**[ 4] Servlet 웹프로그래밍**

목표 : 컨텍스트 패스(context Path), 서블릿 작동순서, 라이프사이클

* 서블릿은 JSP 표준이 나오기 전에 만들어진 표준으로 자바에서 웹 애플리케이션을 개발할 수 있도록 하기 위해 만들어 졌으며 자바 클래스를 웹에서 실행 할 수 있도록 한 웹 프로그램
* javax.servlet.http.HttpServlet 클래스로부터 상속받아서 작성
* 위의 클래스는 톰캣의 servlet-api.jar 에 포함
* 작성과정
  + 서블릿 규칙에 따라 자바 코드 생성
  + 경우에 따라서 web.xml 파일에서 서블릿을 주소와 매핑
  + 웹 컨테이너 재실행
* 서블릿 요청 처리
  + 요청 방식에 따라 doGet이나 doPost 메서드를 재정의해서 처리
  + service 메서드를 재정의해서 사용할 수 있는데 이 메서드는 get 방식이나 post 방식 상관없이 호출되며 이 메서드가 호출되면 doGet이나 doPost 메서드는 호출되지 않는다
* 서블릿에서 요청을 처리하기 위해 오버라이딩 한 메서드는 request 객체를 이용해서 웹 브라우저의 요청 정보를 읽어 오던가 아니면 response를 이용해서 응답을 전송할 수 있다
* 응답을 전송하고자 하는 경우는 response 객체의 setContentType()메서드를 이용해서 타입과 인코딩 방식을 지정해 주어야 한글이 깨지지 않는다.
* 웹 브라우저에 데이터를 전송하려면 getWriter()를 호출해서 문자열 데이터를 출력할 수 있는 PrintWriter를 가져오고 print()나 println()을 이용해서 전송한다

1. 클라이언트측에서 WAS로 요청하는 방식이 두가지 : Get방식, Post방식이 있다.

(HTML코드안에 전송하는 방식 지정해 주는 태그에 인해 지정 태그에 get으로 할지 post로 할지에 따라 정해짐)

* Get방식 – URL값으로 정보가 전송되어 보안에 약함
  + 주소와 매개변수를 붙여서 주소 표시줄에 입력하는 방법(?로 구분)
  + a 태그를 이용해서 페이지를 요청하는 경우
  + form태그에서 명시적으로 GET방식으로 요청하는 경우(method=”get”)
  + 매개변수의 데이터는 255자 이내이며 보안성이 없음
* Post방식 – header를 이용해 정보가 전송되어 보안에 강함
  + form태그에서 명시적으로 POST 방식으로 요청(method=”post”)
  + 데이터의 크기에 제한이 없으며 URL에 표시되지 않으므로 보안성이 우수
  + 한글 파라미터 전달시 request.setCharacterEncoding(“utf-8”)추가

1. doGet() ;

1. doPost() ; 매개변수로 HttpServletRequest와 HttpServletResponse를 받습니다

웹브라우저

↓↑

HttpServletRequest > 클라이언트의 요청 처리 객체

HttpServletResponse > 클라이언트에게 응답 처리 객체

* HTML

<body>

<a href=*"http://localhost:8181/ch04\_servlet/Ex1Test?para=aa"*>Ex1test로 파라미터 넘기기</a>

<hr>

<button onclick="location.href='../Ex1Test?para=aa'">Ex1Test로 파라미터 넘기기</button>

<hr>

<form action=*"../Ex1Test"* method=*"get"*>

<input type=*"text"* name=*"para"* value=*"aa"*>

<input type=*"submit"* value=*"Ex1Test로 파라미터 넘기기(GET방식)"*>

</form>

<hr>

<form action=*"../Ex1Test"* method=*"post"*>

<input type=*"text"* name=*"para"* value=*"aa"*>

<input type=*"submit"* value=*"Ex1Test로 파라미터 넘기기(POST방식)"*>

</form>

</body>

* Servlet 소스

@WebServlet("/Ex1Test")

**public** **class** Ex1Test **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

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

System.***out***.println("get방식");

String para = request.getParameter("para");

response.setContentType("text/html;charset=utf-8");

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<head></head>");

out.println("<body>");

out.println("<h2>GET방식</h2>");

out.println("<h2>전달받은 파라미터 값은 "+para+"</h2>");

out.println("</body>");

out.println("</html>");

}

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

System.***out***.println("post방식");

request.setCharacterEncoding("utf-8");

String para = request.getParameter("para");

response.setContentType("text/html;charset=utf-8");

