Java Web Programming 입문

(Servlet/JSP)

24일차

오늘의 키워드

- 내장 객체 (Implicit Object)
  - 개요
  - request
  - response
  - pageContext

내장 객체 (Implicit Object)

- 내장 객체
  - JSP Page 내에서 Programmer가 선언하지 않아도 사용할 수 있는 Object
    - request, response, out, etc…
  - Java Code를 작성하는 Scriptlet (<% ~ %>) 안에서 사용 가능

- 생성, 선언
  - JSP 내에서 선언하지는 않지만
    - JSP Page 로부터 생성된 Servlet Code 내 jspService() Method 내에 자동으로 선언
  - 내장 객체가 선언되었다는 가정하에 사용

내장 객체	설명	타입
request	클라이언트의 요청 정보를 담고 있는 객체	javax.servlet.http.HttpServletRequest
response	클라이언트의 요청에 대한 응답 정보를 담고 있는 객체	javax.servlet.http.HttpServletResponse
out	페이지 내용을 담고 있는 출력 스트림 객체	javax.servlet.jsp.JspWriter
pageContext	페이지 실행에 필요한 컨텍스트 정보를 담고 있는 객체	javax.servlet.jsp.PageContext
config	JSP 페이지의 서블릿 설정 데이터의 초기화 정보 객체	javax.servlet.ServletConfig
session	세션 정보를 담고 있는 객체	javax.servlet.http.HttpSession
application	웹 애플리케이션의 등록 정보 담고 있는 객체	javax.servlet.ServletContext
page	JSP 페이지를 구현한 자바 클래스 인스턴스 객체	javax.servlet.jsp.HttpJspPage
exception	JSP 페이지의 서블릿 실행 시 처리하지 못한 예외 객체	java.lang.Throwable

내장 객체 (Implicit Object)

- 종류
  - request
    - Client의 요청 정보를 담고 있는 Object
    - javax.servlet.http.HttpServletRequest
  - response
    - Client의 요청에 대한 응답 정보를 담고 있는 Object
    - javax.servlet.http.HttpServletResponse
  - out
    - Page 내용을 담고 있는 Output Stream Object
      - 결과 출력을 위해 사용
      - JSP Page에서 생성되는 모든 결과는 out 을 통해 Client로 전달
    - javax.servlet.jsp.JspWriter
  - pageContext
    - JSP Page 실행에 필요한 Context 정보를 담고 있는 Object
    - javax.servlet.jsp.PageContext
  - config
    - JSP Page 실행 시 사용되는 Servlet Container 차원의 설정 정보 저장/관리
      - JSP Page의 servlet 설정 Data 초기화 정보 Object
    - javax.servlet.ServletConfig

내장 객체 (Implicit Object)

- 종류
  - session
    - Session에 대한 정보를 담고 있는 Object
    - javax.servlet.http.HttpSession
  - application
    - Web Application의 등록 정보를 담고 있는 Object
    - javax.servlet.ServletContext
  - page
    - JSP Page가 변환된 Servlet Object를 참조하는 참조 변수
      - 생성할 Servlet Object를 참조
    - javax.servlet.jsp.HttpJspPage
  - exception
    - JSP Page의 Servlet 실행 시 발생된 Excpetion을 담고 있는 Object
    - java.lang.Throwable

내장 객체 (Implicit Object)

- request & response
  - Client의 요청/응답 정보를 나타내는 내장 객체
    - _jspService() Method에 Parameter로 전달

```
public void _jspService ( HttpServletRequest request,
                          HttpServletResponse response )
              throws java.io.IOException, ServletException {
    ...
}
```

    - Servlet Container 에서 _jspService() Method로 전달
      - Servlet에서 service() Method와 동일한 방식

- 그 외 내장객체
  - _jspService() Method의 지역 변수로 선언
    - JSP Page내 Scriptlet 내부에 작성한 Java Code는 모두
      - _jspService() Method 내에 삽입

```
JspFactory      _jspxFactory    = null;
PageContext     pageContext     = null;
HttpSession     session         = null;
ServletContext  application     = null;
ServletConfig   config          = null;
JspWriter       out             = null;
Object          page            = this;
```

