**JSP 공부 정리**

전채현

**jsp 들어가기 위해서 설치해야 하는 것들**

Eclipse for Enterprise Developer (JEE - 2019-06R)

- General - Workspace - Encoding

- Web - HTML/CSS/JSP -Encoding

- General - Web Browser - Use external web brower - Chrome

- Server - Runtime Environments - Add

톰캣 설치 폴더 지정(bin이 보이는 위치)

톰캣을 구동할 jre설정해주기

===============================================

파일 설치할 때 c드라이브 밑에 폴더 만들어서 하는 것이 나중에 권한에 막힐 일이 없다

===============================================

Apache Tomcat 8.5.xx (.zip) -> 8.5에서 최신버젼

core -> zip을 다운받아야 한다

- 환경변수 설정해줘야 한다

(Web Application Server, WAS)

-> 자바로 만들어짐

- JAVA\_HOME 환경변수가 정상적으로 등록되지 않으면 실행불가

- &JAVA\_HOME%\bin 이 PATH에 등록되지 않으면 실행불가

-> 환경변수에 bin까지 넣어버리면 안된다 (path에서 %CATALINA\_HOME%\bin이렇게 추가하는 것이다)

- CATARINA\_HOME : 톰캣의 설치 경로 (apache-tomcat-8.5.xx)

- PATH에 추가 : %CATARINA\_HOME%\bin

tomcat -> bin

bat확장자는 window

sh 확장자는 linux

start.bat 을 실행하면 브라우저에서 localhost:8080으로 들어가면 tomcat으로 들어가진다

백그라운드에서 tomcat이 실행중일 수도 있으니 다음을 실행 후 재시도

C:\> taskkill /f /im java.exe

tomcat에서 버젼이 중요한 이유 다른 프로그램들과 상호작용하기 때문에 버전에 따라서 문법이 다를 수 있다

-> servlet : xml

===============================================

<%@ %> : 지시자

-> 페이지 전반에 적용되는 설정, 다른 JSP파일 포함, 다른 라이브러리 불러오기

<%! %> : 선언부

-> 멤버 필드나 멤버 메서드를 선언하거나 정의하는 영역

<% %> : 스크립틀릿

-> 메인에 해당하는 자바 코드를 작성하는 함수영역, service()

<%= %> : 표현식

-> 스크립틀릿 out.print()을 이용해 HTML문서에 자바 값을 출력하는 구문

===============================================

내장 객체 : 생성자를 통해서 생성하지 않아도, JSP페이지가 기본저긍로 참조할 수 있는 객체

입출력 : request, response, out

서블릿 : page, config

외부환경 정보 제공 : session, application, pageContext

예외 : Exception

===============================================

request 내장 객체

request : 클라이언트가 웹 서버에 전달하는 요청을 객체화한 형태

보내는 사람 : 클라이언트 (ip)

받는 사람 : 웹 서버(ip)

요청 방식 : 내가 문서를 받고 싶다면 get, 내가 어떤 내용을 담아서 보낸다면 post

요청 내용 : 받고 싶은 문서의 이름이나, 클라이언트가 보내는 내용을 받기 원하는 서버의 페이지 이름 (자원의 위치)

URL : Uniform Resource Locator (웹 브라우저의 주소창을 통해서 전달)

http://search.naver.com/search.naver?query=이지은

http://프로토콜, 서버와 클라이언트가 통신하는 규칙과 약속

search.naver.com서버 도메인(IP), 어떤 서버에게 요청을 전달하는가 (받는 사람)

search.naver문서, 페이지의 이름 (자원의 이름)

?query=이지은파라미터, ?변수이름1=변수값1&변수이름2=변수값...

클라이언트가 서버에게 전달하고자 하는 값 (내가 원하는 문서를 특정하기 위해서)

http://도서관.com/대여.html?책이름=드래곤볼&책번호=3

도서관에 가서 대여 창구에서 "책 이름은 드래곤볼이구요, 번호는 3권입니다. 이 책을 저에게 주세요(GET)"

보내는 사람 : <%=request.getRemoteAddr() %>

받는 사람 : <%=request.getLocalAddr() %>

요청 방식 : <%=request.getMethod() %>

요청 내용 : <%=request.getRequestURL() %>

자원 식별자 : <%=request.getRequestURI() %>

===============================================

**GET vs. POST**

request.getMethod()

요청 방식에 따른 분류

GET (검색?)

- 주소창을 통해서 파라미터를 전달한다

- POST에 비해서 빠르다

- 글자로 표현가능한 데이터만 전달할 수 있다(파일 전송 불가)

- 글자수에 대한 제한이 있는 경우도 있다(최근 환경에서는 사실상 길이 제한은 거의 없음)

- 용량 제한은 어느정도 있다

- 자바스크립트를 통한 주소변경(location.href = 'http://egoing.net';), <a>태그를 통한 주소변경은 모둔 GET방식의 요청이다

POST (로그인, 파일 전송 경우에 사용)

- 요청헤더의 form-data 영역에 파라미터를 담아서 전송한다(주소창을 통하지 않는다)

- 비밀번호와 같이 보안상 민감한 데이터를 다룰때에는 글자로 구성된 데이터라고 해도, POST를 사용한다

- 주소창에 노출이 되지 않을 뿐, 패킷을 열어보면 패스워드도 식별가능하다 (암호화로 보완 가능, HTTPS)

- 데이터 자체가 암호화되는 것이 아니다 개발자도구로 들어가서 보면 다 확인할 수 있다

- 파일 전송 가능(추가적인 속성을 작성해야 함)

===============================================

**useBean**

<jsp:useBean id="user" class="beans.Account"></jsp:useBean>

기본 생성자가 있어야지 -> setProperty를 생성할 수 있다.

Beans 클래스는 작성 규칙이 있다.

1. private 멤버 변수

2. public 메서드

3. 자동으로 생성되는 생성자나, 혹은 직접 작성한 기본 생성자

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. 생성자는 오버로딩을 해서 여러개를 작성할 수 있지만 \*\*\*\*\*\* 기본 생성자는 무조건 있어야 한다.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5. input태그의 name 이름과 beans의 멤버 변수 이름은 가급적 맞추도록 한다.

<jsp:useBean id="user" class="beans.Account"/>

<jsp:setProperty property="\*" name="user"/>

accs[i].equals(user.getId()) && accs[i].equals(user.getPw())

자바의 객체 비교를 위한 구현

// 내가 작성한 클래스의 객체가 같은지 비교하기 위해선

// hashCode와 equals를 overriding해서 직접 구현해야 한다.

===========================================

redirect은 로그인이 필요한 페이지가 있다 그런 페이지를 위해서 redirect가 필요함

===========================================

클라이언트가 서버에 요청을 보내면,

서버는 요청에 알맞는 응답을 생성해서 클라이언트에 전달

요청을 받은 서버가, 현재 페이지에서 요청을 처리할 수 없거나

이미 만들어진 다른 페이지나 다른 사이트에서 처리가능하면

그쪽으로 응답을 재지정하는 메세지를 전달할 수 있다

이러한 이동 방식을 redirect, forward라고 한다.

MVC 패턴 2

===========================================

**redirect, forward**

redirect는 요청정보를 다시 갱신해서 새로운 페이지를 요청하는 방식이다.

이때, 이전 요청 정보의 파라미터 값이 사라진다.

즉, redirect는 파라미터 정보가 유지되지 않는다.

응답으로 어떤 페이지에 가야하는지 알려주기 때문에

클라이언트가 어떤페이지에서 결과를 받았는지 알 수 있다.

forward : 들어온 요청정보를 서버 내부의 다른 페이지로 전달한다.

이때 요청정보를 유지한 상태에서 다른 페이지로 이동한다.

즉, forward는 파라미터 정보가 유지된다.

forward는 그대로 파라미터 정보를 전달한다.

내부에서 바뀐페이지를 클라이언트에게 알려주지 않기 때문에

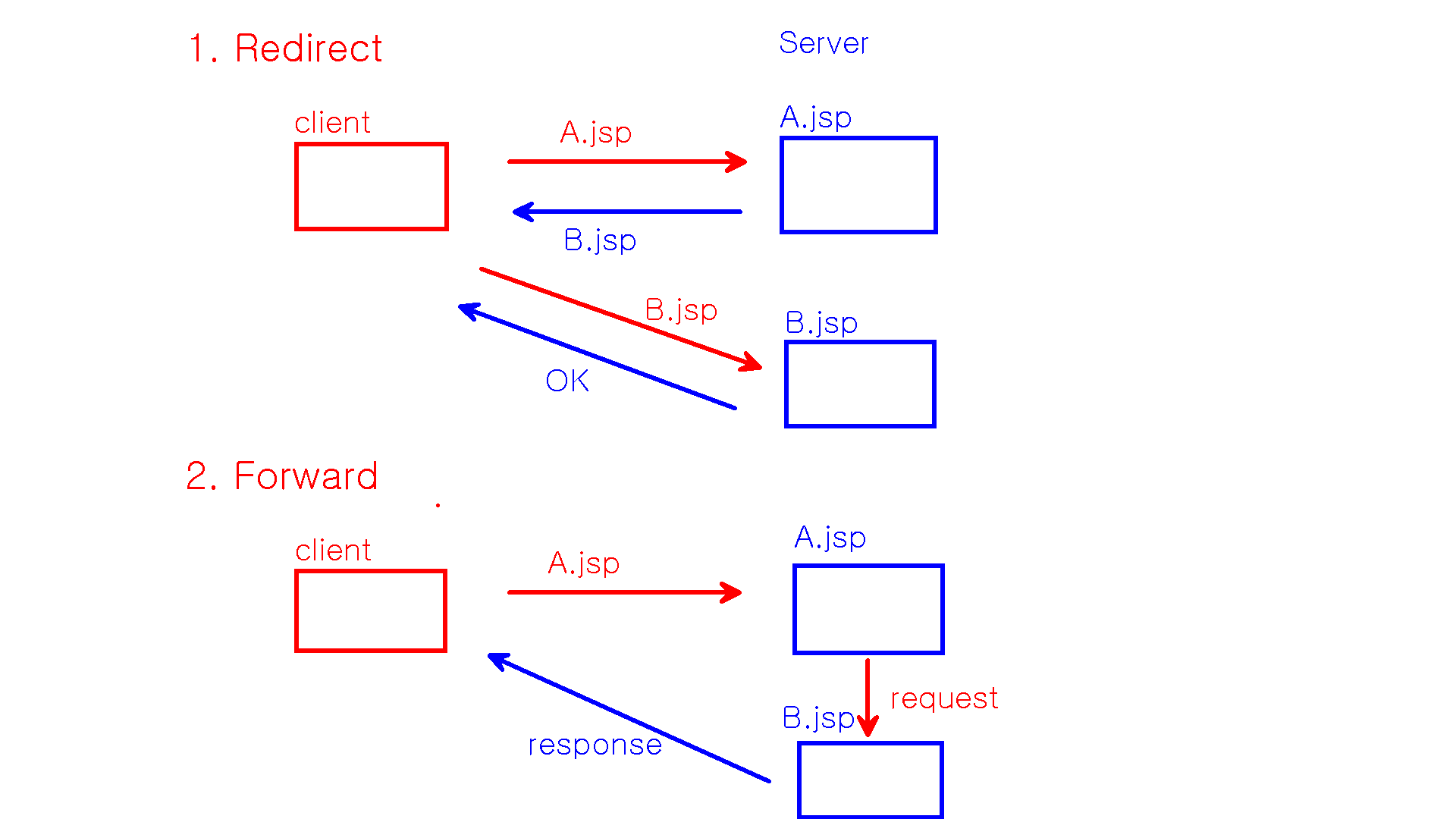
클라이언트는 실제로 어떤 페이지가 응답했는지 알 수 없다.

-> 데이터 관리할 때 사용한다.