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<head></head>");

out.println("<body>");

out.println("<h2>POST 방식</h2>");

out.println("<h2>전달받은 파라미터 값은 "+para+"</h2>");

out.println("</body>");

out.println("</html>");

}

}

1. 파라미터값을 배열로 받는 경우의 예제

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>Insert title here</title>

</head>

<body>

<h2>오늘의 메뉴와 식당 고민</h2>

<form action=*"../Ex2ArrayParameter"*> <!-- method=*"post"도 해 봄 -->*

<p>메뉴

<select name=*"menu"* multiple=*"multiple"* size=*"3"*>

<option>된장찌게</option>

<option>김치찌게</option>

<option>부대찌게</option>

<option selected=*"selected"*>불고기</option>

</select>

</p>

<p>식당 <input type=*"checkbox"* name=*"rest"* value=*"종로"*>종로

<input type=*"checkbox"* name=*"rest"* value=*"광교"*>광교

<input type=*"checkbox"* name=*"rest"* value=*"미아"*>미아

<input type=*"checkbox"* name=*"rest"* value=*"금천"*>금천

</p>

<p>국적 <input type=*"radio"* name=*"nation"* value=*"한국"*>한국

<input type=*"radio"* name=*"nation"* value=*"미국"*>미국

<input type=*"radio"* name=*"nation"* value=*"중국"*>중국

<input type=*"radio"* name=*"nation"* value=*"일본"*>일본

</p>

<p><input type=*"submit"* value=*"입력"*>

</form>

</body>

</html>

@WebServlet("/Ex2ArrayParameter")

**public** **class** Ex2ArrayParameter **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

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

String[] menu = request.getParameterValues("menu");

String[] rest = request.getParameterValues("rest");

String nation = request.getParameter("nation");

response.setContentType("text/html;charset=utf-8");

PrintWriter out = response.getWriter();

out.println("<html>");

out.println("<head>");

out.println("<style>");

out.println("h2 {color:blue;}");

out.println("</style>");

out.println("</head>");

out.println("<body>");

out.println("<h2>오늘은 ");

**for**(String m : menu) {

out.println(m+ " ");

}

out.println("을(를) 먹어야지</h2>");

out.println("<h2>넘어온 식당 : "+ Arrays.toString(rest) + "</h2>");

out.println("<h3>국적은 "+nation+"</h2>");

out.println("</body>");

out.println("</html>");

out.close();

}

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

request.setCharacterEncoding("utf-8");

doGet(request, response);

}

}

1. 컨텍스트 패스(Context Path) ; WAS(Web Application Server)에서 웹어플리케이션을 구분하기 위한 path. **해당 웹 어플리케이션이 이름**. 이클립스에서 프로젝트를 생성하면, 자동으로 server.xml에 추가된다.

그러므로, 프로젝트를 삭제하고 다시 같은 이름의 프로젝트를 생성할 수 없다.

<Context docBase="jsp\_ch02" path="/jsp\_ch02" reloadable="true" source="org.eclipse.jst.jee.server:jsp\_ch02"/>

1. Servlet 작동순서 ; 클라이언트에서 servlet요청이 들어 오면 서버에서는 servlet컨테이너를 만들고, 요청이 있을 때마다 스레드가 생성된다.

**웹브라우저 → [웹서버] → [웹어플리케이션서버] → [Servlet 컨테이너 servlet객체생성, 쓰레드생성]**

WAS

WAS

JVM

1. Servlet 라이프 사이클 (생명주기) ; 언어를 할 때 라이프사이클은 호출할 시점의 정확한 시점을 알 필요가 있다.
   1. Servlet의 사용도가 높은 이유는 빠른 응답 속도 때문. Servlet은 최초 요청 시 객체가 만들어져 메모리에 로딩되고, 이후 요청 시에는 기존의 객체를 재활용하게 된다. 따라서 동작 속도가 빠르다.
   2. Servlet의 라이프사이클
      1. 실행 시기 : Servlet 객체 생성(최초 한번)
      2. 최초요청, 컨테이너 시작 ; init() 호출(최초 한번)
      3. 모든 요청과 응답 ; service(), doGet(), doPost() 호출(요청시 매번, doGet이나 doPost없으면 service실행. 통상적으로 doGet과 doPost를 둔다)
      4. 컨테이너 종료/어플리케이션종료 ; destroy() 호출(마지막한번 자원해제 : Servlet수정, 서버재가동 등)

최초 한번

Servlet 객체 생성

선처리 @PostConstruct

init() 호출

최초 한번

service() doGet() doPost() 호출

요청시 매번

destroy() 호출

마지막 한번(자원해제: servlet수정, 서버 재가동 등등..)

