

MVC Architecture 구현해 보기

Servlets and JSP를 이용한 단계별 구현 방법

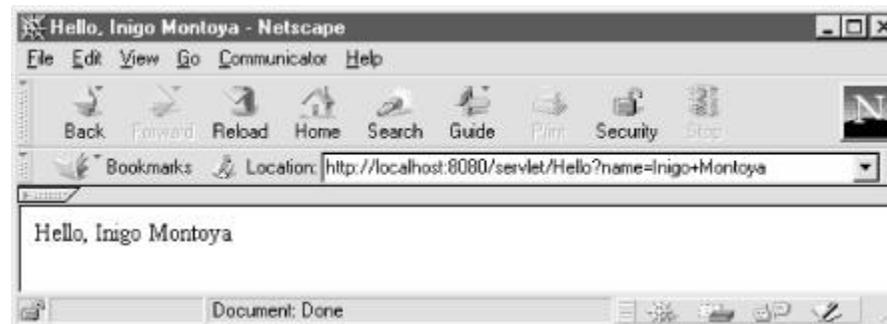
남 광 우

MVC 구현(버전 0): HTML-서블릿 구현

- 화면 설계: 웹에서 이름 입력받아 Hello you!
 - 다음 그림과 같이 웹에서 이름을 입력받아 화면에 출력하는 프로그램을 작성하시오.



자기 이름을 입력하고
버튼을 Click



MVC 구현(버전 0): HTML-서블릿 구현

- 구현 설계: 웹에서 이름 입력받아 Hello you!
 - 웹에서 이름 입력받아 Hello!의 설계

Helloyou.html

Helloyou servlet

- 1) 이름을 입력하고 버튼을 클릭
- 2) Helloyou 서블릿을 호출

- 1) HttpServletRequest의 파라미터로 전송받은 이름을 추출
- 2) 이름과 Html을 복합한 내용을 HttpServletResponse를 통해 전송

MVC 구현(버전 0): HTML-서블릿 구현

- 구현: 웹에서 이름 입력받아 Hello!의 구현

Helloyou.html

```
<HTML>
<HEAD>
<TITLE>Introductions</TITLE>
</HEAD>
<BODY>
  <FORM METHOD=GET ACTION="/Helloyou">
    If you don't mind me asking, what is your name?
    <INPUT TYPE=TEXT NAME="name"><P>
    <INPUT TYPE=SUBMIT>
  </FORM>
</BODY>
</HTML>
```



MVC 구현(버전 0): HTML-서블릿 구현

- 구현: Servlet 구현

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Helloyou extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        String name = req.getParameter("name");
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello, " + name + "</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("Hello, " + name);
        out.println("</BODY></HTML>");
    }

    public String getServletInfo() {
        return "A servlet that knows the name of the person to whom it's" +
            "saying hello";
    }
}
```

MVC 구현(버전 0): HTML-서블릿 구현

- 구현 : HelloYou의 deployment
 - web.xml의 설정
 - servlet 구현 클래스를 webapps/ROOT/WEB-INF/classes에 카피
 - webapps/ROOT/WEB-INF/ 디렉토리에 web.xml을 작성
 - http://localhost:8080/helloyou를 이용하여 호출

web.xml의 예

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>helloyou</servlet-name>
    <servlet-class>Helloyout</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>helloyou</servlet-name>
    <url-pattern>/helloyou</url-pattern>
  </servlet-mapping>
</web-app>
```

MVC 구현(버전 0): HTML-서블릿 구현

- 구현 : HelloYou의 deployment
 - Tomcat의 디렉토리 구조

The image shows a Windows Explorer window displaying the directory structure of an Apache Tomcat installation. The path is `E:\bin\tomcat6\apache-tomcat-6.0.10\webapps\examples\WEB-INF`. The directory tree is as follows:

- apache-tomcat-6,0,10
 - bin
 - conf
 - lib
 - logs
 - temp
 - webapps
 - docs
 - examples
 - jsp
 - servlets
 - WEB-INF
 - classes
 - jsp
 - jsp2
 - lib
 - tags
 - host-manager
 - manager
 - ROOT

The right pane shows the contents of the selected `WEB-INF` directory:

이름	크기	종류	수정된 날짜
classes		파일 폴더	2007-04-10 오후 ...
jsp		파일 폴더	2007-04-10 오후 ...
jsp2		파일 폴더	2007-04-10 오후 ...
lib		파일 폴더	2007-04-10 오후 ...
tags		파일 폴더	2007-04-10 오후 ...
web.xml	11KB	XML 문서	2007-02-13 오후 ...