-> 요청주소와 응답화면이 별도로 관리되어야 하는 경우(MVC2???뭐지 이제 알지 , Spring)

-> 요청정보가 유지되어서 다른 페이지로 이동되어야 하는 경우

-> 요청정보를 유지해서 전달하기 때문에 요청정보에 대한 내용을 아는 내부 사이트로만 이동 가능



**그림1. redirect vs. forward**

==========================================

redirect랑 forward개념 - 클라이언트랑 서버에서 주고 받는 관계

redirect는 클라이언트가 서버로 요청을 보내면 서버는 요청에 알맞는 정보를 전달해요

그래서 파라미터 값이 저장되지 않는 것이죠

forward는 클라이언트가 서버로 요청을 보내면 서버 내부에서 다른 페이지로 이동해서

클라이언트에게 정보를 제공하기 때문에 파라미터값이 유지돼요.

다만 클라이언트는 어떤 서버에서 요청되었는지 알 수 없어요!!

===========================================

attribute

session -> 로그인 정보 유지시킬 때

===========================================

method = "post"

한글 깨짐 시

<% request.setCharacterEncoding("UTF-8"); %>를 추가해 준다. 어디에??

===========================================

header.jsp main.jsp footer.jsp 이런 순서대로 만들어진다.

<%@ include file="header.jsp" %>

<%-- include는 다른 jsp 파일의 내용을 이 파일에 가져와서 한번에 컴파일 한다 --%>

<%-- 즉, 반복 사용되는 메뉴 같은 페이지를 불러서 재사용하는 방식이다 --%>

<main>

<div id="home">

<img src="src/main.png">

</div>

</main>

<%@ include file="footer.jsp" %>

이런 방식으로

include를 사용해서 블록을 만들어 준다. 종이를 오려쓰는 느낌

<jsp: forward> 알아보자!

<%@ include file="header"%>

<%@ include file="footer"%>

이런식으로 원하는 위치에 순서대로 입력한다.

===========================================

**paramenter, attribute**

parameter : 클라이언트가 입력한 정보를 서버에 전달

attribute : 서버에서 임의의 객체를 생성해서, jsp의 "내장객체"에 저장해 두고 필요시에 깨내쓰는 데이터

===========================================

**JSP 내장 객체 4가지**

pageContext : 페이지가 실행될 때 생성되며, 다른 페이지로 바뀌면 사라진다

request : pageContext처럼, 해당 페이지 내에서만 유효하지만 단, forward시 데이터가 유지된다

session : 웹 브라우저를 종료하기 전까지 유지된다. 주로 사용자마다 계정정보를 유지시킬 때 사용한다 ->

로그인 계속 유지되는 것

application : 웹 프로그램이 다시 시작하거나 종료도기 전까지 유지한다. 서버를 꺼지면 사라진다.

아래로 올수록 살아남는 시간이 길다

===========================================

**HTTP 프로토콜의 통신 방식**

session -> session.getAttribute();

- 정보가 서버에 저장이 된다.

- 보안성이 비교적 우수하다

- 쿠기보다 느리다

- 주로 로그인 정보를 유지하는 데 사용한다

cookie

- 정보가 클라이언트에 저장이 된다.

- 보안성이 비교적 떨어진다.

- 수명이 있다.

- 세션보다 빠르다.

- 쇼핑몰의 장바구니, 팝업창의 며칠간 띄우지 않기, 자동 로그인, 로그인 정보 유지

===========================================

**void setAttribute가 업캐스팅된다.**

업캐스팅 -> 학생 경찰, 의사를 사람이라고 부르는 느낌

하위클래스를 한번에 처리할 수 있다.

업캐스팅 문제점

- 자식클래스를 가져다 사용할 수 없다.

다운캐스팅 해야하는 이유 - 자식 클래스의 기능을 사용하기 위해서

===========================================

null -> 데이터 없다. 메모리 값이 비어있다. 객체가 없다.

===========================================

if(login != null){

session.setAttribute("login", login);

// login 객체를 세션에 저장

// setAttribute ("이름", Object value);

// 이때, 모든 객체를 받기 위해 Object로 업캐스팅 된다. -> 꺼낼 때 다운캐스팅을 해야한다.

}

response.sendRedirect("ex02.jsp");

===========================================

오늘의 수업은 세션을 활용해서 로그인을 유지하는 것

일단 헤더파일에 Login result = session.getAttribute("login");을 만들어서

login이라는 이름으로 세션들을 저장해서 관리합니다.

그리고 헤더파일에 div로 묶여 있는 로그인 표시가 로그인이 되면 로그아웃으로 변경되도록 구현할 것입니다.

<div><a href =<%=(result == null)? "login.jsp" : "logout.jsp"%>">

<%=(result==null)? "로그인" : "로그아웃"%>

</a>

</div>

이런식으로 조건식을 사용해서 간단하게 result의 값에 따라서 두가지를 구현할 수 있어요

그러면 이제 logout 파일을 만들겁니다.

로그아웃 파일은

<%

session.removeAttribute("login"); //이걸 사용해서 세션을 지워서 로그아웃 상태로 만들어버리는 거죠

%>

<script>

alert('로그아웃 되었습니다');

location.href="main.jsp";

</script>

이런식으로 로그아웃 파일을 구현해 줍니다.

그리고 로그인프로 파일에서

if(result != null){

session.setAttribute("login", result);

// 로그인이 성공하면 null이 아니기 때문에 setAttribue에 값을 저장하는 것입니다.

response.sendRedirect("main.jsp");

}

<script>

alert('로그아웃 되었습니다.');

location.href='main.jsp';

</script>

이렇게 로그인이 되었을 때와 아닐 때를 구분해 줍니다.

그 다음은 메인파일로 와서 중간에 로그인 중이 아님을 로그인이 되었을 경우 닉네임을 호출하도록 해보겠습니다.

메인파일 fieldset 위에 div를 만들어서 해볼게요

<div>

<%

if(result == null){

out.println("로그인 중이 아닙니다.");

}

else{

out.println(result.getNick() + "님 환영합니다");

}

%>

</div>

이렇게 해서 로그인이 되면 닉네임을 호출하도록 구현을 했습니다.

그리고 로그인 파일에서

<%// 주소창으로 login.jsp로 이동하면 값을 해킹할 수 있기 때문에 해주는 것이다.

// 세션이 남아있기 때문에 다시 메인페이지로 보내는 것이다.

if(result != null){

response.sendRedirect("main.jsp");

}

%>

이렇게 구현을 해줄게요

왜?? 로그인이 되어 있는 상태에서는 세션이 남아있기 때문에 login.jsp로 이동하면 값을 해킹당할 수 있기 때문에 세션이 있으면 main.jsp로 가버려 이런 것입니다.

\*\*\*\* 로그인 결과 파일에서 로그인이 성공하면

session.setAttribute("login");으로 세션을 만들어주고

로그인 창에서

String login = session.getAttribute("login"); 으로 가져오면 되는 것이다.

그리고 로그아웃 할 떄

session.removeAttribute("login"); 해주면 된다.

이렇게하면 세션을 구현할 수 있다.

===========================================

구현이 안된애들은 hashCode와 equals 를 override해서 비교를 해야 한다

HashSet도 마찬가지 이다.

===========================================

게시판도 로그인된 상태에서 글을 쓸 수 있게 한다. <= session을 이용해서 구현한다.

세션과 쿠키를 이용해서 로그인 정보

<p><input name="auto" type="checkbox" value="yes">로그인 정보 저장</p>

auto라는 파라미터로 yes면 저장

Cookie public Cookie(String name, String value)

<div class="block">Constructs a cookie with the specified name and value.

-> 쿠키는 이런 이름과 value 형태로 저장한다.

<p><input name="id" placeholder="ID" required value="<%=id %>"></p>

<p><input type="password" placeholder="PW" name="pw" required value="<%=pw %>"></p>

이런식으로 value로 로그인 폼에 id pw를 미리 채워둘 수 있다.

===========================================

코드는 비트로 조합된 부호를 말한다.

최초의 코드는 아스키코드이다.

===========================================

**JSP DB 연결**

자바코드로 DB연결 시켜야해!

이 과정을 connection이라고 해

connection은 클래스 관리해

라이브러리 파일을 자바에 넣어줘야 해

statement가 있어야 해!

connection을 다 사용하면 끊어줘야 해!

아이디 패스워드를 전송해주기 때문에 끊어줘야 해!

ojdbc 8 이라는 라이브러리를 오라클에서 다운 받아야 해요!!

ojdbc 8은 다이나믹 웹 프로젝트 에서 web-inf에 보면 lib가 보인다

거기 안에 넣어주면 프로젝트에 라이브러리를 넣어줄 수 있다.

실무로 가면 maven으로 간다

라이브러리 의존성이 높아지게 되는데 이미 완성된 것을 가지고 와서 쓰는 것이 효율적이기 때문이다.

의존성이 높아지면 라이브러리에서 설정에 따라서 실행이 되는게 있고 안되는게 있다

그래서 maven이 프로젝트 다운하면 자동으로 라이브러리를 다운시켜주는 것이다.

===========================================

항상 이렇게 많이 적을 수 없으니까

DAO를 사용함 그리고 DTO(=VO)를 클래스로 만들어서 불러서 사용하자

-> connection해주는 것.

-> sql구문은 변하지 않으니까

===========================================

**멤버변수** - 클래스의 속성, 특성을 말하는 것입니다.

===========================================

DB의 DATE 자료형은 자바에서 JAVA.UTIL.DATE 클래스를 받아서 처리 가능하다.

===========================================

**java.sql.SQLException:** 부적합한 열 이름

-> 이런 오류는

dto.setId(rs.getString("id")) 이렇게 가져오는데 "id"가 db랑 이름이 달라서 생기는 오류이다.

dto 객체에서 setid 메서드로 아이디을 가져오는 데 어디서 가지고 올거냐

rs에 저장해둔 getString으로 문자열로 가지고 오겠다. db에서 id라는 이름을 가지고 오겠다라는 의미지

===========================================

커밋을 안누르면 실제 DB에 안들어가기 때문에 항상 저장을 해야한다.

===========================================

DTO -> data transform object

날짜를 출력할 때

DTO에서 creation 자료형을 date로 줘서 구현한다.

이때 , dto.setCreation(rs.getDate("creation"));

이렇게 날짜 정보를 가지고 올때는 getDate로 가지고 온다 okokokok!

내가 한 방식은 메인 파일에

Date creation = new Date();를 가지고 와서

사용했다. 근데 뒤에 00-00-00-00이 나와서 db에서 이렇게 온다고 생각을 했었다 착각함

===========================================

ArrayList<QuizDTO> list = new ArrayList<QuizDTO>();

여기서 for -each문 말고 인덱스로 자료를 가지고 오려면

for(int i =0; i < list.size(); i++){

list.get(i).getId()

이렇게 가지고 와야 한다.

list에서 get가지고 온다 i번째 그리고 getId로 id를 가지고 온다.

for - each로 돌리려면

for(QuizDTO list : accs){

accs.getId() 이렇게 가지고 오면 된다.

===========================================

**Exception in thread "main" java.sql.SQLException:** 마지막 행 다음의 결과 집합

-> 이때 해결책은 DB에서 가져오는 아이디와 DTO에서 만든 아이디를 다르게 해준다.

SQL 구문 예외부적합한 열 이름

java.sql.SQLException: 부적합한 열 이름

-> 이렇게 뜨면

대부분 getInt나 getString에서 오타

============================================

**org.apache.jasper.JasperException: java.lang.NullPointerException**

->> 서버가 연결에서 null 값이 발견

-->> 이때 실수한건 db 아이디가 c##itbank인데 c#itbank로 입력해서 DB연결 접속 오류가 발생

===========================================

insert into 구문으로 회원가입을 구현할 것이다.

<p><input name ="id", placeholder="ID" required></p>

required가 빈칸이 없다는 것을 말하기 때문에 따로 검사할 필요가 없다.

===========================================

JDBC에서 SELECT의 결과는 테이블을 전달받는데 자바에서 ResultSet 객체에 저장 받을 수 있다.

L> executeQuery를 사용한다.

insert, delete, update 같은 구문은 실행한 결과 수를 반환하기 때문에 정수로 받아서 처리한다.

L> executeUpdate를 사용한다. 반환형이 int

ok무슨말인지 알겠다.

