JSP (Java Sever Page)

- 서블릿과 MVC 패턴 구현

CONTENTS

- 서블릿의 기초
- 서블릿 구현
- URL 매핑 규칙

서블릿

- 서블릿은 JSP 표준이 나오기 전에 자바에서 웹 어플리케이션 개발을 위해 만들어진 표준.
- 서블릿 개발 과정은 JSP 비해 복잡함.
- MVC 패턴을 지원하는 프레임워크의 경우 기반 코드를 서블릿에서 개발하기 때문에 서블릿을 직접 구현하지 않더라도 서블릿 자체에 대해서 이해하는 것은 중요.

■ 서블릿 개발과정

1. 서블릿 규약에 따라 자바 코드를 작성 (HttpServlet 상속 후 메서드 구현)
2. 자바코드를 컴파일 해서 클래스파일을 생성한다.
   (이클립스 환경에서 개발할 경우 이 단계는 하지 않음.)
3. 클래스파일을 /WEB-INF/classes 디렉토리에 패키지에 알맞게 위치시킨다. (이클립스 환경에서 개발할 경우 이 단계는 하지 않음.)
4. web.xml 파일에 서블릿 클래스를 설정한다.
5. 톰캣 등의 웹컨테이너를 재생시킨다.
6. 웹 브라우저에서 확인한다.

서블릿 구현

NowServlet.java

```
package test;

import java.io.IOException;
import java.io.PrintWriter;
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 NowServlet extends HttpServlet { // HttpServlet 클래스를 상속받아 클래스 작성
// 처리하고자 하는 HTTP 방식(mothod)에 따라 알맞게 메서드를 오버라이딩 해주어야 함.
// doGet, doPost
        @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException {
// HttpServletReques 와 HttpServletResponse 파라미터는 JSP에서의 request와 response 와 같다.
                response.setContentType("text/html; charset=euc-kr");
```

서블릿 구현

NowServlet.java

```
                PrintWriter out = response.getWriter();
                out.println("<html>");
                out.println("<head><title>현재시간</title></head>");
                out.println("<body>");
                out.println("현재 시간은");
                out.println(new Date());
                out.println("입니다.");
                out.println("</body></html>");
        }

}
```

