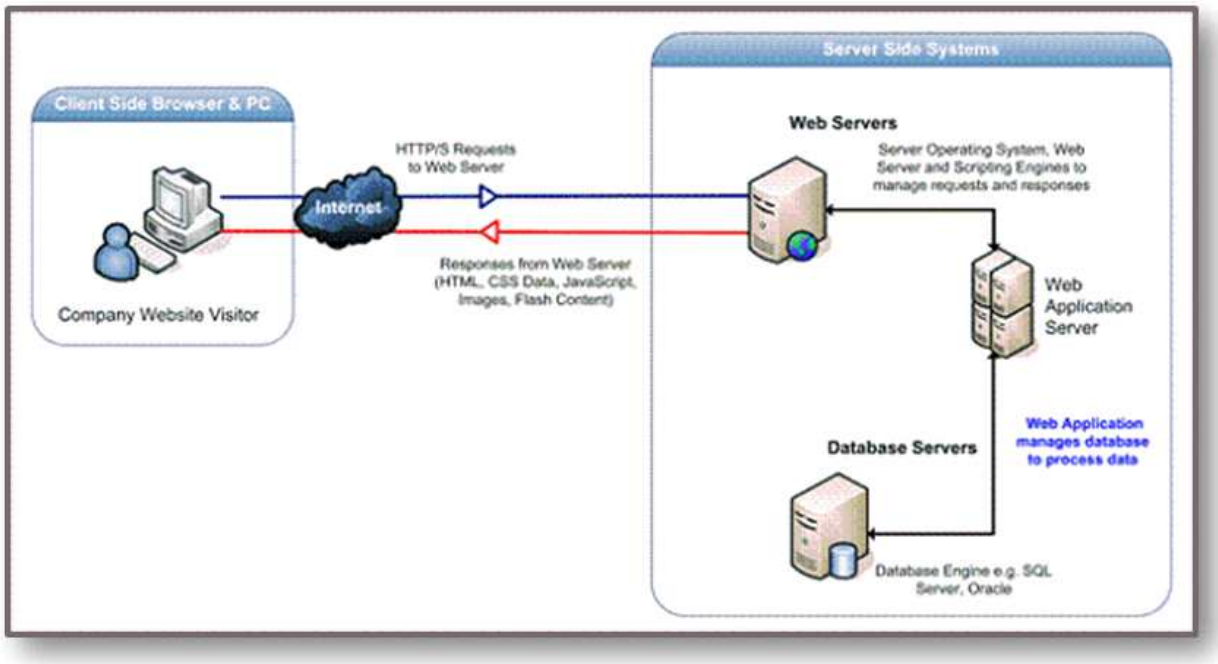
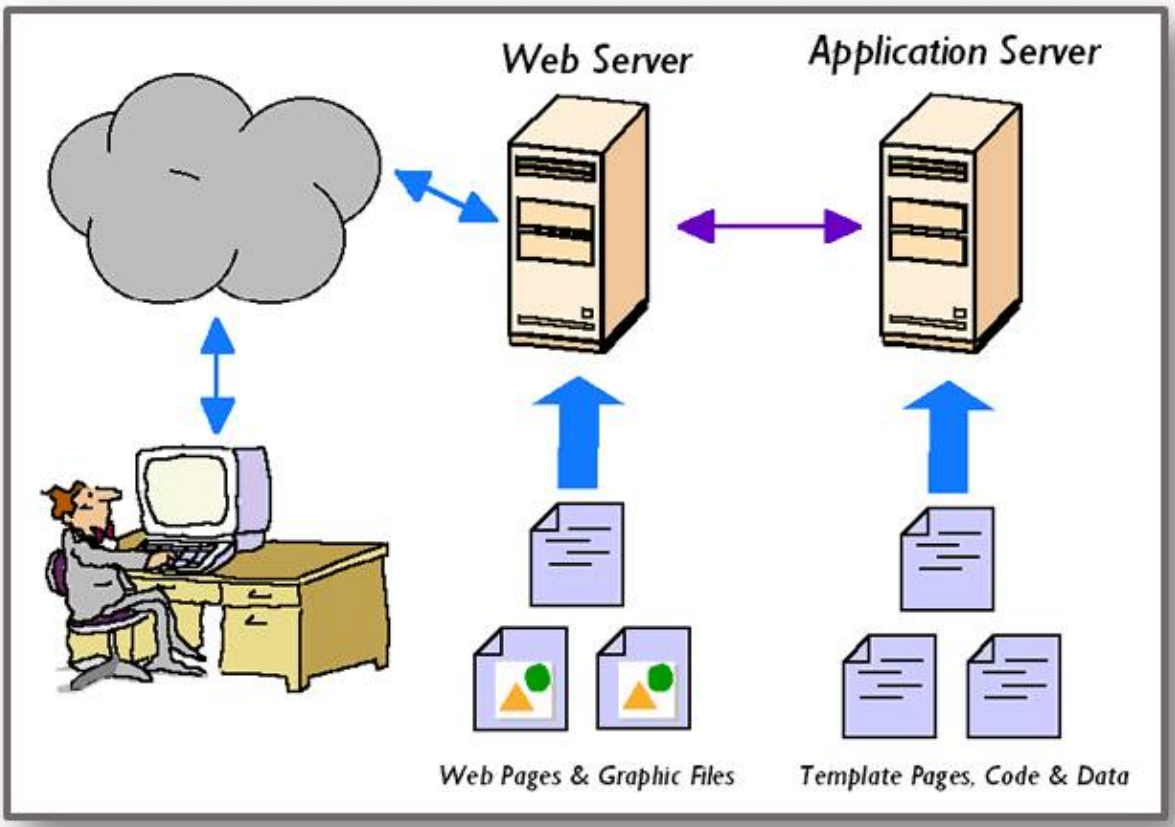
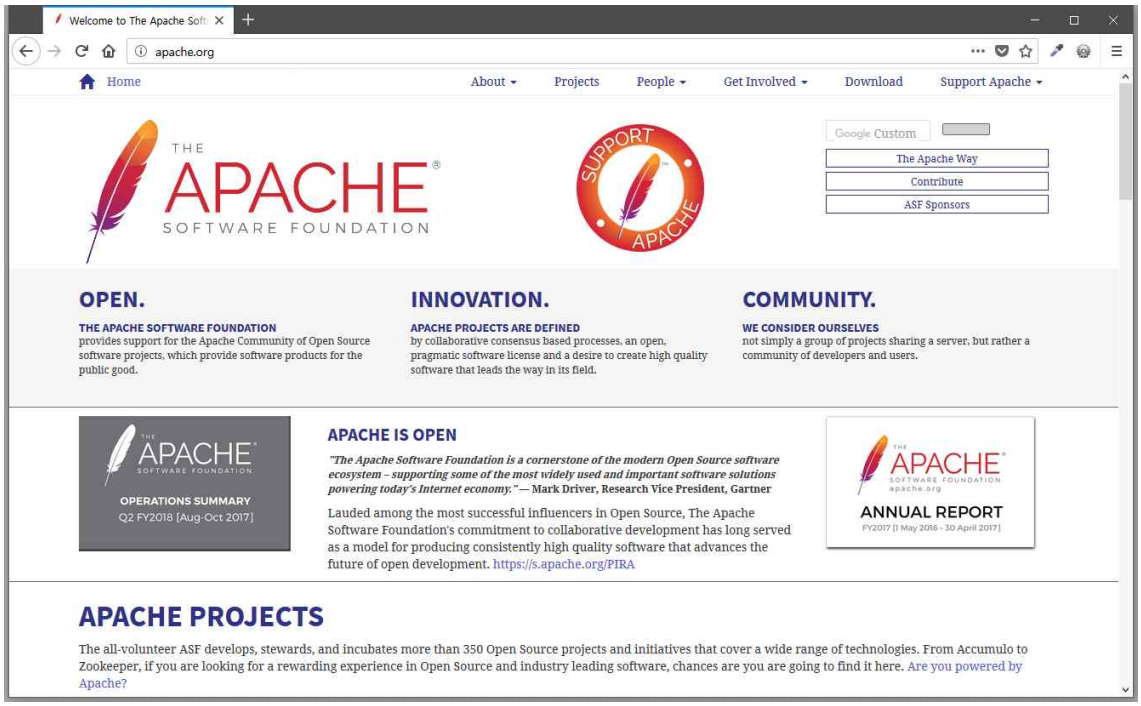
■ 웹서버 / 웹 어플리케이션 서버