===========================================

String sql = "insert into account values('%s','%s','%s','%s','%s','%s','%s')";

// 회원가입 날짜를 넣을 때문 simpledateformat을 사용해서 따로 생성하지 않고 한다.

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

String time = sdf.format(new Date());

sql = String.format(sql, user.getId(), user.getPw(), user.getName(),

user.getBirth(), user.getEmail(), user.getPhone(), time);

===========================================

**PreparedStatement** -> SQLinjection을 막아줄 수 있다.

DB의 성능을 높여줄 수 있다

DBCP -> DB를 관리하는 라이브러리

화면 변화없이 특정부분만 바꾸고 싶을 때는 AJAX -> 비동기

Tomcat DBCP

===========================================

**JDBC, DBCP란?** 웹 어플리케이션의 DB접속에 대한 고찰

자바 웹 어플리케이션에서 DB 접속과 관련된 라이브러리가 JDBC, DBCP 입니다.

JDBC는 Java DataBase Connectivity 의 약자로 자바에서 데이터베이스에 연결하기 위한 인터페이스죠.

Oracle, MySQL, MsSQL 등 데이터 베이스는 JDBC를 사용하기 위한 각각의 Driver를 제공합니다.

JDBC는 이 Driver를 통해 DB에 접속을 합니다.

그럼 JDBC만 있으면 DB접속이 되는데 DBCP는 왜 쓰지? 라는 궁금증이 생기실 겁니다.

답은 '효율' 입니다.

효율적인 DB 커넥션을 위해 DBCP를 사용하는 것이죠.

그럼 DBCP가 뭔지 알아보겠습니다.

DBCP는 DataBase Connection Pool 의 약자로 DB와 커넥션을 맺고 있는 객체를 관리하는 역할을 합니다.

JDBC만을 사용할 경우라면 DB접속 시 아래와 같은 순서가 반복되게 됩니다.

===========================================

**JDBC 드라이버 로드**

1. DB 접속을 위한 JDBC 드라이버 로드

2. getConnection Method로 부터 DB 커넥션 객체를 얻음

3. 쿼리 수행을 위한 PreparedStatement 객체 생성

4. excuteQuery를 실행해서 결과를 받아옴.

여기서 비효율적인 부분은 1번과 2번 입니다.

DB 연결 시 마다 Driver를 로드하고 커넥션 객체를 얻는 작업을 반복하죠.

이 부분을 효율적으로 처리하도록 바꾸는 것이 DBCP의 역할 입니다.

DBCP를 사용하게 되면,

WAS 실행 시 미리 일정량의 DB Connection 객체를 생성하고 Pool 이라는 공간에 저장해 둡니다.

그리고 DB 연결 요청이 있으면, 이 Pool 이라는 공간에서 Connection 객체를 가져다 쓰고 반환 하게 됩니다.

DBCP를 사용하므로써 설정할 수 있는 옵션은 아래와 같습니다.

maxActive : 동시에 사용할 수 있는 최대 커넥션 개수

maxIdle : Connection Pool에 반납할 때 최대로 유지될 수 있는 커넥션 개수

minIdle : 최소한으로 유지할 커넥션 개수

initialSize : 최소로 getConnection() Method를 통해 커넥션 풀에 채워 넣을 커넥션 개수

===========================================

**Statement 와 PreparedStatement**

SQL 을 실행할 수 있는 객체입니다.

Statement

1. 단일로 사용될 때 빠른 속도를 지닙니다.

2. 쿼리에 인자를 부여할 수 없습니다.

3. 매번 컴파일을 수행해야 합니다.

PreparedStatement

1. 쿼리에 인자를 부여할 수 있습니다.

2. 처음 프리컴파일 된 후, 이후에는 컴파일을 수행하지 않습니다.

3. 여러번 수행될 때 빠른 속도를 지닙니다.

===========================================

**JDBC, DBCP : 자바 DB접속과 관련된 라이브러리**

JDBC : java database connectivity의 약자로, DB에 접급하기 위한 라이브러리

그럼 JDBC만 쓰면 접속 되는데 DBCP는 왜 쓰는거지>>?? 효율성 때문에

DBCP : DB connection Pooling, DB의 Connection을 관리하는 라이브러리

JDBC만 사용하는 경우 아래의 루틴을 반복한다.

1. JDBC 드라이버 로딩을 한다.

2. DB 접속을 위해서 Connection을 생성한다.

3. Connection을 통해서 statement를 생성한다.

4. 실행후 Connection을 닫음

여기서, 비효율적인 구간은 1,2,4 -> 매번 드라이버 로딩 후 커넥션 생성 후 제거

이를 효율적으로 바꾸는 것이 DBCP가 된다.

DBCP 사용시

WAS(=tomcat)이 미리 일정량의 Connection을 미리 생성하고 Pool이라는 공간에 저장한다.

DB연결 요청이 발생시 (ex, 로그인, 회원가입, 게시판 글 작성 및 조회 등등)

Pool이라는 공간에 Connection 객체를 가져다 쓰고, 다시 Pool에 반납하는 과정을 가진다.

이게 사람이 많을수록 효율적이다.

===========================================

**EL과 JSTL**

EL 표현식 -> 출력하기 위한 방식이다.

${param.이름} : 파라미터

${이름}: Attribute

${\*Scope.이름}: 지정 Attribute

EL 태그는 파라미터의 형변환을 알아서 해주고, null값을 출력하지 않는다.

이름 : ${param.name } ${param.파라미터 명}

나이 : <${param.age> } 자동으로 파라미터의 자료형을 맞춰준다

나이 + 12 : <${param.age> + 12 }

EL은 멤버 변수를 바로 접근 가능한데, getter가 있어서 가능한 것

${list.get(1).name }

${list[1].name }

list는 원래 get()메서드로 index에 접근하는데 ,EL은 []도 지원한다.

${map.get("df").name }

${map["df"].name }

===========================================

**JSTL**

jsp는 자바코드와 html 코드가 섞여있는 구조로 복잡한 구조로 이루어짐

이때, jsp에서 자주 사용되는 자바코드를 태그로 만들어서 사용가능하다.

커스텀 태그 : JSP에서 기본 제공되는 태그가 아니라, 자신이 직접 추가한 태크

커스템 태크 라이브러리 : 잘 만들어진 커스텀 태그를 모아서, 압축 후 배포한 형태이다

다른 개발자들은 이를 받아서 사용할 수 있다.

JSTL : 커스텀 태그를 개별적으로 작성되다보니, 일관성이 없더라

이를 표준화한 것을 JSTL이라고 한다.

===========================================

**커스텀태그설명**

핵심 기능(core)일반 프로그래밍 언어에서 제공하는 변수 선언 및 제어문을 제공한다

그 외, redirect같은 웹에서의 이동 기능도 있다

핵심 기능만 많이 사용한다

형식화(format)숫자, 날짜, 시간의 형식을 지정하는데 사용

데이터베이스(sql)데이터베이스에 입력/수정/삭제/조회 기능을 제공한다

XMLXML 문서 처리를 위한 기능 제공한다

함수(functions)문자열을 처리하거나, 기타 유용한 함수를 제공한다

core 기능

c:set으로 Attribute를 추가한다

scope 지정하지 않으면 기본 pageContext에 저장된다

<c:set var="p1" value="홍길동"/>

pageContext.setAttribute("p1", "홍길동");

<c:set var = "p2" value="김길동" scope="page"/>

pageContext나 pageScope로 작성하는 것이 아니니 주의

<c:set var = "p3" value="이길동" scope="request"/>

request.setAttribute("p1", "홍길동");

<c:set var = "p4" value="박길동" scope="session"/>

session.setAttribute("p1", "홍길동");

<c:set var = "p5" value="최길동" scope="application"/>

application.setAttribute("p1", "홍길동");

이런식으로 <c:set> </c:set> 이렇게 해서 사용한다

다만 이걸사용하려면 taglib를 해줘야 한다.

===========================================

**EL과 JSTL로 게시판 만들기**

IF-ELSE 구문 만들기

<c:if test="${not empty param.age }">

<c:choose> <%-- choose는 내부 when 조건을 가지고 판별한다 --%>

<c:when test="${param.age > 20 }">

<h2>성인입니다.</h2>

</c:when>

<c:when test="${param.age >= 17 }">

<h2>고등학생입니다</h2>

</c:when>

<c:when test="${param.age >= 14 }">

<h2>중학생입니다.</h2>

</c:when>

<c:when test="${param.age >= 8 }">

<h2>초등학생입니다.</h2>

</c:when>

<c:otherwise>

<h2>미취학 아동입니다.</h2>

</c:otherwise>

</c:choose>

</c:if>

===========================================

EL 태그는 Attribute에 저장되어 있어서 스크립틀릿안에 있는 것을 가지고 올 수 없다.

그래서 스크립틀릿 안에서 변환을 해줘야 사용이 가능하다.

===========================================

**eclipse vs. intellij**

eclipse

장점

- 무료다!

- 여러 프로젝트를 동시에 관리할 수 있다.

단점

- 다른 IDE 툴들에 비해 느림, 하지만 하드웨어의 비약적인 발달로 인해 엄청많이 차이가 나지는 않음.

- 자바로 만들어져서 자바가상머신(JVM) 위에서 실행되므로 다른 IDE에 비교해서 속도가 느리다.

- 지속적인 패치에도 불구하고 자잘한 버그들이 있음

- C/C++, PHP, Rust, Groovy, Scala, Python 등 다양한 언어를 지향한다고 되어 있지만 JAVA를 제외한 언어는 개발하기 불편

intellij

장점

- 개발의 질을 한층 높여줄수 있는 플러그인을 간편하게 설치할 수 있음

- 직관적이고 예쁜 UI

- 다양한 벡엔드&프론트엔드 프로젝트 생성

- 우수한 스마트 코드 작성 기능들 덕분에 개발자의 생산성을 대폭 높여준다.

- 자동완성(auto completion), 소스코드분석, 지능형리펙토링, 디버거, 테스트러너 기능이 이클립스와 비교하여 우수하다.

- 형상관리를 git으로 하고있다면 intelliJ가 편하다.

단점

- 단축키, 툴적응에 약간의 시간이 필요하다

- eclipse로 주로 이루어지는 프로젝트 개발환경과 구조와 맞지 않는 경우 존재

- 유료다.. 결제 금액도 기업이 아닌 개인결제 기준으로 1년에 149달러, 풀패키지는 249달러로 꽤 비싸다

기능이 제한된 무료 버전(커뮤니티 버전)이 있지만, 유료인 Ultimate 버전을 사용해야 인텔리제이의 장점을 살릴 수 있다

===========================================

**maven vs. gradle**

1. 빌드 관리 도구란?

- 우리가 프로젝트에서 작성한 java 코드와 프로젝트 내에 필요한

각종 xml, properties, jar 파일들을 JVM이나 WAS가 인식할 수 있도록 패키징 해주는 빌드 과정 == "빌드 자동화 도구" 라고 할 수 있다!

- 프로젝트 생성, 테스트 빌드, 배포 등의 작업을 위한 전용 프로그램이라 할 수 있다.

- 애플리케이션을 개발하면서, 일밙겅르로 개발에 필요한 다양한 외부 라이브러리들을 다운로드하고,

해당 라이브러리를 사용하여 개발해야하는 상황이 많다.

이 때 각 라이브러리들을 번거롭게 모두 다운받을 필요없이,

빌드도구 설정파일에 필요한 라이브러리 종류와 버전들, 종속성 정보를 명시하여 필요한 라이브러리들을

설정파일을 통해 자동으로 다운로드 해주고 이를 간편히 관리해주는 도구이다.

===========================================

**Maven???**

- Maven은 Java용 프로젝트 관리도구로 Apache의 Ant 대안으로 만들어졌다.

- 빌드 중인 프로젝트, 빌드 순서, 다양한 외부 라이브러리 종속성 관계를 pom.xml파일에 명시한다.

- Maven은 외부저장소에서 필요한 라이브러리와 플러그인들을 다운로드 한다음, 로컬시스템의 캐시에 모두 저장한다.