후처리 @PreDestroy

서버종료및재가동

/애플리케이션 종료

1. Servlet 선처리, 후처리
   1. Servlet의 라이프 사이클중 init()과 destroy()메소드와 관련하여 선처리(init()전)와 후처리(destroy()후) 작업이 가능
      1. init()호출 전 선처리 : @PostConstruct 후 메소드
      2. destroy()호출 후 후처리 : @PreDestroy 후 메소드

@WebServlet("/LifeCycleEx")

**public** **class** LifeCycleEx **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** LifeCycleEx() {

**super**();

}

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

// **TODO** Auto-generated method stub

System.***out***.println("doGet");

}

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

// **TODO** Auto-generated method stub

System.***out***.println("doPost");

}

@Override

**public** **void** init() **throws** ServletException {

// **TODO** Auto-generated method stub

System.***out***.println("init");

}

@Override

**public** **void** destroy() {

// **TODO** Auto-generated method stub

System.***out***.println("destroy");

}

@PostConstruct

**private** **void** initPostConstruct(){

System.***out***.println("@PostConstruct");

}

@PreDestroy

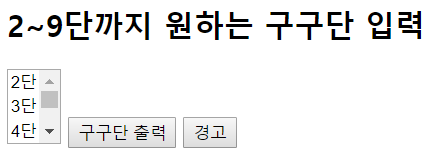
**private** **void** destoryPreDestroy(){

System.***out***.println("@PreDestroy");

}

}

Quiz 1. 2~9단까지 원하는 구구구단 수를 입력 받아 해당 구구단을 출력하는 서블릿을 구현하시오



2단

2\*1 = 2

2\*2 = 4

2\*3 = 6

…

5단

5\*1 = 5

5\*2 = 10

5\*3 = 15

…

exam/quiz1.html

css/quiz1.css

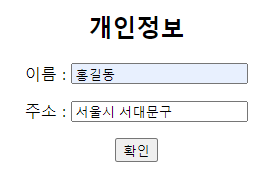
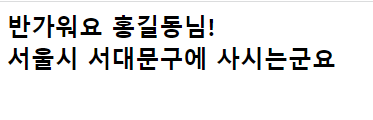
@charset *"UTF-8"*;

.dansu {padding:*5px* *10px*;text-align: *center*;}

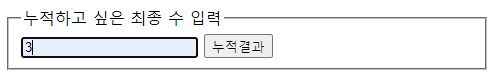
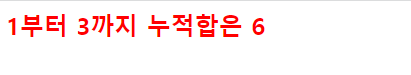
exam/Quiz1.java (서블릿)

겟 or 포스트

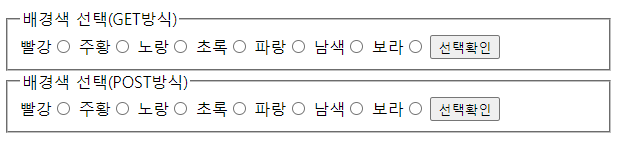
Quiz 2. 이름과 주소를 입력받기 위한 HTML 문서를 작성하고 입력한 내용을 출력해 주는 두번째 화면을 구현하는 서블릿을 구현하시오(post방식이용) 오른쪽이 서블릿.

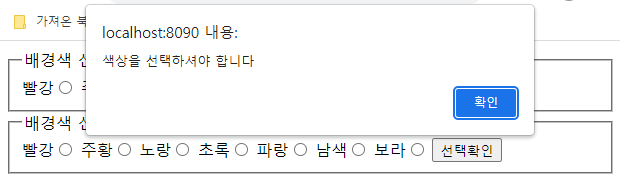
 

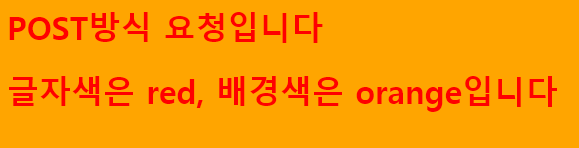
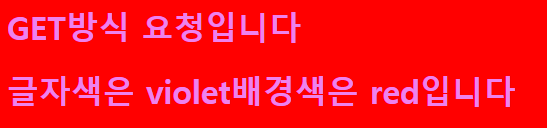
Quiz 3. get방식으로 처리

