8_12_2022__Web_Jsp, DB

Step01_Servlet

~Step02_DB

Servlet, jsp

▼ 실습예제(Servlet)

```
package test.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import test.dto.MemberDto;

@WebServlet("/member/list")
public class MemberServlet extends HttpServlet {
  @Override
  protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

    // DB 에서 불러온 회원 목록이라고 가정하자
    List<MemberDto> list = new ArrayList<>();
    list.add(new MemberDto(1, "두부", "페어펙스"));
    list.add(new MemberDto(2, "유키", "하이데저트"));
    list.add(new MemberDto(3, "소주", "리치몬드"));
    resp.setCharacterEncoding("utf-8");
    resp.setContentType("text/html; charset=utf-8");
    PrintWriter pw = resp.getWriter();
    pw.println("<!doctype html>");
    pw.println("<html>");
    pw.println("<head>");
    pw.println("<meta charset='utf-8'>");
    pw.println("<title></title>");
    pw.println("</head>");
    pw.println("<body>");
    pw.println("<h1>회원 목록 페이지 입니다.</h1>");
    pw.println("<table border =\"1\">");
      pw.println("<thead>");
        pw.println("<tr>");
          pw.println("<th>번호</th>");
          pw.println("<th>이름</th>");
          pw.println("<th>주소</th>");
        pw.println("</tr>");
      pw.println("<thead>");
      pw.println("<tbody>");
        for(MemberDto tmp:list) {
          pw.println("<tr>");
            pw.println("<td>"+tmp.getNum()+"</td>");
            pw.println("<td>"+tmp.getName()+"</td>");
            pw.println("<td>"+tmp.getAddr()+"</td>");
          pw.println("</tr>");
          }
      pw.println("<tbody>");
    pw.println("</table>");
    pw.println("<a href=\"/Step01_Servlet/\">인덱스로 가기</a>");
    pw.println("</body>");
    pw.println("</html>");
    pw.close();
  }
  /*
```

```
   * 회원 목록을 table 형식으로 출력해 보세요
   */

}
```

▼ 실습예제(jsp)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>/index.html</title>
</head>
<body>
  <div class="container">
    <h1>인덱스 페이지 입니다.</h1>
    <ul>
      <!--
      1. 아래의 링크를 눌렀을 때 오늘의 날씨가 이쁘rn게 나오도록 해 보세요.
      2. 1번을 성공했다면 날씨를 5개정도 정해 놓고 그중에서 랜덤하게 하나가 나오도록 프로그래밍 해 보세요.
     -->
      <li><a href="weather">오늘의 날씨 보기</a></li>
      <li><a href="weather.html">오늘의 날씨 보기2</a></li>
      <li><a href="weather.jsp">오늘의 날씨 보기3</a></li>
      <li><a href="friend/list">친구 목록 보기</a></li>
      <li><a href="friend/list.jsp">친구 목록 보기2</a></li>
      <li><a href="comment.jsp">jsp 페이지에서의 주석</a></li>
      <li><a href="/Step01_Servlet/member/list">회원 목록 보기</a></li>
      <li><a href="/Step01_Servlet/member/list.jsp">회원 목록 보기2</a></li>

    </ul>
    <!--
           폼을 제출(submit) 하면 ( type="submit" 인 전송 버튼을 누르면)
           action="요청경로"
           method="전송방식"

           서버에 "요청경로" 대로 요청이 되고
           폼에 입력한 내용은 "전송방식" 으로 전송이 된다.  (get or post)

           - form 사용법
           1. action 속성의 값은 폼을 제출했을때 요청되는 경로가 된다.
           2. method 속성의 값은 전송 방식을 지정한다. 생략하면 default 값은 get 이다.
           3. form 의 자손요소중에 type="submit" 버튼을 누르면 폼이 제출된다.
           4. 전송 방식은 get 방식과 post 방식이 있는데
              get 방식 전송은 입력한 정보를 주소창에 달고가는 방식이고
              post 방식 전송은 요청의 몸통에 달고가는 방식이기 때문에 주소창에 보이지 않는다.
       -->
    <form action="/Step01_Servlet/send" method="get">
      <input type="text" name="msg" placeholder="서버에 할말 입력..." />
      <button type="">전송</button>
    </form>
  </div>
</body>
</html>
```

Form

- 폼을 제출(submit) 하면 (type="submit" 인 전송 버튼을 누르면)
- 서버에 "요청경로(action)" 대로 요청이 되고 폼에 입력한 내용은 "전송방식(method)" 으로 전송이 된다. (get or post)
  - get 방식 요청
    - 요청 파라미터가 주소창에 같이 보인다. (형식 → ?XXX=XXX)
    - name속성의 밸류가 파라미터와 함께 나오기 때문에 꼭 추가해야 한다.
    - 자동 디코딩이 되어 한글이 안깨진다.
  - post 방식
    - 주소창에 클라이언트에서 적은 내용이 보이지 않는다.

- form 사용법
  1. action 속성의 값은 폼을 제출했을때 요청되는 경로가 된다.
  2. method 속성의 값은 전송 방식을 지정한다. 생략하면 default 값은 get 이다.
  3. form 의 자손요소중에 type="submit" 버튼을 누르면 폼이 제출된다.
  4. 전송 방식은 get 방식과 post 방식이 있는데 get 방식 전송은 입력한 정보를 주소창에 달고가는 방식이고 post 방식 전송은 요청의 몸통에 달고가는 방식이기 때문에 주소창에 보이지 않는다.

▼ 예제

1. /send 요청을 처리할 SendServlet 클래스 제작.

   ▼ 코드