내장 객체 (Implicit Object)

- _jspFactory Object
  - JSP에서 가장 먼저 초기화되는 변수
    - JspFactory Class의 Static Method인 getDefaultFactory() Method 이용하여 생성

```
_jspxFactory = JspFactory.getDefaultFactory();
```

- pageContext Object
  - _jspxFactory Object로부터
    - 다른 내장 객체를 생성하는 pageContext 내장 객체 생성

```
pageContext =
    _jspxFactory.getPageContext(
        this, request, response, null, true, 8192, true
    );
```

  - getPageContext() Method
    - 현재 Servlet Object
      - 요청하는 Servlet을 지정
    - Request & Response Object
      - _jspService() Method의 Parameter인 요청 & 응답 Object

내장 객체 (Implicit Object)

- pageContext Object
  - getPageContext() Method
    - Error Page URL
      - Error가 생겼을 경우 출력할 Error Page 의 URL 주소
    - Session 사용 여부
    - 출력 Buffer 크기
      - Client로 전송할 Data의 Buffer 크기를 지정
    - autoflush 지원 여부
      - Client로 전송할 Data를 위해 autoflush의 지원 여부를 결정

- 다른 내장 객체 초기화

```
application = pageContext.getServletContext()    ;
config      = pageContext.getServletConfig()     ;
session     = pageContext.getSession()           ;
out         = pageContext.getOut()               ;
```

내장 객체 (Implicit Object)

- request & response Object
  - Servlet의 service() Method

```
public void doGet( HttpServletRequest request,
                   HttpServletResponse response ) {
    ...
}
```

  - JSP로부터 자동 생성된 _jspService() Method

```
public void _jspService( HttpServletRequest request,
                         HttpServletResponse response ) {
    ...
}
```

  - Servlet과 JSP가 약간 다른 구조를 가지지만…
    - 동작원리는 같음
    - Client의 요청이 들어왔을 때 service() 계열 Method 호출
      - Parameter에는 HttpServletRequest, HttpResponse Object 사용

request

- request
  - 는…
    - HttpServletRequest Type Object
    - Client로부터 전달된 요청 정보를 담고 있는 Object
    - HttpServletRequest Interface의 Instance
  - 기능
    - Client로부터 전달된 Parameter 처리
    - Server의 정보 출력
    - Client로부터 Query로 전달된 Parameter 출력
    - HTTP Request Message의 Header 출력
    - Cookie 및 속성 처리

request

- clientInfo.jsp

```
<%@ page contentType = "text/html; charset=euc-kr" %>
<html>
    <head>
        <title>request 객체를 이용한 정보 출력</title>
    </head>
    <body>
        <p>도메인 이름       = <%=request.getServerName()          %></p>
        <p>서버 이름         = <%=request.getServerPort()          %></p>
        <p>요청 URI          = <%=request.getRequestURI()          %></p>
        <p>쿼리 문자열       = <%=request.getQueryString()         %></p>
        <p>클라이언트 IP     = <%=request.getRemoteAddr()          %></p>
        <p>요청 프로토콜     = <%=request.getProtocol()            %></p>
        <p>요청 방식         = <%=request.getMethod()              %></p>
        <p>문자 인코딩       = <%=request.getCharacterEncoding()   %></p>
        <p>데이터 길이       = <%=request.getContentLength()       %></p>
        <p>콘텐츠 타입       = <%=request.getContentType()         %>
    </body>
</html>
```