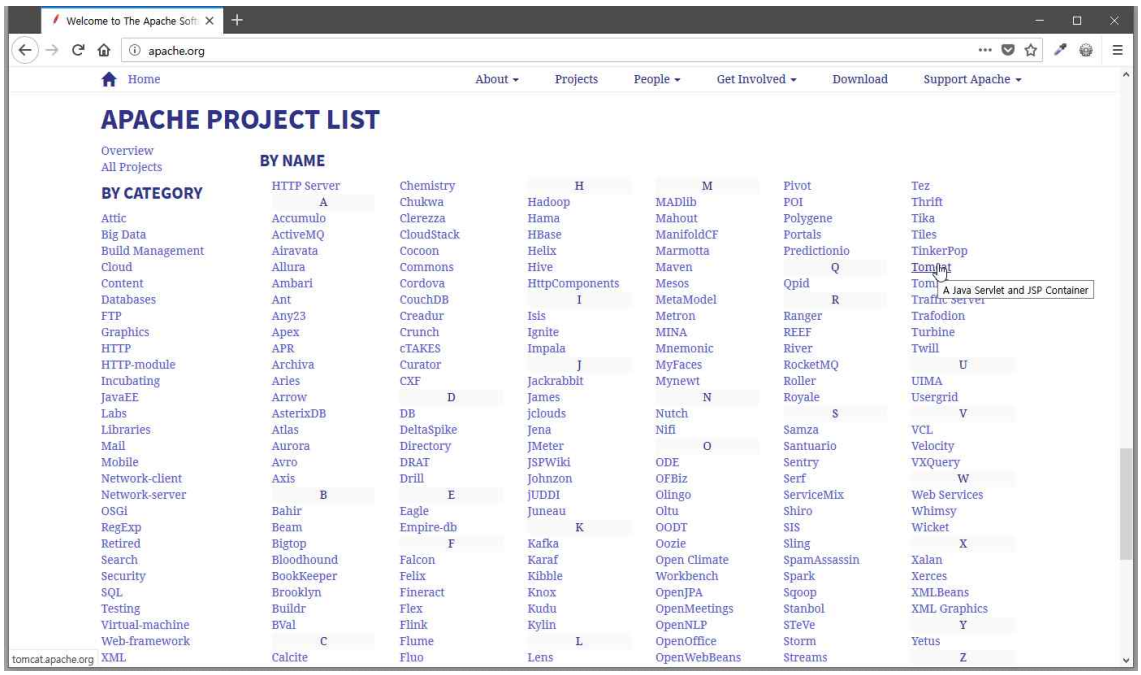


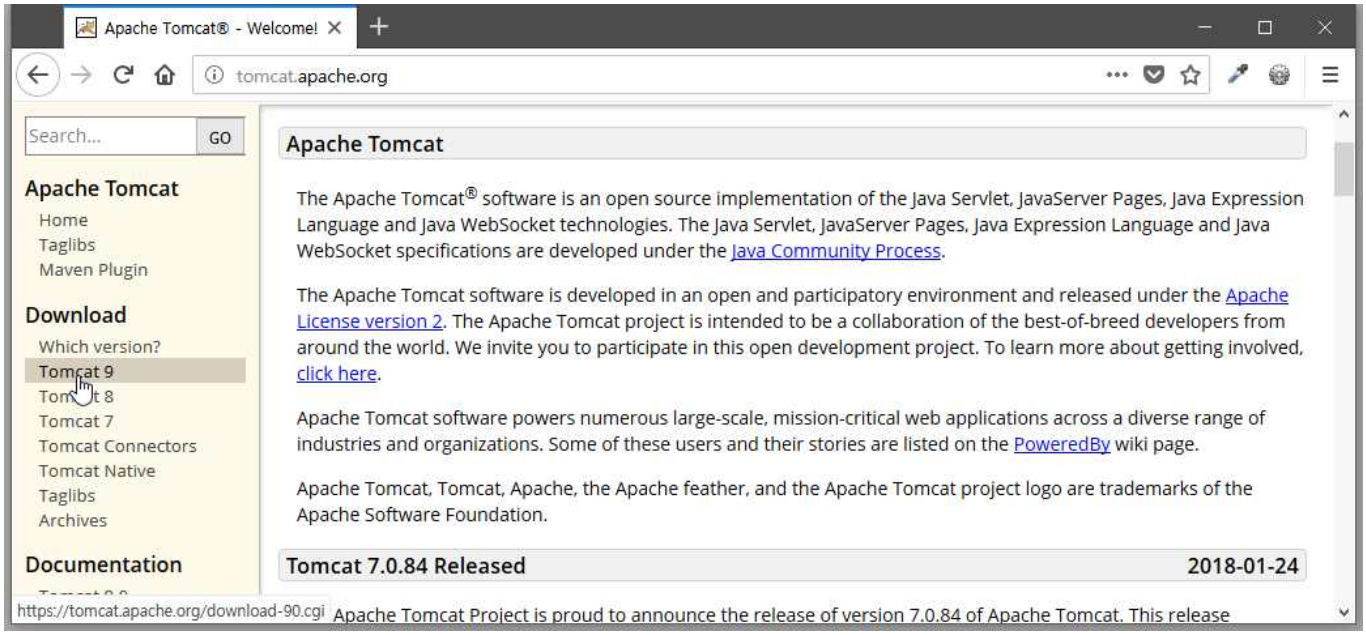


Tomcat의 다운로드 및 설치

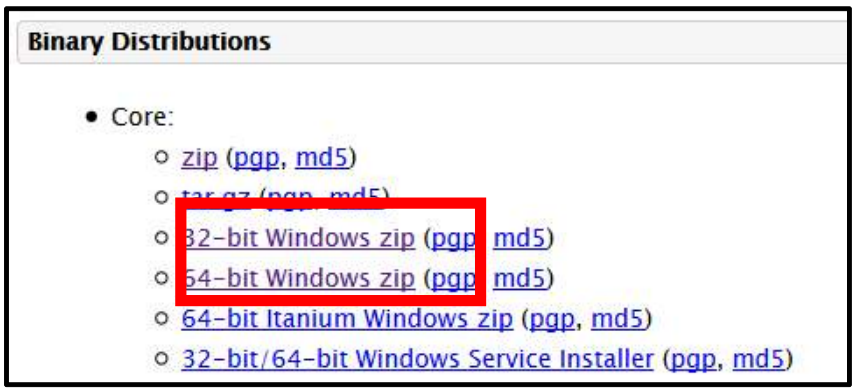
■ apache.org



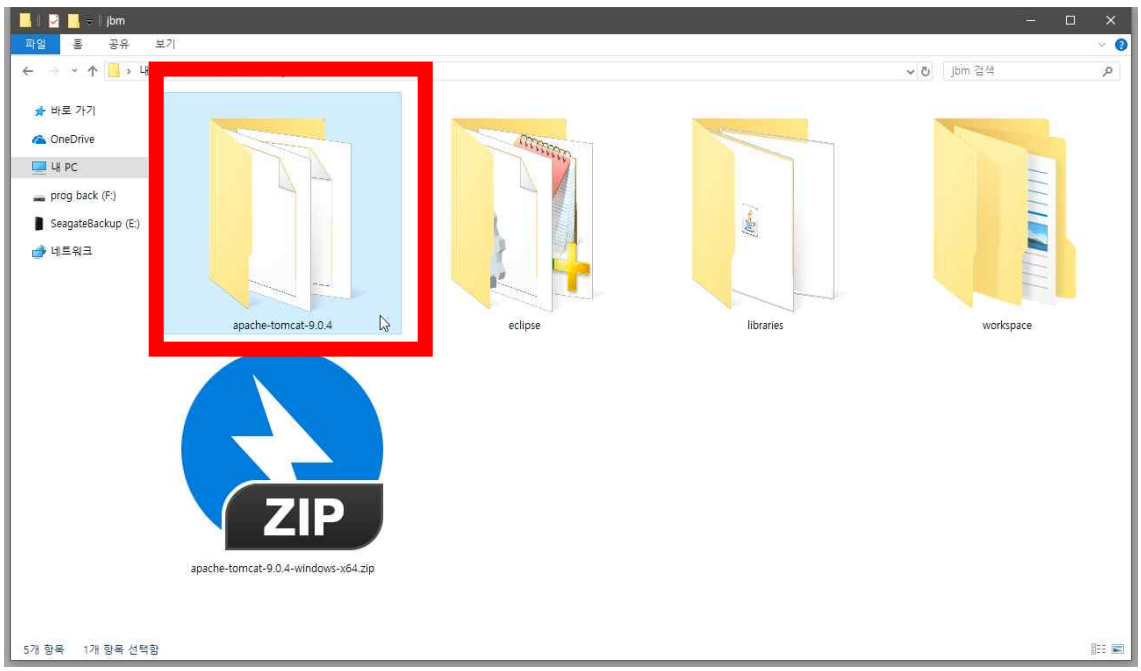




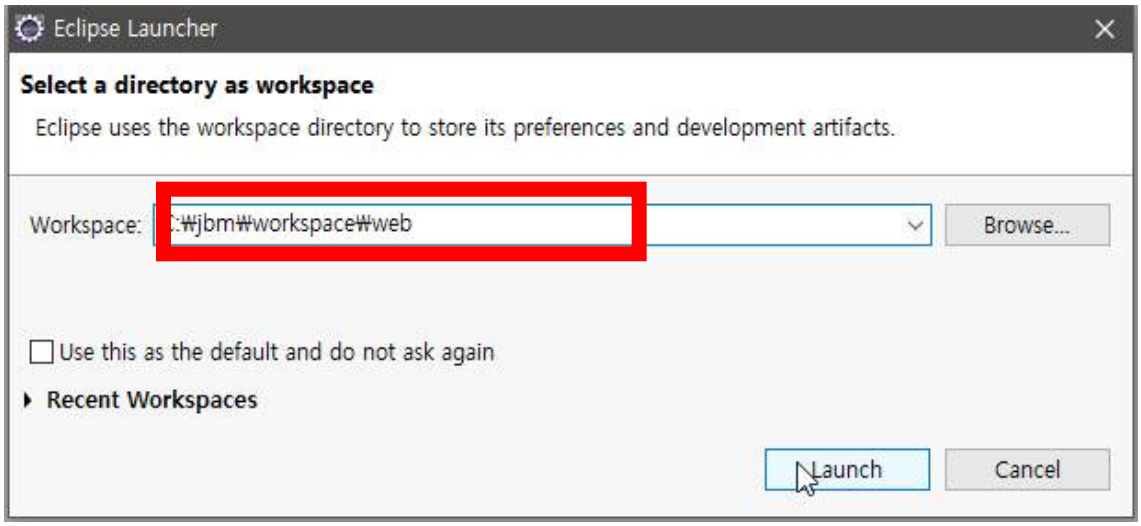
■ 윈도우 버전에 맞게 다운로드



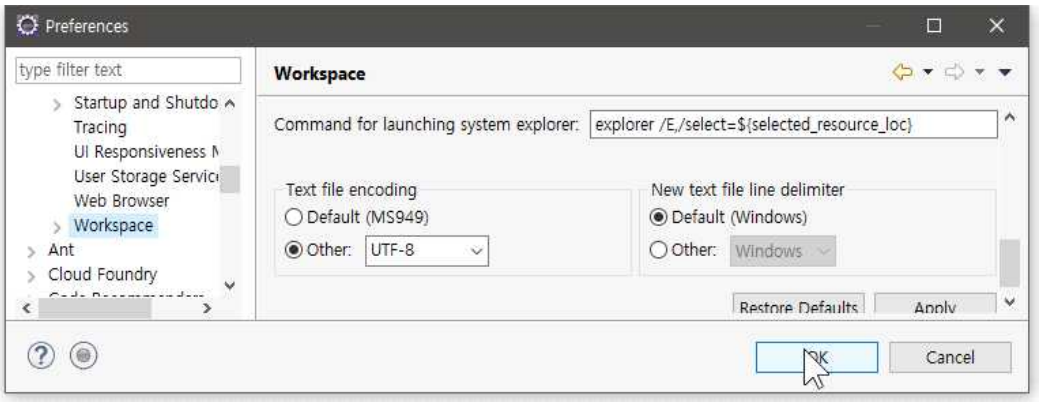
■ 압축풀어서 jbm 폴더에 가져다 놓기



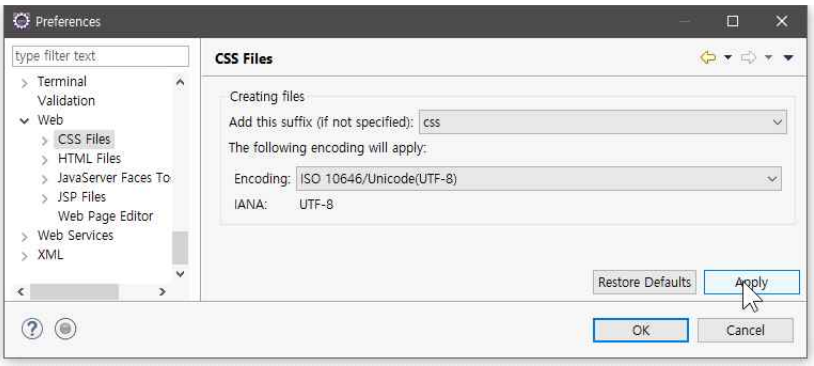
■ workspace 변경 및 perspective 변경

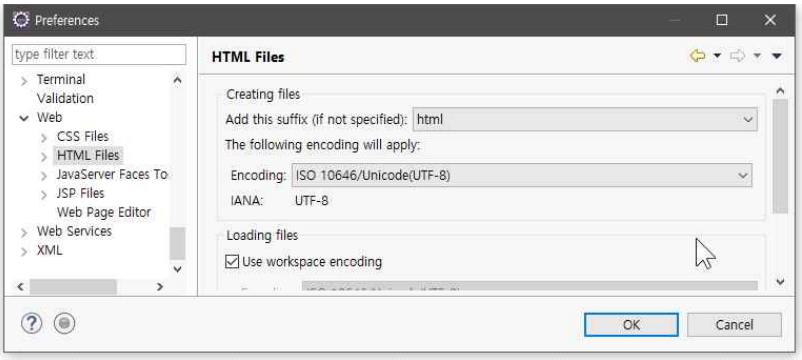


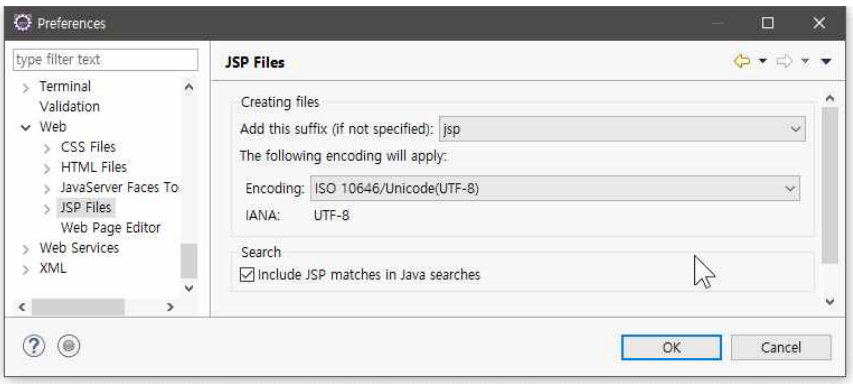
■ workspace의 text file encoding : UTF-8으로 변경