서블릿 구현 : web.xml 매핑

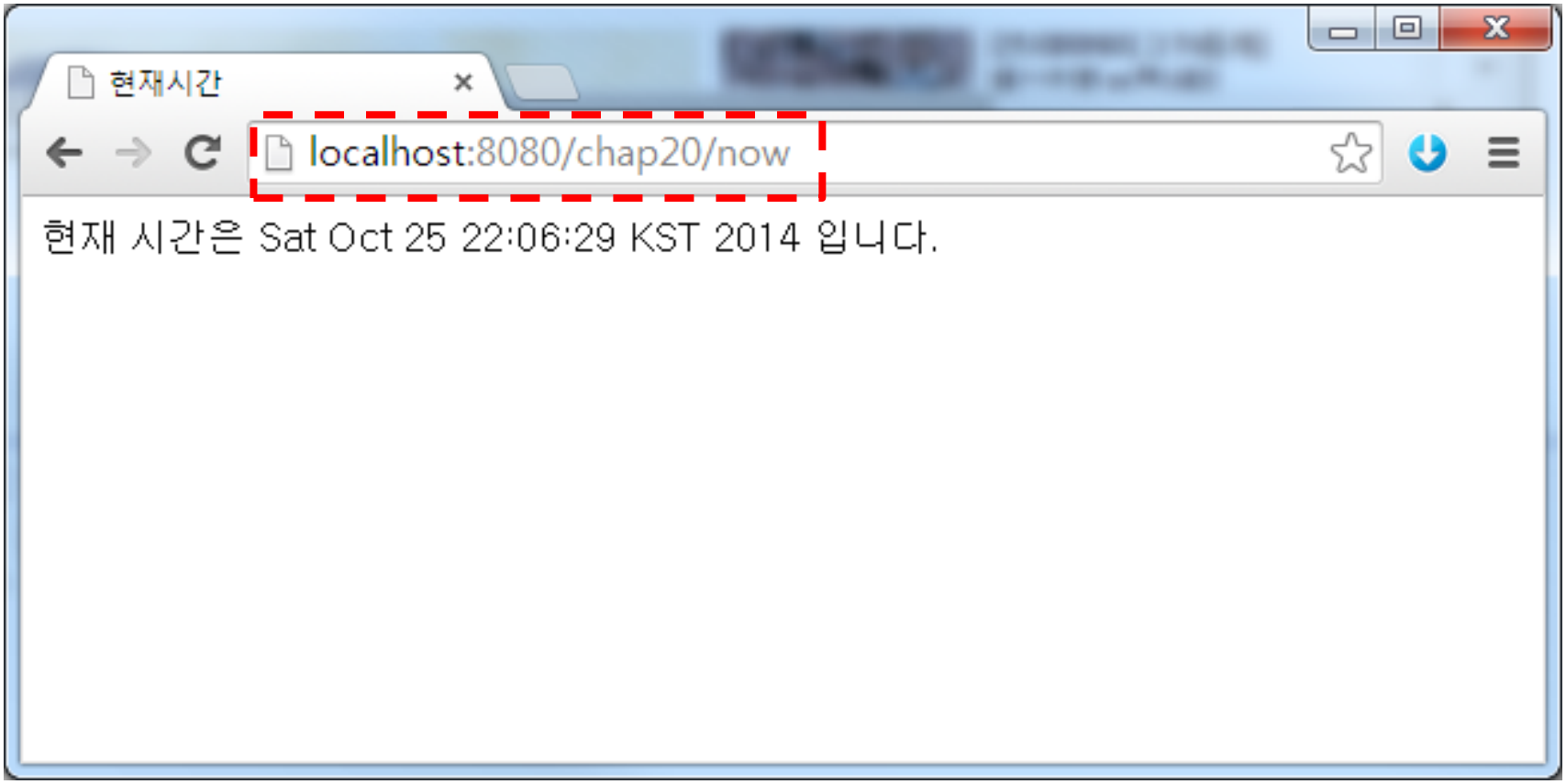
NowServlet.java

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="chap20" version="3.0">
<!-- 서블릿으로 사용할 클래스 등록 -->
        <servlet>
                <servlet-name>now</servlet-name>
                <servlet-class> test.NowServlet</servlet-class>
        </servlet>
<!-- URL을 통해 서블릿을 호출할 수 있도록 서블릿과 URL과의 매핑 -->
        <servlet-mapping>
                <servlet-name>now</servlet-name>
                <url-pattern>/now</url-pattern>
        </servlet-mapping>


</web-app>
```

서블릿 구현 : 결과


현재시간
localhost:8080/chap20/now
현재 시간은 Sat Oct 25 22:06:29 KST 2014 입니다.

서블릿 구현 :이노테이션으로 매핑하기

HelloServlet.java

```
package test;

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(urlPatterns = "/hello")
// 두개 이상의 URL 패턴을 처리 할 경우
//@WebServlet(urlPatterns = {"/hello“, "/hello1“})
public class HelloServlet extends HttpServlet {