  - http://localhost:8080/MySample/clientInfo.jsp?name=test

request

- request 내장 객체의 Method와 수행 결과

메소드 원형	메소드의 기능	실행 결과
String getServerName()	서버의 도메인 이름을 반환	localhost
int getServerPort()	서버의 포트 번호를 반환	8080
String getRequestURI()	요청된 URL에서 경로를 반환	/MySample/clientInfo.jsp
String getQueryString()	쿼리 문장을 반환	name= **test**
String getRemoteAddr()	클라이언트의 주소를 반환	127.0.0.1
String getProtocol()	클라이언트가 요청한 프로토콜을 반환	HTTP/1.1
String getMethod()	요청 방식(GET, POST)을 반환	GET
String getCharacterEncoding()	요청 메시지의 인코딩 형식을 반환	null
long getContentLength()	요청 메시지 중 데이터의 길이를 반환	-1
String getContentType()	요청 메시지의 콘텐츠 타입(Contents Type)을 반환	null

request

- Parameter 출력 Method
  - Client로부터 전달된 Parameter값 추출

메소드 원형	메소드의 기능
String getParameter(String name)	name 이름을 가진 파라미터의 값을 반환
String[] getParameterValues(String name)	name 이름을 가진 모든 파라미터 값을 반환 (여러 개의 값을 가질 수 있는 양식에서 사용)
Enumeration getParameterNames()	파라미터의 이름 집합을 Enumeration 객체로 반환
Map getParameterMap()	파라미터를 Map 객체 형태로 반환

  - getParameter() Method
    - Text Field나 Radio Button 처럼
      - 단일 값을 가지는 양식 Data 값을 추출할 때 사용
  - getParameterValues() Method
    - Check Box나 다중 선택 Box 처럼
      - 다중 선택이 가능한 양식으로부터 값을 얻기 위해 사용

request

- requestParameter.html

```
<html>
    <head>
        <title>파라미터 출력</title>
    </head>
    <body>
    <form action="requestParameter.jsp" method="get">
        <p>이름 : <input type="text" name="name"></p>
        <p>취미 :
            <input type="checkbox" name="hobby" value="Ski">스키
            <input type="checkbox" name="hobby" value="Inline">인라인
        </p>
        <input type="submit" value="Submit">
    </form>
    </body>
</html>
```

request

- Parameter 출력 Method
  - getParameter() Method
    - Text Field나 Radio Button 처럼
      - 단일 값을 가지는 양식 Data 값을 추출할 때 사용

```
String name = request.getParameter("name");
```

  - getParameterValues() Method
    - Check Box나 다중 선택 Box 처럼
      - 다중 선택이 가능한 양식으로부터 값을 얻기 위해 사용

```
String[] hobby = request.getParameterValues("hobby");
```

```
for ( int i = 0; i < hobby.length; i++ ){
    out.print(hobby[i] + " ");
}
```

  - getParameterNames() Method
    - JSP Page로 전달된 전체 Parameter 이름 집합을 얻기 위한 Method
    - Parameter 이름 집합을 Enumeration Object로 반환

```
Enumeration param = request.getParameterNames();
```

```
while(param.hasMoreElements()) {
    String name = (String)param.nextElement();
    String[] value = request.getParameterValues(name);
}
```

request

- Parameter 출력 Method
  - getParameterMap() Method
    - Parameter 이름과 값이 함께 있는 Map Object 반환

```
Map paramMap = request.getParameterMap();
```

    - Map Object 의 사용

```
Set paramKeySet = paramMap.keySet();

Iterator keyIterator = paramKeySet.iterator();

while(keyIterator.hasNext()) {
    String name = (String)keyIterator.next();
    String[] value = (String[])paramMap.get(name);
}
```

request

- requestParameter.jsp

```jsp
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ page import="java.util.*"%>

<html>
    <head>
        <title>request 객체를 이용한 파라미터 출력</title>
    </head>
    <body>
        <h2>getParameter()</h2>
        이름 : <%=request.getParameter("name")%>
        <h2>getParameterValues()</h2>
        취미 : <%
                    String[] hobby = request.getParameterValues("hobby");
                    for(int i = 0; i < hobby.length; i++) {
                        out.print(hobby[i] + " ");
                    }
                %>
        <h2>getParameterValues()</h2>
        <%
            Enumeration param = request.getParameterNames();
            while(param.hasMoreElements()) {
                String name = (String)param.nextElement();
                out.print(name + " : ");
                String[] value = request.getParameterValues(name);
                for(int i = 0; i < value.length; i++) {
                    out.print(value[i] + " ");
                }
                out.print("<br>");
            }
        %>
        <h2>getParameterMap()</h2>
        <%
            Map paramMap = request.getParameterMap();
            Set paramKeySet = paramMap.keySet();
            Iterator keyIterator = paramKeySet.iterator();
            while(keyIterator.hasNext())      {
                String name = (String)keyIterator.next();
                out.print(name + " : ");
                String[] value = (String[])paramMap.get(name);
                for(int i = 0; i < value.length; i++) {
                    out.print(value[i] + " ");
                }
                out.print("<br>");      }
        %>
    </body>
</html>
```