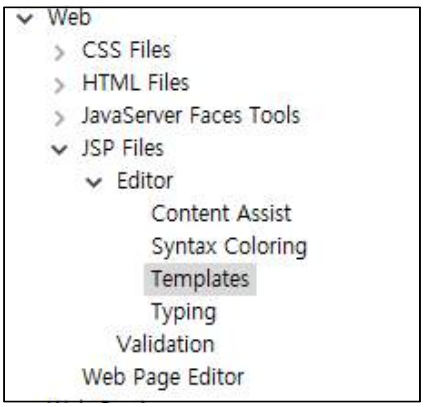
■ 코딩폰트 설정 / css / html / jsp 파일 인코딩 설정

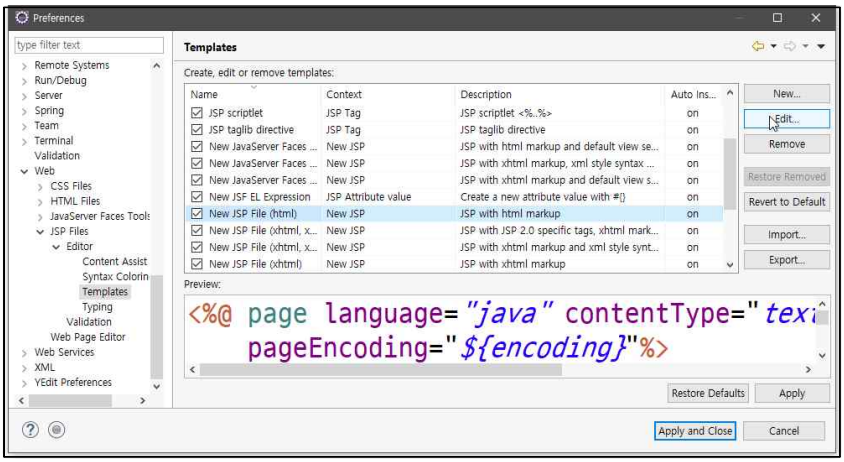


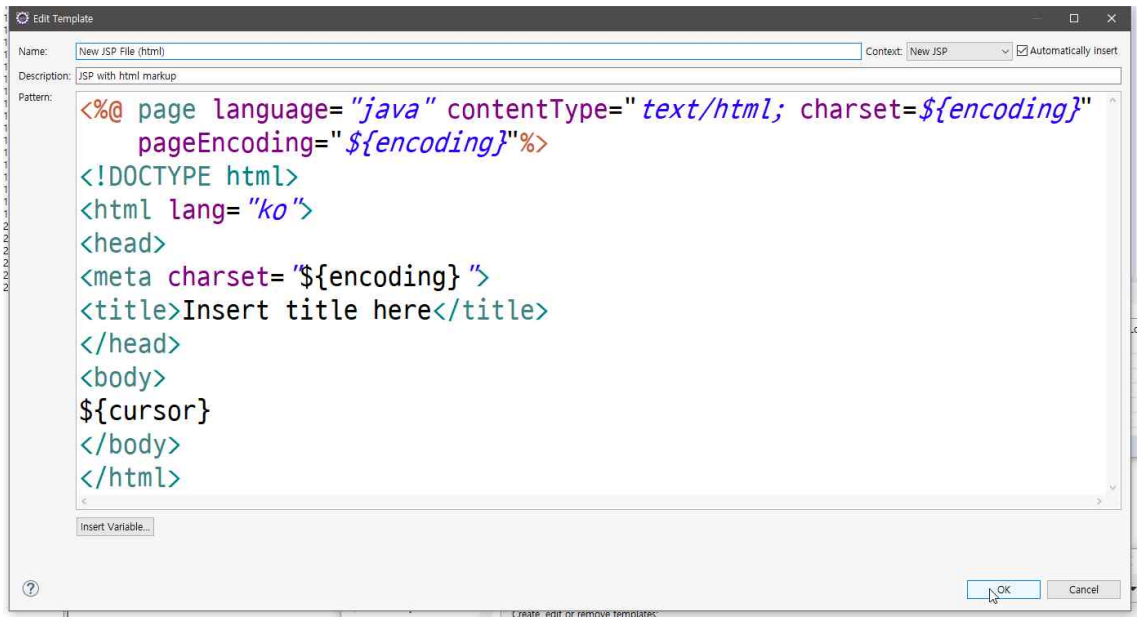




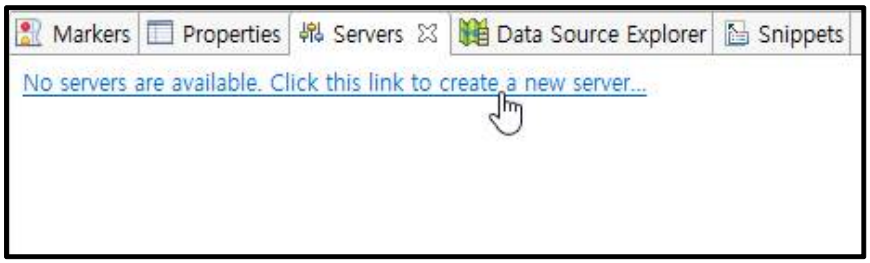
■ jsp(HTML 버전) 기본템플릿 변경

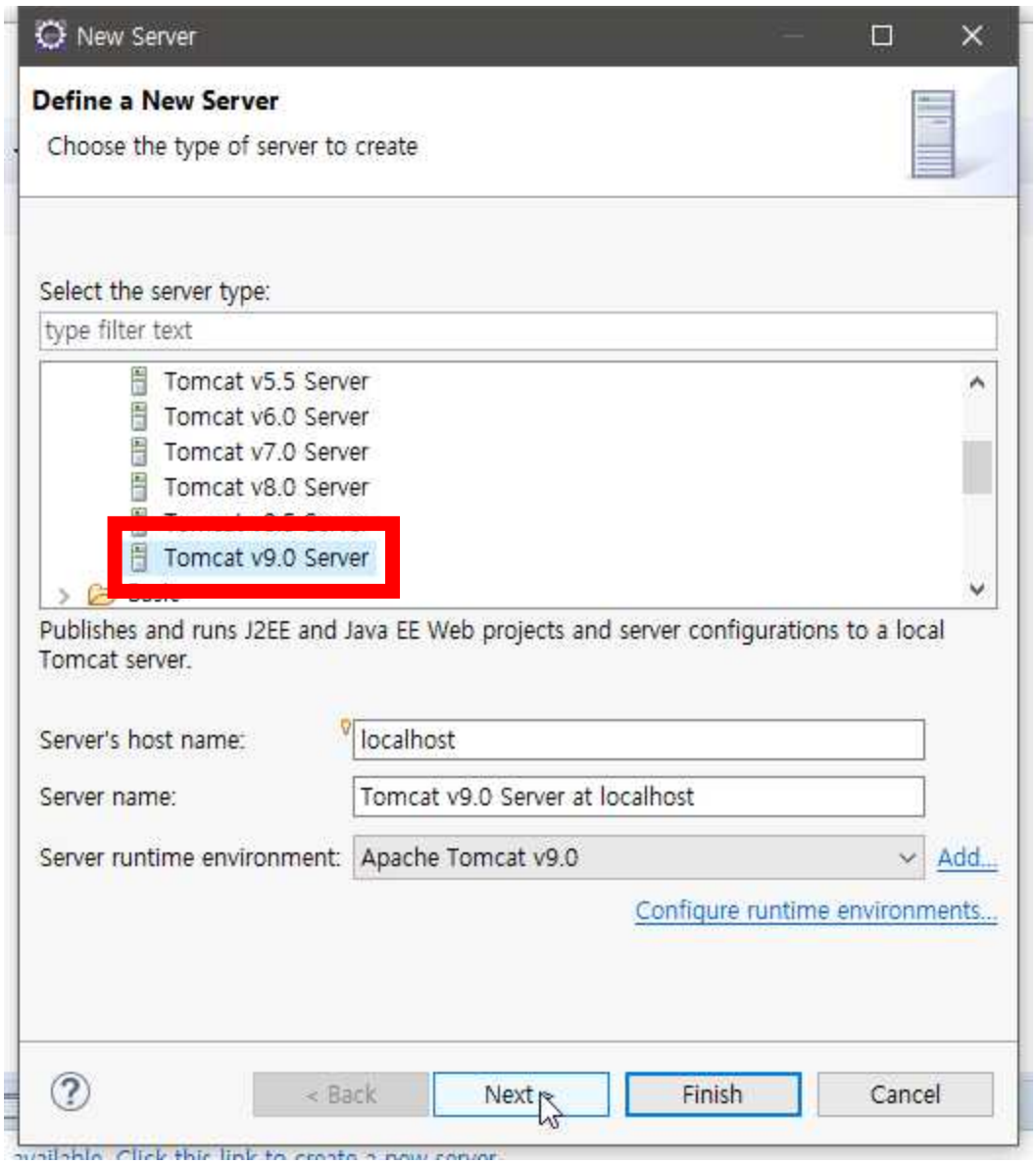


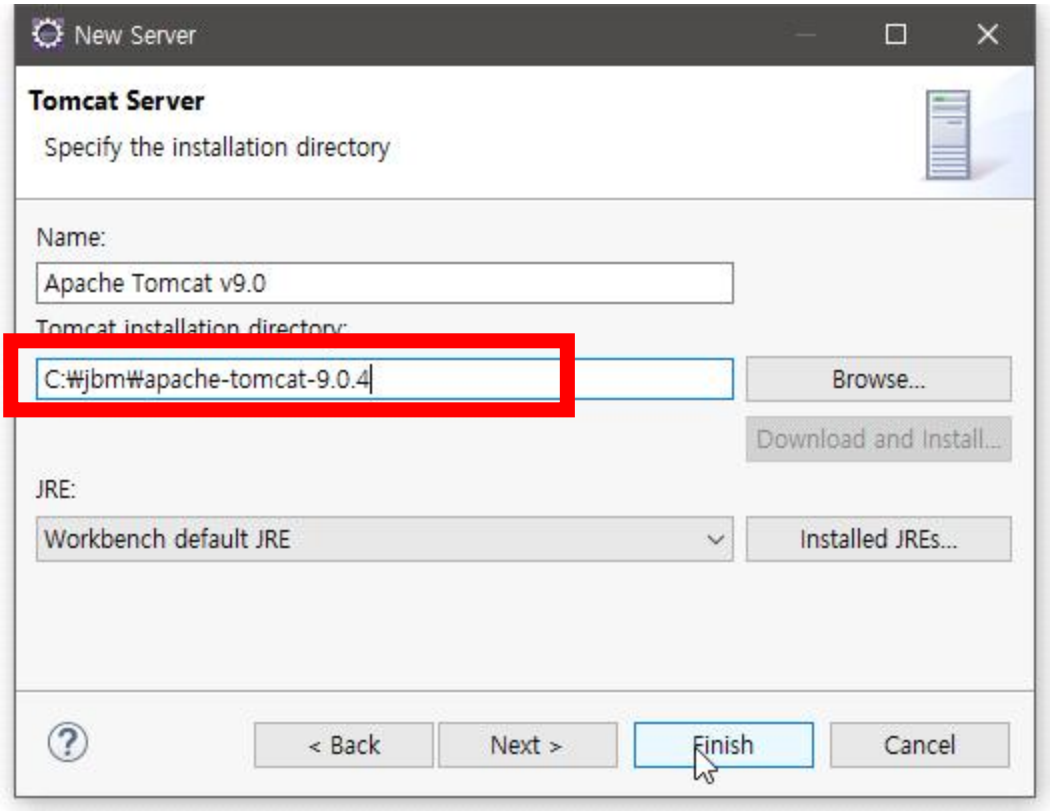




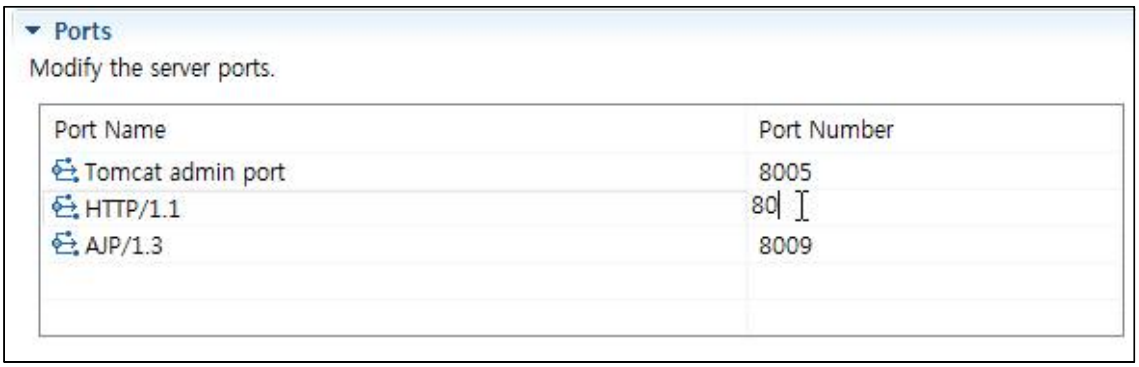
■ eclipse에서 server 설정



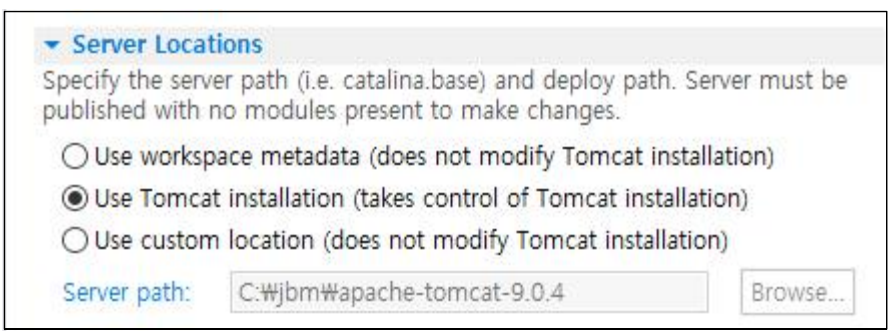




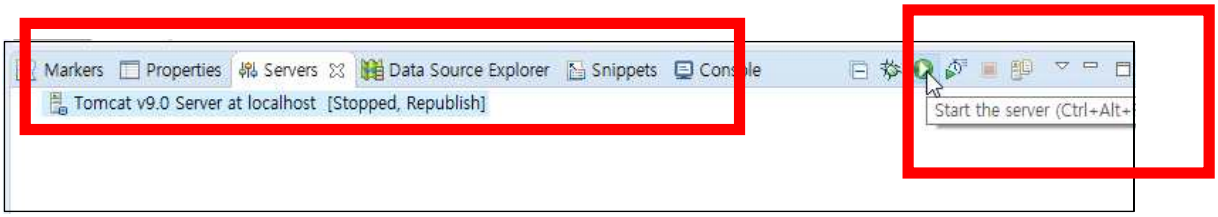
■ 8080포트를 오라클 apex 서비스가 차지하고 있기 때문에 포트 변경(80)



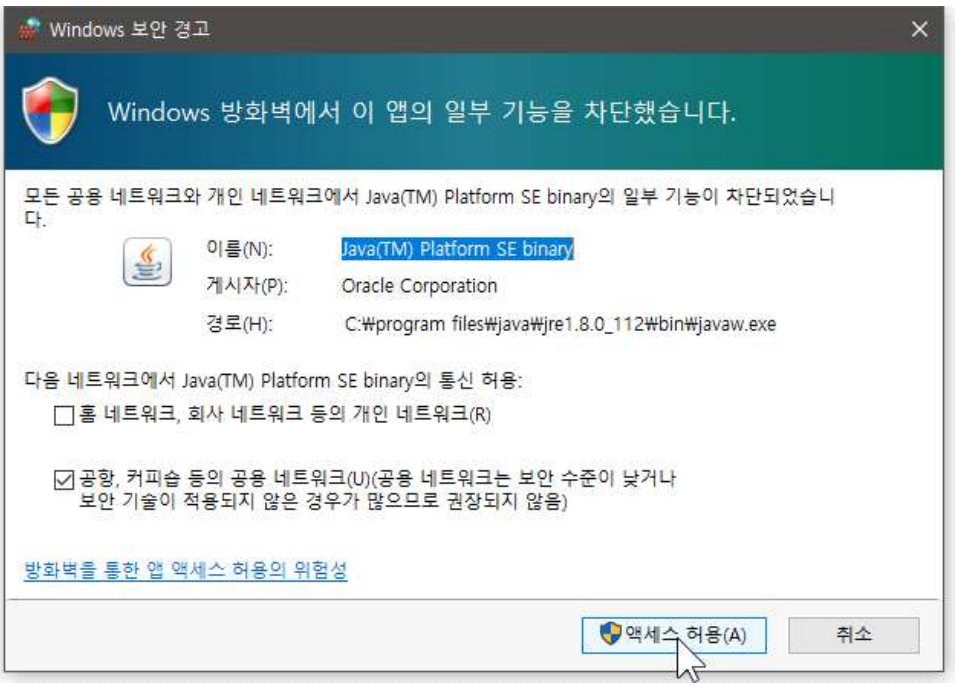
■ 기본 웹서버 실행 경로를 tomact경로로



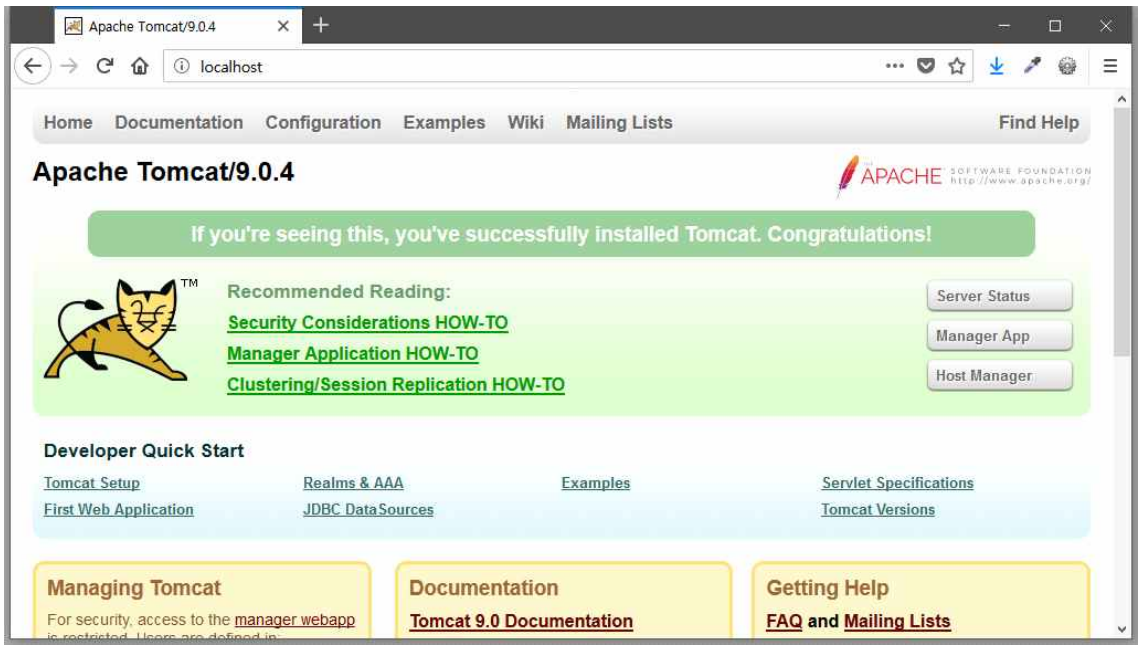
■ 톰캣 서버 실행



■ 방화벽 허용



■ localhost(127.0.0.1)로 주소 입력



HTTP 프로토콜

■ HTTP 프로토콜은 웹 서버와 웹 브라우저 사이의 통신에 사용하는 프로토콜이다.

■ HTTP 프로토콜은 요청(Request)과 응답(Response)으로 이루어진 프로토콜이다.

■ HTTP 요청은 HTTP 메소드, 요청 URL, 서버에 전달한 정보로 이루어져 있다.

■ HTTP 응답은 요청에 대한 서버의 응답으로서 상태코드, 컨텐츠 타입정보, 컨텐츠로 구성

1. 요청(Request)

```
Host: www.naver.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie:            npic=MBOVllsiTyi/qUX6pmeBsP2cvfLyfP+47v+cuhj4PiT/xdPxYDh0XPgflNhRNWHdCA==;
NNB=7AJVORCNLJQFQ; NM_THEMECAST_NEW=tcc_fod%2Ctcc_lif; nx_ssl=2
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

파일 업로드 Content-Type :multipart/form-data

2. 응답(Response)