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

서블릿 구현 :이노테이션으로 매핑하기

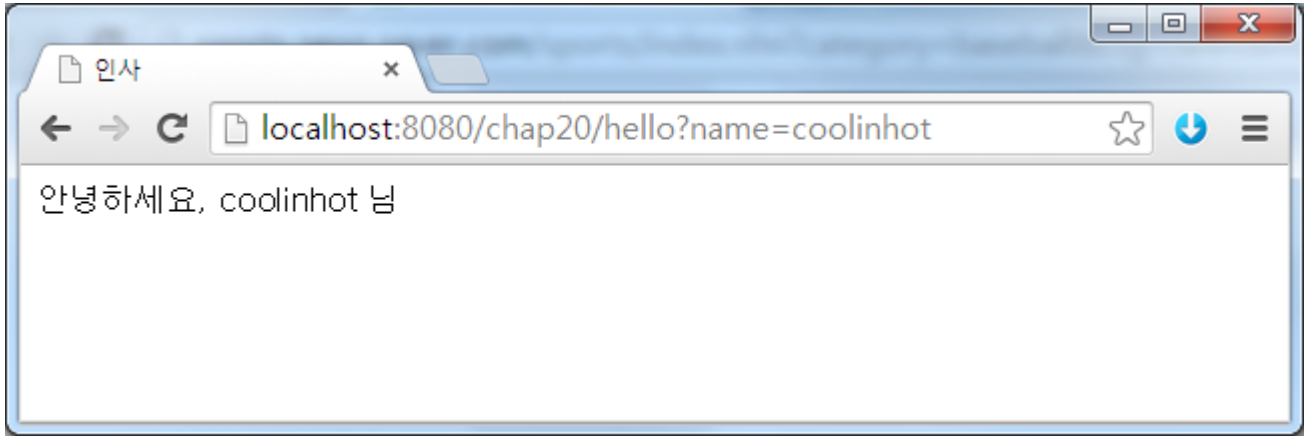
HelloServlet.java

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

                PrintWriter out = response.getWriter();
                out.println("<html>");
                out.println("<head><title>인사</title></head>");
                out.println("<body>");
                out.println("안녕하세요, ");
                out.println(request.getParameter("name"));
                out.println("님");
                out.println("</body></html>");
        }

}
```

서블릿 구현 :이노테이션으로 매핑하기 결과



서블릿 구현 : HTTP 각 방식별 구현 메서드

```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
}
```

서블릿 구현 : HTTP 각 방식별 구현 메서드

```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
}

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
// TODO Auto-generated method stub
}
```

서블릿 구현 : 서블릿의 초기화 파라미터, 초기화, 로딩

- 서블릿이 사용이 웹 URL을 통한 실행이 아닌
- <init-param>태그로 지정한 값은 초기화 파라미터라고 불리는 설정값을 지정할 때 사용.
  - <param-name> : 파라미터의 이름
  - <param-value> : 초기화 파라미터의 값
- 서블릿 클래스는 getInitParameter()메서드를 이용해서 초기화 파라미터 값을 얻어온다.
- init() 메서드 오버라이딩.
- 이노테이션에서 설정

```
@WebServlet(urlPattern = "/hello", loadOnStartup =1)