Annotations in the image:

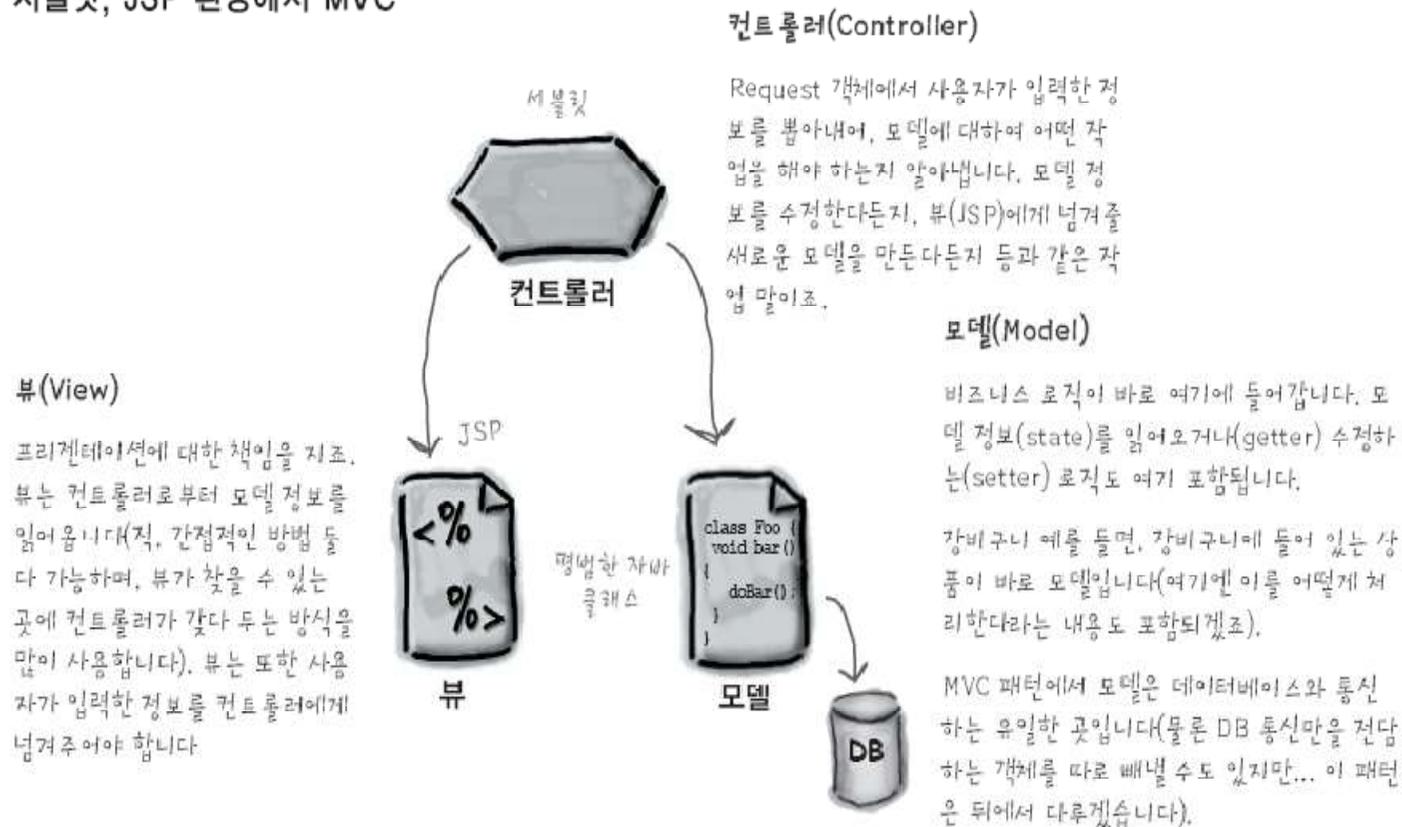
- Tomcat서버 실행파일**: Points to the `bin` directory.
- Web 응용 디렉토리**: Points to the `webapps` directory.
- Tomcat의 웹 Root**: `http://localhost:8080/`. Points to the `ROOT` directory.
- web.xml**: `Servlet 클래스 설정 파일`. Points to the `web.xml` file.
- Html 파일의 디렉토리**: Points to the `examples` directory.
- classes : 구현한 Servlet 클래스들을 옮겨놓을 디렉토리**: Points to the `classes` directory inside `WEB-INF`.
- 각 Web 응용의 Root**:
 - `http://localhost:8080/examples`
 - `http://localhost:8080/docs`
 - `http://localhost:8080/manager`Points to the `examples`, `docs`, and `manager` directories.