```
package test.servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/send")
public class SendServlet extends HttpServlet {
  @Override
  protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

    super.service(req, resp);
  }

}
```

2. 만든 서블렛에서 클라이언트가 form 전송하는 문자열을 추출해서 서버측 콘솔창에 출력.

   전송하는 문자열은 msg라는 파라미터 명으로 전달됨

   ▼ 코드

```
package test.servlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/send")
public class SendServlet extends HttpServlet {
  @Override
  protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    System.out.println("클라이언트에서 요청이 왔다!");
    req.setCharacterEncoding("utf-8");
    String a = req.getParameter("msg");
    System.out.println("msg : "+a);
  }
}
```

3. 클라이언트에게 html 형식으로 “메세지 잘 받았어 클라이언트야!” 라는 문자열을 응답.

   ▼ 코드(완성)

```
package test.servlet;

import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/send")
public class SendServlet extends HttpServlet {
  @Override
  protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    System.out.println("클라이언트에서 요청이 왔다!");
    req.setCharacterEncoding("utf-8");
    String a = req.getParameter("msg");
    System.out.println("msg : "+a);

    resp.setCharacterEncoding("utf-8");
    resp.setContentType("text/html; charset=utf-8");
    PrintWriter pw = resp.getWriter();
    pw.println("<!doctype html>");
    pw.println("<html>");
    pw.println("<head>");
    pw.println("<meta charset='utf-8'>");
    pw.println("<title> 클라이언트의 요청을 받았다 !</title>");
    pw.println("</head>");
    pw.println("<body>");
    pw.println("<p>메세지 \""+a+"\" 잘 받았어 클라이언트야!</p>");
    pw.println("<br>");
    pw.println("<a href=\"/Step01_Servlet/\">인덱스로 가기</a>");
    pw.println("</body>");
    pw.println("</html>");
    pw.flush();//방출
    pw.close();//닫아주기
  }

}
```

Step02_DB

DB에 저장된 정보를 jsp사용해 가져오려 한다.

▼ 톰캣

1. Context configuration

In a similar manner to the mysql config above, you will need to define your Datasource in your Context. Here we define a Datasource called myoracle using the thin driver to connect as user scott, password tiger to the sid called mysid. (Note: with the thin driver this sid is not the same as the tnsname). The schema used will be the default schema for the user scott.

Use of the OCI driver should simply involve a changing thin to oci in the URL string.

```
<Resource name="jdbc/myoracle" auth="Container"
              type="javax.sql.DataSource" driverClassName="oracle.jdbc.OracleDriver"
              url="jdbc:oracle:thin:@127.0.0.1:1521:mysid"
              username="scott" password="tiger" maxTotal="20" maxIdle="10"
              maxWaitMillis="-1"/>
```

2. web.xml configuration

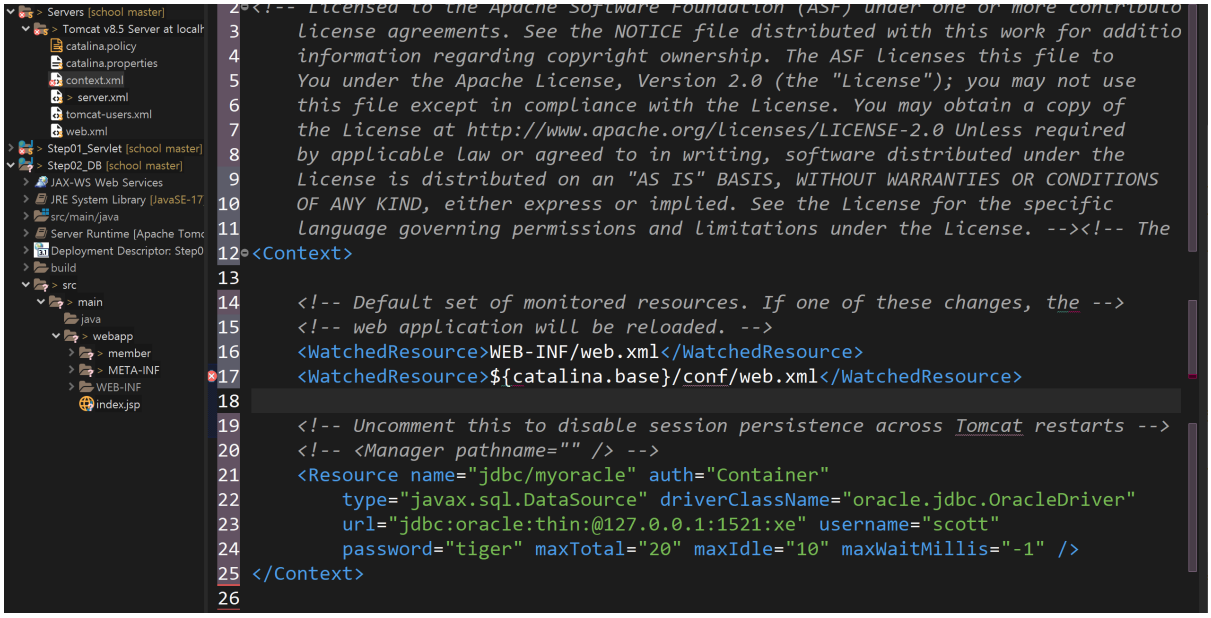
You should ensure that you respect the element ordering defined by the DTD when you create you applications web.xml file.