- 예를 들어, "Spring Boot Data JPA Starter" 모듈을 사용하여 프로젝트를 개발하고 싶다면

메이븐레포지토리에서 해당 모듈을 검색하여 xml 설정파일에 추가하여 사용할 수 있다.

(참고로 maven용 코드 이외에도 gradle용 코드도 탭에 있다.)

===========================================

**gradle??**

- Apacahe Maven과 Apache Ant에서 볼수 있는 개념들을 사용하는 대안으로써 나온 프로젝트 빌드 관리 툴이다. (완전한 오픈소스)

- Groovy 언어를 사용한 Domain-specific-language를 사용한다. (설정파일을 xml파일을 사용하는 Maven보다 코드가 훨씬 간결하다.)

- 2007년에 처음 개발되었고, 2013년에 구글에 의해 안드로이드 프로젝트의 빌드 시스템으로 채택되었다.

- 꽤 큰규모로 예상되는 multi-project 빌드를 도울 수 있도록 디자인되었다.

- Gradle은 프로젝트의 어느부분이 업데이트되었는지 알기 때문에, 빌드에 점진적으로 추가할 수 있다.

-> 업데이트가 이미 반영된 빌드의 부분은 즉 더이상 재실행되지 않는다. (따라서 빌드 시간이 훨씬 단축될 수 있다!)

===========================================

**spring boot**

maven 과 gradle은 의존성 관계

실무에서는 기록을 남기기 위해서 logging로 출력을 해야한다.

**로깅과 관련된 두가지**

**logback**

**slf4j -> interface**

테스트할 때 junit을 사용한다.

(assertj 테스트를 도와주는 것)

Gradle은 의존관계가 있는 라이브러리를 함께 다운로드한다.

**스프링 부트 라이브러리**

- spring boot starter-web

-> spring boot starter-tomcat : 톰캣

-> spring webmvc : 스프링 웹 MVC

- spring boot starter thymeleaf : 타임리프 템플릿 엔진(view)

- spring boot starter (공통) : 스프링부트 + 스프링코어 + 로깅

-> spring boot

--->spring core

-> spring boot starter logging

---> logback, slf4j

**테스트 라이브러리**

- spring boot starter-test

-> junit : 테스트 프레임워크

-> mockito : 목 라이브러리

-> assertj : 테스트 코드를 좀 더 편하게 작성하게 도와주는 라이브러리

-> spring-test : 스프링 통합 테스트 지원

===============================================

**Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.**

->> 그 파일 때문에 에러난게 아니라,

아마 pom.xml에 spring-boot-starter-data-jpa 라는 의존성을 넣었는데, DB관련 의존성을 넣지 않아서 에러가 난거 같습니다.

spring-boot-starter-data-jpa 의존성을 삭제하거나,

위 메시지에서 제시하는대로 임베디드 DB 라이브러리 중 아무거나 하나 추가하시면 저 에러는 사라질 거에요.

===============================================

**Web server failed to start. Port 8080 was already in use.**

이 경우 지금 8080포트가 켜져 있어서 실행이 안되는 것이다

그래서 cmd로 들어가서 netstat -ano 명령어로 8080의 pid 를 알아낸다.

그리고 taskkill /pid ???? /f 명령어를 입력해서 포트를 닫아준다.

===============================================

**[ attribute란? : 메소드를 통해 저장되고 관리되는 데이터 ]**

PageContext / Request에서 사용될때

setAttribute("key", value) → 값을 넣는다.

getAttribute("key") → 값을 가져온다.

removeAttribue("key") → 값을 지운다.

session에서 사용될때

set / get / remove 동일하고 추가로++

invalidate( ) → 값을 전부 지운다

===============================================

JSTL 안에서는 javascript가 사용이 불가하지만 javascript안에서는 JSTL 사용이 가능하다는 것이다.

===============================================

**VO( Value Object )**

Value Object는 DTO와 동일한 개념이나 차이점은 read only 속성을 갖는다.

Value Object는 관계 데이터 베이스의 레코드에 대응되는 자바 클래스이다.

형태는 db 레코드를 구성하는 필드들을 Value Object의 Attribute로 하고

해당 변수에 접근할 수 있는 Getter, Setter 메소드의 조합으로 클래스가 형성되어진 클래스이다.

특성은 대체로 불변성이고 equlas()로 비교할 때 객체의 모든 값을 비교해야 한다.

프로그램의 사용자가 화면에서 어떤 데이터를 입력하거나 조회 요청이 왔을 때

입력된 데이터나 조회하는 조건을 VO에 담아서 DAO에 요청하면 DAO는 저장소(일반적으로 Database )로부터

데이터를 입력하거나 조회한 후 그 결과를 돌려주게 된다.

VO는 간단한 독립체( Entity )를 의미하는 작은 객체를 의미한다.

VO의 같음은 그 정체성에 의해 결정되지 않는데,

그 뜻은 두 VO 들은 그 두 가지가 같은 값을 같고 있을 때 같은 것이지 같은 객체라고 해서 같지 않다는 것이다.

작기 때문에, 같은 독립체를 대변하는 복수의 같은 VO들이 존재할 수 있다.

가끔 하나의 인스턴스에 의존하고 그에 기반한 레퍼런스를 사용하기 보다는 새 객체를 생성하는 것이 더 간편하다.

필요성

Network traffic을 줄임으로 인해 효과적이다.

기대효과

Network traffic이 줄어든다.

장단점

장점은 비 서버 측 클라이언트도 네트워크 오버헤드 없이 영속성 데이터에 액세스 할 수 있다는 점이다.

데티어 전달을 위해 가장 효율적인 방법이지만, 클래스의 선언을 위해 많은 코드가 필요하다.

즉 파일수가 많아지게 되고 관리도 힘들어진다.

===============================================

**DTO( Data Transfer Object : 데이터 전송 객체 )**

데이터가 포함된 객체를 한 시스템에서 다른 시스템으로 전달하는 작업을 처리하는 개체이다.

Data에 접속하는 객체이다. 여기서 Data란 일반적인 Database도 될 수 있고, 파일도 될 수 있으며, 메모리도 될 수 있고, 기타 다른 저장소도 될 수 있다.

DTO는 프로세스 사이에서 데이터를 전송하는 객체를 의미한다. 이것은 이용하는 이유는 프로세스 간의 커뮤니케이션이 주로 개별 호출이 부담스러운 작업일 경우가 많은 원격 인터페이스(예:웹 서비스)에 의해 이루어지기 때문이다.

대부분의 개별 호출이 클라이언트와 서버 간의 왕복 시간을 소모하기 때문에, 호출 횟수를 줄이는 방법 중 하나는 몇 번의 호출에 의해 전송될 데이터를 모으는 DTO를 이용해서 한번만 호출하게 하는 것이기 때문이다.

VO와 DTO의 비교

DTO의 나머지 속성은 VO와 똑같다고 생각된다.

Core J2EE Patterns라는 책에서는 Value Object와 Transfer Object를 동일한 뜻으로 사용하지만, 반대로 Martin Fowler는 저서 Patterns of Enterprise Applicatino Architecture에서 약간 다른 의미로 이야기 한다.

DTO는 메소드 호출 횟수를 줄이기 위해 데이터를 담고 있는 녀석으로, VO는 값이 같으면 동일 오브젝트라고 볼 수 있는 것으로 표현하고 있다.

DTO a = new DTO(1);

DTO b = new DTO(1);

이라고 했을 때, a != b 이지만,

VO a = VO(1);

VO b = VO(1);

이라고 했을 때는 a == b 라고 정의하는 형태이다.

대부분의 검색에서 사람들은 VO와 DTO를 같은 개념으로 이야기 하고 있어서,

아직도 VO와 DTO가 "이런거다"라기 보다 거의 똑같은 개념으로 생각하고 있다.

===============================================

**DAO( Data Access Object : 데이터 접근 객체 )**

데이터 접근을 목적하는 객체이다.

커넥션 같은 것을 하나만 두고 여러 사용자가 DAO의 인터페이스를 사용하여 필요한 자료에 접근 하도록 하는 것이 DAO의 개념이다.

DB에 대한 insert, update, delete, select를 처리한다.

DAO는 특정 타입의 데이터베이스나 다른 지속적인 메커니즘( Persistence Mechanism )에 추상 인터페이스를 제공하는 객체이다.

어플리케이션 호출을 데이터 저장 부분( Persistence Layer )에 매핑함으로써

DAO는 데이터베이스의 세부 내용을 노출하지 않고 특정 데이터 조작 기능을 제공한다.

이 고립성은 단일 책임 원칙( Single Responsibilty Principle )에 기반한다.

이 패턴은 대부분의 프로그래밍 언어에, 영속성이 필요한 대부분의 소프트웨어에, 대부분의 데이터베이스에 적용될 수 있다.

단일 책임 원칙 : 객체 지향 프로그래밍에서 모든 컨텍스트( 클래스, 기능, 변수 등 )은 하나의 책임만 가져야 한다는 것이며,

이는 컨텍스트에 의해 완전히 캡슐화 되어야 한다는 것이다. 그리고 모든 서비스들은 해당 책임에 맞춰 조정되어야 한다.

다른 말로는 "클래스를 수정해야 할 이유는 오직 하나여야 한다"고 한다.

필요성

모든 데이터베이스에 공통적으로 접속 할 수 있는 ODBC가 나왔지만 여전히 로우 레벨의 API를 포함하고 있었기 때문에 개발 장벽이 높았다.

이런 이유 때문에 개발자들은 정작 데이터베이스에 들어 있는 데이터를 어떻게 이용할지에 초점을 맞추기 보다,

어떻게 데이터베이스에 접속해서 데이터베이스와 교류하는지에 더 초점을 기울였다.

즉, 데이터를 활용하는 논리적 고민보다 기술적 고민에 더 많은 신경을 썼었다. 이런 이유로 DAO란 대안이 나오게 되었다.

기대 효과

사용자는 자신이 필요한 interface를 DAO에게 던지고 DAO는 이 인터페이스를 구현한 객체를 사용자에게 편리하게 사용 할 수 있도록 반환해준다.

장단점

DB에 대한 접근을 DAO가 담당하도록 하여 데이터베이스 액세스를 DAO에서만 하게 되면

다수의 원격 호출을 통한 오버헤더를 VO나 DTO를 통해 줄일 수 있고, 다수의 DB 호출 문제를 해결할 수 있다.

또한 단순히 읽기만 하는 연산이므로 트랜젝션 간의 오버헤드를 감소할 수 있다.

그러나 Persistent Storage를 너무 밀접하게 결합해서 작성을 하게 되면 Persistent Storage를 다시 작성할 경우가 생기는데 이러한 경우 유지 보수의 문제가 생길 수도 있다.