request

- Http Request Message Header 출력

메소드 원형	메소드의 기능
String getHeader(String name)	지정한 이름의 헤더 값을 반환
Enumeration getHeaders(String name)	지정한 이름의 헤더 목록을 Enumeration 객체로 반환
Enumeration getHeaderNames()	헤더의 이름 집합을 Enuemration 객체로 반환

  - getHeaderNames() Method

```
Enumeration headerEnum = request.getHeaderNames();
```

  - Enumeration Object로부터 HTTP Header 값 출력

```
while(headerEnum.hasMoreElements()) {
    String headerName = (String)headerEnum.nextElement();
    String headerValue = request.getHeader(headerName);
}
```

request

- printHeader.jsp

```jsp
<%@ page contentType = "text/html; charset=euc-kr" %>
<%@ page import="java.util.*"%>

<html>
    <head>
        <title>request 객체를 이용한 헤더 출력</title>
    </head>
    <body>
        <%
            Enumeration headerEnum = request.getHeaderNames();
            while(headerEnum.hasMoreElements())      {
                String headerName = (String)headerEnum.nextElement();
                String headerValue = request.getHeader(headerName);
                out.println(headerName + " : " + headerValue + "<br>");
            }
        %>
    </body>
</html>
```

response

- response 내장 객체
  - 는…
    - HttpServletResponse Type Object
    - Client로 전달될 응답 메시지를 담고 있음
    - request Object와 함께 Servlet Container에 의해
      - _jspService() Method의 Parameter로 전달
  - 기능
    - 응답 Page에 대한 설정 정보 지정
    - Client로 전달될 Response Message의 Header 설정
    - 다른 Web Page 로의 Redirect

response

- Response Page 설정
  - response Object의 응답 Page 설정 Method
    - JSP Page로부터 생성된 응답 HTML 문서 속성 설정

메소드 원형	메소드의 기능
void setContentType(String type)	출력될 페이지의 콘텐츠 타입을 설정
void setCharacterEncoding(String charset)	문자의 인코딩 스타일을 지정

    - setContentType(), setCharacterEncoding() Method 이용

```
response.setContentType("text/html");
response.setCharacterEncoding("euc-kr");
```

    - JSP 상에서라면 page 지시문을 사용

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

- 응답 Header 설정을 위한 response Object의 Method

메소드 원형	메소드의 기능
boolean containsHeader(String name)	이름이 name인 헤더의 항목이 존재하는지 검사한다.
void addHeader(String name, String value)	이름이 name이고 그 값이 value인 헤더를 추가한다.
void setHeader(String name, String value)	name 헤더의 값을 value로 지정한다.

response

- response Object 를 이용한
  - Redirect
    - 다른 Web Page로의 이동

```
void sendRedirect(java.lang.String location)
```

```
int level = Integer.parseInt(request.getParameter("level"));
if(level == 1)
    response.sendRedirect("level_1.jsp");
else
    response.sendRedirect("level_etc.jsp");
```

response

- levelCheck.jsp

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

<html>
    <head>
        <title>회원 레벨 검사</title>
    </head>
    <body>
        <%
            int level = Integer.parseInt(request.getParameter("level"));
            if(level == 1)
                response.sendRedirect("level_1.jsp");
            else
                response.sendRedirect("level_etc.jsp");
        %>
    </body>
</html>
```

  - http://localhost:8080/MySample/levelCheck.jsp?level=1
    - sendRedirect() Method에 의해 추가된 HTTP Header
      - HTTP/1.1 302 Moved Temporarily
      - Location : http://127.0.0.1:8080/MySample/level_1.jsp

response

- level_1.jsp

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

<html>
    <head>
        <title>회원 레벨 1</title>
    </head>
    <body>
        <h2>1등급 회원 입니다.</h2>
    </body>
</html>
```