MVC 개념

□ MVC : Model, View, Controller

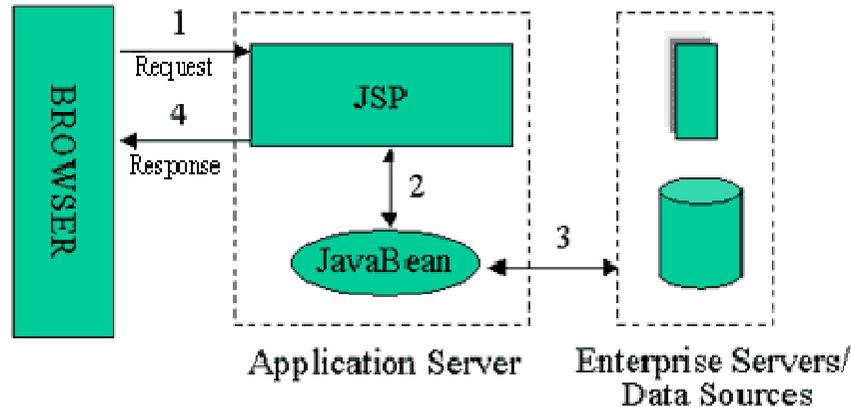
- 디자인 패턴중의 하나로서, 하나의 프로그램을 Model, View, Controller 세개의 컴포넌트로 구분하여 구현하는 것

서블릿, JSP 환경에서 MVC



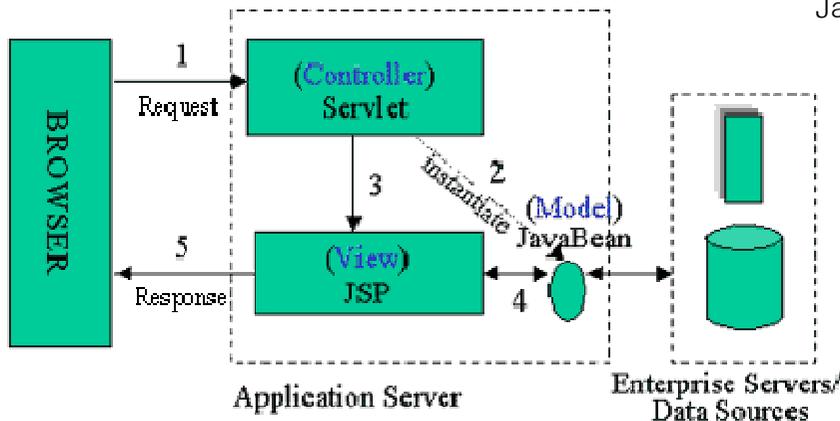
MVC 개념

□ JSP 에서의 MVC 구현 방법 변화



JSP Model 1 Architecture

Model 1에서는 JSP에서 사용자 Request(Controller)와 Response(View)를 담당하고 Data source의 접근 처리는 JavaBean(Model)에서 담당

