이션 서버는 로직이나 데이터베이스와의 연동을 통해 데이터를 처리
* 처리 결과를 웹 서버로 보내 사용자의 브라우저에 응답

**2. 서블릿**

*서블릿?*

* PHP, ASP와 같은 언어에 비해 응답 속도가 빠름
* Javax.servlet.http 패키지에서 제공하
* 는 HttpServlet클래스를 상속받아 구현
* JVM에서 동작하기 때문에 클래스 파일이 생성되어야 함

**기본형태**

public class Sample extends HttpServlet{

private stataic final long serialVersionUID = 1L;

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

//get방식으로 받은 데이터를 처리

}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{

//post방식으로 받은 데이터를 처리

}

}

|  |
| --- |
| ***? get & post***  get 방식 : 주소 창을 통해 서버로 데이터를 전송. 보안에 취약. 255자 이하의 데이터  ex) http://www.samsung.com/servletSample**?data1=test&data2=123**  post 방식 : html header를 타고 데이터를 전송. 보안에 강함. 255자 이상의 데이터 |

* 보통 <form>태그를 통해 전달 요청

ex) <form action=”CallServlet”>

<input type=”text” name=”전송할 텍스트”>

<input type=”submit” value=”전송>

</form>

* 서블릿을 사용하기 위해 서블릿 클래스 내에

@WebServlet(“/서블릿이름”)

처럼 선언하거나 web.xml 파일내에

<servlet>

<servlet-name>서블릿이름</servlet-name>

<servlet-class>서블릿주소</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>서블릿이름</servlet-name>

<url-pattern>/서블릿이름</url-pattern>

</servlet-mapping>

와 같이 입력해 주어야 함

* 서블릿은 두 개의 매개 변수(HttpServletRequest, HttpservletResponse)를 통해 데이터를 처리함.
* HttpServletRequest는 클라이언트의 요청(request)을 처리
* HttpServletResponse는 요청 처리 결과를 클라이언트에게 응답하기 위해 사용
* 클라이언트에게 응답할 페이지에 대한 환경 설정을 해주어야 한글이 깨지지 않음.

response.setContentType(“text/html;charset=UTF-8”);

* 서블릿에서 HTML 문서로 바로 출력할 수 있으며 response.getWriter()메소드로 출력 스트림 객체를 얻어 와야함.

PrintWriter out = response.getWirter();

out.print(“<HTML>”);

*서블릿의 동작원리?*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 웹브라우저 | 요청→ | 웹서버 | → | 서블릿  컨테이너 | → | 스레드 | → | 서블릿  객체 |
| ←응답 | ← | ← | ← |

1. 브라우저에서 서블릿을 요청
2. WAS안의 웹 서버가 서블릿 요청을 인식하여 서블릿 컨테이너에게 서블릿을 수행하도록 넘겨줌
3. 서블릿은 스레드를 기동하여 해당 서블릿 객체를 생성하여 이를 수행
4. 서블릿 객체의 작업이 종료되면 기동되었던 스레드 종료
5. 서블릿 수행 결과가 웹 서버에 전송
6. 클라이언트에게 전송

*서블릿의 라이프 사이클?*

* 서블릿은 처음 요청시 서블릿 클래스를 찾아 메모리에 로딩하여 인스턴스를 생성하고 이후 메모리에 계속 남아있어 새로 생성하지 않아도 재사용 가능하여 속도가 빠름

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instance생성 |  | init() |  | doGet()  doPost() |  | destroy() |
| → | → | → |
| *서블릿 객체 생성* |  | *최초로 한번만 호출* |  | *요청될 때마다 호출* |  | *톰캣 해제시 자원해제* |

* <form>태그에서 method를 통해 전달 방식 결정

<form action=”CallServlet” method=”**get/post**”>

<input type=”text” name=”전송할 텍스트”>

<input type=”submit” value=”전송>

</form>

* 전달 받은 데이터는 서블릿에서 getParameter메소드를 통해 사용

int data = request.getParameter(“data”);

* 전달 받는 데이터의 경우 한글 깨짐을 방지하기 위해 get과 post 각각 사용하는 방법이 다름
* post의 경우

request.setCharacterEncoding(“UTF-8”);

만을 추가하여 사용

* get의 경우

서버 설정 파일인 server.xml의 <Connector>태그를 수정하여 사용

<Connector URIEncoding=”UTF-8” …..>

*<form>태그?*

<input type=”text”> 텍스트 입력 상자

<input type=”password”> 암호 입력 상자

<textarea rows=10 cols=10> 여러 줄 입력할 수 있는 글 상자

<input type=”radio”> 배타적 선택을 하는 라디오 버튼

<input type=”checkbox”> 복수 선택이 가능한 체크 박스

<select> 목록 상자

<option value=”a”>a</option>

<option value=”b”>b</option>

</select>

\*\*HTML5 지원 태그\*\*

<input type=”number” min=0 max=50 step=1> 숫자 입력 상자

<input type=”email”>이메일 무결성 자동 체크 입력 상자

**3. JSP**

*JSP?*

* JSP는 HTML태그에 자바로 프로그래밍하여 브라우저에 보여주는 페이지
* <% %>내에 자바 코드를 추가하여 사용
* JSP파일은 톰캣 서버가 번역하여 그 결과를 HTML 태그로 변환한 후 웹브라우저에 내려 보내짐.
* 서블릿이 자바 코드 내에 HTML태그를 작성한다면 JSP는 HTML문서 내에 자바 코드를 입력하기에 디자이너가 이해하기에 용이.