- level_etc.jsp

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

<html>
    <head>
        <title>회원 레벨 etc</title>
    </head>
    <body>
        <h2>일반 회원 입니다.</h2>
    </body>
</html>
```

pageContext

- pageContext 내장 객체
  - 는…
    - 각 Servlet에 하나씩 존재
      - 하나의 Servlet 안에서만 유효
    - 하나의 Servlet을 관리, 정보 공유를 도와주는 Object
  - 기능
    - 다른 내장 객체 생성
      - JSP Page를 실행하여 생성되는 Servlet 하나당 하나의 PageContext가 존재
      - 하나의 JSP Page가 생성 될 때 해당 Page의 다양한 정보와 특성들을 관리
        - JSP Page를 실행시켜 생기는 Servlet 하나당 하나의 PageContext가 존재
      - Page 범위 내에서의 정보 공유
  - 특성
    - pageContext Object를 이용한 정보 공유의 범위는
      - 현재 Page내로 제한됨

pageContext

- ServletContext, PageContext Object
  - ServletContext
    - 하나의 Web Application에 하나의 ServletContext Object가 존재
  - PageContext
    - 하나의 Servlet에 하나의 PageContext Object가 존재
- ServletContext, PageContext Obejct의 정보공유
  - ServletContext Object
    - 하나의 Web Application으로 제한
  - PageContext Object
    - 하나의 Servlet으로 제한

pageContext

- 같은 Page에서의 정보 공유
  - pageContext 내장 객체의 속성, Method

메소드 원형	메소드의 기능
void setAttribute(String name, Object value)	이름이 name이고 값이 value인 속성을 PageContext에 등록한다.
Object getAttribute(String name)	이름이 name인 속성 값을 반환한다.
void removeAttribute(String name)	이름이 name인 속성을 제거한다.

  - setAttribute() Method

```
String contextValue  = "Using PageContext";
pageContext.setAttribute("contextName",contextValue );
```

  - getAttribute() Method

```
Object obj = pageContext.getAttribute("contextName");
String value = (String)obj;
out.println("<h3>PageContext Value = " + value + "</h3>");
```

pageContext

- pageContextAttr.jsp

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

<%pageContext.setAttribute("contextName","Using PageContext" );%>

<html>
    <head>
        <title>PageContext를 이용한 정보 공유</title>
    </head>
    <body>
        <h2>PageContext의 속성 호출</h2>
        <%
            Object obj = pageContext.getAttribute("contextName");
            if(obj != null)
                out.println("<h3>PageContext Value = " + (String)obj + "</h3>");
            else
                out.println("Attrubute Not found!!");
            %>
        <h2>PageContext의 속성 재호출</h2>
        <%
            Object obj2 = pageContext.getAttribute("contextName");
            if(obj2 != null)
                out.println("<h3>PageContext Value = " + (String)obj + "</h3>");
            else
                out.println("Attrubute Not found!!");
        %>
    </body>
</html>
```

pageContext

- 다른 JSP Page에서 속성 호출
- callOtherPageContext.jsp

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

<html>
    <head>
        <title>다른 페이지에서 속성</title>
    </head>
    <body>
        <h2>다른 페이지에 등록된 속성 호출</h2>
        <%
        Object obj = pageContext.getAttribute("contextName");
        if(obj != null)
            out.println("<h3>PageContext Value = " + (String)obj + "</h3>");
        else
            out.println("Attrubute Not found!!");
        %>
    </body>
</html>
```

pageContext

- pageContext Object의 Forward
  - forward() Method는…
    - 현재 Page에서의 요청/응답을 처리할 제어권을
      - 해당 URL로 넘기는데 사용
    - forward() Method를 호출한 원래 Page의 요청/응답은
      - 무시되고 forwarding된 Page의 요청과 응답을 처리