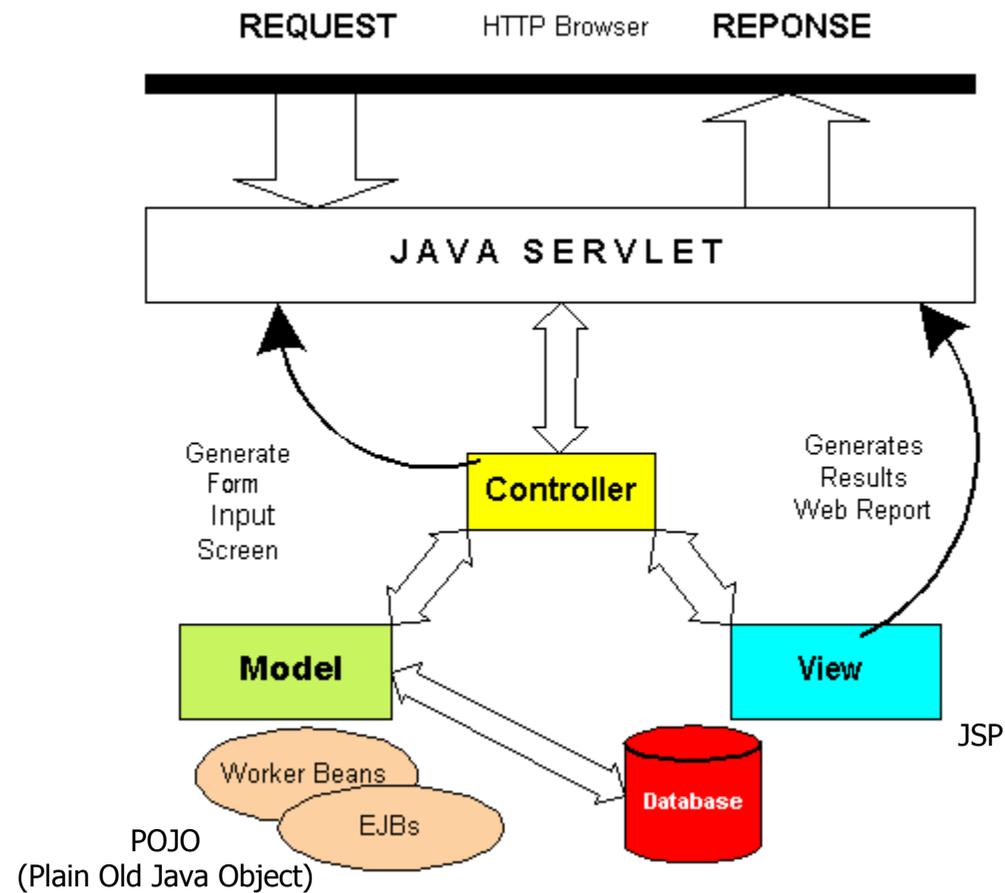


JSP Model 2 Architecture

Model 2에서는 사용자 Request를 Servlet(Controller)에서 담당하고 Response는 JSP(View), Data source의 접근 처리는 JavaBean(Model)에서 담당