===============================================

**페이징 알고리즘 아이디어**

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<div class="paging">

<div class="element">

<c:set var="page" value="${currentpage}"/>

<!-- page maxpage를 넘었을 경우 제한 -->

<c:if test="${page > maxpage}">

<c:set var="page" value="${maxpage}"/>

</c:if>

<!-- 페이지를 8개씩 나누기 위해 현재 페이지에 보여줄 max값 설정 -->

<fmt:formatNumber value="${page/8 - (page/8 % 1)}" type="pattern" pattern="0" var="full"/>

<c:set var="full" value="${full \* 8}"/>

<!-- prev(전페이지), next(다음페이지) 개수 설정 -->

<c:set var="scope" value="${page%8}"/>

<c:choose>

<c:when test="${scope == 0}">

<c:set var="prev" value="8"/>

<c:set var="next" value="1"/>

</c:when>

<c:when test="${scope < 9}">

<c:set var="prev" value="${scope}"/>

<c:set var="next" value="${9-scope}"/>

</c:when>

</c:choose>

<!-- prev 버튼 -->

<c:if test="${page > 8}">

<fmt:formatNumber value="${(page-1)/8 - (((page-1)/8) % 1)}" type="pattern" pattern="0" var="prevb"/>

<c:set value="${(prevb-1)\*8 + 1}" var="prevb"/>

<span id="prevBtn" class="prev button" value="${prevb}"></span>

</c:if>

<!-- 전 페이지 -->

<c:if test="${page != 1}">

<c:set var="j" value="${prev}"/>

<c:forEach var="i" begin="1" end="${prev-1}">

<c:set var="j" value="${j - 1}"/>

<c:if test="${(page - j) > 0}">

<span class="no">${page - j}</span>

</c:if>

</c:forEach>

</c:if>

<!-- 현재 페이지 -->

<span class="no selected">${page}</span>

<!-- 다음 페이지 -->

<c:if test="${page != maxpage}">

<c:forEach var="i" begin="1" end="${next-1}">

<c:if test="${maxpage >= page+i}">

<span class="no">${page+i}</span>

</c:if>

</c:forEach>

</c:if>

<!-- next 버튼 -->

<c:if test="${maxpage >= page + next}">

<fmt:formatNumber value="${(page-1)/8 - (((page-1)/8) % 1)}" type="pattern" pattern="0" var="nextb"/>

<c:set value="${(nextb+1)\*8 + 1}" var="nextb"/>

<span id="nextBtn" class="next button" value="${nextb}"></span>

</c:if>

</div>

</div>

===============================================

**페이징 알고리즘 아이디어**

<c:set var="page" value="${(param.p == null) ? 1 : param.p}"/>

<c:set var="startNum" value="${page - (page-1) % 5 }"/>

<c:set var="lastNum" value="23"/>

<c:set var="isLast" value="4" />

<div>

<c:if test="${startNum >= 6 }">

<a href="?p=${startNum - 5 }&t=&q=" class="btn btn-prev">이전</a>

</c:if>

<!--<c:if test="${startNum == 0}">

<span class="btn btn-prev" onclick="alert('이전 페이지가 없습니다.');">이전</span>

</c:if>

-->

</div>

<%-- <!-- 직전 -1 으로 보내는 경우 -->

<div>

<c:if test="${startNum - 1 > 0 }">

<a href="?p=${startNum - 1 }&t=&q=" class="btn btn-prev">이전</a>

</c:if>

<c:if test="${startNum <= 0}">

<span class="btn btn-prev" onclick="alert('이전 페이지가 없습니다.');">이전</span>

</c:if>

</div> --%>

<ul class="-list- center">

<c:if test="${startNum + 5 >= lastNum }">

<c:set var="isLast" value="${lastNum - startNum }" />

</c:if>

<c:forEach var="i" begin="0" end="${isLast }">

<li><a class="-text- orange bold" href="?p=${startNum + i}&t=&q=" >${startNum + i}</a></li>

</c:forEach>

</ul>

<div>

<!-- 마지막번호는 DB레코드 수와 연관이 있다!

우선은 lastpage 개수를 임의로 지정. -->

<c:if test="${startNum + 5 < lastNum }">

<a href="?p=${startNum + 5}&t=&q=" class="btn btn-next">다음</a>

</c:if>

<%-- <c:if test="${startNum + 5 >= lastNum }">

<span class="btn btn-next" onclick="alert('다음 페이지가 없습니다.');">다음</span>

</c:if> --%>

</div>

===============================================

**다수의 파라미터 한번에 처리하는 방법**

처리만 담당

Ex01DTO의 객체를 만든다

Ex01DTO dto = new Ex01DTO();

dto 참조변수에 request.getparameter로 입력받은 값을 넣어준다

dto.setAddress(request.getParameter("address"));

dto.setAge(request.getParameter("age"));

dto.setEmail(request.getParameter("email"));

dto.setId(request.getParameter("id"));

dto.setPw(request.getParameter("pw"));

getparameteMap을 사용해서 key : value형태로 출력

Map<String, String[]> map = request.getParameterMap();

Set<String> keyset = map.keySet();

for(String key : keyset){

String value = map.get(key)[0];

out.print(key + ":" + value);

}

getparameterName을 사용해서 key : value형태로 출력

Enumeration도 String 형식

Enumeration<String> enu =request.getParameterNames();

while(enu.hasMoreElements()){// while문을 돌면서

String key = enu.nextElement();// 키를 가져오고

String value = request.getParameter(key);// request.parameter로 key의 value를 요청한다

out.print(key + ": " + value);

}

id<%=dto.getId() %>

pw<%=dto.getPw() %>

age<%=dto.getAge() %>

address<%=dto.getAddress() %>

email<%=dto.getEmail() %>

자바빈 사용 파라미터의 위치를 class로 확인하고 id의 이름으로 가져온다

자바빈으로 dto객체를 생성

setproperty에서 property로 모든 값을 가져온다고 지정하고 name이 dto2인 녀석들의 모든 값을 가져오겠다

<jsp:useBean id="dto2" class="day2.Ex01DTO"/>

<jsp:setProperty property="\*" name="dto2"/>

===============================================

객체 생성 (기본생성자)

<jsp:useBean id="dto2" class="day02.Ex01DTO" />

클래스가 가지는 setter와 input의 name이 일치한다면 파라미터를 받아서 객체에 대입

<jsp:setProperty property="\*" name="dto2" />

===============================================

**여러개 checkbox**

<fieldset>

<legend>설문</legend>

<form>

<input type="checkbox" value="네트워크" name="part">네트워크

<input type="checkbox" value="웹 (프런트)" name="part">웹 (프런트)

<input type="checkbox" value="웹(백엔드)" name="part">웹(백엔드)

<input type="checkbox" value="시스템" name="part">시스템

<input type="checkbox" value="앱개발" name="part">앱개발

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

</form>

</fieldset>

checkbox같은 경우 name은 동일한데 값이 여러개인 경우가 있다

이럴때는 배열의 형태로 처리해서 배열을 출력하도록 하면 된다

이때 주의할 점은 아무것도 입력을 안할 경우도 있으니 part != null 이라는 조건을 걸어주는 것이 좋다

arr.length가 0이면 NPE가 발생한다

배열의 형태로 사용하려면 request.getParameterValues를 사용해야한다

String [] arr = request.getParameterValues("part");

out.print("희망 분야 : ");

for(int i =0; part != null && i < arr.length; i++){

out.print(arr[i]);

out.print(i== arr.length-1 ? "<br>" : ",");

}

===============================================

**rdirectURL vs. forwardURL**

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

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

redirectURL이 null이 아니라면 response 보내라 sendRedirect(파라미터로 입력 받아온 주소로)

if(redirectURL != null){

response.sendRedirect(redirectURL);

}

forwardURI가 null이 아니라면 일단 post로 받았으니까 인코딩해주고

RequestDispatcher를 불러오는데 이건 request.getRequestDispatcher메서드를 사용해서 forwardURI 파라미터를 입력받아서 객체화 한다

그리고 만들어 두었던 RequestDispatcher의 참조변수인 rd의 forwar메서드를 이용해서 화면을 전환시켜주는 것이다

if(forwardURI != null){

request.setCharacterEncoding("UTF-8");

이동하고 싶은 주소를 전달하면서 요청을 지속적으로 따라갈 수 있는 객체를 받는다

RequestDispatcher rd = request.getRequestDispatcher(forwardURI);

여기서 요청받은 request를 이동한 페이지에서도 그대로 사용할 수 있도록 request를 같이 전달한다

rd.forward(request, response);

}

===============================================

**request.setAttribute(String name, Object o)**

- 지정한 name으로 객체 o를 attribute로 등록한다. 모든 유형의 객체를 저장할 수 있다

request.getAttribute(String name)

- 지정한 name으로 객체를 찾아서 반환한다. 반환되는 타입은 Object이며, 필요하다면 형변환(Casting)을 해줘야 한다

(반환형이 Object이다보니까 상위 이기 때문에 받으려는 타입에 맞게 형변환을 해줘야 한다)

- 만약 지정한 name의 객체가 없으면 null을 반환한다

--> 서로 다른 JSP페이지 간의 원할한 객체 전달을 위해서이다

--> Attribute는 페이지 간의 데이터를 주고 받기 위한 임시 저장소이다

===============================================

**pageContext** : jsp 페이지마다 생성되는 내장 객체이다

- 다른 내장객체에 접근하기 위해서 활용할 수 있다

- attribute를 저장할 수 있다

- <jsp:useBean id="join" class="day03.MemberDTO"/>태그로 객체를 생성하면 기본적으로 pageContext에 객체를 저장한다

-> 왜냐하면 scope의 기본값이 pageContext이기 때문에 객체를 생성하면 pageContext에 객체가 저장되는 것이다

ServletRequest myRequest = pageContext.getRequest();

String test1 = "내장객체 request와 pageContext에서 불러낸 myRequest는 동일한 객체인가?";

System.out.println(test1 + (myRequest == request));

HttpSession mySession = pageContext.getSession();

String test2 = "내장객체 session과 pageContext에서 불러낸 mySession이 동일한 객체인가?";

System.out.println(test2 + (mySession == session));

ServletContext ctx = pageContext.getServletContext();

String test3 = "내장 객체 application과 pageContext에서 불러낸 ctx가 동일한 객체인가?";

System.out.println(test3 + (ctx == application));

JspWriter myOut = pageContext.getOut();

myOut.println("<h4>여기에 작성하면 문서에 기록 된다 </h4>");

===============================================

**useBean -> setProperty**

<jsp:useBean id="join" class="day03.MemberDTO"/>

<jsp:setProperty property="\*" name="join"/>

이렇게 useBean으로 객체를 생성하고 setProperty로 클래스의 setter와 input의 name이 일치한다면 파라미터를 객체에 삽입하는 과정을 거치면

page객체에 파라미터의 값이 들어가 있는 상태이다

그럼

<%= ((MemberDTO)pageContext.getAttribute("join")).getUserid()%>

-> 이런식으로 표현식을 사용해서 getUserid를 가져올 수 있지만

-> EL태그를 사용하면 더 쉽게 값을 가져올 수 있다

${ join.userid } 이렇게 해주면 join객체 안에 있는 getUserid 값을 문자열로 가져온다

-> ${} 로 접근하는 join은 스크립틀릿 안에 지역변수가 아니라 attribute안에 저장된 객체를 참조하는 것이다

-> 주의 (${} 를 주석에 넣으면 이걸 인식해서 500대 오류를 발생시킨다 주의할 것)

-> EL Tag는 값이 null이면 null이라고 출력하지 않는다

-> ${ emtpy login ? '' : '로그아웃'} 이런식으로 해당 객체의 유무로 삼항연산자를 사용할 수 있다

===============================================

**EL Tag 정리하기**

1. 내장 객체의 attribute에 접근할 수 있다

- attribute에 접근하여, 내부 멤버 요소를 참조할 때, getter 형식은 생략한다

- 별도의 형 변환이 필요없다

- pageContext -> request -> session -> application 순서로 탐색한다

- 특정 scope의 attribute로 지정하여 접근할 수 있다

- 만약, attribute의 자료형이 Collection이면, 내부 요소에도 접근이 가능하다

2. 파라미터에도 접근하여 출력할 수 있다

- ${ param.name} 으로 바로 사용자가 입력한 name으로 접근할 수 있다 // param의 값을 주소에서 변경해도 값을 가져올 수 있다

3. 연산자를 사용할 수 있다

- 산술연산자 : +, -, \*, /, %

- 비교 연산자 : >, <, >=, <=, ==, !=

- null 여부 : empty(== null), not empty(!= null)

- EL Tag 내부에서는 + 연산이 무조건 숫자에 대한 연산으로 취급된다

- 문자열 상수는 ' 로 묶어준다

4. EL Tag에서 pageContext를 출발점으로, 다른 내장객체에 접근이 가능하다

===============================================

**기능 구현 작업 순서**

네이버의 회원가입과 로그인 기능을 모방하여 페이지를 구현하세요

1. 메인페이지(로그인 및 가입 링크)

2. 회원가입 페이지 (약관 동의 생략) -> (가입 성공 ? 로그인 : 회원가입 페이지)

3. 로그인 페이지

- 회원 정보를 저장할 수 있는 ArrayList 를 application에 attribute로 만들어야 한다

- 각 회원 정보는 DTO 클래스를 통해서 구성하며, userid, userpw, username, year, month, date, gender

(O)

- 로그인에 성공하면 회원 정보가 담긴 DTO객체를 session에 attribute로 저장한다

(O)