*JSP 동작 순서?*

1. 브라우저에서 JSP페이지를 웹 서버에 요청
2. 웹 서버는 JSP에 대한 요청을 서블릿 컨테이너에게 넘김.
3. 서블릿 컨테이너는 해당 JSP를 찾아 서블 파일(.java)을 생성
4. 서블릿 파일이 컴파일 되어 클래스 파일을 생성
5. 컴파일된 서블릿 클래스 파일을 웹 서버로 전송
6. 사용자는 서블릿의 출력 결과인 HTML 형태로 응답 결과를 확인.

*JSP 기본태그?*

<% %> 자바 코드를 기술

<%! %> 변수와 베소드를 선언

<%= %> 계산식이나 함수를 호출한 결과를 문자열 형태로 출력

<%-- --%> 주석

<%@ %> JSP페이지의 속성을 지정

*JSP 내장객체?*

* JSP 내장 객체는 프로그래머가 객체를 생성하는 과정 없이 바로 사용 가능

out.print(“HELLO”); 출력스트림

getRequestURL(); 요청 URL을 구함

getRequestURI(); 요청 URL에서 쿼리 스트링을 제외한 부분을 구함.

getSession(flag); 요청 괸련 세션 객체를 구함.

getRequestDispacher(path); 지정 로컬 URL에 대한 RequestDispacher 객체를 구함.

getMethod();요청방식이 GET방식인지 POST방식인지 알려줌

getProtocol(); 사용중인 프로토콜을 알려줌

getParameter(String name); 지정한 이름의 파라미터를 구함

getParametervalues(String name); checkbox와 같이 여러 개의 값이 넘어올 때 사용.

response.sendRedirect(URL); 웹 서버가 지정한 페이지로 이동하도록 지시.

request.setAttribute(“속성이름”, 속성값); request 객체에 속성 값으로 저장해서 보내줌.

* 내장 객체의 영역 : 객체의 유효기간이라고도 불리며 해당 객체가 얼마 동안 살아있는가를 지정해 주는 것.

|  |  |
| --- | --- |
| Page | 하나의 JSP페이지를 처리할 때 사용되는 영역 |
| request | 하나의 요청을 처리할 때 사용되는 영역 |
| session | 하나의 브라우저와 과련된 영역 |
| application | 하나의 웹 애플리케이션과 관련된 영역 |

*액션태그?*

* 자바 코드 기술에 비해 가독성을 높인 코드
* XML문법을 따르므로 종료태그를 포한 하여야 함

<jsp:forward> 다른 사이트로 이동할 때 사용

<jsp:include> 자원을 현재 페이지의 내용에 포함

<jsp:param> 파라미터를 추가 forward, include, plugin과 같이 사용

<jsp:useBean> 빈을 생성하고 사용하기 위한 환경을 정의

<jsp:setProperty> 빈에서 속성 값을 할당

<jsp:getProperty> 빈에서 속성 값을 얻어옴.

ex>

<jsp:forward page=”aaa.jsp”>

<jsp:param name = “url” value”aaa.jsp:/>

</jsp:forward>

**4. 세션(session)**

*세션?*

* 다른 페이지로 이동하더라도 그 전 페이지의 정보를 유지하기 위한 기법
* 클라이언트의 하드웨어에 저장한 쿠키와 달리 서버 상에 존재하여 보안 유지가 가능
* 저장할 수 있는 데이터에 한계가 없다.
* 브라우저 단위당 한 개씩 존재

*자주 사용되는 메소드?*

getAttribute(String name) 이름에 해당되는 객체를 가져오며, 없을 경우 null을 반환

Object형으로 형 변환을 하여 사용하여야 함.

getAttributeNames() 세션에서 모든 객체의 이름을 Enumeration형으로 얻음.

getCreationTime() 세션이 만들어진 시간을 반환

getId() 해당 세션 가리키는 고유 id값을 String형으로 변환

getMaxInactiveInterval() 사용자가 다음 요청을 보낼 때까지의 세션을 유지하는 최대시간

isNew() 해당 세션이 처음 생성되었으면 true, 이전에 생성되었으면 false를 반환

removeAttribute(String name) 지정된 이름에 해당하는 객체를 세션에서 제거

setAttribute(String name, Object value) 세션에 지정된 이름에 객체를 추가

invalidate() 해당 세션을 없애고 세션에 속해있는 값들을 없앤다.

ex> 다른 서블릿에서 생성한 세션이 존재시 계속 진행, 없을 경우 로그인 페이지로 이동

HttpSession session = request.getSession();

String id = (String) session.getAttribute("id");

if(id == null){

response.sendRedirect("login.jsp");

return;}

**5. JDBC**

*JDBC?*

* 자바 프로그램에서 데이터베이스를 조작하기 위해 사용되는 API를 제공하는 클래스들
* 자바에서 JDBC를 이용하여 SQL을 오라클과 주고 받을 수 있음.

*JDBC 연결순서?*

1. JDBC 드라이버 로드 (DriverManager)
2. 데이터베이스와 연결 (Connection)
3. SQL문 실행 (Statement)
4. 데이터베이스와 연결 끊음 (ResultSet)

ex>

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn

=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","hr","hr");

String sql = “SELECT \* FROM employees”;

PreparedStatement pstmt = conn.prepareStatement(sql);

ResultSet rs = pstmt.executeQuery();

//insert, update, delete와 같이 결과값이 없는 경우 => executeUpdate();

while(rs.next()){ //마지막 행이 넘어갈 경우 rs.next()가 false 반환

out.println(rs.getString(“컬럼명”);

}

rs.close();

pstmt.close();

conn.close();

* PreparedStatement클래스를 통해 sql 변수를 효율적으로 조작

String sql = “SELECT \* FROM employees WHERE seq = ? name=?”;

pstmt.setInt(1, 14);

pstmt.setString(2, “Mike”);