MVC 개념

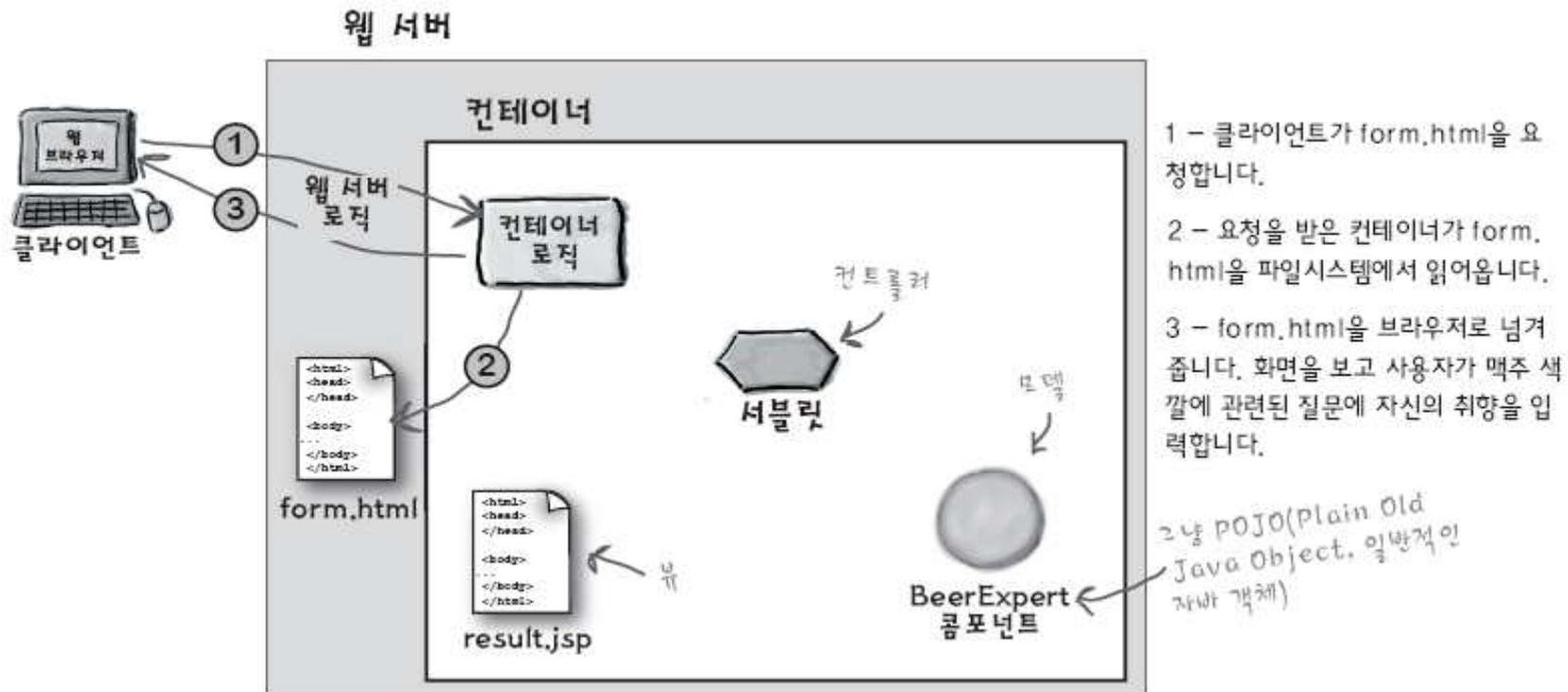
- JSP와 Servlet을 이용한 구현



JSP기반 MVC 구현

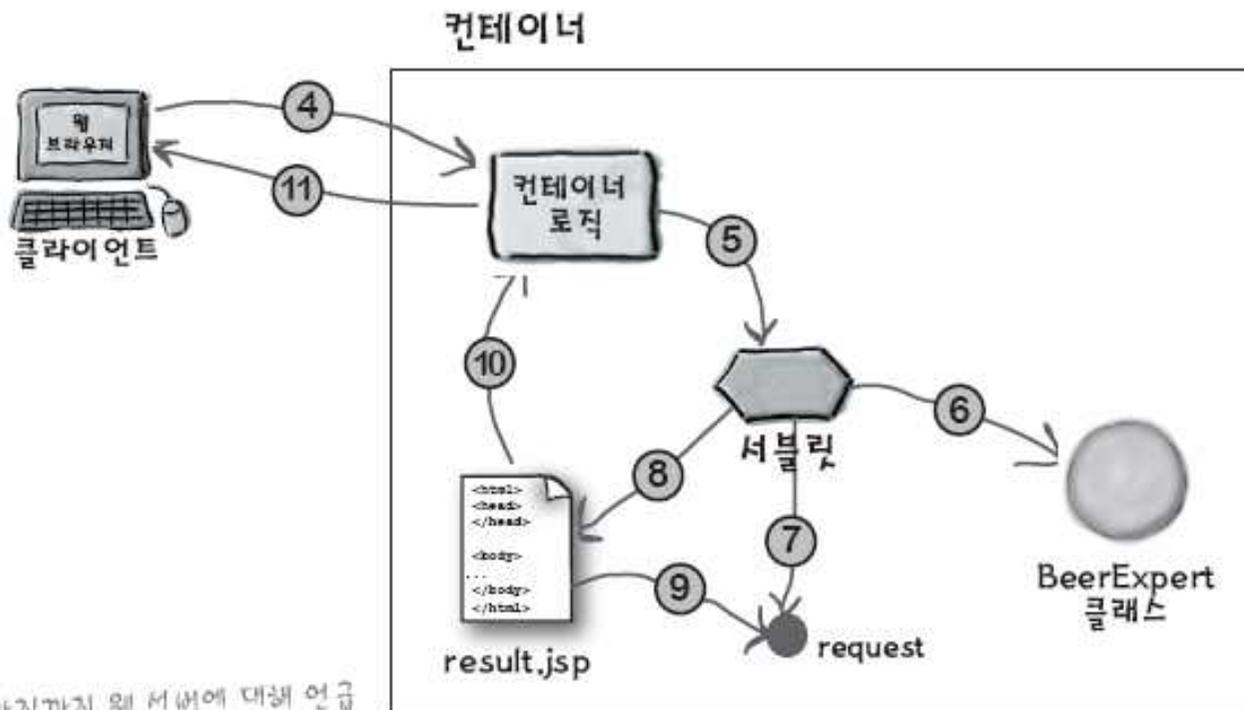
□ Web Application 실행 순서

■ HTML(정적 콘텐츠) 요청/응답 순서



JSP기반 MVC 구현

- Web Application 실행 순서
 - 서블릿 요청/응답 순서



4 - 사용자가 선택한 정보를 컨테이너로 보냅니다.