@WebServlet(urlPattern = "/hello",
   initParam = {
      @WebInitParam(name="greeting", value="Hello"),
      @WebInitParam(name="title", value="제목"),
   })
```

@WebServlet 어노테이션 주요 속성

- **name**
  - 서블릿의 이름을 설정하는 속성으로 기본값은 빈 문자열("")이다.

    @WebServlet(name="서블릿이름")
- **urlPatterns**
  - 서블릿의 URL을 설정하는 속성으로 속성으로 String 배열을 지정, 기본값은 빈 배열({})
  - **서블릿에 대해 한 개의 URL을 설정하는 경우**

    @WebServlet(urlPatterns="/url") 또는 @WebServlet(urlPatterns={"/url"})
  - **서블릿에 대해 여러 개의 URL을 설정하는 경우**

    @WebServlet(urlPatterns={"/url1", "/url2", "/url3"})
- **value**
  - urlPatterns와 같은 용도로 사용하고, 어노테이션에서 단일 속성 'value'는 생략 가능

    @WebServlet(value="/url") 또는 @WebServlet("/url");
- **여러 개의 속성 동시에 지정하기**

  @WebServlet(value="/url", name="서블릿이름")

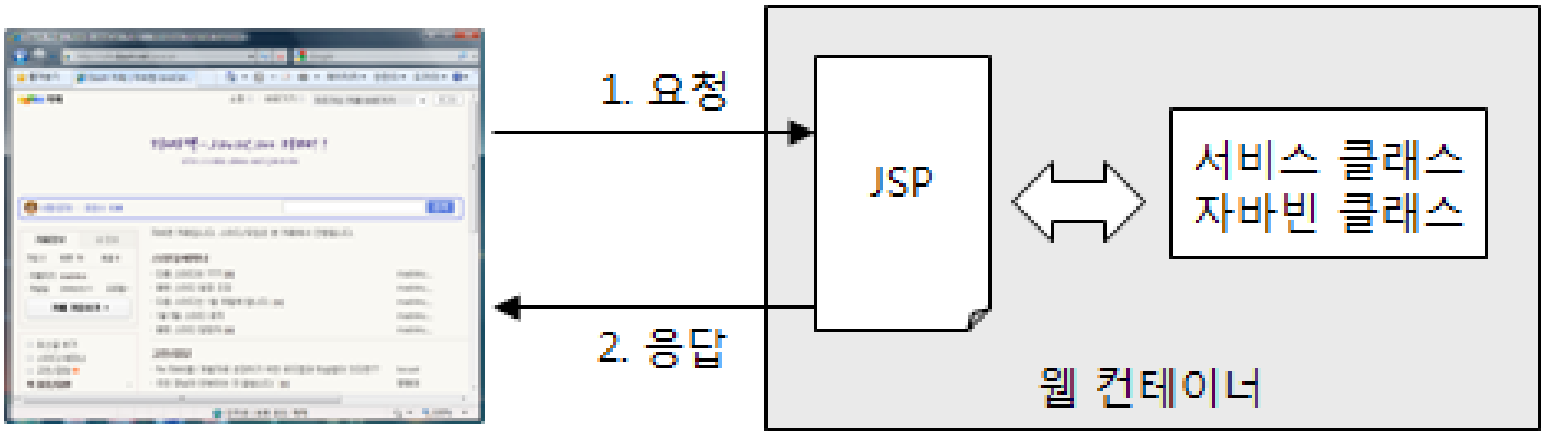
서블릿 구현 : URL 패턴 매핑 규칙

- '/'로 시작하고 '/*'로 끝나는 url-pattern은 경로 매핑을 위해서 사용
- '*.'로 시작하는 url-pattern은 확장자에 대한 매핑을 할 때 사용.
- 오직 '/'만 포함하는 경우 어플리케이션의 기본 서블릿으로 매핑

CONTENTS

모델 1 구조

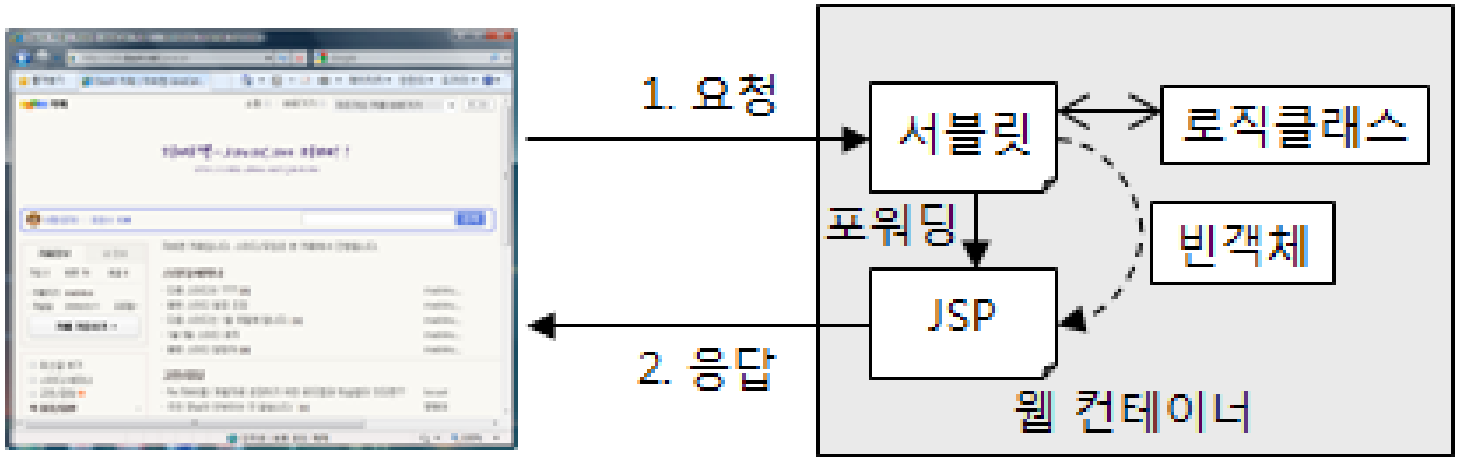
- JSP를 이용한 단순한 모델



- JSP에서 요청 처리 및 뷰 생성 처리
  - 구현이 쉬움
  - 요청 처리 및 뷰 생성 코드가 뒤섞여 코드가 복잡함