```
Cache-Control: no-cache, no-store, must-revalidate
Connection: close
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Mon, 06 Feb 2017 16:05:26 GMT
P3P: CP="CAO DSP CURa ADMa TAIa PSAa OUR LAW STP PHY ONL UNI PUR FIN COM NAV INT DEM STA PRE"
Pragma: no-cache
Server: nginx
Transfer-Encoding: chunked
X-Frame-Options: SAMEORIGIN
```

※ 파일 다운로드시 Content-type : binary/octet-stream

Container(컨테이너)

■ 컨테이너는 웹서버와 서블릿 사이의 통신을 지원한다.

■ 컨테이너는 서블릿의 생명주기를 관리한다.

■ 컨테이너는 새로운 요청이 들어올 때 마다 자바 스레드를 생성해서 사용자의 요청을 처리함

■ 컨테이너는 선언적인 방법으로 보안관련 내용을 설정할 수 있다.

서블릿

■ 서블릿은 웹 서버에서 실행되는 자바 프로그램이다.

■ 서블릿은 사용자의 요구에 따라서 동적으로 웹 컨텐츠를 생산해서 클라이언트 측에 전송하는 자바 기술이다.

■ javax.servlet.http.HttpServlet 클래스를 상속받아서 사용자정의 서블릿 프로그램을 작성한다.

서블릿의 실행과정

■ 사용자가 url을 클릭해서 컨테이너에 새로운 요청을 한다.

■ 컨테이너는 요청을 접수한 다음, HttpServletRequest와 HttpServletResponse 객체를 생성한다.

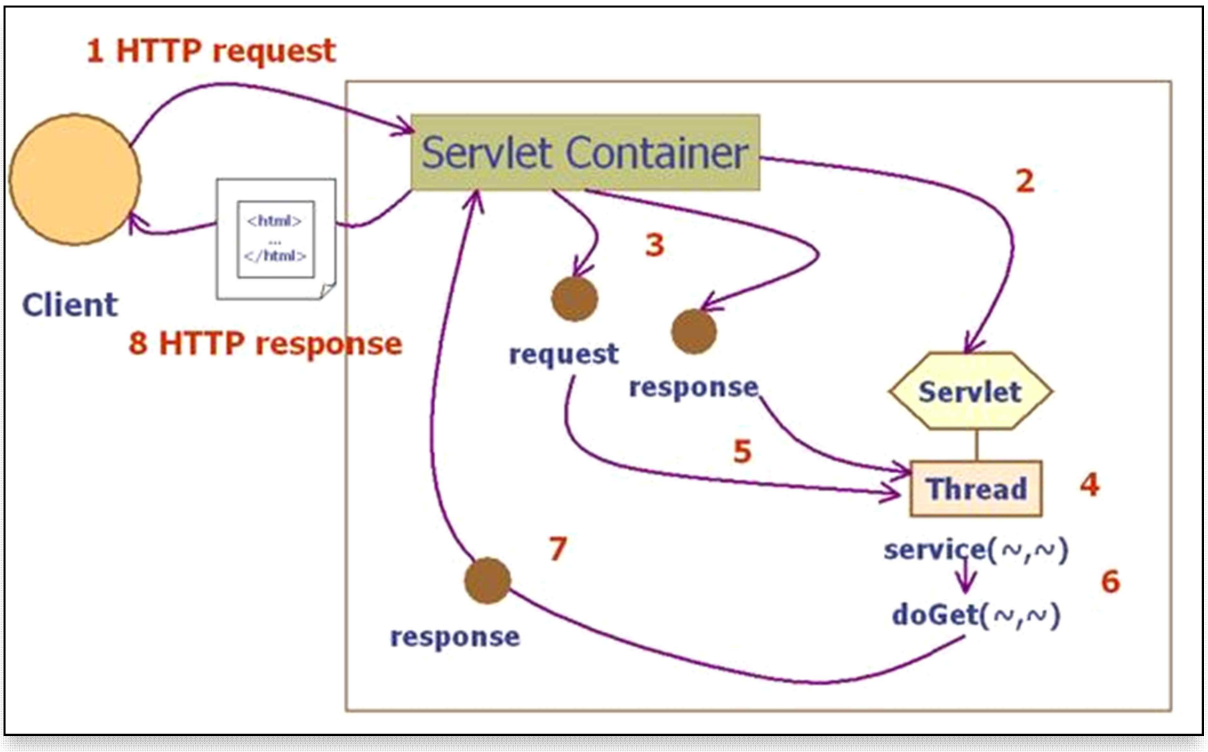
■ 접수된 url을 분석해서 해당 서블릿 객체를 생성하고, 사용자의 요청을 처리하기 위해 스레드를 생성한다.

■ 서블릿의 service() 메소드를 호출하고, 요청방식에 따라서 어떤 메소드를 호출할지 결정한다.

■ 서블릿의 doGet()메소드를 호출해서 사용자의 요청을 처리하고, 응답을 작성한다.

■ 작성된 응답을 클라이언트에 전송한다.

■ 사용자의 요청을 처리하기 위해 생성한 스레드를 소멸시킨다.



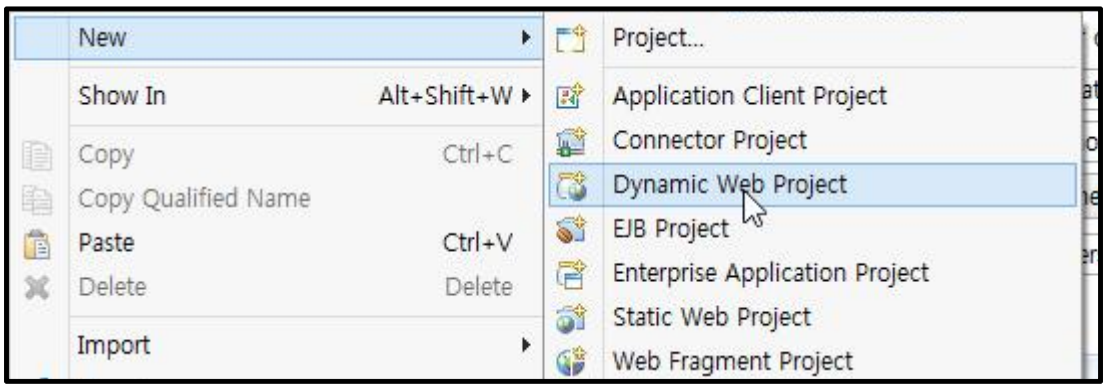
배포 서술자(web.xml)

■ 배포 서술자는 서블릿과 jsp를 어떻게 실행하는냐에 관한 정보가 들어있다.

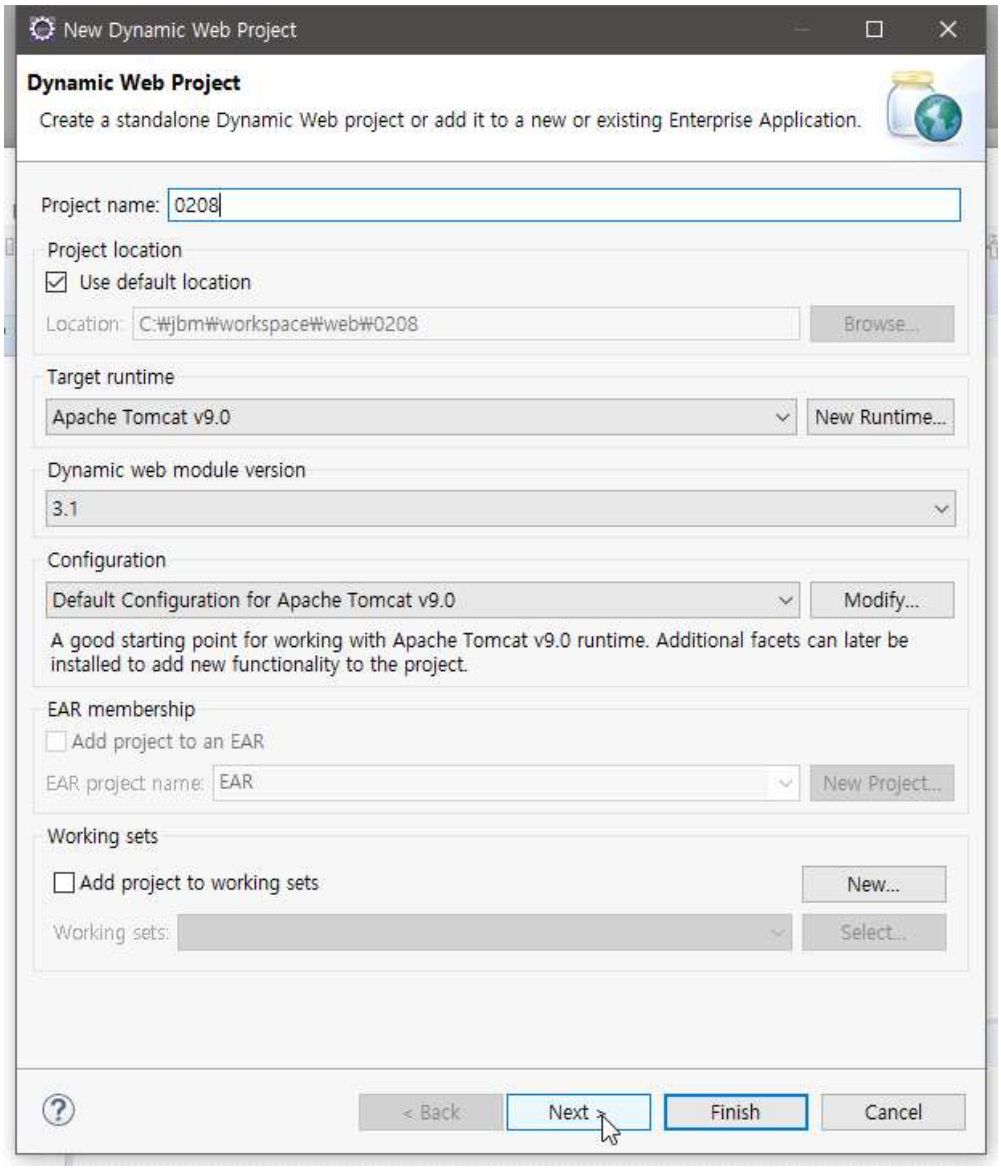
■ 배포 서술자에서는 서블릿과 URL을 매핑시키는 작업을 한다.

■ 배포 서술자는 URL 매핑외에 보안역할 설정, 오류 페이지 설정, 초기화 구성 등의 내용을 설정할 수 있다.

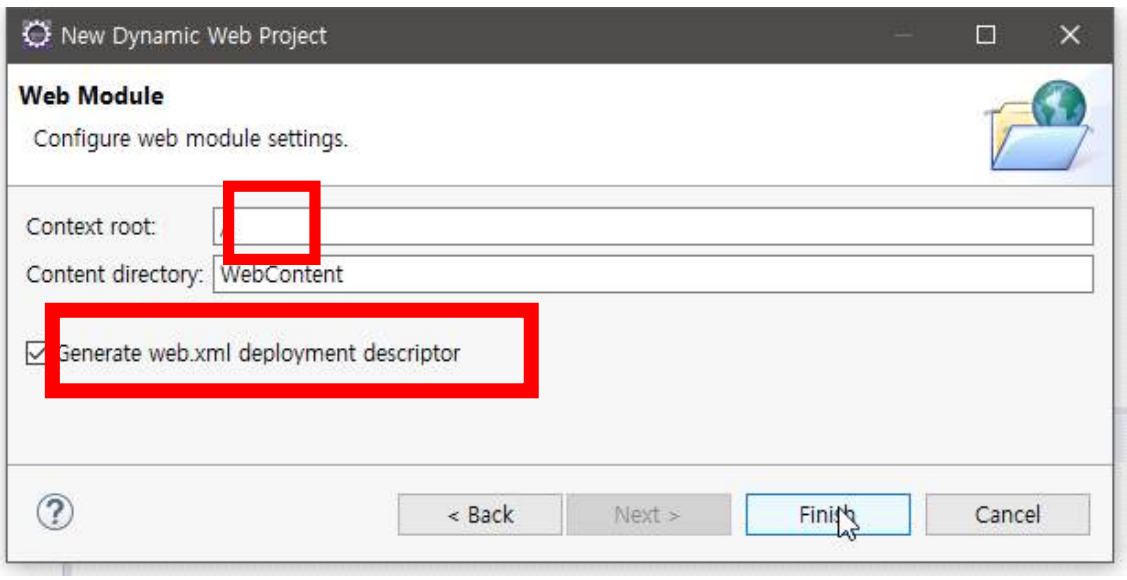
■ 프로젝트 생성



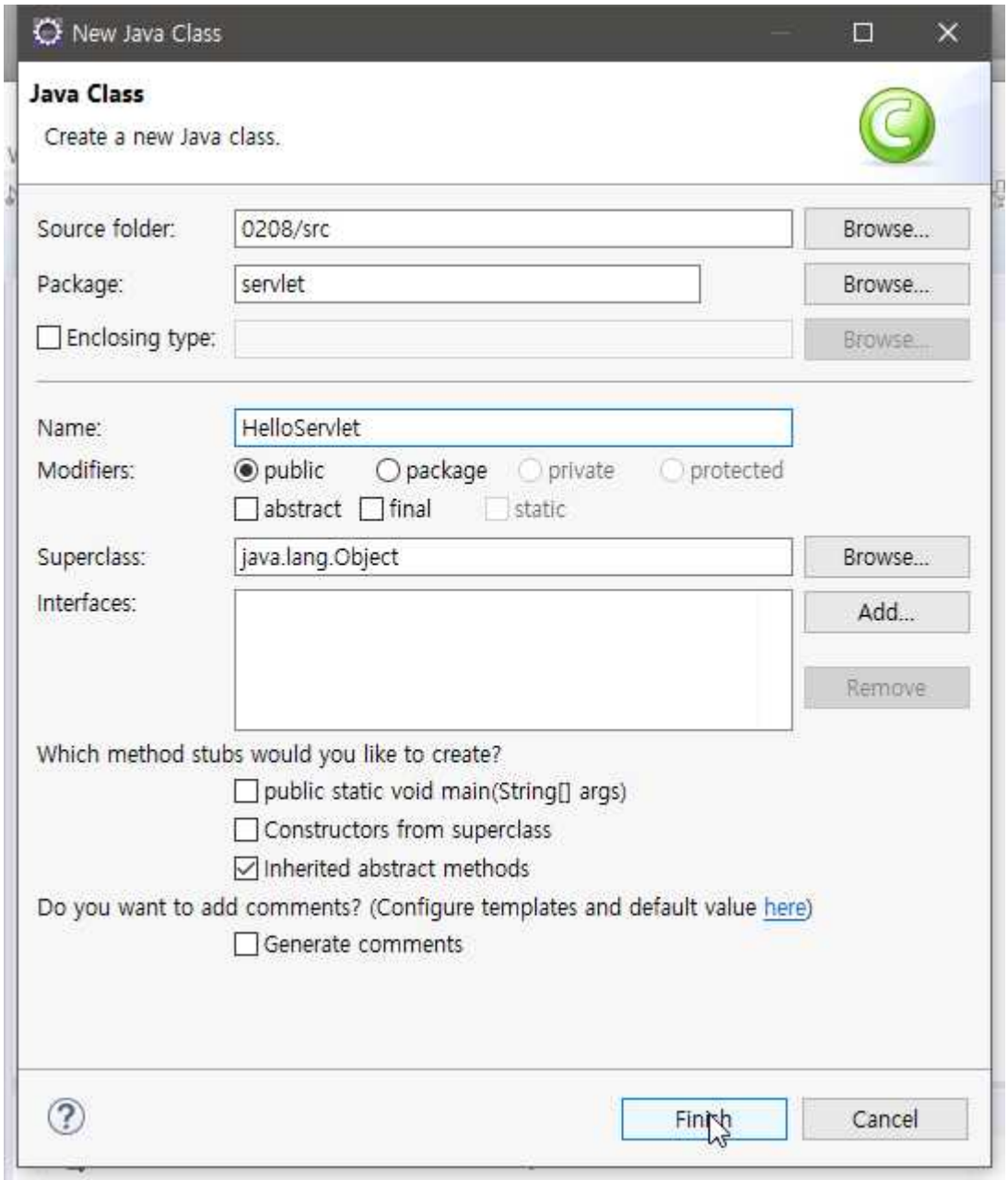
■ Dynamic Web Project 생성



■ 루트경로를 '/'로 & web.xml 생성



■ HelloServlet 생성하기

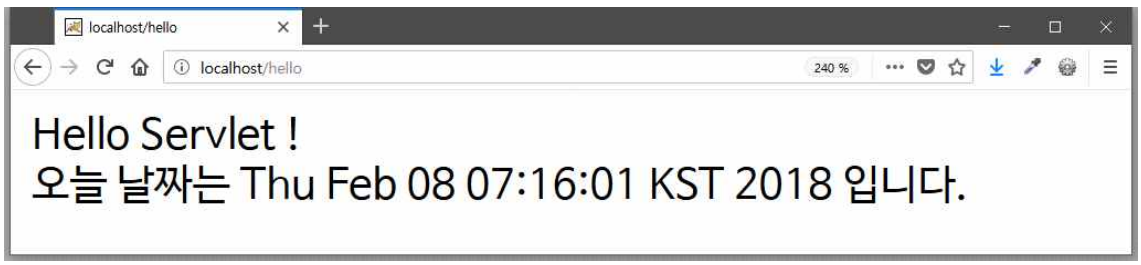


■ HelloServlet 작성

HelloServlet 예제

```
package hello;

public class HelloServlet extends HttpServlet {

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

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

                PrintWriter out = response.getWriter();
                Date date = new Date();
                out.println("<html>");
                out.println("<body>");
                out.println("Hello Servlet !");
                out.println("<br>");
                out.println("오늘 날짜는 " + date + " 입니다.");
                out.println("</body>");
                out.println("</html>");
        }
}
```

■ 서블릿 실행하기



■ web.xml(배포서술자) 보기

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
 http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <display-name>0208</display-name>
  <welcome-file-list>
       <welcome-file>hello</welcome-file>
  </welcome-file-list>

  <servlet>
       <servlet-name>hello</servlet-name>
       <servlet-class>servlet.HelloServlet</servlet-class>
  </servlet>

  <servlet-mapping>
       <servlet-name>hello</servlet-name>
       <url-pattern>/hello</url-pattern>
  </servlet-mapping>

</web-app>
```

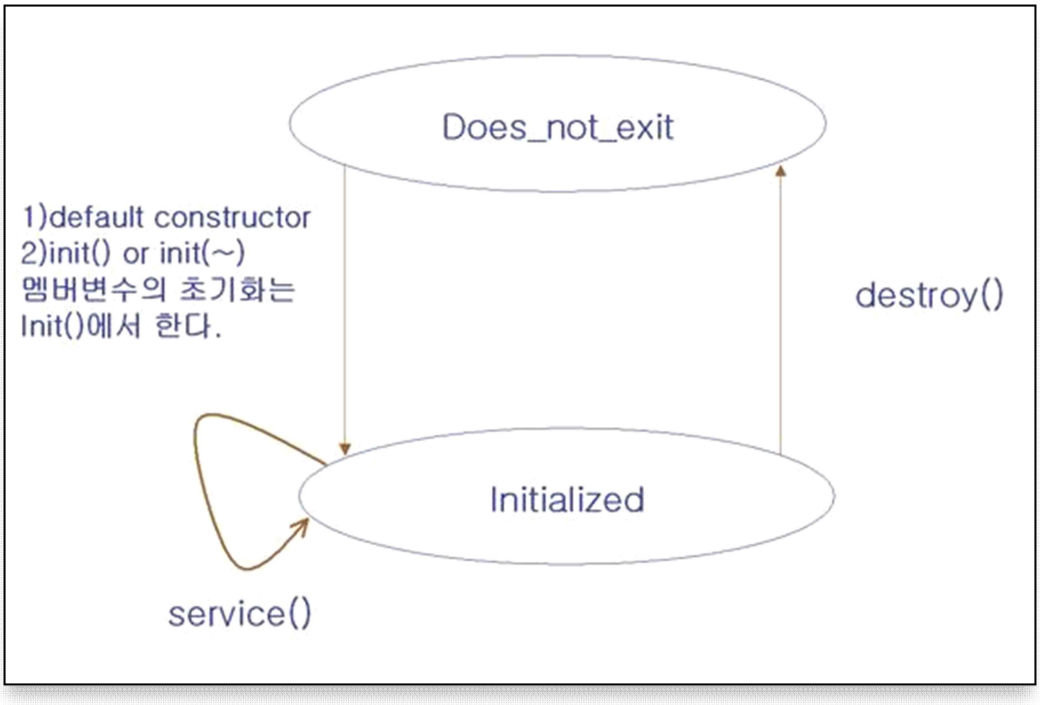
1) welcome-file-list : 첫페이지(예:http://naver.com/에서 실행되는 페이지)

2) servlet : 우리가 작성한 서블릿의 패키지 설정

3) servlet-mapping : url과 우리가 작성한 서블릿간의 맵핑

서블릿의 라이프 사이클

■ 서블릿의 상태



■ 서블릿의 라이프 사이클 관련 메소드

init() : 디폴트 생성자를 이용해서 서블릿 객체를 생성함으로 init()메소드를 사용해서 서블릿 객체를 초기화한다.

서블릿의 일생동안 단 한번 호출된다.

service() : HTTP 메소드를 참조하여 doGet()을 호출할지, doPost()를 호출할지 결정한다.

요청이 있을때 마다 호출된다.

destroy() : 서블릿이 소멸될때 호출된다. 자원해제와 관련된 작업을 한다.

서블릿의 일생동안 단 한번 호출된다.