5 - 컨테이너는 URL을 분석하여 담당 서블릿을 찾아 요청을 넘깁니다.

6 - 서블릿은 BeerExpert 클래스를 호출합니다.

7 - BeerExpert 클래스는 맥주에 대한 조언을 서블릿으로 넘겨줍니다. 서블릿은 이 정보를 Request 객체에 저장합니다.

8 - JSP에 이 Request 객체를 포워딩(forward)합니다.

9 - JSP는 서블릿이 넣어 놓은 정보를 Request 객체에서 추출합니다.

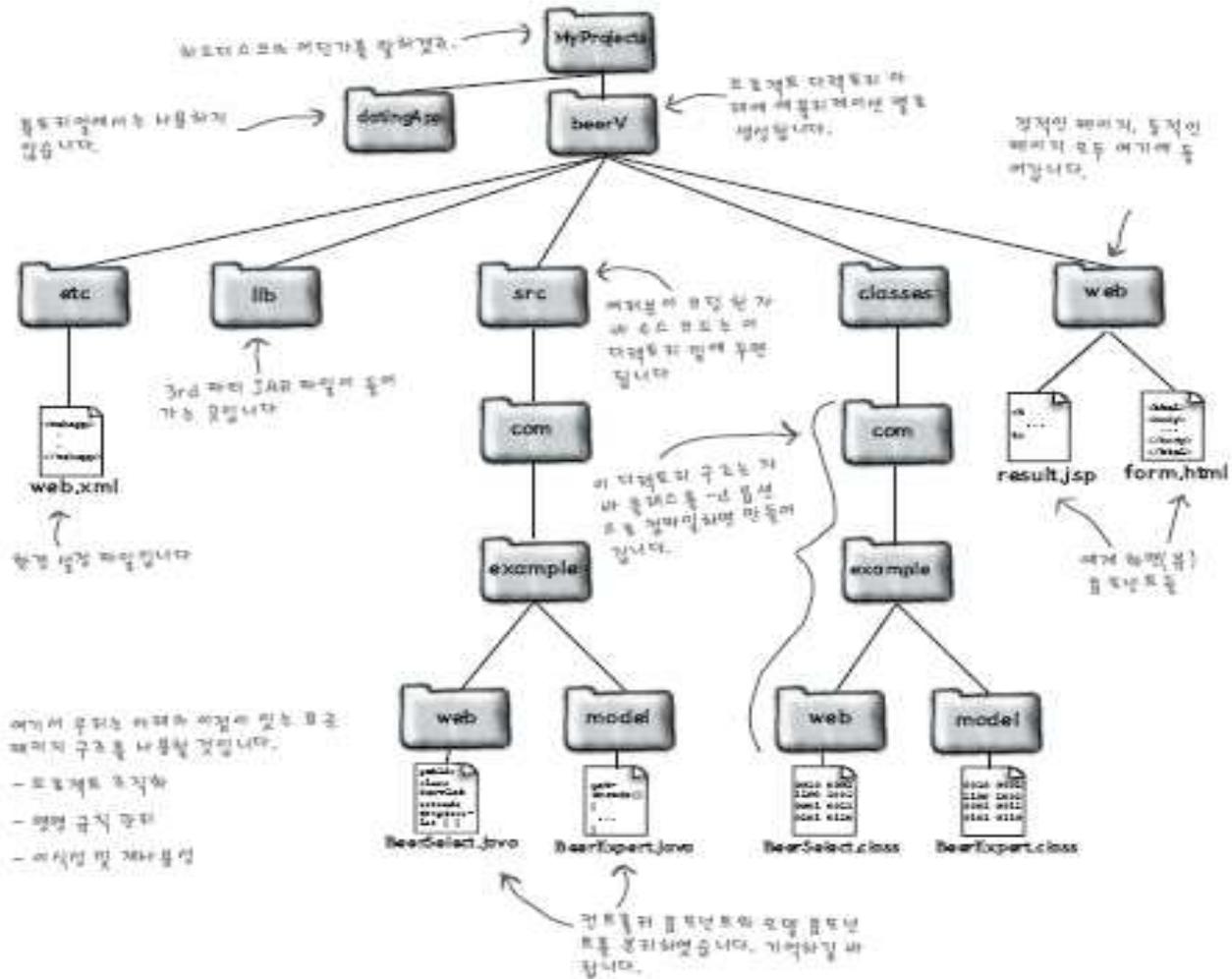
10 - JSP는 여기에 바탕하여 HTML 페이지를 작성합니다.