```
void forward(java.lang.String relativeUrlPath)
```

    - 특징
      - Forward된 해당 Page로 제어권을 모두 넘김
      - Forward된 Page로 넘어가기 전 out 내장 객체의 clear() Method 호출

pageContext

- pageForward.jsp

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

<html>
    <head>
        <title>pageContext의 Forward</title>
    </head>
    <body>
        <h2>다른 페이지로 포워딩</h2>
        <%
            pageContext.forward("forwardReceive.jsp");
            out.println("<h3>제어권이 다시 돌아오지 않는다.</h3>");
        %>
    </body>
</html>
```

- forwardReceive.jsp

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

<html>
    <head>
        <title>pageContext의 Forward</title>
    </head>
    <body>
        <h2><%=request.getParameter("name")%></h2>
    </body>
</html>
```

pageContext

- pageForward.jsp 실행
  - http://localhost:8080/MySample/pageForward.jsp?name=test
  - forwarding이 실행되기전 문장들이 출력되지 않음
    - forward() Method가 호출되면
      - 자동으로 지금까지 Buffer에 저장된 내용을 지워주는
        - out.clear() Method가 실행되기 때문
      - 다른 Page로 Forwarding될 때
        - request, response에 대한 제어권이 같이 Forwarding 되므로
          - 호출된 Jsp Page에 동일하게 request, response 내장객체를 사용할 수 있음
      - Forwarding이 실행된 후
        - 다시 제어권이 넘어오지 않으므로
          - 마지막 문장도 출력되지 않음
- forward() Method의 특징
  - Forwarding 실행 시 out.clear()가 호출되어 Buffer 비워짐
  - Forwarding 된 Page로 request, response에 대한 제어권이 넘어감
  - Forwarding 후 다시 본래 Page로의 제어권이 넘어오지 않음

pageContext

- pageContext Object의 Include
  - include() method 는…
    - forward() 와는 달리 현재 Page에서의 요청과 응답에 대한 제어권을
      - 해당 Page로 잠시 넘겼다가 처리가 끝나면 돌려받음
    - include() Method 호출 시
      - 해당 Page로 요청과 응답 제어권을 넘겨 처리한 후
      - 처리가 끝나면 그 결과와 함께
      - 다시 원래의 Page로 처리 결과와 제어권을 돌려주고
      - 처리를 마무리하도록 함

```
void include(java.lang.String relativeUrlPath)
```

  - 특징
    - include() 요청이 일어나면
      - 요청 시까지 해오던 작업들이 들어있던 Buffer가 flushing
        - 요청이 전달되는 Page로 넘어감
      - 해당 Page에서 임시로 제어권을 가지고 작업
        - 작업이 끝나면 제어권, 작업 결과를 다시 원래 Page로 돌려줌

pageContext

- pageInclude.jsp

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

<html>
    <head>
        <title>pageContext의 Include</title>
    </head>
    <body>
        <h2>다른 페이지의 인클루드</h2>
        <%
            pageContext.include("included.jsp");
        %>
        <h3>제어권이 돌아왔습니다.</h3>
    </body>
</html>
```

- included.jsp

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

<html>
    <head>
        <title>pageContext의 Include</title>
    </head>
    <body>
        <h2><%=request.getParameter("name")%></h2>
    </body>
</html>
```

pageContext

- pageInclude.jsp의 실행
  - http://localhost:8080/MySample/pageInclude.jsp?name=test
  - 문장 출력 후,
    - included.jsp Page가 호출되어 실행
  - 호출된 Page로 넘어간 제어권이 다시 원래 Page로 돌아옴
    - 나머지 코드 실행
- include() Method의 특징
  - 다른 JSP Page가 호출될 때 Buffer가 비워지지 않음
  - 호출된 Page로 request, response에 대한 제어권이 같이 넘어감
  - 호출된 후 다시 원래의 Page로 제어권이 넘어옴

오늘 숙제

- 집에서 해보자!