■ 서블릿 라이프 사이클 예제

LifeCycleServlet 예제

```
package servlet;

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

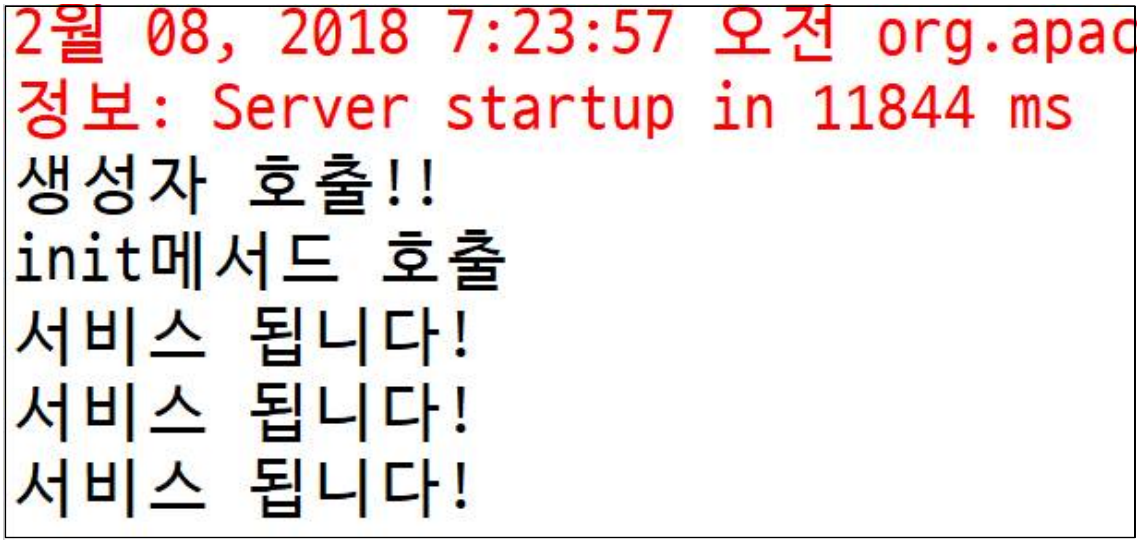
```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LifeCycleServlet extends HttpServlet {
	public LifeCycleServlet() {
		System.out.println("생성자 호출!!");
	}

	//init()메서드 : 생성된후에 서블릿을 초기화할때
	//필요한 메서드
	@Override
	public void init() {
		System.out.println("init메서드 호출");
	}

	//service()메서드 : 요청이 올때마다 호출되는 메서드
	@Override
	protected void service(HttpServletRequest arg0,
				HttpServletResponse arg1)
			throws ServletException, IOException {

		System.out.println("서비스 됩니다!");


	}

	//객체가 없어질때 destory() 메서드
	@Override
	public void destroy() {

		System.out.println("객체가 죽었어요ㅠ_ㅠ");

	}

}
```

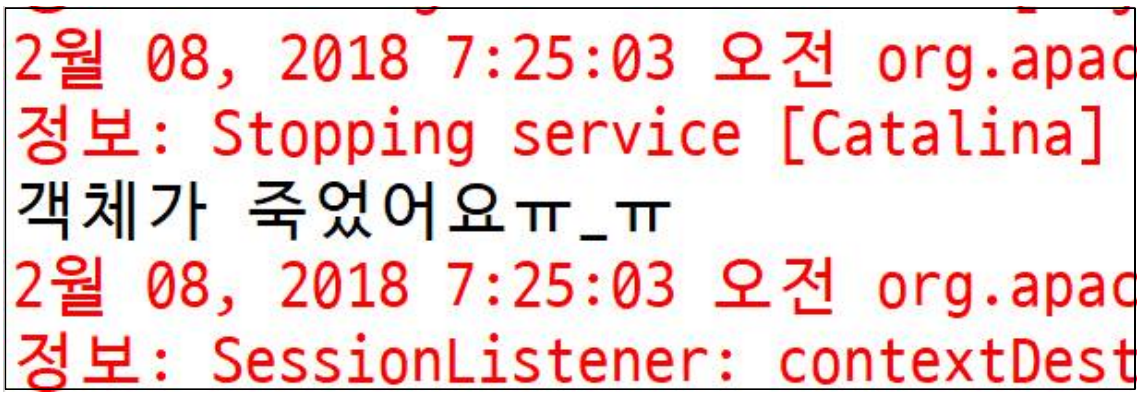
web.xml 설정하기

```
<servlet>
	<servlet-name>lifeCycle</servlet-name>
	<servlet-class>servlet.LifeCycleServlet</servlet-class>
</servlet>
<servlet-mapping>
```

```
        <servlet-name>lifeCycle</servlet-name>
        <url-pattern>/life</url-pattern>
</servlet-mapping>
```

■ 실행 결과





HttpServletRequest

■ 컨테이너는 HTTP요청으로 전달된 client의 요청관련 데이터를 HttpServletRequest객체를 생성해서 이름/값의 쌍으로 저장하고, 서블릿에 전달한다.

■ HttpServletRequest의 주요 메소드

반환형	메소드 명	설명
String	getParameter(String name)	요청 파라미터에서 지정된 이름의 파라미터 값을 반환한다.
String[]	getParameterValues(String name)	요청 파라미터에서 지정된 이름의 파라미터의 모든 값을 배열의 형태로 반환한다.
HttpSession	getSession()	HttpSession객체를 반환한다. HttpSession객체가 존재하지 않으면 새로운 세션 객체를 생성한 후 반환한다.
HttpSession	getSession(boolean value)	HttpSession객체를 반환한다. false로 지정된 경우 해당 클라이언트에 대해 생성된 세션객체가 없으면 null을 반환한다.
Cookie[]	getCookies()	클라이언트의 요청에 포함된 쿠키를 배열로 반환한다.
String	getRequestURL()	요청에 사용된 URL을 문자열로 반환한다.
String	getQueryString()	요청에 사용된 쿼리 문자열을 반환한다.
String	getMethod()	요청에 사용된 요청방식(GET, POST, ...)을 문자열로 반환한다.
void	setCharacterEncoding(String enc)	form을 통해서 전달된 데이터의 지정된 인코딩 방식으로 인코딩하는 메소드

HttpServletResponse

■ HttpServletResponse 객체는 요청을 시도한 클라이언트로 전송할 응답(컨텐츠)을 나타내는 객체다.

■ HttpServletResponse 객체는 클라이언트로 전송될 응답헤더정보를 설정하고, 응답을 위한 스트림을 얻거나, 세션관리를 위한 추가정보들을 설정할 수 있다.

■ HttpServletResponse의 주요 메소드

반환형	메소드 명	설명
void	setContentType(String type)	응답으로 보내질 컨텐츠의 타입과 캐릭터셋을 지정한다.
OutputStream	getOutputStream()	요청에 대한 바이트 출력용 스트림을 얻는다.
PrintWriter	getWriter()	요청에 대한 문자 출력용 스트림을 얻는다.

void	sendRedirect(String url)	지정된 url로 요청을 redirect 시킨다.

■ HttpServletRequest와 HttpServletResponse을 이용한 방명록

방명록 작성 페이지(write.html)

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>방명록 작성하기</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <form method="post" action="/myWeb/visitInsert">
      <table align="center" width="500" border="1">
        <tr>
            <th width="70">작성자</th>
        <td width="430">
           <input type="text" name="writer" size="50">
        </td>
        </tr>
        <tr>
        <th>내용</th>
        <td>
           <textarea rows="7" cols="50" name="memo"></textarea>
         </td>
        </tr>
        <tr>
         <td colspan="2" align="center">
           <input type="submit" value="저장">  
           <input type="reset" value="취소">
         </td>
         </tr>
    </table>
    </form>
  </body>
</html>
```

방명록 추가 서블릿(VisitInsertServlet.java)

```
package visit;
```

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class VisitInsertServlet extends HttpServlet {


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

        request.setCharacterEncoding("utf-8");

        //client가 http요청으로 전송한 값 읽기
        String writer = request.getParameter("writer");
        String memo = request.getParameter("memo");

        System.out.println("작성자 : " + writer);
        System.out.println("내용 : " + memo);

                StringBuffer sql = new StringBuffer();
                sql.append("insert into visit(no, writer, memo, regdate) ");
                sql.append("values (visit_seq.nextval, ?, ?, sysdate)");

                Connection con = null;
                PreparedStatement pstmt = null;
                try{
                        Class.forName("oracle.jdbc.OracleDriver");
                        con
                        = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
                                                                "scott", "tiger");
                        pstmt = con.prepareStatement(sql.toString());
                        pstmt.setString(1, writer);
                        pstmt.setString(2, memo);

                        pstmt.executeUpdate();
                }catch(SQLException e){
                        e.printStackTrace();
                }catch(ClassNotFoundException e){
                        e.printStackTrace();
```

```
                }finally{
                        try{if(pstmt != null)pstmt.close();}catch(SQLException e){}
                        try{if(con != null)con.close();}catch(SQLException e){}
                }

        response.sendRedirect("visitList");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }


    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

방명록 리스트 보기 서블릿(VisitListServlet.java)

```
package visit;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class VisitListServlet extends HttpServlet {

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

```
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {


            out.println("<html>");
            out.println("<head><title>방명록 리스트</title></head>");
            out.println("<body>");

            StringBuffer sql = new StringBuffer();
            sql.append("select no, writer, memo, regdate ");
            sql.append("from visit ");
            sql.append("order by no desc ");

            Connection con = null;
            PreparedStatement pstmt = null;
            ResultSet rs = null;

            try{
                Class.forName("oracle.jdbc.OracleDriver");
                con
                  = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
                                                        "scott", "tiger");
                pstmt = con.prepareStatement(sql.toString());
                rs = pstmt.executeQuery();

                while(rs.next()){
                    int no = rs.getInt("no");
                    String writer = rs.getString("writer");
                    String memo = rs.getString("memo");
                    Date regdate = rs.getDate("regdate");

                    out.println("<table align=center width=500 border=1>");
                    out.println("<tr>");
                    out.println("<th width=50>번호</th>");
                    out.println("<td width=50 align=center>"+ no +"</td>");
                    out.println("<th width=70>작성자</th>");
                    out.println("<td width=180 align=center>"+ writer +"</td>");
                    out.println("<th width=50>날짜</th>");
                    out.println("<td width=100 align=center>"+ regdate+"</td>");
                    out.println("</tr>");
                    out.println("<tr>");
                    out.println("<th width=50>내용</th>");
                    out.println("<td   colspan=5>       <textarea   rows=7   cols=50>"+   memo
+"</textarea></td>");
                    out.println("</tr>");
```

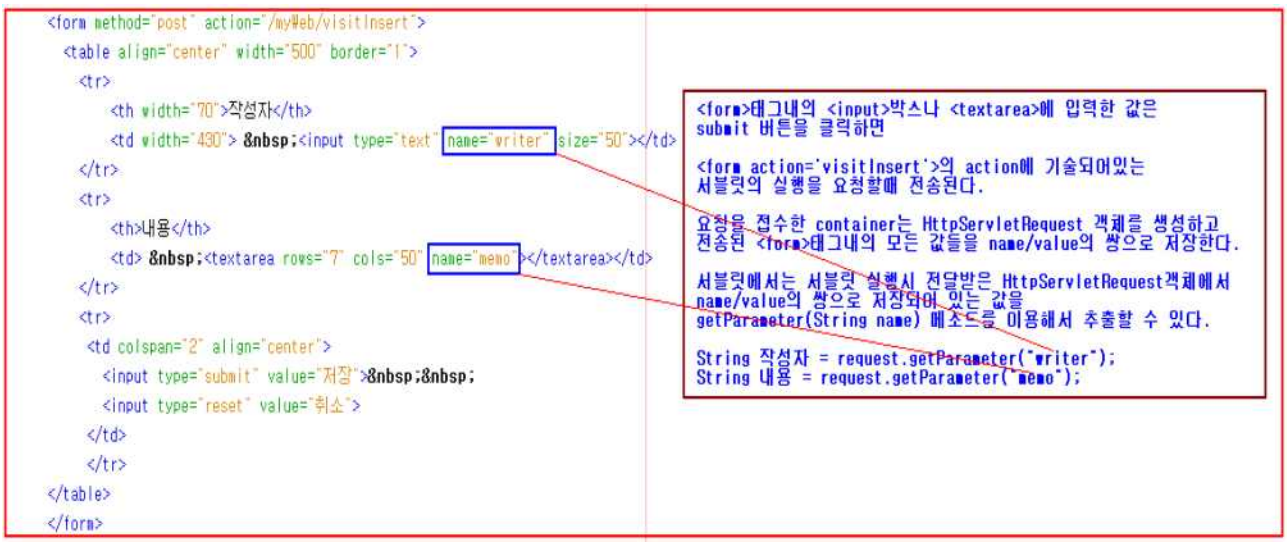
```
                    out.println("</table>");
                    out.println("<p>");
                }

            }catch(SQLException e){
                e.printStackTrace();
            }catch(ClassNotFoundException e){
                e.printStackTrace();
            }finally{
                try{if(pstmt != null)pstmt.close();}catch(SQLException e){}
                try{if(con != null)con.close();}catch(SQLException e){}
            }

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

        } finally {
            out.close();
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

web.xml 설정

```
<servlet>
        <servlet-name>visitInsert</servlet-name>
        <servlet-class>visit.VisitInsertServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>visitInsert</servlet-name>
        <url-pattern>/visitInsert</url-pattern>
</servlet-mapping>

<servlet>
```

```
        <servlet-name>visitList</servlet-name>
        <servlet-class>visit.VisitListServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>visitList</servlet-name>
        <url-pattern>/visitList</url-pattern>
</servlet-mapping>
```

■ Form 요소와 HttpServletRequest



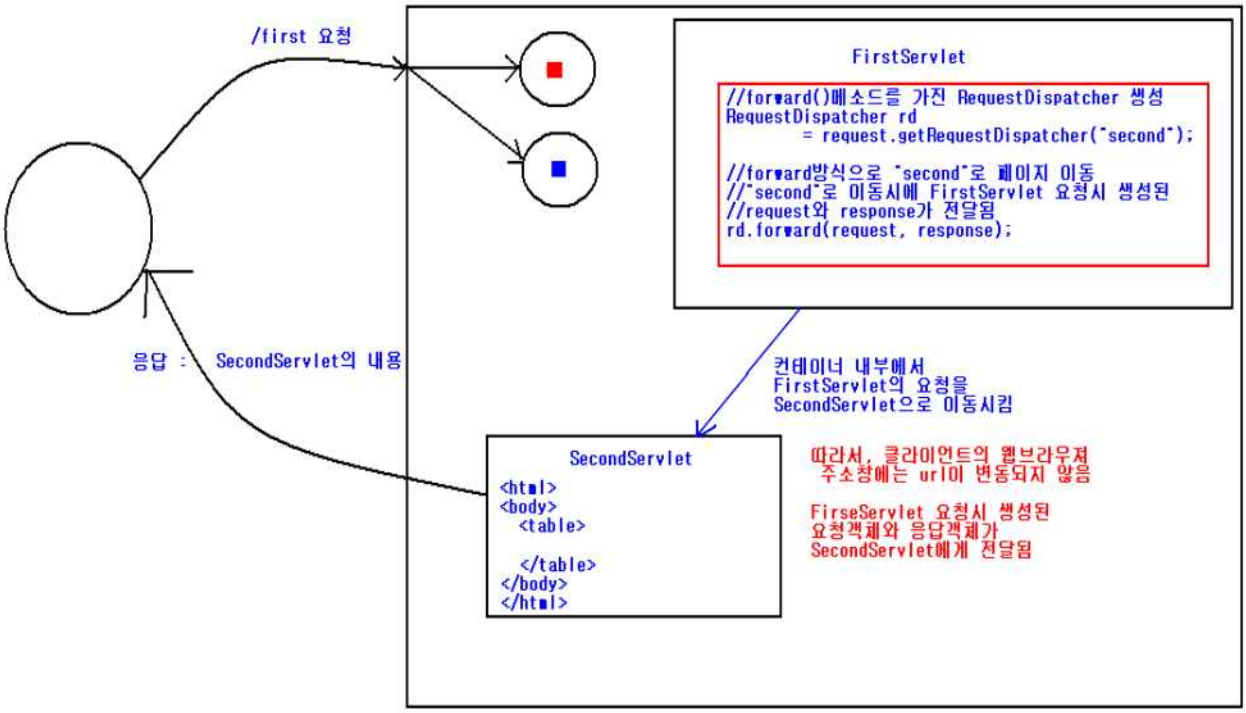
페이지 이동방법

■ 메소드 호출을 통해서 페이지를 이동할 수 있는 방법에는 두 가지 방법이 존재한다.