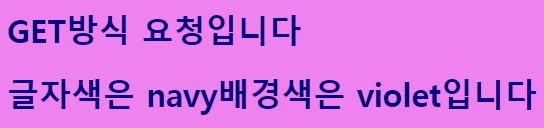
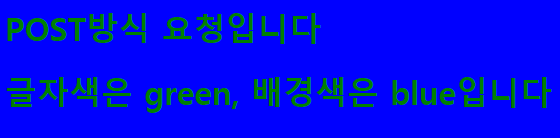
 

Quiz 4. 위 2개 html

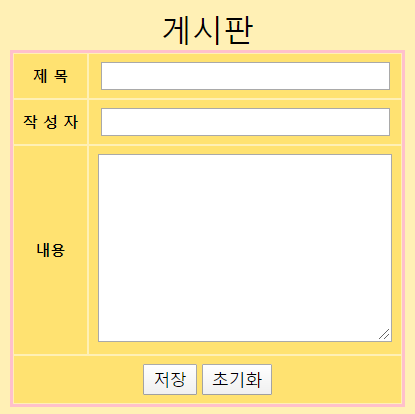




Quiz 5. 게시판 글쓰기 기능의 데이터(제목, 작성자, 내용textarea) 입력을 위한 HTML 문서를 작성하고 입력한 내용을 출력해 주는 두번째 화면을 구현하는 서블릿을(255 넘어가는것post) 구현하시오.

Quiz5.css

@charset *"UTF-8"*;

**body** { background-color: *#FFF0B5*;font-size: *9pt*; }

**table** {border: *pink* *solid* *2px*; margin:*0* *auto*;}

**tr** {background-color: *#FFE271*;}

**tr***:HOVER* {background-color: *orange*;cursor: *pointer*;}

**td,th** {text-align: *center*; padding: *5pt*;}

**caption** { font-size: *25px*;}

1. Servlet Parameter
   1. Form태그의 submit 버튼을 클릭하여 데이터를 서버로 전송하면, 해당파일(Servlet)에서는 HttpServletRequest객체를 이용하여 Parameter값을 얻을 수 있다.
      1. request.getParameter("파라미터이름");
      2. request.getParameterValues("파라미터배열이름");
      3. request.getParameterNames() ; 파라미터 이름들이 배열로
2. 한글처리
   1. Tomcat 서버의 기본 문자 처리 방식은 IOS-8859-1 방식 입니다. 따라서 개발자가 별도의 한글 인코딩을 하지 않으면 한글이 깨져 보이는 현상이 있습니다.
   2. response객체인지, request객체인지 따라서 한글처리 방식에 차이가 있습니다
      1. response객체 : response.setContentType("text/html;charset=utf-8");
      2. request객체(POST방식) : request.setCharacterEncoding("utf-8");



#D4A190(버튼과 선들) #A47160(회원가입 글씨) #3F3F48(th 글씨)

@charset *"UTF-8"*;

*#joinForm\_wrap* {width:*430px*; margin: *10px* *auto*; border: *1px* *solid* *#D4A190*; padding:*20px*;}

*#join\_title*{

width:*400px*;

margin:*10px* *auto*;

padding-bottom:*20px*;

border-bottom:*1px* *solid* *#D4A190*;

color:*#A47160*;

font-size:*1.5em*;

text-align:*center*;

font-weight:*bold*;

}

*#joinForm\_wrap* **table**{margin:*0* *auto*; color:*#A47160*;}

*#joinForm\_wrap* **table** **tr**{height:*40px*; }

*#joinForm\_wrap* **table** **th**{

width:*120px*;

font-size:*0.9em*;

color:*#3F3F48*;

text-align:*left*;

padding-left:*10px*;

}

*#joinForm\_wrap* **table** **td** {width: *300px*;}

*.name***,** *.id***,** *.pw***,** *.pwChk***,** *.birth***,** *.email***,** *.mailSend* {

border: *1px* *solid* *#D4A190*; height:*20px*;

width:*95%*; padding:*3px*;

}

*.mailSend* {height: *60px*;}

*.joinBtn\_style* {

color: *#ffffff*;

background: *#A47160*;

border-style: *none*;

outline: *none*;

width:*32%*; height:*120%*;

box-sizing: *border-box*;

}

11. Servlet 초기화 파라미터 : ServletConfig의 메소드 이용

* 특정 Servlet이 생성될 때 초기에 필요한 데이터들이 있습니다. 예를 들어 특정 경로 및 아이디 정보 등이다.
* 이러한 데이터들을 초기화 파라미터라고 하며, 어노테이션을 이용하거나, web.xml에 기술하고 Servlet파일에서는 getInitParameter(“파라미터”) 메서드를 이용해서 접근(사용)한다.
* Servlet파일에 초기화 파라미터 기술하는 법
  1. Servlet 클래스 제작 -> @WebInitParam에 초기화 파라미터 기술 -> ServletConfig 메소드 이용해서 데이터 불러오기
  2. Servlet소스 내