11 - 컨테이너는 이 페이지를 한껏 기대하고 있는 사용자에게 넘겨줍니다.

아직까지 웹 서버에 대해 언급하지 않았지만, 이따는 가정하에 시나리오를 작성해본 것입니다.

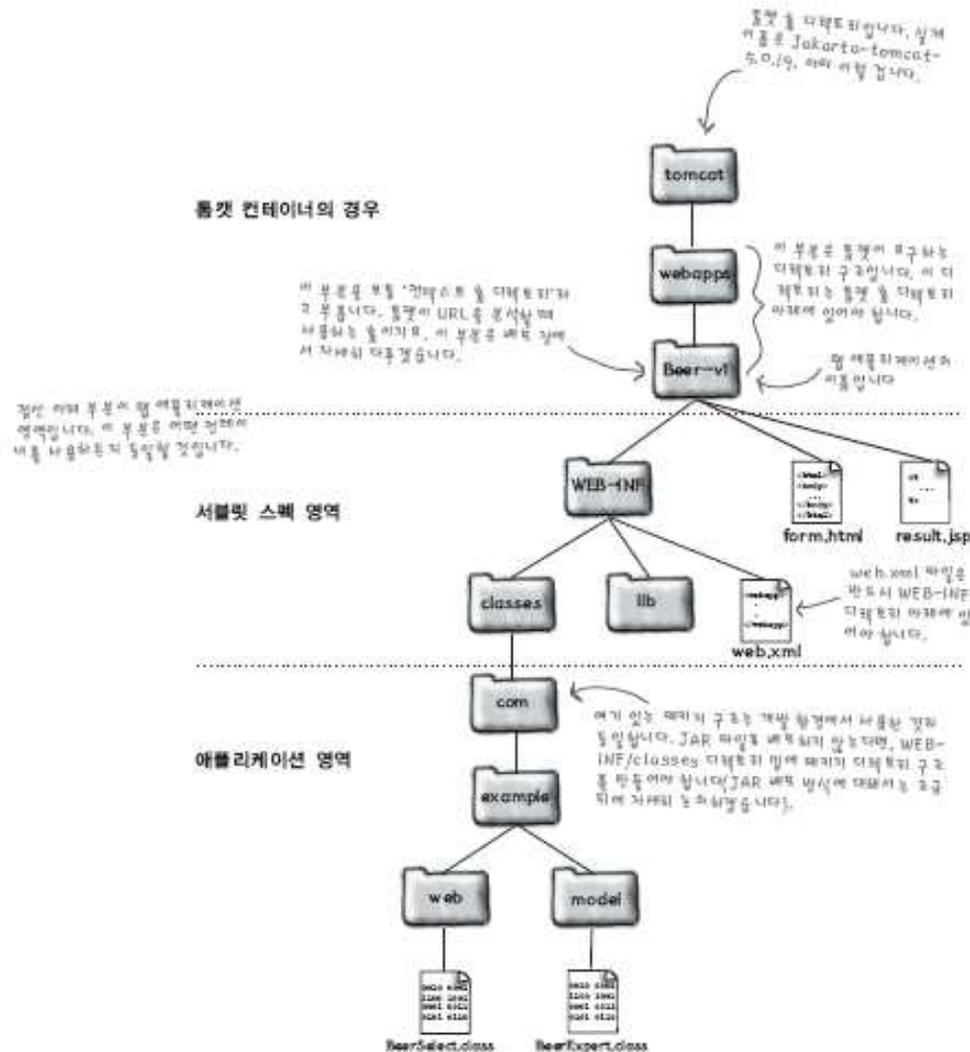
JSP기반 MVC 구현

Web Application 개발 환경 구성



JSP기반 MVC 구현

□ Web Application 배포 환경 구성



MVC 구현 프로그램

- 요구사항 : MVC를 이용한 구현 프로그램
 - 프로그램 명 : 맥주 선택 웹 응용
 - 구현 목표 : 맥주의 색 스타일을 선택하여 제출하면, 적절한 맥주를 추천해주는 웹 프로그램 작성