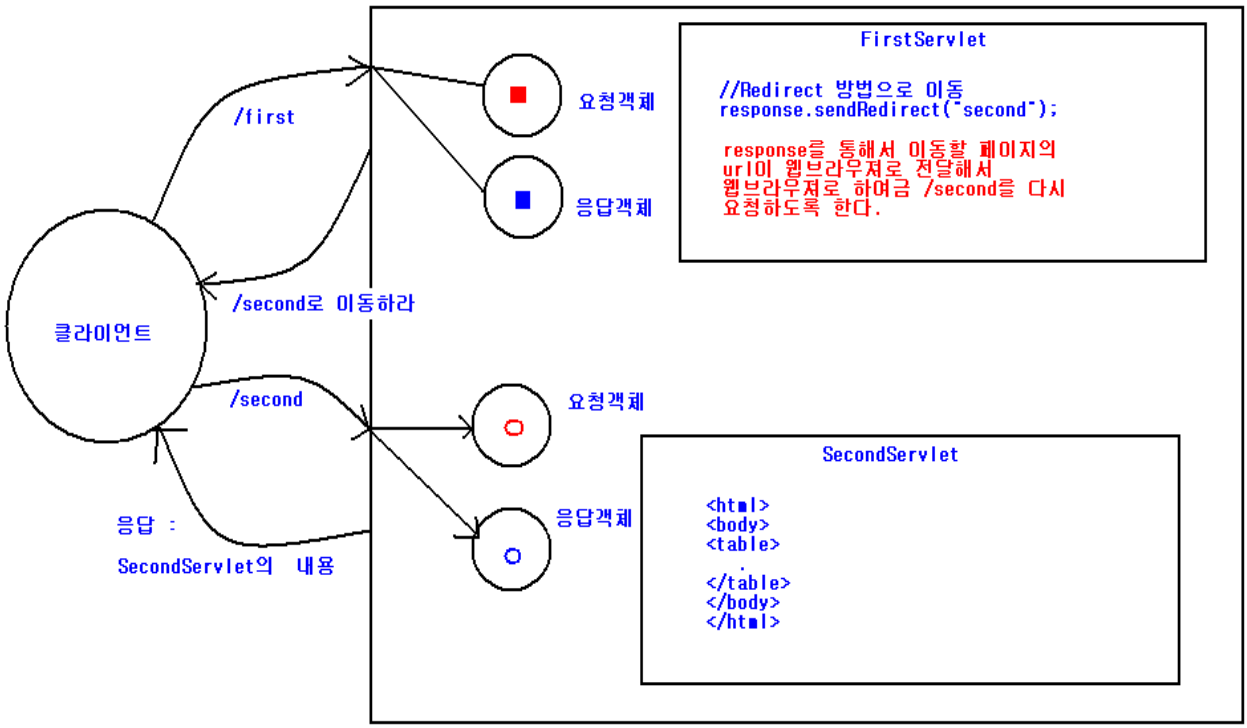
■ forward 방법의 이동과 redirect 방법의 이동

구분	forward	redirect
url	url이 바뀌지 않는다	url이 바뀐다
요청객체와 응답객체	유지된다	유지되지 않는다
속도	빠르다	느리다
소속	요청객체	응답객체

■ forward 방법



■ redirect 방법



■ 이동방법 예제

FirstServlet

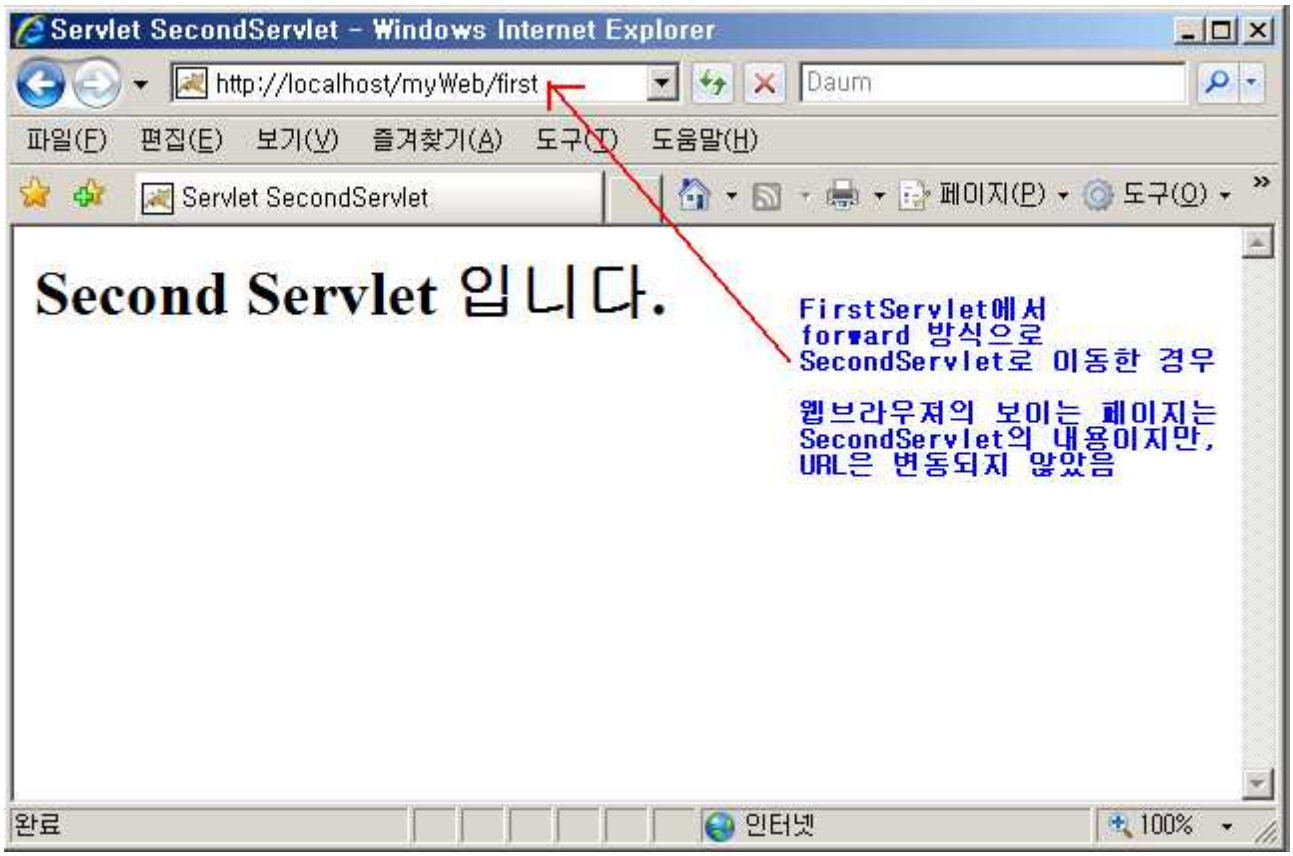
```
package forwardRedirect;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FirstServlet extends HttpServlet {

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

        System.out.println("FirstServlet 수행됨");

        //페이지 이동

        //1. forward 방식으로 이동
        RequestDispatcher rd = request.getRequestDispatcher("second");
        rd.forward(request, response);
    }



    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

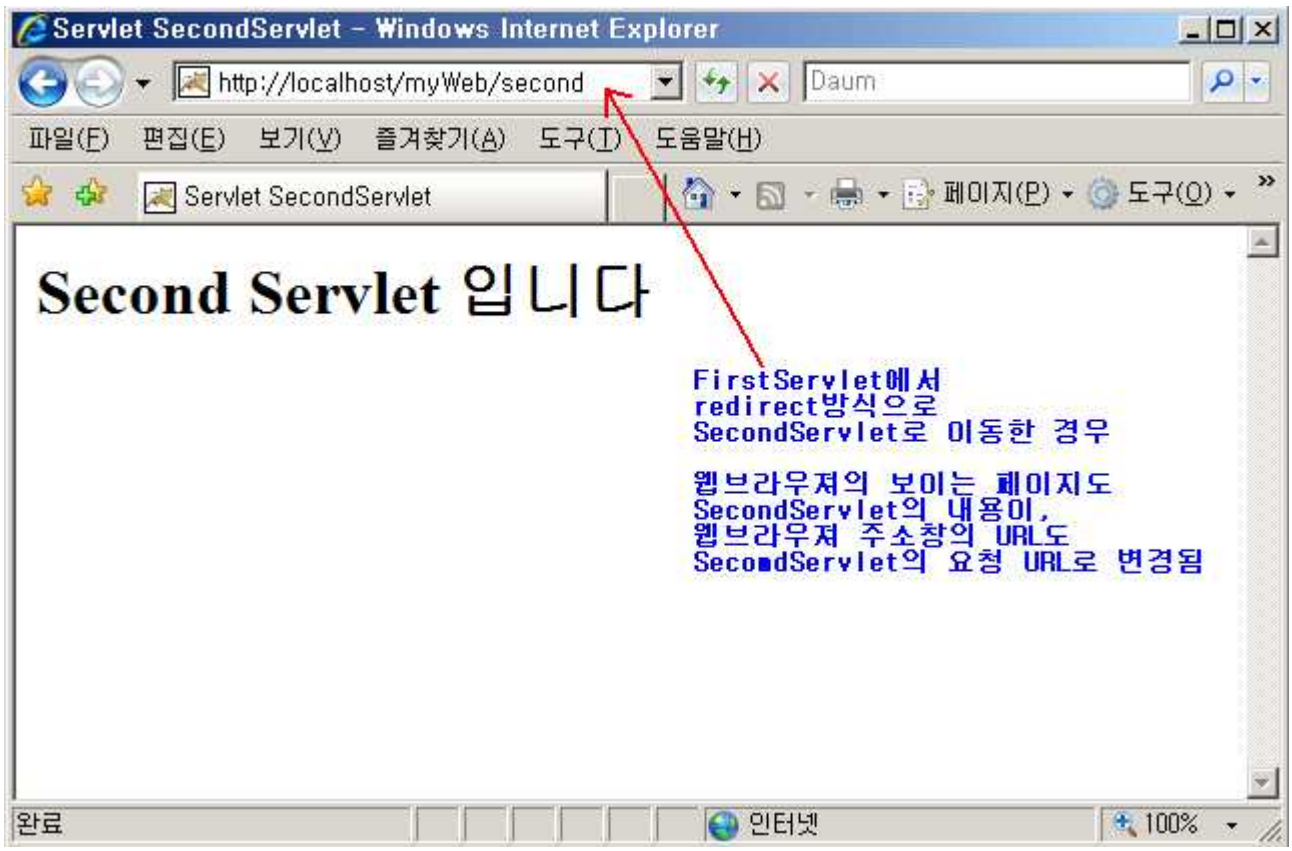
SecondServlet

```
package forwardRedirect;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SecondServlet extends HttpServlet {



    protected void processRequest(HttpServletRequest request,
            HttpServletResponse response)  throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {

            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet SecondServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1> Second Servlet 입니다</h1>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }


    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }


    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }


}
```

■ 실행 결과


Servlet SecondServlet - Windows Internet Explorer
http://localhost/myWeb/first
Daum
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
Servlet SecondServlet
페이지(P) 도구(O)
Second Servlet 입니다.
FirstServlet에서
forward 방식으로
SecondServlet로 이동한 경우
웹브라우저의 보이는 페이지는
SecondServlet의 내용이지만,
URL은 변동되지 않았음
완료
인터넷
100%


Servlet SecondServlet - Windows Internet Explorer
http://localhost/myWeb/second
Daum
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
Servlet SecondServlet
페이지(P) 도구(O)
Second Servlet 입니다
FirstServlet에서
redirect방식으로
SecondServlet로 이동한 경우
웹브라우저의 보이는 페이지도
SecondServlet의 내용이,
웹브라우저 주소창의 URL도
SecondServlet의 요청 URL로 변경됨
완료
인터넷
100%

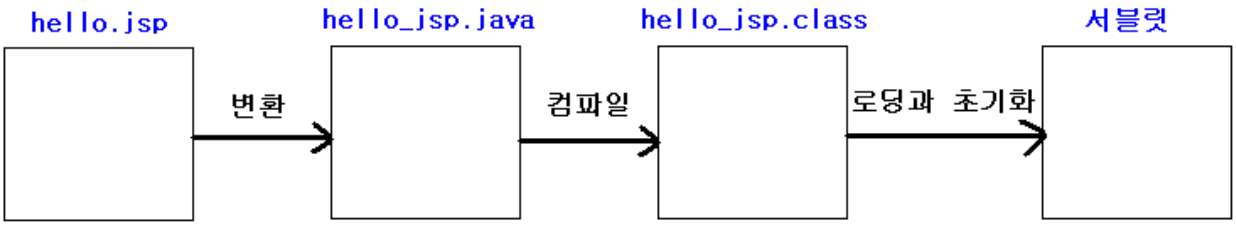
JSP(java server pages)

■ JSP는 서블릿과 동일하게 동적인 웹 페이지를 개발하기 위한 기술이다.

■ JSP는 HTML 태그내에 자바 코드를 삽입하는 형식이므로, HTML과 자바코드의 분기가 가능하고, 서블릿에서 HTML을 작성하는 것과 같은 번거로움이 없다.

■ 작성된 JSP는 클라이언트의 요청이 접수되면, 웹 어플리케이션 서버에 의해서 서블릿 소스코드로 변환되어 서블릿 클래스로 컴파일된 후에 실행된다.

■ JSP의 동작 원리

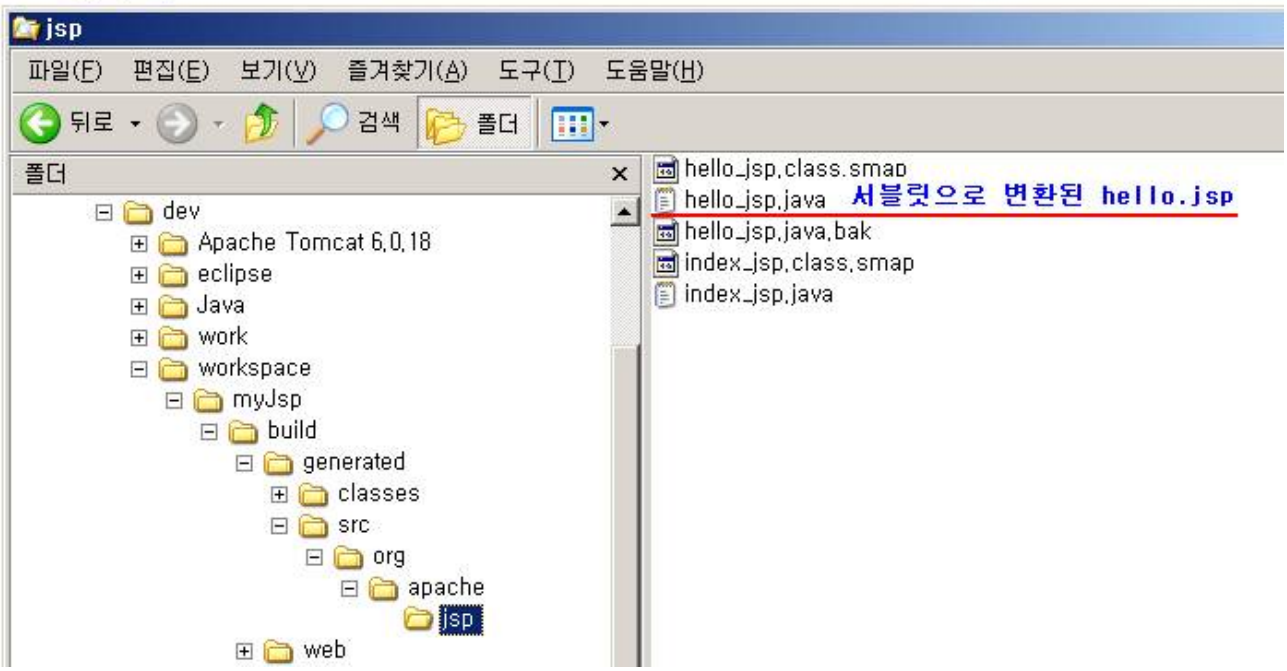


■ hello.jsp 작성하기

hello.jsp

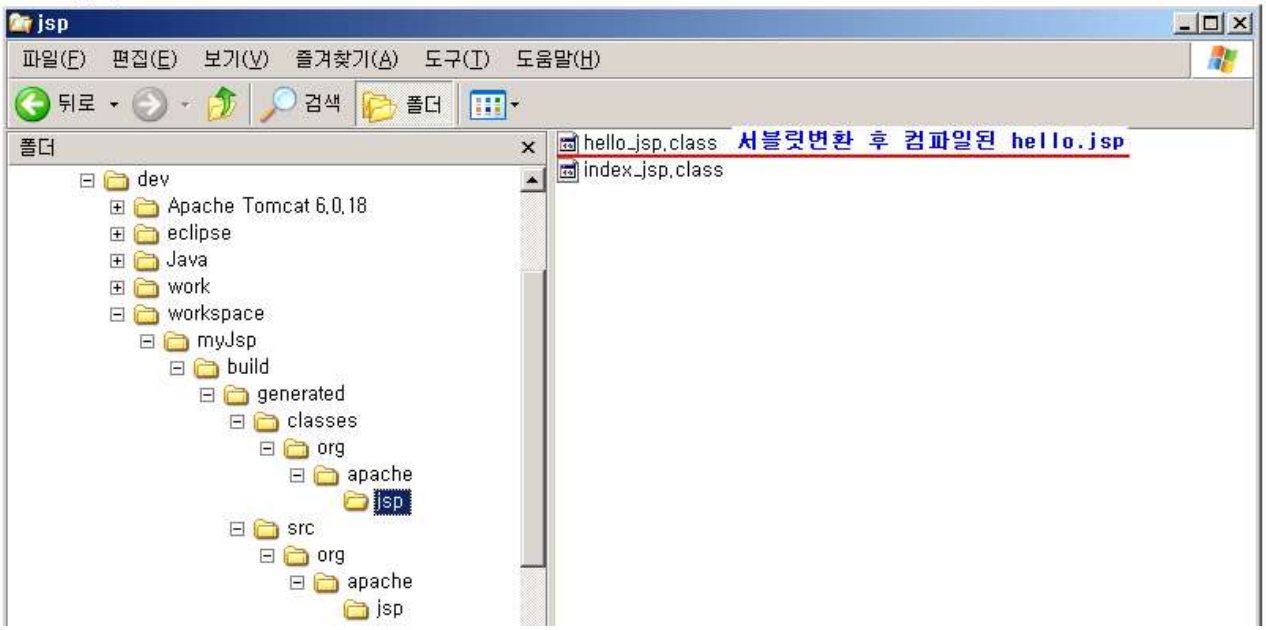
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>My First JSP Page!</h1>
    </body>
</html>
```

hell_jsp.java




```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

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

    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;

    try {
      response.setContentType("text/html;charset=UTF-8");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("\n");
      out.write("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"\n");
      out.write("    \"http://www.w3.org/TR/html4/loose.dtd\">\n");
      out.write("\n");
      out.write("<html>\n");
      out.write("    <head>\n");
      out.write("        <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\n");
      out.write("        <title>JSP Page</title>\n");
      out.write("    </head>\n");
      out.write("    <body>\n");
      out.write("        <h1>My First JSP Page!</h1>\n");
      out.write("    </body>\n");
      out.write("</html>\n");
```

hello_jsp.class



■ hello.jsp 실행하기

JSP 기본 문법

■ 주석

▮ html 주석 : 웹브라우져가 해석하지 않는다. 소스보기 창에 나타남

<!-- 주석문 -->

▮ JSP 주석 : JSP가 서블릿 코드로 변환될 때 변환에 참여하지 않음

<%-- 주석문 --%>

▮ 자바 주석 : JSP내의 자바 코딩 영역에서 사용하는 주석

// 주석문

■ JSP 지시어(Directive)