@WebServlet(urlPatterns= {"/ServletInitParam"},

initParams= {@WebInitParam(name="id",value="root"),

@WebInitParam(name="pw",value="mysql"),

@WebInitParam(name="jdbcdriver", value="com.mysql.jdbc.Driver")})

**public** **class** ServletInitParam **extends** HttpServlet {

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

String id=getInitParameter("id");

String pw = getInitParameter("pw");

String jdbcdriver = getInitParameter("jdbcdriver");

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<h2>ID : "+id+"</h2>");

out.println("<h2>PW : "+pw+"</h2>");

out.println("<h2>jdbcdriver : "+jdbcdriver+"</h2>");

out.close();

System.***out***.println("id:"+id);

System.***out***.println("pw:"+pw);

System.***out***.println("jdbcdriver:"+jdbcdriver);

}

}

* Web.xml에 초기화 파라미터 기술하는 법
  1. Servlet 클래스 제작 -> web.xml파일에 초기화 파라미터 기술 -> GenericServlet 메소드 이용해서 데이터 불러오기
  2. 소스

<servlet>

<servlet-name>ServletInitParam</servlet-name>

<servlet-class>com.tj.ex2.ServletInitParam</servlet-class>

<init-param>

<param-name>id</param-name>

<param-value>root</param-value>

</init-param>

<init-param>

<param-name>pw</param-name>

<param-value>mysql</param-value>

</init-param>

<init-param>

<param-name>jdbcdriver</param-name>

<param-value>com.mysql.jdbc.Driver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>ServletInitParam</servlet-name>

<url-pattern>/ServletInitParam</url-pattern>

</servlet-mapping>

12. 데이터 공유 : ServletContext의 메소드 이용

* 여러 Servlet에서 특정 데이터를 공유해야 할 경우 context parameter를 이용해서 web.xml에 데이터를 기술하고, Servlet에서 공유하면서 사용 할 수 있다. Servlet에 대한 기술보다는 위에 기술해야 한다.

<context-param>

<param-name>contextId</param-name>

<param-value>scott</param-value>

</context-param>

<context-param>

<param-name>contextPw</param-name>

<param-value>tiger</param-value>

</context-param>

<context-param>

<param-name>contextPath</param-name>

<param-value>d:/mega-IT/IDE</param-value>

</context-param>

String cId = getServletContext().getInitParameter("contextId");

String cPw = getServletContext().getInitParameter("contextPw");

String cPath = getServletContext().getInitParameter("contextPath");

13. 웹 어플리케이션 감시 : ServletContextListener

* 웹 어플리케이션의 생명주기를 감시하는 리스너(Listener)인 ServletContextListener가 있다.
* 리스터의 해당 메소드가 웹 어플리케이션의 시작과 종료시 호출된다(contextInitialized메소드와 contextDestroyed()메소드)
* ServletContextListener에 implements 받은 리스터 클래스 제작하고 web.xml 파일에 리스너 클래스 기술한다

**public** **class** ListenerEx **implements** ServletContextListener{

@Override

**public** **void** contextDestroyed(ServletContextEvent arg0) {

System.***out***.println("★ ★ ★ context 종료될 때 ★ ★ ★");

}

@Override

**public** **void** contextInitialized(ServletContextEvent arg0) {

System.***out***.println("★ ★ ★ context 시작될 때 ★ ★ ★");

}

}

<listener>

<listener-class>com.tj.ex2.ListenerEx</listener-class>

</listener>

Web.xml에 명시하지 않으면 리스너 클래스에 어노테이션으로 처리할 수 있다.

@WebListener

**public** **class** ListenerEx **implements** ServletContextListener {

@Override

**public** **void** contextDestroyed(ServletContextEvent arg0) {

// **TODO** Auto-generated method stub

System.***out***.println("종료될 때");

}

@Override

**public** **void** contextInitialized(ServletContextEvent arg0) {

// **TODO** Auto-generated method stub

System.***out***.println("시작될 때");

}

}

**처음 웹 어플리케이션 실행 될 때 contextInitialized()실행 -> 선처리작업 -> init()메소드 실행 -> doGet()이나 doPost() 실행**

**종료될 때 Destroye()실행 -> 후처리작업 -> contextDestroyed()실행**