- 로그인 상태에서는 로그인 폼 대신 , 로그 아웃 메뉴 링크가 있어야 한다

(O)

- 로그아웃 클릭하면 <% session.invalidate(); %> 코드로 로그아웃 처리한다

(O)

- css 가능한 만큼만 진행

작업 순서

1. 데이터 모델링 - 내가 프로그램에서 어떤 형태의 데이터를 다룰 것인가

(MemberDTO userid, userpw, uswrname, year, month, date, gender)

(O)

2. 화면 구현 - 프로그램에서 사용될 주요 기능을 메뉴형태로 배치(회원 가입과 로그인)

(O)

3. 기능 구현 - 가입은 (insert) , 로그인(select)

4. 페이지 간 연결 확인 및 예외 처리

- jsp:useBean, jsp:setProperty, EL Tag를 적극 활용하세요

- 처리는 자바코드(스크립틀릿)으로 진행하고, 출력은 EL Tag나 표현식을 활용하세요

- 회원가입 form의 모든 데이터는 반드시 입력해야 하는 요소입니다(not null, required)

===============================================

**응답코드**

200OK (요청의 성공)

300

301이름이 바뀌거나, 경로가 바뀌어서 찾을 수 없다

400클라이언트 에러

401인증 X

403접근 권리 X

404요청받은 리소스를 찾을 수 없다

405

415

500서버문제(서버 내부 문제) -> JSP나 자바코드가 잘못된 것

요청받은 문서가 존재하면 해당 문서를 컴파일 -> HTML텍스트 파일 형태로 보내준다

(.jsp -> .java -> .class 로 해서 다시 컴파일 해서 HTML + jsp로 해서 클라쪽으로 보내준다)

===============================================

새 파일에 문제파일의 코드를 한줄 한줄 붙여넣기해서 어떤 문장이 문제가 되는지 찾아봤는데 결국

"<!-- ${ } = ~~~~ --> " 라고 수업내용필기를 위해 주석 처리한 문장 속

${ }가 문제였다.

주석이 문제가 될거라곤 생각도 못했는데 삭제하니 오류가 해결되었다.

-> 500번 오류 발생함

===============================================

**URLEncoder, URLDecoder**

http 라는 프로토콜이 비연결지향성이라서 그렇다

한번 주고 받고 끝이다 다음 대화를 이어가기 위해서는 계속 요청과 응답이 있어야 한다

지속적인 대화를 주고 받고 싶으면 임시 저장소가 필요한데 그걸 쿠키와 세션이다

클라이언트 쪽은 쿠키, 서버는 세션

parameter -> [client cookie] <- string

attribute -> [server session] <- object

===============================================

css파일은 tomcat 서버 입장에서 컴파일 대상이 아니므로, 내용이 바뀌어도 새로고침이 자동으로 수행되지 않는다

===============================================

**jsp 내장객체 include**

<jsp:include page="ex02-header.jsp"/>

<%-- 먼저 각자 컴파일 결과를 합친다 --%>

<%-- 초3 되고, 퓨전하는 것 --%>

<%-- page를 객체화 해서 넣는 것 --%>

<%-- 컴파일하면 어차피 HTML이 아니고 class로 들어올 테니까 --%>

<%-- 그래서 삽입가능 --%>

===============================================

**include 액션태그**

- 액션 태그를 사용해서 include를 구현할 수도 있다

- 다른 페이지의 내용을 불러와서 현재 페이지에 포함시킨다는 개념은 동일하다

- 단, 액션태그 include는 서로 다른 두 파일을 컴파일 해서 만든 HTML 소스를 합친다

- 컴파일 단계에서 서로 다른 파일이므로, 변수이름이 겹쳐도 문제가 되지 않는다

- 두 파일은 변수를 공유할 수 없다

- 상단/하단에 포함시키기보다는, 중간에 이미 만든 페이지의 내용을 불러오는 용도로 사용한다

<%@ include file="ex01-header.jsp" %>

<%-- 먼저 합치고, 이후 컴파일 --%>

<%-- 퓨전하고 초3 --%>

include (반복되는 헤더와 푸터를 나눠서 가져올 수 있다)

include : 다른 문서의 내용을 현재 문서에 포함시키는 지시자

다른 웹 문서를 가져와서 소스코드를 합친 상태로 컴파일한다

상단 메뉴, 하단 메뉴 등의 공통된 페이지 요소를 처리하기에 좋다

서로 다른 두 메뉴는 컴파일 이전에 소스코드가 합쳐지므로, 변수이름이 중복되면 안된다

문서를 가져와서 합친 상태에서 컴파일하기 때문에

include 대상 파일에서 선언한 변수는 include를 처리하는 파일에서 모두 사용가능

header에서 선언한 변수는 main이든 footer든 어디서도 참조가능하다

===============================================

public String getPrice(){

return String.format("%,d원",price)

}

public void setPrice(String price){

this.price = Integer.parseInt(price);

}

**getter의 반환형과 setter의 매개변수 자료형이 일치하지 않으면 setProperty가 작동하지 않는다**

===============================================

list.get(i).getName().contains(searchName)

항상 이 .equals 나 contains의 순서를 생각해야 한다

-> 논리적으로 생각해보면 앞의 것이 searchName을 포함한다면 true이렇게 반환

searchName은 입력될지 안될지 모르기 때문에 값이 존재할지 안할지 모른다는 것이다

그래서 뒤에서 받는 것이 좋다

===============================================

<input type="text" name="name" value="${ param.name }">

--> 검색을 할때 검색창에 검색한 이름을 남기고 싶으면 입력받은 파라미터를 value로 가지고 있으면 된다

===============================================

textarea {resize : none;} => textarea size 고정

===============================================

<input type="hidden" name="index" value="${index}">

--> EL Tag로 받아서 숫자처럼 바로 사용할 수 있다

--> type="hidden" 으로 넘겨주고 싶은 값을 숨겨서 전달할 수 있다

===============================================

list에서 전달받은 index값을 꺼내보고, 지정한 인덱스에 새로받은 값을 넣어서 반환되는 이전 값이 일치하면 성공

const flag = ${ list.get(param.index) == list.set(param.index, modify)}

===============================================

삭제하는 방법

1. 목록을 띄워놓고 삭제할거 삭제버튼으로 삭제하기

2. 검색을 해서 삭제할 것인지

===============================================

배열이나 리스트에서 index는 고유하다(다른값과 중복되지 않는다. unique, primary key)

--> 중복이 없는 검색이나 고유값을 날릴경우

삭제할 때 input에 속성값에서 범위 지정해주기

<input min="1" max = "${list.size()}">

===============================================

<a href="deleteProcess.jsp?idx=${param.idx}><button>삭제</button></a>

<button id="deleteBtn">삭제</button>

alert() : 팝업창으로 메시지를 출력하고, 확인버튼만 받는다면

comfirm() : 팝업창으로 메시지를 출력하고, 확인/취소를 받아서 그 결과를 따라 true/false를 반환한다

deleteBtn을 onclick하면 함수가 실행되서 comfirm을 호출해서 idx번째를 삭제하겠는지 확인

===============================================

const flag = ${ list.remove(list[param.idx-1]}

// list.remove(int index) : 이 방식으로 시도했는데 String으로 식별해서 아래 함수 호출

// list.remove(Object o) : 리스트에서 배열처럼 접근 가능 idx를 정수로 확인 가능

=============================================================

header.jsp 에 선언된 리스트를 참조할 수 있으나, index와 regist가 서로 다른 객체이므로,

서로 다른 객체의 이름만 같은 멤버필드는 같은 대상이라고 할 수 없다

따라서, 서로 다른 페이지간에 공유할 객체를 session이나, application 내장 객체에 저장해야 한다

===============================================

required 는 보안요소 X

===============================================

**location.href location.replace**

기능 새 로운 페이지로 이동된다.기 존페이지를 새로운 페이지로 변경시킨다 .

형태 속성 메서드

주소 히 스토리 기록된다 기 록되지 않는다 .

사용예 location.href='abc.php' location.replace('abc.php')

location.href는 객체의 속성이며, loaction.replace()는 메서드(함수)로 작동된다.

href는 페이지를 이동하는 것이기 때문에 뒤로가기 버튼을 누른경우 이전 페이지로 이동이 가능하지만,

replace는 현재 페이지를 새로운 페이지로 덮어 씌우기 때문에 이전 페이지로 이동이 불가능하다.

href는 일반적인 페이지 이동시 이용을 하면 되고,

replace의 경우는 이전페이지로 접근이 필요없는경우 보안상 덮어씌우는 것도 괜찮을듯 하다.

===============================================

history.back()// history.go(-1), 이렇게 이동하면 form에서 입력했던 값이 남아있다 (password 제외)

===============================================

const mod = ${list.set(idx, modify)} boolean 값이 아님

set을 할 때 그냥 idx하니까 0번째부터 확인하고 넣어버린다

modify에서 입력받은 idx를 사용하기 위해서는 param.idx로 사용해야 한다

===============================================

**fmt**

Date date = new Date();

pageContext.setAttribute("date", date);

SimpleDateFormat과 유사하다

default : <fmt:formatDate value="${date }" dateStyle="default"/>

short : <fmt:formatDate value="${date }" dateStyle="short"/>

medium : <fmt:formatDate value="${date }" dateStyle="medium"/>

long : <fmt:formatDate value="${date }" dateStyle="long"/>

full : <fmt:formatDate value="${date }" dateStyle="full"/>

pattern : <fmt:formatDate value="${date }" pattern="yyyy-MM-dd a hh:mm:ss"/>

===============================================

**c:redirect , c:url**

<%= request.getContextPath() %>

${ pageContext.request.contextPath }

<c:url value="css/style.css"></c:url>

파일 이름 앞에 문서 경로를 자동으로 붙여준다

<c:redirect url="http://www.naver.com"/>

===============================================

숫자1 : <fmt:formatNumber value="123456789" type="number"/><br>

숫자2 : <fmt:formatNumber value="1000" type="currency" currencySymbol="￦"/><br>

숫자3 : <fmt:formatNumber value="0.3" type="percent"/><br>

숫자4 : <fmt:formatNumber value="12345.678" pattern=".00"/><br>

<결과값>

숫자1 : 123,456,789 <- 기본 천단위 끊어서 표현

숫자2 : ￦ 1,000.00 <- 원표시 붙이고 (,)붙이고 소수점 2개 까지 표현

숫자3 : 30% <- %로 표현

숫자4 : 12345.68 <- 설정 소수점에서 반올림으로 표현

================================================

**Session Cookie**

쿠키는 클라이언트에 문자열 정보를 저장한다

세션은 서버에 정보를 저장한다(JSP)

서버가 하나의 거대한 건물이라면, 건물에 찾아오는 여러 손님(client)을 1:1로 상대하는 직원이 세션에 해당한다

서버에서 application 내장 개체는 하나이지만, 세션은 여러개가 존재할 수 있다

쿠키로 처리 가능한 일이라면, 세션으로도 처리가 가능하다

keep alive 보완하기 위해서 사용

세션 생성 시간 : <%=session.getCreationTime() %>

세션 생성 시간 : <%=new Date(session.getCreationTime()) %>

세션이 유효한 시간 (초) : <%= session.getMaxInactiveInterval() %>

인터넷뱅킹 로그인 연장뜨는 것 / 요청이 한번이라도 있으면 유효기간 연장 / 아무런 작동이 없으면 세션은 날라감

마지막으로 세션에 접근한 시간 : <%= new Date(session.getLastAccessedTime()) %>

다수의 세션을 구분하기 위한 ID : <%= session.getId() %>

서버에서 세션을 구분하기 위한 id이기 때문에 중요하다

새 탭을 열어도 동일한 ID를 불러줌 / 그래서 컴터를 종료하지 않으면 남아있음

마지막 사용 세션의 ID를 저장한 쿠키 : ${ cookie.JSESSIONID.value }

쿠키는 클라이엔트에게 한번 갔다 와야하기 때문에 늦다/ 그래서 첫 방문이면 쿠키가 없는 것이다

세션이 새로 생성된 세션인가 ㅣ <%= session.isNew() %>

세션이 방금 생성되었기 때문에 attribute가 없다

session.invalidate();