JSP페이지 내에서 JSP Container에게 해당 페이지를 어떻게 처리할 것인가에 대한 정보를 알려주는데 사용된다

▮ **page 지시어** : 페이지 관련 환경을 정의한다.

<%@ page contentType="text/html" pageEncoding="UTF-8"%>

주로 문자 인코딩, 응답페이지의 컨텐츠타입 등을 정의한다.

1. **language** => jsp페이지에서 사용되는 스크립팅 언어를 지정한다. (현재 자바만 지원)
Default) <%@ page language="java" %>
xml 표현법) <jsp:directive.page language="java" />

2. **extends** => jsp페이지가 상속받을 부모 클래스를 지정한다. Extends 속성을 생략하면 JSP컨테이너는 HttpJspBase 클래스를 구현한 클래스로 자동으로 지정한다.
extends 속성은 거의 사용하지 않은 이유는 JSP컨테이너가 자동으로 HttpJspBase 클래스를 구현한 클래스를 지정해주기 때문이다.
Example) <%@ page extends="single.SingleThreadServlet1" %>
xml 표현법) <jsp:directive.page extends="single.SingleThreadServlet1" />

3. **import** => import할 패키지를 명시하면 된다. 기본적으로 3개의 패키지를 import하고 있다.
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
Example) <%@ page import="java.util.ArrayList" %>

4. **session** => HttpSession 객체를 사용할 것인지를 결정한다.
session속성 값이 true 세션객체를 사용하고,false 사용하지 않는다.
Default) <%@ page session="true" %>

5. **buffer** => JSP페이지의 출력 버퍼의 크기를 설정한다. 디폴트는 8kb이다.
buffer 속성이 none으로 설정되면,JspWriter 객체를 이용한 출력 버퍼시 버퍼를 이용하지 않는다. 즉, JSP 페이지로 부터 출력되는 모든 내용은 즉시 클라이언트에 응답한다.
Default)<%@ page buffer="8kb" %>

6. **autoFlush** => 버퍼가 다 찼을 경우 자동으로 비울 것인가,아닌가를 지정한다.
디폴트는 true이다.
autoFlush 속성이 true로 설정되면, 버퍼가 다 찼을 경우 자동으로 비워지고,버퍼의 내용을 웹 브라우저로 전송한다. 그러나 false인 경우는 버퍼가 다 찼을 경우 처리중인 JSP페이지를 중단하고 예외가 발생된다.
buffer속성이 none으로 설정되면, autoFlush속성은 false로 설정할 수 없다.그 이유는 버퍼가 없는 상태에서는 버퍼에 대한 예외가 발생될 수 없기 때문이다.
Default) <%@ page autoFlush="true" %>

7. **isThreadSafe** => JSP페이지가 SingleThreadModel를 지원할 것인가를 결정한다. 디폴트는 true이다.
isThreadSafe속성이 false이면 SingleThreadModel를 구현하겠다는 것이고,
true이면 SingleThreadModle 구현 하지 않겠다는 의미이다.(이것보다는 synchronized()사용-권장)
Default) <%@ page isThreadSafe="true" %>

8. **info** =>해당 jsp파일의 정보를 기술한다.
Jsp파일의 정보를 가져오는 메소드는 getServletInfo()이다.
Example) <%@ page info="안녕하세요 JSP페이지 입니다." %>

9. **errorPage** => JSP 페이지에서 에러가 발생 했을 때 이동할 에러페이지 경로이다.
만약 JSP페이지에서 전체를 try ~ catch로 묶어주면 에러가 발생 하더라도 에러페이지로 이동하지 않는다.
Example) <%@ page errorPage="/error/error.jsp" %>

10. **isErrorPage** => JSP페이지에서 Throwable객체인 exception를 사용할 것인지를 정한다. 디폴트는 false이다.isErrorPage속성을 true로 설정하면 현재 JSP페이지에서 exception객체를 사용할 수 있고,false 이면 사용할 수 없다.
Default)<%@ page isErrorpage="false" %>

11. **contentType** => JSP페이지의 응답 MIME 타입과 문자 인코딩 타입을 지정한다.
JSP페이지의 디폴트 MIME타입은 "text/html" 이고,문자 인코딩 타입은 "euc-kr" 이다.
Default) <%@ page contentType="text/html;charset=euc-kr" %>

12. **pageEncoding** => JSP페이지에서 문자 인코딩 방식을 결정한다. 디폴트는 "ISO-8891-1"이다.
Default)<%@ page pageEncoding="ISO-8859-1" %>

13. **isELIgnored** => JSP페이지에서 EL(Expression Langugae)를 사용할 것인가를 기술한다.
isELIgnored값이 false 이면 EL를 사용하겠다는 의미이고, true이면 EL를 사용하지 않겠다는 의미이다.
DD에서도 EL를 사용할 것인지 아닌지를 설정할 수 있다.
isELIgnored 속성과 DD 설정이 동시에 이루어진 경우는 isELIgnored속성이 우선한다.
Default) isELIgnored="false"

▮ **include 지시어** : 변환 시점에 현재 페이지에 포함할 코드나 jsp 문서를 정의한다.

<%@ include file="menu/top.jsp" %>

file의 속성값으로 각종 파일(텍스트, html, jsp 등)이 올 수 잇고, url은 상대경로로 지정한다.

포함할 파일의 내용이 변환 시점에 그대로 복사되어서 포함되기 때문에 코딩 시에 변수명의 중복여부에 주의하여야 한다.

▮ **taglib 지시어** : JSP에서 이용 가능한 태그 라이브러리를 정의한다.

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

■ JSP의 스크립팅 구성요소

JSP 페이지에서 자바코드를 삽입하는데 사용되는 요소들이다.

▮ **선언문(declarations)** : JSP페이지에 삽입된 코드를 통해서 멤버변수나 멤버 메소드를 선언할 때 사용된다. JSP life Cycle 메소드 중에서 jspInit(), jspDestroy()등을 재정할 때 사용할 수 있다. 형태 <%! 선언문; %>

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
   <%!
        private String name = "홍길동";

        public void test(){
            System.out.println("테스트 메소드 입니다.!");
        }
   %>
<html>
    <head>
```

▮ 스크립틀릿(scriptlets) : JSP페이지에서 작성된 자바 코드를 지원한다.

scripting_jsp.java

```
public final class scripting_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

  private String name = "홍길동";   멤버 변수로 선언됨

  public void test(){
    System.out.println("테스트 메소드 입니다.!");     멤버 메소드로 선언됨
  }


  private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();

  private static java.util.List _jspx_dependants;

  private javax.el.ExpressionFactory _el_expressionfactory;
  private org.apache.AnnotationProcessor _jsp_annotationprocessor;

  public Object getDependants() {
    return _jspx_dependants;
  }
```

_jspService()메소드에 원하는 자바코드를 JSP 페이지에 정의할 수 있도록 지원한다.

스크립틀릿에서 선언된 변수는 로컬변수로 설정되며, 메소드를 호출할 수는 있으나, 선언할 수는 없다.

형태 : <% 자바 코드 %>

```
<%
    //로컬 변수 선언
    int age = 30;

    //메소드 호출
    test();
%>
```

▮ 표현식(expression) : JSP페이지내에서 직접 클라이언트로 출력될 내용을 표시할 때 사용된다. _jspService()메소드 내에 삽입되며, 표현식에서는 세미콜론을 사용하지 않는다. 형태 : <%= 표현식 %>

scripting_jsp.java

```
out.write("        <h1>회원정보</h1>\n");
out.write("        <li>이름 : ");
out.print(name );                표현식 <%=name %>는 out.print(name); 로 변환된다.
out.write("</li>\n");
out.write("        <li>나이 : ");
out.print(age );
out.write("</li>\n");
```

scripting_jsp.java

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

  PageContext pageContext = null;
  HttpSession session = null;
  ServletContext application = null;
  ServletConfig config = null;
  JspWriter out = null;
  Object page = this;
  JspWriter _jspx_out = null;
  PageContext _jspx_page_context = null;


  try {
    response.setContentType("text/html;charset=UTF-8");
    pageContext = _jspxFactory.getPageContext(this, request, response,
              null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;

    out.write("\n");
    out.write("\n");
    out.write("\n");
    out.write("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\"\n");
    out.write("    \"http://www.w3.org/TR/html4/loose.dtd\">\n");
    out.write("    ");
    out.write("\n");
    out.write("\n");
    out.write("    ");

      //로컬 변수 선언
      int age = 30;
      //메소드 호출
      test();

    out.write("\n");
```

스트립틀릿내에서 구현한 코드는
_jspService() 메소드내에 존재한다.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>회원정보</h1>
        <li>이름 : <%=name %></li>          표현식
        <li>나이 : <%=age %></li>           표현식
    </body>
</html>
```

scription.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
   <%!
        private String name = "홍길동";
```

```
        public void test(){
            System.out.println("테스트 메소드 입니다.!");
        }
    %>

    <%
        //로컬 변수 선언
        int age = 30;

        //메소드 호출
        test();
    %>

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>회원정보</h1>
        <li>이름 : <%=name %></li>
        <li>나이 : <%=age %></li>
    </body>
</html>
```

실행결과

JSP 내장객체

■ JSP 컨테이너는 많이 사용하는 서블릿 객체를 JSP 개발자가 JSP 코드의 스크립틀릿이과 표현식 태그안에서 사용할 수 있도록 하기 위해 기본적으로 제공해주는 객체들이다.

■ 내장객체는 JSP 컨테이너에 의해 생성되고 제공되기 때문에 JSP개발자는 이를 선언하거나 생성하지 않아도 사용할 수 있다. JSP 컨테이너가 제공하는 기본객체는 9가지가 있다.

session과 exception객체는 page 지시어의 설정의 따라서 핸들링 가능하지만, 나머지 객체는 컨테이너가 자동으로 생성해준다

1) **request**
클래스 또는 인터페이스 : javax.servlet.http.HttpServletRequest
범위(scope) : Request
설명 : request 내장 객체는 서블릿의 service 메서드의 매개변수인 HttpServletRequest와 동일하게 사용한다.

2) **response**
클래스 또는 인터페이스 : javax.servlet.http.HttpServletResponse
범위 : Request
설명 : response 기본 객체는 서블릿의 service 메서드의 매개변수인 HttpServletResponse와 동일하게 사용한다.

3) **pageContext**
클래스 또는 인터페이스 : javax.servlet.jsp.PageContext
범위(scope) : Page
설명 : JSP 페이지를 위한 페이지 컨텍스트로, pageContext 기본 객체는 다른 기본객체에 접근할 수 있는 메서드를 제공한다.

4) **session**
클래스 또는 인터페이스 : javax.servlet.http.HttpSession
범위 : Session
설명 : 세션 객체를 나타낸다.

5) **application**
클래스 또는 인터페이스 : javax.servlet.ServletContext
범위 : Application
설명 : application기본 객체는 JSP페이지의 서블릿 컨텍스를 말한다. Application 기본객체는 웹 어플리케이션 당 하나씩 존재한다.

6) **out**
클래스 또는 인터페이스 : javax.servlet.jsp.JspWriter
범위 : Page
설명 : out 기본 객체는 JSP 페이지의 출력을 위한 것이다.
웹 브라우저 응답을 출력하기 위한 JspWriter객체 이다.

7) **config**
클래스 또는 인터페이스 : javax.servlet.ServletConfig
범위(scope) : Page
설명 : config 기본 객체는 JSP페이지에서 사용할 초기 파라미터를 저장하고 있다.

8) **page**
클래스 또는 인터페이스 : java.lang.Object
범위 : Page
설명 : page 기본 객체는 JSP 그 자체를 의미한다. 자바 코드로는 this이다.

9) **exception**
클래스 또는 인터페이스 : java.lang.Throwable
범위 : Page
설명 : exception 기본객체는 JSP페이지에서 발생한 에러를 처리할 때 사용한다.
Page 디렉티브에서 isErrorPage가 true로 설정되는 경우만 exception 내장 객체를 사용할 수 있다.

JSP LifeCycle

hello_jsp.java

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

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

내장객체 선언

```
  try {
    response.setContentType("text/html;charset=UTF-8");
    pageContext = _jspxFactory.getPageContext(this, request, response,
              null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;
```

내장객체 초기화

■ JSP 컨테이너는 JSP 파일을 HttpJspPage 인터페이스를 구현한 서블릿 클래스로 변환하여 생성한다.