```
<resource-ref>
 <description>Oracle Datasource example</description>
 <res-ref-name>jdbc/myoracle</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
</resource-ref>
```

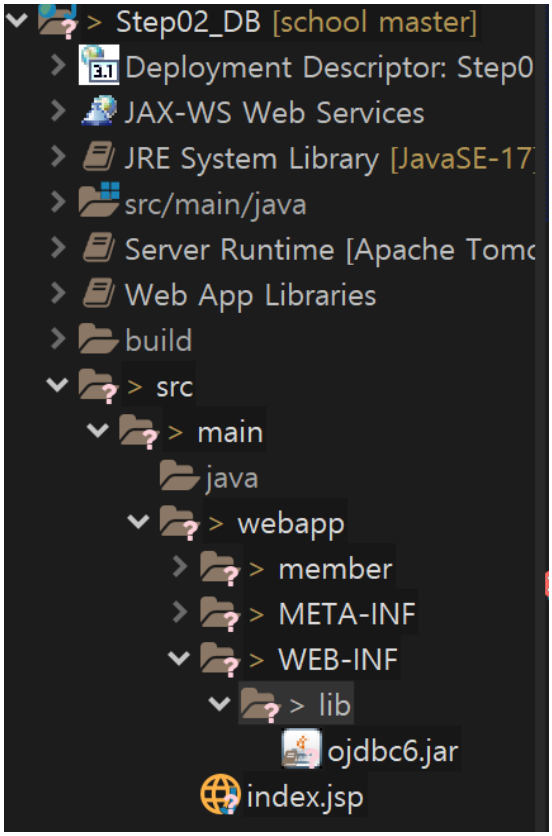
3. Code example

You can use the same example application as above (assuming you create the required DB instance, tables etc.) replacing the Datasource code with something like

```
Context initContext = new InitialContext();
Context envContext  = (Context)initContext.lookup("java:/comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/myoracle");
Connection conn = ds.getConnection();
//etc.
```



톰캣웹사이트에서 복사해옴



ojdbc6.jar 복사해넣기

2. web.xml configuration

You should ensure that you respect the element ordering defined by the DTD when you create you applications web.xml file.

```
<resource-ref>
 <description>Oracle Datasource example</description>
 <res-ref-name>jdbc/myoracle</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
</resource-ref>
```

javaEE tools

```
package test.util;

import java.sql.Connection;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

public class DbcpBean {
  // 필드
  private Connection conn;

  // 생성자
  public DbcpBean() {
    try {
      Context initContext = new InitialContext();
      Context envContext = (Context) initContext.lookup("java:/comp/env");
      DataSource ds = (DataSource) envContext.lookup("jdbc/myoracle");
      // 리턴되는 Connection 객체를 필드에 저장하기
      conn = ds.getConnection();
      System.out.println("Connection 얻어오기 성공");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
  //메소드
  public Connection GetConn() {
```

```
    return conn;
  }

}
```

Data Base Connection Pool Bean

- 자바에서 객체를 콩(bean)으로 부르기도 한다.
- 최초 Connection이 필요한 시점에 미리 Connection 여러개를 얻어내서 Connection Pool에 넣어 놓고 필요할 때 가져다 쓰고 다 쓴 다음 반납하는 구조로 사용한다.
- Connection Pool에서 Connection 한개를 가져오는 방법

```
Connection conn = neww DbcpBean().getConn();
```

- 다 사용한 다음 반납하는 방법

```
conn.close(); // close()메소드를 호출하면 자동으로 Pool에 반납된다.
```