-> 세션의 유효시간을 사용자가 변경하는 것

===============================================

**Session**

서버에는 여러개의 세션이 존재하고, 구분을 위하여 각각ID를 가지고 있다

클라이언트 입장에서 세션이란 서버에 둘 수 있는 임시 사물함과 같은 개념이다

일반적으로, 브라우저를 종료하면 기존 세션이 사라지지는 않으나 다른 세션을 만나게 된다

이때, 내가 마지막으로 사용했던 세션의 ID를 쿠키에 적어두었다가, 다음 요청시 ID를 서버에 제시할 수있다

서버는 클라이언트가 제시한 마지막 사용한 세션의 ID로 세션 객체를 찾아보고, 다음 요청시 ID를 서버에 제시할 수 있다

만약 클라이언트가 제시한새로운 세션을 생성하여 연결해준다

세션의 ID를 저장하는 세션 쿠키는 만료시간이 '브라우저종료' 시점으로 설정되어 있다

세션 쿠키의 이름을 똑같이 만들어서 유효시간만 새로 설정해서 보내면 세션쿠키가 유지될 수 있다

===============================================

**cookie**

- 웹 처리에 사용되는 어떤 데이터를 클라이언트(호스트가 아니라 프로그램)에 문자열 형식으로 저장하는 방식

- http 프로토콜은 요청-응답 이후 연결이 유지되지 않는다(keep alive 제외)

- 페이지가 변경되어도, 다음번에 사이트에 재방문해도 유지되어야 하는 정보

- 클라이언트에 저장하면 cookie, 서버에 저장하면 session

- 초창기에는 Cookie에 ID/Password, 주민등록번호, 열어본 게시물의 목록-게시글 번호가 저장되는 것(방문기록)

- Cookie는 브라우저가 관리하므로, 사용자의 컴퓨터에 저장이 된다

- 개인정보 도난의 위험이 있어서, 최근에는 사이트마다 Cookie 수집에 대한 허용을 확인한다

- 쿠키는 웹 프로그램이라서 크롬의 Cookie와 엣지의 Cookie가 다르다

- 쿠키에 저장된 정보를 기반으로 자동로그인되는 것도 있다

- 쿠키를 만드는 건 서버가 만들고 들고있는건 클라이언트가 들고 있다

- 클라이언트는 쿠키를 보유하고 있다가, 해당 사이트에 재방문시 받았던 쿠키를 지참하여 방문한다

- 서버는 클라이언트가 가져온 쿠키를 식별하여 클라이언트 마다 서로 다른 처리를 수행할 수 있다

- 브라우저별로 다르게 쿠키를 관리한다

- 쿠키를 사용할 때는 입력값을 최대한 적게하는 것이 좋다

- 보낼때는 request 꺼내올 때는 response

===============================================

DAO에서 넘길 수 있는건 row, list, dto 3개로 고정하고 나머지는 웹에서 판단하게 할 것

===============================================

dao의 login함수를 호출하면서 사용자의 파라미터가 담긴 user 객체를 전달하고

함수호출의 반환값을 login이라는 객체로 다시 받고

그 로그인 객체를 session에 attribute형식으로 저장하면서 이름은 login, 값은 DB에서 받아온 dto객체

===============================================

invalidate하면 jsessionid 값이 변경되는 것을 확인할 수 있다

1FE47D062A5EDAA01635C0ADEF169D47

E6AAF2757F7F2C75BB155F96339D6094

체크박스 체크하면 on 안하면 null

===============================================

쿠키를 삭제하려면 동일한 이름의 쿠키를 생성하고, 유효시간을 0으로 설정한 이후 클라이언트에게 보낸다

쿠키문자열은 인코딩이 제한되기 때문에 띄워쓰기를 포함하지 않고 언더바( \_ ) 나 다른기호로 대체한다

===============================================

<c:set var="login" value="${ dao.loginChk(user) }" scope="session"/>

dao의 login함수를 호출하면서 사용자의 파라미터가 담긴 user 객체를 전달하고

함수호출의 반환값을 login이라는 객체로 다시 받고

그 로그인 객체를 session에 attribute형식으로 저장하면서 이름은 login, 값은 DB에서 받아온 dto객체

===============================================

**Cookie**

<%

DAO dao = new DAO();

AccountDTO result = dao.selectAccountOne(user);

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

if(result != null){

session.setAttribute("login", result);

Cookie cid = new Cookie("id", result.getId());

Cookie cpw = new Cookie("pw", result.getPw());

cid.setMaxAge(60 \* 60 \* 24);

cpw.setMaxAge(60 \* 60 \* 24);

if(auto == null){

cid.setMaxAge(0);

cpw.setMaxAge(0);

}

response.addCookie(cid);

response.addCookie(cpw);

}

response.sendRedirect("main.jsp");

쿠키를 저장해서 넘겨줄 준비

서버로 넘어가는 것

짐칸에 살짝 놓아두는 느낌

도메인이 일치하면 쿠키를 지참해서 가는것이다

쿠키는 인코딩을 따로 처리하기 때문에 띄어쓰기 같은 문제가 생김

클라이언트에게 미리 찍어주는 쿠폰같은 느낌

response.addCookie(c);

쿠키의 구성은 parameter와 유사하지만 띄어쓰기를 포함하지 않아야하므로,

사용자의 입력보다는, 서버에서 만들어주는 고정 문자열을 쿠키에 저장하는 경우가 많다

사용자의 입력은 어떤 입력이 들어올지 예측하기 힘들고

DB에 저장된 데이터를 조회하기 위한 PK는 대부분 문자열로 표현 가능한 형태이기 때문에

만약, 띄어쓰기를 포함하려면 String 클래스의 replaceAll("old", "new") 메서드로 치환이 가능하다

(라디오박스는 몇번인지를 쿠키에 넣어서 처리하고, option은 selected를 쿠키에 담고, 체크박스는 checked를 담아서 처리)

===========================================

**회원탈퇴 과정**

1. 링크를 클릭했으므로, 전달되는 파라미터가 없다

2. 나는 현재 로그인 한 계정의 탈퇴를 처리하고 싶다(session 의 login객체)

3. 각 계정을 중복없이 정확하게 구분할 수 있는 기준값은 PK

4. ${ login.userid } 를 가지고 와서 dao에 전달해서 삭제쿼리문을 실행하고 결과값을 가지고 온다

5. 삭제에 성공해도 로그인된 세션이 남아있으니까, 삭제를 성공하면 세션을 같이 날려준다

6. 만약 삭제 실패라면 탈퇴처리가 안되었다고 팝업을 알리고 back()

탈퇴하면 로그아웃으로 넘겨서 session.invalidate()를 수행하도록 한다

<c:if test="${ dao.delete(param.pwChk) == 1 }">

<%

session.invalidate();

%>

<script>

alert('회원탈퇴 완료~')

location.href='${ cpath }'

</script>

<!-- <c:redirect url="${ cpath}/logout.jsp"/> -->

</c:if>

-> 이런식으로 redirect를 걸어서 로그아웃으로 가도록 한다 탈퇴하면 그러면 세션이 자동으로 날아가니까

===========================================

**InitialContext()**

이 클래스는 네이밍 조작을 실행하기 위한 개시 컨텍스트 입니다.

네이밍 조작은 모두, 컨텍스트를 기준으로 합니다.

초기 컨텍스트에는 Context 인터페이스가 구현되어 이름을 해결하는 개시점이 제공됩니다.

초기 컨텍스트를 구축합니다.

===========================================

**Web.xml 이란?**

- 서블릿 클래스는 JSP 페이지와 달리, 설치뿐만 아니라 등록을 하는 과정을 필요로 한다.

- 여기서 서블릿 클래스를 등록하는 위치의 이름을 Web Application Deployment Descriptor라고 하는데 (DD) 이 역할을 하는 위치가 바로 Web.xml이다.

- Web.xml 파일의 경우, 웹 어플리케이션 디렉토리마다 하나씩만 존재할 수 있다.

- DD는 WAS 구동 시, /WEB-INF 디렉토리에 존재하는 Web.xml을 읽어 웹 어플리케이션의 설정을 구성하기 위해 존재한다.

===========================================

**context.xml**

<?xml version="1.0" encoding="UTF-8"?>

<Context>

<Resource

name="jdbc/oracle"// 사용자가 지정하는 임의의 이름

auth="Container"

type="javax.sql.DataSource"

driverClassName="oracle.jdbc.OracleDriver"

url="jdbc:oracle:thin:@192.168.1.100:1521:xe"// 원격접속의 경우 반드시 @로 설정한다.

username="c##itbank"// 연결한 db 사용자 이름

password="it"// 연결한 db의 비밀번호

maxActive="20"// 커넥션 객체의 최대 갯수 (default 8)

maxIdle="10" // 사용하지 않는 idle(대기)상태의 최대 수

maxWait="-1" />

</Context>

===========================================

**web.xml**

<?xml version="1.0" encoding="UTF-8"?>

<resource-ref>

<description>ConnectionPool</description>

<res-ref-name>jdbc/oracle</res-ref-name>--> meta-INF에 있는 것

<res-type>javax.sql.DataSource</res-type>

<res-auth>Container</res-auth>

</resource-ref>

-> \*\*\* 현재 보고 있는 day09 프로젝트가 웹 프로그램으로 동작하기 위한 기본설정을 넣어주는 것이다 \*\*\*

--> 어떤 객체를 받아올지 모르니까 object타입이다

--> 프로젝트를 실행할 때 web.xml을 읽어서 resource를 읽어오래 그래서 meta-inf에서 context.xml을 읽어서

--> /jdbc/oracle이라는 이름을 찾아서 가지고 오는데 이 과정에서 객체가 어떤 타입인지 모르니까 object타입이다

--> 이런 객체를 준비해주세요 주문서같은 느낌

-> 텍스트 형태의 파일로 객체를 지정할 수 있다

-> 따로 만들어주지 않으면 Servers에서 web.xml을 가지고 와서 사용하는 것이다

===========================================

**싱글톤(Singleton)**

- 객체를 요청하면 어디서나 동일한 객체를 참조할 수 있도록 하는 디자인패턴

자바형태의 객체로 만들어두고 이름으로 자원을 참조한다

프로젝트만들자마자 이 객체가 생기고 준비상태가 된다

기존방식 : DAO마다 1개의 커넥션을 보유하고 메서드 호출이 끝나면 연결이 끊어짐

-> 메서드 호출을 여러번 하려면 같은 클래스로 여러번 객체를 생성하는 불편함

ConnectionPool 도입 후 : DAO가 여러번 메서드를 호출해도 커넥션을 새로 받아서 사용할 수 있게 된다.

-> 서로 다른 페이지가 아니라면, DAO는 하나만 있어도 충분하다

<%@ include %> 를 활용하면 여러 페이지가 하나의 DAO를 참조할 수도 있다

- 소스코드를 합친 이후에 컴파일이 되는 형식

- 실제 서버에서는 여러개의 DAO객체가 만들어지는 형태

DTO : 속성을 중심으로 처리하는 방식 -> 서로 다른 객체가 다른 값을 가지게 된다

DAO : 기능을 중심으로 처리하는 객체 -> 서버상의 메모리에 오직 하나의 객체만 있어도 된다(참조할 수 있게 만들어줘야지 - 주소값을 저장해둠)

(밥을 먹는데 한숟갈마다 숟가락을 바꿔서 쓰는것 -> 지금은 한숟가락으로 밥을 먹는 것)

객체를 서로 다른 페이지에서 생성해도, 여러번 새로고침해도, 서로 다른 클라이언트에서 접근해도

오로지 하나의 객체만 참조하게 된다( 불필요한 메모리 낭비가 사라진다)

싱글톤은 오로지 하나의 객체만 참조하는 것을 확인하기

getInstance()메서드를 통해서 브라우저가 변경되어도 해시코드 값이 같은 것을 확인할 수 있다

===========================================

**파일 업로드 정리**

package upload;

import java.io.File;

import java.io.IOException;

import javax.servlet.http.HttpServletRequest;

import com.oreilly.servlet.MultipartRequest;

import com.oreilly.servlet.multipart.DefaultFileRenamePolicy;

import com.oreilly.servlet.multipart.FileRenamePolicy;

// FileUtil 클래스는 파일을 업로드하기 위해 따로 class를 만든거다

public class FileUtil {

private static String encoding = "UTF-8";

private static String saveDirectory;

private static int postSize = 1024 \* 1024 \* 20;

private static FileRenamePolicy policy = new DefaultFileRenamePolicy();

private static MultipartRequest mpRequest;

여기서 static을 사용한 이유 ?

static 은 class가 생성되어 메모리에 올라갈 때 자동으로 생성되어서 값의 변화가 불가? class가 생성되는 타이밍에 메모리 할당되는거라서 정적이다

뭐가 정적이죠?? static으로 묶으면 다른 외부에서 값을 변화시킬 수도 참조해서 값을 변경할수도 없다 why?? FileUtil class가 생성되는 시점에 메모리에 꽝하고 박혀있는 상태니까 함부로 값의 변경이 힘들겠죠??