모델 2 구조

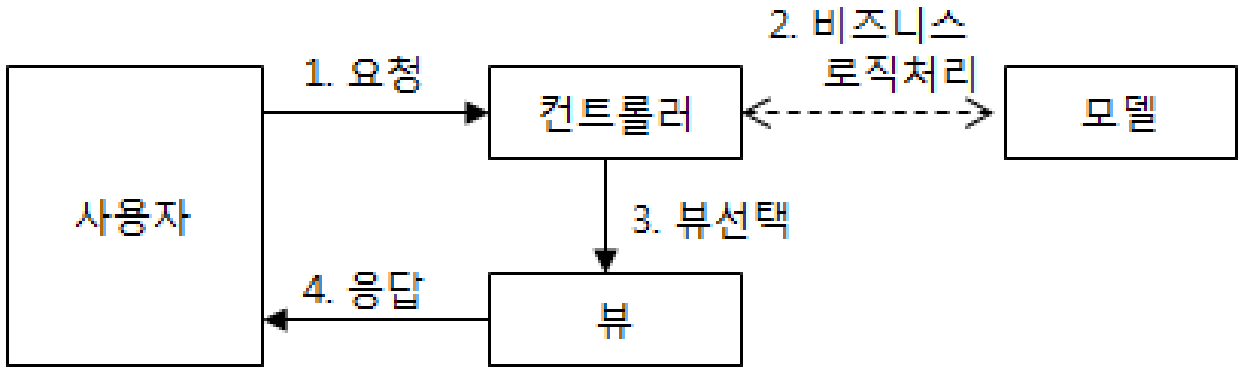
- 서블릿이 요청을 처리하고 JSP가 뷰를 생성



- 모든 요청을 단일 서블릿에서 처리
  - 요청 처리 후 결과를 보여줄 JSP로 이동

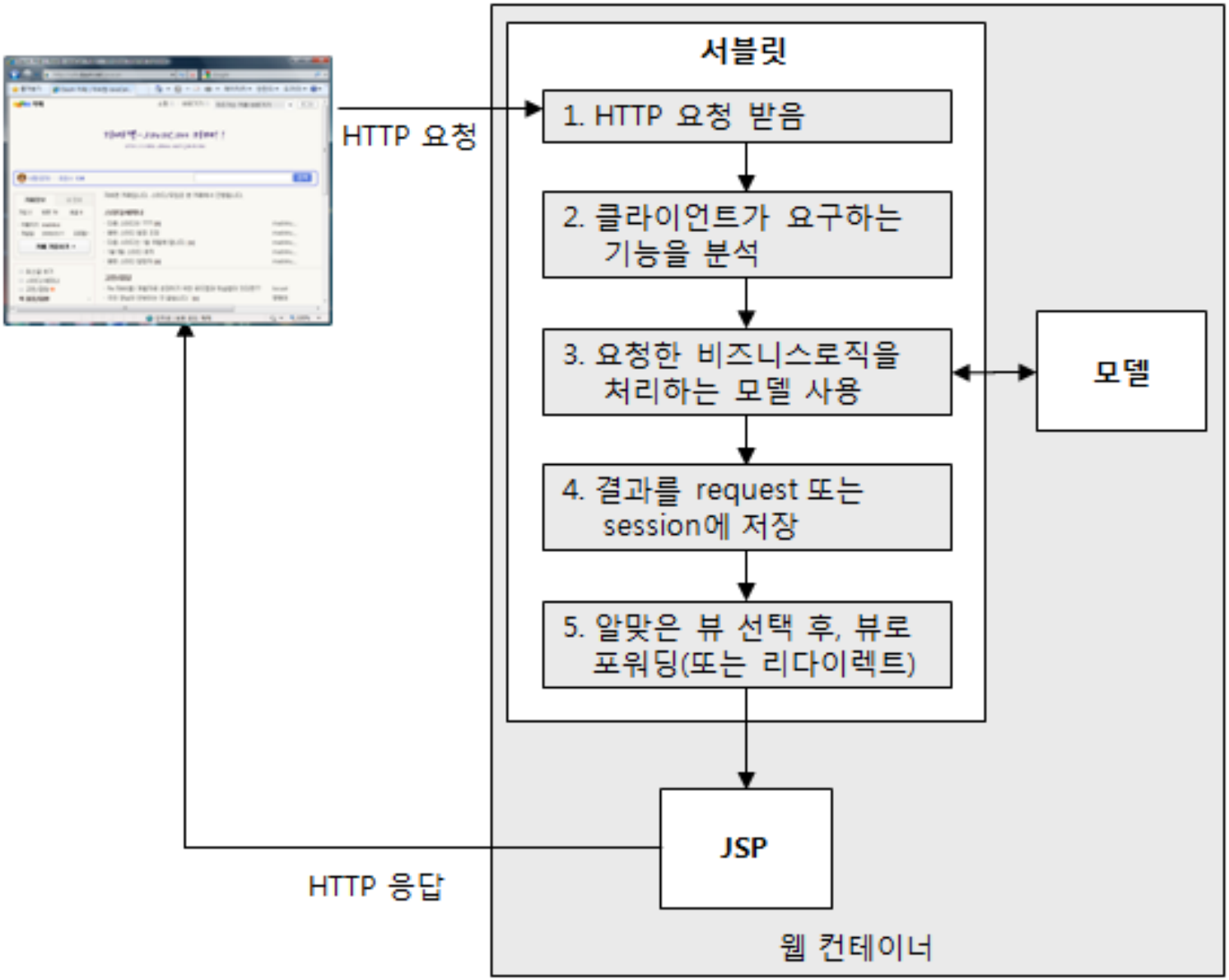
MVC 패턴

- Model-View-Controller
  - 모델 - 비즈니스 영역의 상태 정보를 처리한다.
  - 뷰 - 비즈니스 영역에 대한 프레젠테이션 뷰(즉, 사용자가 보게 될 결과 화면)를 담당한다.
  - 컨트롤러 - 사용자의 입력 및 흐름 제어를 담당한다.

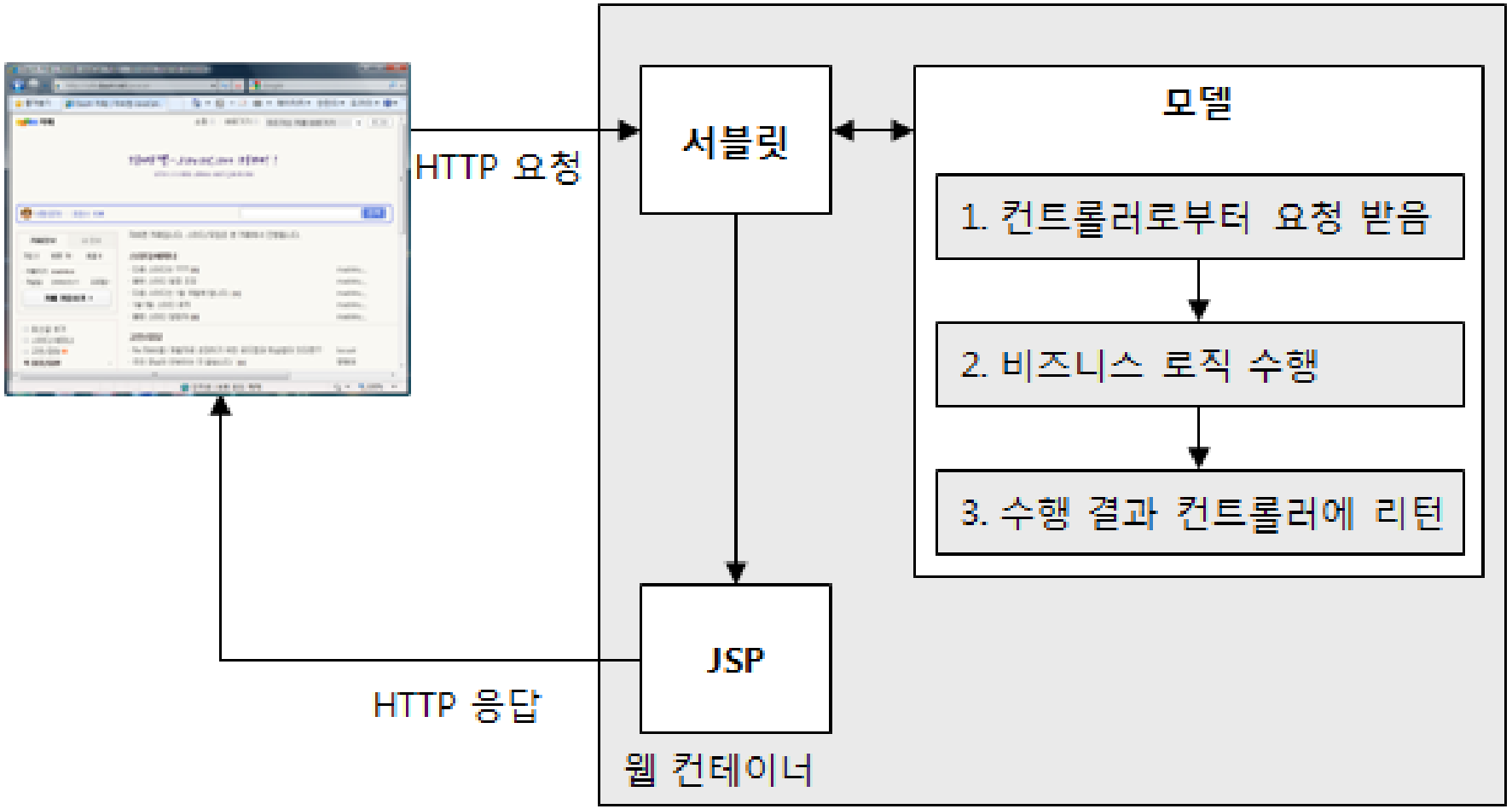


- 특징
  - 로직을 처리하는 모델과 결과 화면을 보여주는 뷰가 분리됨
  - 흐름 제어나 사용자의 처리 요청은 컨트롤러에 집중
- 모델 2 구조와 매핑: 컨트롤러-서블릿, 뷰-JSP

MVC의 컨트롤러 : 서블릿



MVC의 모델 : 로직 수행 클래스



MVC 패턴 컨트롤러 기본 구현