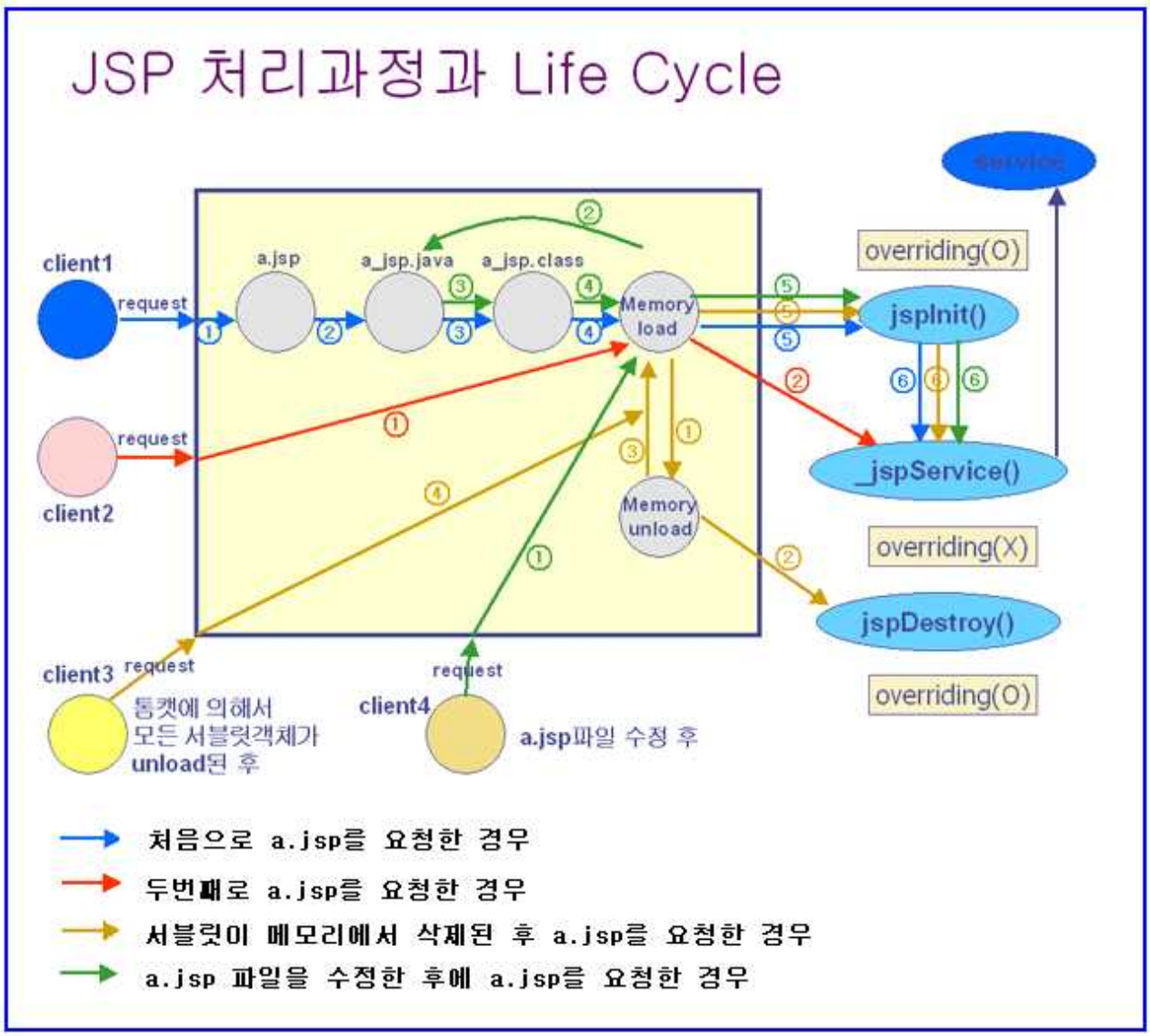
■ JSP의 LifeCycle 관련 메소드

1. **jspInit()** : 서블릿의 init()에 해당합니다. **재정의 가능**합니다.

2. **jspDestroy()** : 서블릿의 destroy() 에 해당합니다. **재정의 가능**합니다.

3. **_jspService()** : 서블릿의 service() 에 해당합니다. **재정의 불가능**합니다.
요청이 들어올때마다 이 메소드는 새로운 스레드에서 실행된다. JSP에서 작성한 코드를 받아서 컨테이너가 _jspService()메소드를 만들기 때문에 절대 재정의해서는 안된다.

■ JSP 요청시 처리과정과 Life Cycle



life.jsp

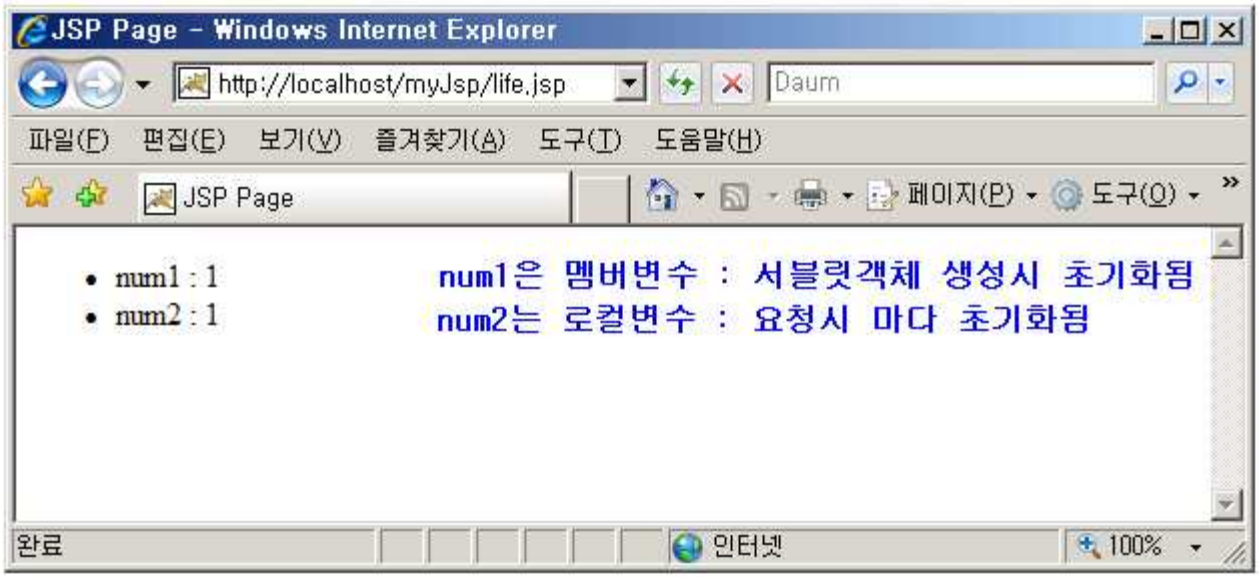
```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
   <%!
    private int num1 = 0;

    public void jspInit(){
        System.out.println("jspInit() 호출됨");
    }

    public void jspDestroy(){
        System.out.println("jspDestroy() 호출됨");
    }
   %>
   <%
    int num2 = 0;

    num1++;
    num2++;
   %>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
       <ul>
        <li>num1 : <%=num1 %></li>
        <li>num2 : <%=num2 %></li>
       </ul>
    </body>
</html>
```

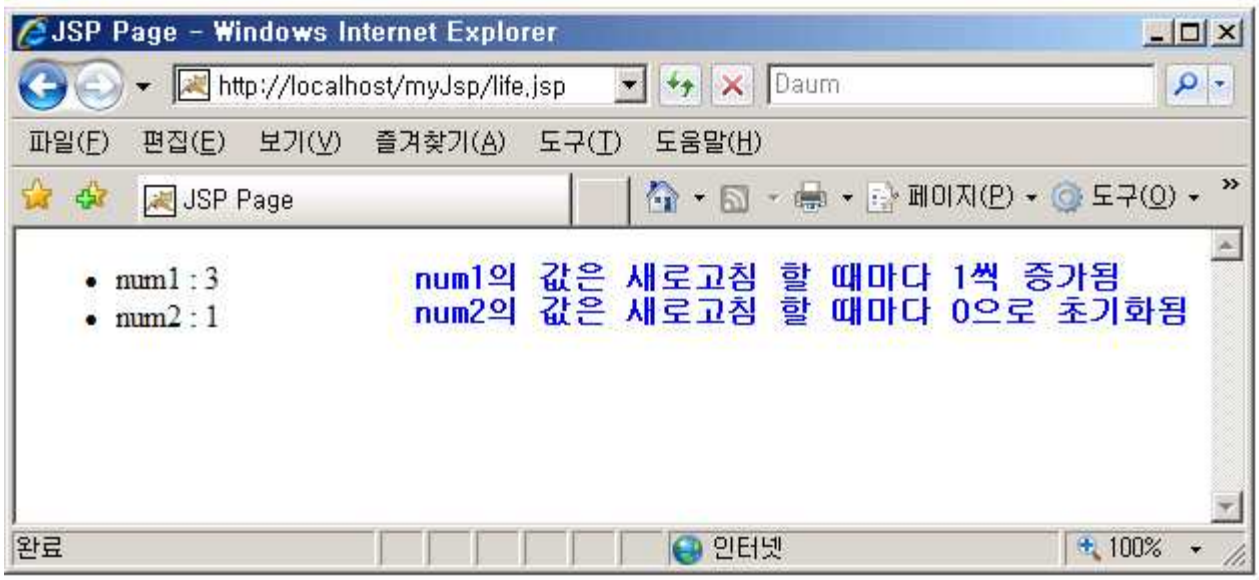
jspInit() 재정의

jspDestroy() 재정의

tomcat의 콘솔화면

실행결과

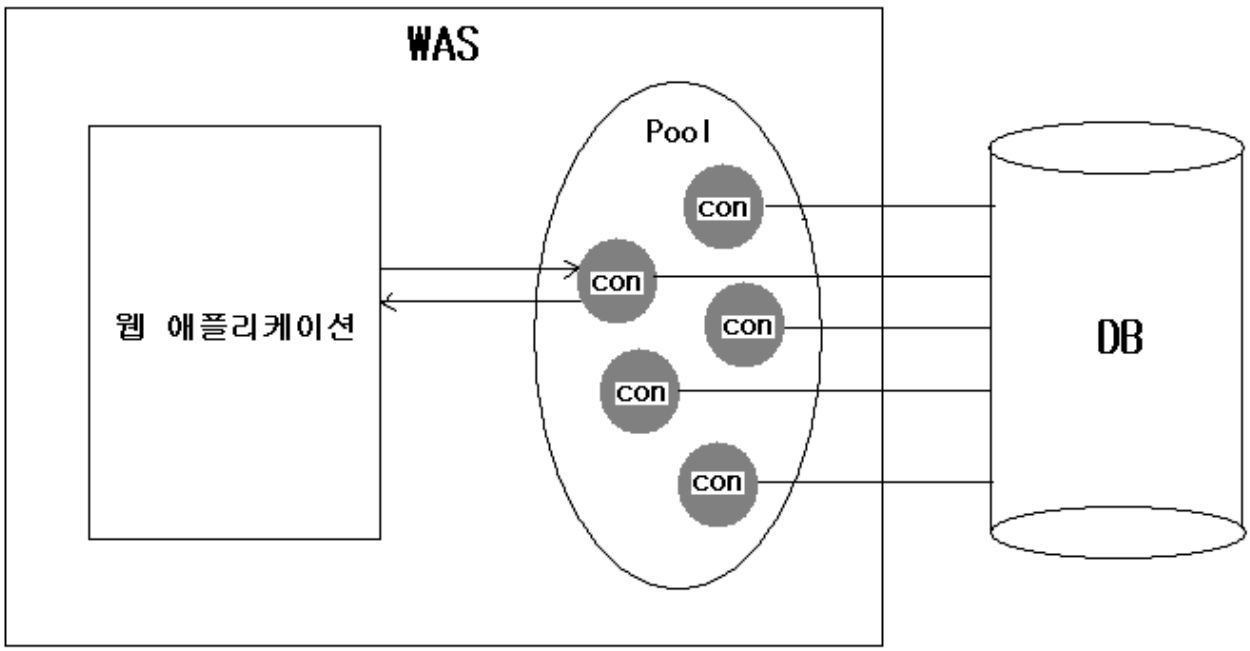




Connection Pool

■ JDBC 프로그램은 Connection 자원을 획득하는데 많은 시간이 소요된다.

■ Connection Pool은 미리 DB와 연결을 유지하고 있는 Connection객체를 생성해서 Vector와 같은 타입의 객체에 넣어두고, 필요할 때 꺼내서 사용하고, 사용을 마친 Connection 객체는 다시 Pool로 반납하는 구조다.



■ JDBC 2.0 부터는 javax.sql.DataSource 를 사용해서 ConnectionPool를 구현하도록 하고 있다.

■ Tomcat를 비롯한 대부분의 WAS 제품의 내부적으로 javax.sql.DataSource를 구현한 Connection Pool를 내장하고 있으며, JNDI(Java Naming and Directory Interface)의 lookup 서비스를 통해서 WAS가 구현해 놓은 Connection Pool 객체를 사용할 수 있도록 서비스 하고 있다.

■ Connection Pool 구현하기

▮ Tomcat 홈 디렉토리\lib 폴더에 jdbc 드라이버 복사하기

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
    <Resource name="jdbc/myoracle"                    jndi에 등록시 사용할 이름
              auth="Container"
              type="javax.sql.DataSource"              Connection Pool의 타입
              driverClassName="oracle.jdbc.OracleDriver"    JDBC 드라이버 명
              url="jdbc:oracle:thin:@localhost:1521:orcl"     JDBC 접속 url
              username="scott" password="tiger"
              maxActive="20"                  최대 연결 갯수, 0인 경우 무제한
              maxIdle="10"                 최대 유휴 연결 갯수, 0인 경우  무제한
              maxWait="-1" />              최대 연결 시도 시간, -1인 경우 무제한

</Context>
```

▮ Web application project내의 META-INF에 context.xml 파일을 추가한다.

■ mybatis를 웹서비스에 적용

1) tomcat(was)에 Connection Pool 설정 (윗부분 참고)

2) config.xml 파일에 dataSource 설정

- 지금까지 우리가 했던 설정

```
<environments default="development">
        <environment id="development">
                <transactionManager type="JDBC"/>
                <dataSource type="POOLED">
                        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
                        <property name="driver" value="oracle.jdbc.OracleDriver"/>
                        <property name="username" value="test"/>
                        <property name="password" value="1111"/>
                </dataSource>
        </environment>
</environments>
```

- 커넥션풀을 was가 생성한 dataSource로 설정

```
<environments default="development">
        <environment id="development">
                <transactionManager type="JDBC"/>
                <dataSource type="JNDI">
                        <property name="data_source" value="java:comp/env/jdbc/oracle"/>
                </dataSource>
        </environment>
</environments>
```

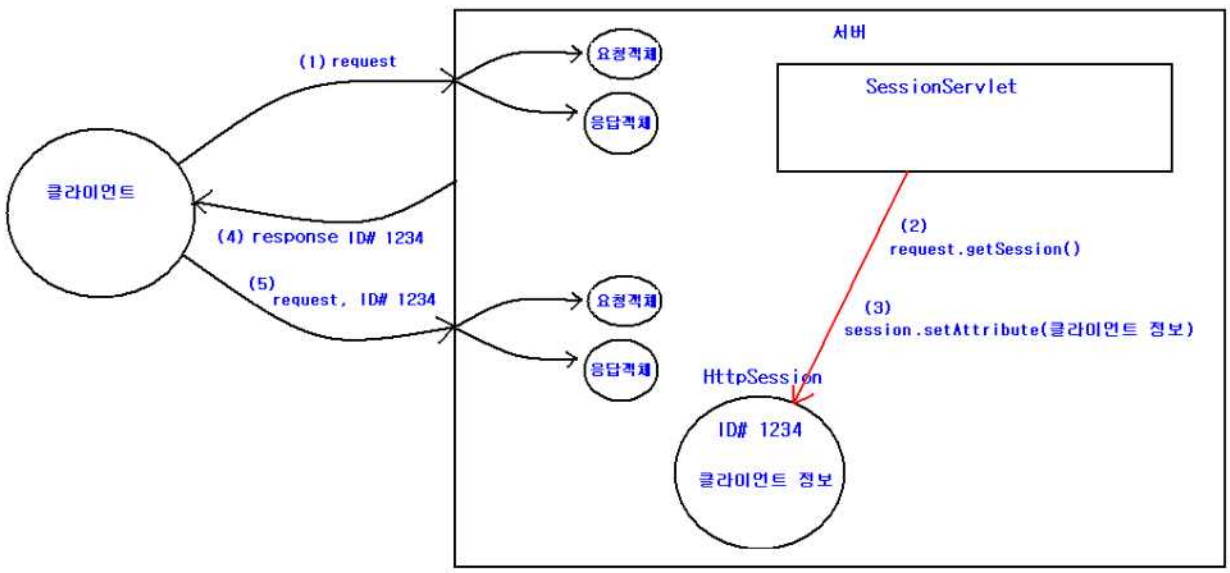
HttpSession 객체

■ HTTP 프로토콜은 무상태(stateless) 연결 프로토콜이다.

■ 클라이언트가 서버와 연결을 맺고, 요청을 보낸 뒤, 서버가 요청을 처리한 후 응답을 보내면 클라이언트와 서버 사이의 연결은 끊어진다.

■ 서버에서 클라이언트의 정보를 유지하기 위하여 HttpSession 객체를 사용한다.

■ HttpSession의 동작 원리



(1) 클라이언트의 요청 접수

(2) 클라이언트의 정보를 저장할 HttpSession객체 생성, 고유한 id 부여됨

(3) HttpSession객체내에 클라이언트의 정보를 저장

(4) 고유한 id값을 응답에 넣어서 클라이언트로 보냄

(5) 클라이언트는 다음 번 요청부터 고유한 id값을 서버로 전송한다.

서버에서는 id에 해당하는 HttpSessio객체를 찾아서 요청을 처리한다.

■ HttpSession의 주요 메소드

반환형	메소드 명	설명
void	setAttribute(String name, Objec obj)	HttpSession객체에 지정된 이름으로 객체를 저장한다.
Object	getAttribute(String name)	HttpSession객체에서 지정된 이름으로 저장된 객체를 반환한다.
void	removeAttribute()	HttpSession객체에서 지정된 이름의 객체를 삭제한다.
void	setMaxInactiveInterval(int sec);	HttpSession객체의 타임아웃 시간을 지정한다.
void	invalidate()	HttpSession객체을 무효화시킨다.
String	getId()	세션 id를 반환한다.