그리고 static으로 묶으면 우리가 jsp에서 불러올 때도 <%= FileUtil.getDTO(request) %> 이렇게 불러옵니다

근데 원래 메서드를 호출하려면 해당 객체를 만들어서 객체안에 메서드를 호출해야합니다

BoardDAO dao = new BoardDAO();

int row = dao.insert(dto); 이런식으로 new 키워드로 BoardDAO 생성자를 호출해서 객체를 생성하고 BoardDAO타입의 dao참조변수에 저장

원래는 이런식으로 객체를 참조해서 메서드를 불러와야 하는데 static으로 묶으면 <%= FileUtil.getDTO(request) %> 이렇게 써서 간편하게 객체를

호출 할 수 있으니까 편리하고 속도가 빠르다!!

그렇다고 static을 막쓰면 안됨

static은 자주 변하지 않는 값이나 설정 정보같은 정보를 매번 호출하기 귀찮으니까 메모리에 쾅하고 찍는 느낌으로 사용한다

이 메서드는 HttpServletRequest 타입의 request 참조변수를 매개변수로 받고 있다

request는 jsp 내장객체로 input타입의 값을 getParameter등과 같은 메서드를 이용해서 가져올 수 있고

equest.getContextPath() 같이 해당 프로젝트의 최상위 폴더를 찾을 수도 있고 내장객체라서 다양한 것을 사용할 수 있는데r

우리는 여기서 request 내장 객체가 가지고 있는 메서드를 활용해서 파일 업로드 함수를 만들 것이다

public static UploadDTO getDTO(HttpServletRequest request) throws IOException {

new 키워드로 UploadDTO 생성자를 호출해서 객체를 생성했다

객체는 UploadDTO 타입의 dto 참조변수를 객체로 선언

UploadDTO dto = new UploadDTO();

saveDirecoty : 직역하면 저장될 디렉토리 / 그럼 저장될 위치를 알려달라는 것이다

request.getServletContext().getRealPath()는 웹 어플리케이션(server : tomcat의 작동 경로)

실제 서버의 경로를 가져오는 것이다

개발하고 있는 개발폴더에 파일이 업로드되는 일은 없다 (우리가 프로젝트를 살펴보면 text폴더를 없다)

서비스되고 있는 웹 어플리케이션에 업로드된는 것이다

saveDirectory = request.getServletContext().getRealPath("./text");

new 키워드로 File 생성자에 saveDirectory를 넣어서 객체를 생성한다

저 위치에 폴더를 생성하기 위한 것

그리고 예외상황이 발생할 수 있다 why?? 폴더가 없을 수 있자나

그래서 dir이 exist() 존재가 false라면 dir을 mkdir(디렉토리를 만들어라 : 폴더를 만들어라)

File dir = new File(saveDirectory);

if(dir.exists() == false) {

dir.mkdir();

}

mpRequest는 파일을 업로드하기 위한 객체

new 키워드로 MultipartRequest 생성자를 호출해서 객체를 만드는데 여긴 5가지 매개변수를 넣어줘야지 객체를 생성해준다

해당 매개변수는 위에 다 정의했다

saveDirectory : 저장될 위치

postSize : 업로드 될 파일 크기

encoding : 인코딩방식 "utf-8" 지정 안하면 DB에 글자가 꺠져서 들어간다

policy: 동일한 파일이 업로드 될 경우 이름이 겹치면 제약조건에 걸리는 경우나 구분하기 힘들기 때문에 규칙을 정해서 동일한 파일이 들어오면

파일이름 뒤에 1, 2, 3, ... 이렇게 숫자를 붙여서 파일을 구분할 수 있게 한다

mpRequest = new MultipartRequest(request, saveDirectory, postSize, encoding, policy);

여기서 dto에 값을 넣는데 뭔데 갑자기 값을 넣냐라고 할 수 있는데

request는 getParameter로 입력 받은 값을 가져올 수 있다 근데 지금 위에보면 request를 입력해야 객체를 만들 수 있다

그럼 requset를 활요하는 거지

그래서 input타입의 name이 memo 인 것의 값을 가져와서 dto에 저장하겠다

dto.setMemo(mpRequest.getParameter("memo"));

파일은 input타입중에 file로 우리가 지정한게 있다

거기서 name이 uploadfile인 것을 찾아서 그 파일의 이름을 가져와서 dto에 저장한다

why?? 우리는 DB에 경로만 저장할 거라서 파일의 이름만 DB에 저장해서 ${ cpath }/text/${ ls.uploadFileName } 이런 식으로 경로를 가져올 꺼라서

파일의 이름만 가져오는 것이다

dto.setUploadFileName(mpRequest.getFile("uploadfile").getName());

return dto;

}

}

===========================================

**페이징 정리**

1~5까지를 하나의 section(덩어리로)으로 만들어서 처리하기

section은 ( param.page -1 ) / 5로 처리

why? 1~5 까지 0으로 만들기 위해서 5로 나눠서 몫을 0으로 맞춤

(1~5까지 동일한 숫자를 뽑아내기 위해서)

pagesectionbeginend

1015

2015

3015

4015

5015

begin => section\*5 + 1

if

end가 딱 떨어질 경우

end => begin + 4

()

else

end가 딱 안 떨어지는 경우

end => pageCount

-> 여기서 pageCount가 begin+4 보다 작으면 pageCount를 출력하고 아니면 begin+4를 출력하도록 한다

61610

71610

81610

91610

101610

1121112

1221112

이전은 section이 0이 아니라면 띄워주기

다음은 pageCount 를 5로 나눈 값이 end를 5로 나눈 값이 다르면 띄워주기

===========================================

**servlet 정리**

package myServlet;

import java.io.IOException;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

서블릿은 웹의 요청과 응답을 처리할 수 있는 자바 클래스

JSP 이전에 서블릿 기술이 먼저 등장했다

JSP도 실행될 떄는 서블릿의 형태로 바뀌고 나서 실행된다

서블릿을 만들 때에는 HttpServlet 클래스를 상속받는다

표현 방식이 다른 것이다

jsp는 서블릿 기반으로 만든 것

자바에서 html속성쓰려면 힘듬..

서블릿은 문서 이름 차제를 호출하는 것이다

확장자를 맘대로 사용 가능하다

주소창에 꼭 파일 이름이 올 필요가 없다는 것이다 (web.xml에서 서블릿 안에 url-Pattern의 이름을 다르게 설정해주면 주소창에 다르게 출력)

서블릿을 등록하고 서블릿을 맵핑하는 것이다

서블릿을 구동하는 방법은 xml에 작성

요청에 대한 처리는 자바코드에서 처리하고, 응답은 자바에서 html쓰기 불편하니까 .jsp로 넘기자

서블릿을 만들때는 HttpServlet 클래스를 상속받아서 사용한다

public class Servlet02 extends HttpServlet {

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

System.out.println("doGet : 요청메서드가 get이므로 적절할 jsp로 포워딩");

}

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

System.out.println("doPost : 요청메서드가 post이므로, 필요한 처리를 진행하고 포워드 혹은 리다이렉트");

}

@Override

protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

System.out.println("service : 요청 메서드를 확인하고 그에 맞는 세부 함수를 호출함");

req.setCharacterEncoding("UTF-8");

resp.setCharacterEncoding("UTF-8");

프로젝트 최상위 경로

String contextPath = req.getContextPath();

요청 받은 경로 (자원 식별자)

String requestURI = req.getRequestURI();

contextPath : /day20

requestURI : /day20/ex02

urlPattern : 요청내용 -> requestURI에서 substring 한 것

requestURI의 총 길이에서 contextPath의 길이정도를 뺀 문자열을 urlPattern으로 하겠다

urlPattern이라는 문자열은 requestURI라는 문자열을 substring한 문자열

substring(0,4) : 0부터 3까지

substring(6) : 6부터 마지막까지

String urlPattern = requestURI.substring(contextPath.length(),requestURI.length());

System.out.println("contextPath : " + contextPath);

System.out.println("requestURI : " + requestURI);

System.out.println("urlPattern : " + urlPattern);

requestURL(요청 내용) : 프로젝트 + 파일경로 / WEB\_INF의 실제 jsp 경로를 가져옴 그래서 주소를 redirect시킬 수 있는 것

http://localhost:8080/day20/ex02.jsp

requestURI(자원 식별자) : localhost 뒤에 프로젝트 이름부터 끝까지

day20/ex02.jsp

차이점 : URL는 앞에 http://localhost가 붙고 URI는 안붙고

Web.xml 이란?

- 서블릿 클래스는 JSP 페이지와 달리, 설치뿐만 아니라 등록을 하는 과정을 필요로 한다.

- 여기서 서블릿 클래스를 등록하는 위치의 이름을 Web Application Deployment Descriptor라고 하는데 (DD) 이 역할을 하는 위치가 바로 Web.xml이다.

- Web.xml 파일의 경우, 웹 어플리케이션 디렉토리마다 하나씩만 존재할 수 있다.

- DD는 WAS 구동 시, /WEB-INF 디렉토리에 존재하는 Web.xml을 읽어 웹 어플리케이션의 설정을 구성하기 위해 존재한다.

Servlet 실행 순서

1. 사용자가 URL을 클릭하면 HTTP Request를 Servlet Container에 보낸다.

2. Servlet Container는 HttpServletRequest, HttpServletResponse 두 객체를 생성한다. ( init() )

3. 사용자가 요청한 URL을 분석하여 어느 서블릿에 대한 요청인지 찾는다. (DD를 참조하여 분석)

4. 컨테이너는 서블릿 service() 메소드를 호출하며, POST, GET여부에 따라 doGet() 또는 doPost()가 호출된다.

5. doGet() or doPost() 메소드는 동적인 페이지를 생성한 후 HttpServletResponse 객체에 응답을 보낸다.

6. 응답이 완료되면 HttpServletRequest, HttpServletResponse 두 객체를 소멸시킨다. ( destory() )

\*DD (배포서술자, Deployment Descriptor) = web.xml

String viewName = "";

String prifix = "WEB-INF/view";

String suffix = ".jsp";

requestURI에서 뽑아낸 urlPattern이 / 이면 index

if(urlPattern.equals("/")) {

viewName = "index";

}

그게 아니라면 접두사는 "WEB\_INF/view" 를 붙이고 접미사는 ".jsp"를 붙여서 문자열을 완성시킨다

else {

viewName = prifix + urlPattern + suffix;

}

System.out.println("viewName :" + viewName);

그리고 포워드시켜서 주소는 가만히 있는데 화면만 바꿔치기 시키는 것이다

RequestDispatcher rd = req.getRequestDispatcher(viewName);

rd.forward(req, resp);

}

@Override

public void destroy() {

System.out.println("destory : 서버를 종료하거나, 소스코드 수정에 의해 새로 컴파일이 필요하여 기존 객체를 소멸시킴");

}

@Override

public void init() throws ServletException {

메모리 상에 클래스를 띄워두는 역할

System.out.println("init : WAS(톰캣)에 컴파일된 클래스가 없어서 서블릿 초기화를 진행");

}

}