```
public class ControllerServlet extends HttpServlet {
    // 1단계, HTTP 요청 받음
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        processRequest(request, response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        processRequest(request, response);
    }
    private void processRequest(HttpServlet request, HttpServletResponse response)
    throws IOException, ServletException { //1 단계에서 일단 이 메서드로 요청을 전달
        // 2단계, 요청 분석
        // request 객체로부터 사용자의 요청을 분석하는 코드
        ...
        // 3단계, 모델을 사용하여 요청한 기능을 수행한다.
        // 사용자에 요청에 따라 알맞은 코드
        // 4단계, request나 session에 처리 결과를 저장
        request.setAttribute("result", resultObject); // 이런 형태의 코드
        ...
        // 5단계, RequestDispatcher를 사용하여 알맞은 뷰로 포워딩
        RequestDispatcher dispatcher = request.getRequestDispatcher("/view.jsp");
        dispatcher.forward(request, response);
    }       }
```

MVC 패턴 컨트롤러 기본 구현

mvctest.SimpleController.java

```
package mvctest;

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;

public class SimpleController extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse
response) // 1단계, HTTP 요청 받음
                                                throws ServletException, IOException {
                processRequest(request, response);
        }
```

MVC 패턴 컨트롤러 기본 구현

mvctest.SimpleController.java

```
public void doPost(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {// 1단계, HTTP 요청 받음
        processRequest(request, response);
}

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

        // 2단계, 요청 파악
        // request 객체로부터 사용자의 요청을 파악하는 코드
        String type = request.getParameter("type");

        // 3단계, 요청한 기능을 수행한다.
        // 사용자에 요청에 따라 알맞은 코드
        Object resultObject = null;
        if (type == null || type.equals("greeting")) {
                resultObject = "안녕하세요.";
```

MVC 패턴 컨트롤러 기본 구현

mvctest.SimpleController.java

```
                } else if (type.equals("date")) {
                        resultObject = new java.util.Date();
                } else {
                        resultObject = "Invalid Type";
                }

                // 4단계, request나 session에 처리 결과를 저장
                request.setAttribute("result", resultObject);

                // 5단계, RequestDispatcher를 사용하여 알맞은 뷰로 포워딩
                RequestDispatcher dispatcher =

request.getRequestDispatcher("/simpleView.jsp");
                dispatcher.forward(request, response);
        }
}
```

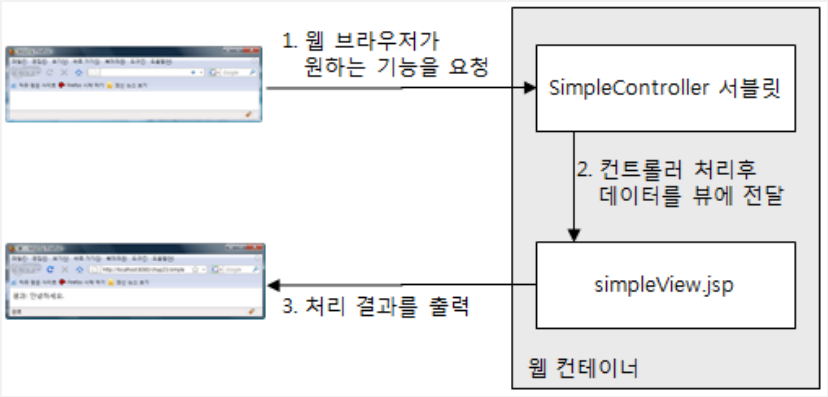
컨트롤러 구현

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

    // 2단계, 요청 파악
    // request 객체로부터 사용자의 요청을 파악하는 코드
    String type = request.getParameter("type");

    // 3단계, 요청한 기능을 수행한다.
    // 사용자에 요청에 따라 알맞은 코드
    Object resultObject = null;
    if (type == null || type.equals("greeting")) {
        resultObject = "안녕하세요.";
    } else if (type.equals("date")) {
        resultObject = new java.util.Date();
    } else {
        resultObject = "Invalid Type";
    }
    // 4단계, request나 session에 처리 결과를 저장
    request.setAttribute("result", resultObject);

    // 5단계, RequestDispatcher를 사용하여 알맞은 뷰로 포워딩
    RequestDispatcher dispatcher = request.getRequestDispatcher("/simpleView.jsp");
    dispatcher.forward(request, response);
}
```



MVC 패턴 컨트롤러 기본 구현

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                            http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        version="3.0">

        <servlet>
                <servlet-name>SimpleController</servlet-name>
                <servlet-class>mvctest.SimpleController</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>SimpleController</servlet-name>
                <url-pattern>/simple</url-pattern>
        </servlet-mapping>

</web-app>
```

■ 컨트롤러 구현

simpleView.jsp

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

<html>
<head><title>뷰</title></head>
<body>

결과: <%= request.getAttribute("result") %>

</body>
</html>
```

■ 클라이언트의 요청 명령을 구분하는 방식

- 컨트롤러가 알맞은 로직을 수행하려면 클라이언트가 어떤 기능을 요청하는 지 구분할 수 있어야 함

- 요청 기능 구분 방식
  - 특정 이름의 파라미터에 명령어 정보를 전달
    - http://host/chap21/servlet/ControllerServlet?cmd=BoardList&...
  - 요청 URI를 명령어로 사용
    - http://host/chap21/boardList?...

요청 URI 사용시 컨트롤러 구현

```
private void process(HttpServletRequest request,      HttpServletResponse response)
throws ServletException, IOException {
```

String command = request.getParameter("cmd");

```
private void process(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
```

String command = request.getRequestURI();

if (command.indexOf(request.getContextPath()) == 0) {

**   command = command.substring(request.getContextPath().length());**

}

■ 모델 1 vs 모델 2

모델	장점	단점
모델 1	-배우기 쉬움 -자바 언어를 몰라도 구현 가능 -기능과 JSP의 직관적인 연결. 하나의 JSP가 하나의 기능과 연결	-로직 코드와 뷰 코드가 혼합되어 JSP 코드가 복잡해짐 -뷰 변경 시 논리코드의 빈번한 복사 즉, 유지보수 작업이 불편함
모델 2	-로직 코드와 뷰 코드의 분리에 따른 유지 보수의 편리함 -컨트롤러 서블릿에서 집중적인 작업 처리 가능.(권한/인증 등) -확장의 용이함	-자바 언어에 친숙하지 않으면 접근하기가 쉽지 않음 -작업량이 많음(커맨드클래스+뷰JSP)

MVC 패턴 구현

- 요구사항
  - 회원가입, 로그인, 방명록을 MVC 패턴으로 구현.
  - 요청 URI 형태로 구현.