UI-01

UI-02

MVC 구현 프로그램

□ 버전 1

- HTML(View)- Servlet(Controller)을 이용하여 구현
 - JSP View, Model 구현 없음
- Servlet에서 응답 로직을 처리하여 응답화면 생성

□ 버전 2

- HTML(View)-Servlet(Controller)-POJO(Model)을 이용하여 구현
 - JSP 구현 없음
- POJO(Model)을 이용하여 판단 로직 구현

□ 버전 3

- HTML,JSP(View)-Servlet(Controller)-POJO(Model) 구현
 - MVC 모두 구현
- Servlet에서 JSP 포워딩

MVC 구현(버전 1): Servlet 구현

- 화면 설계 : UI-01
 - HTML을 이용한 첫번째 화면 구현

```
<html><body>
<h1 align="center">Beer Selection Page</h1>
<form method="POST"
  action="SelectBeer.do">
  Select beer characteristics<p>
  Color:
  <select name="color" size="1">
    <option>light
    <option>amber
    <option>brown
    <option>dark
  </select>
  <br><br>
  <center>
    <input type="SUBMIT">
  </center>
</form></body></html>
```

← 왜 GET을 안 쓰고 POST를 사용 했을까요?

← 호출할 서블릿이 무엇인지를 지시하는 HTML 코드. SelectBeer.do라 그밖에 안 쓴 것을 보면 알겠지만, 앞에 디렉토리 구조가 없습니다. 그렇다면 논리적인 이름이라는 말인데...

← (select) 태그에 대한 코드입니다. 다양한 옵션이 있으니 스스로 공부하기 바랍니다. 'size=1'이 무엇을 의미하는지 모르진 않겠죠?

MVC 구현(버전 1): Servlet 구현

- 배치 app 설계
 - web.xml 의 구현

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <servlet>
    <servlet-name>Ch3 Beer</servlet-name>
    <servlet-class>com.example.web.BeerSelect</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Ch3 Beer</servlet-name>
    <url-pattern>/SelectBeer.do</url-pattern>
  </servlet-mapping>

</web-app>
```

이 이름은 가공의 이름입니다. DD
내에서만 사용합니다.

서블릿 클래스의 완전한 이름을
기술합니다.

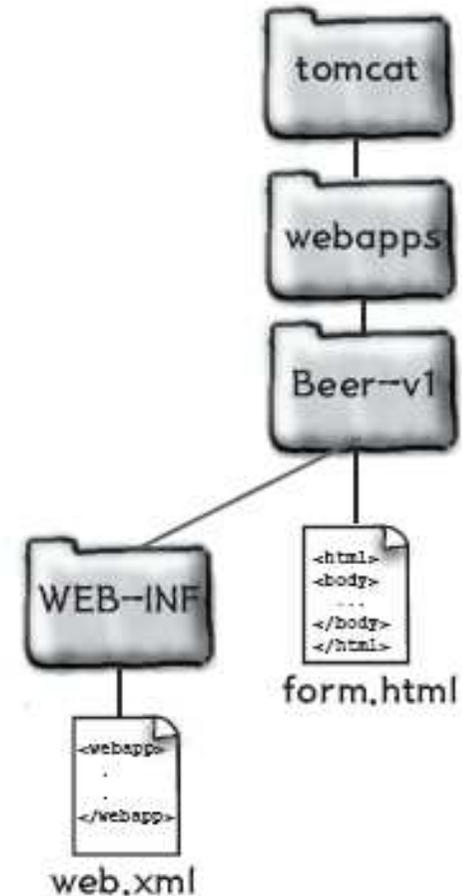
클라이언트가 사용할 이름입니다. .do는 큰 의미가
없습니다.

/를 빼뜨리지 마세요.

MVC 구현(버전 1): Servlet 구현

□ 배치 테스트

- 배포 서술자(DD)를 배포 환경에 복사
 - tomcat / webapps / Beer-v1 / WEB-INF / 에 복사



MVC 구현(버전 1): Servlet 구현

□ 실행 설계

■ 서블릿 매핑: 논리적인 이름을 서블릿 클래스 파일에 매핑하기

① 다이아나는 폼 화면에서 색깔을 선택하고, 제출(서밋) 버튼을 클릭합니다. 브라우저는 다음의 URL을 생성합니다.

`/Beer-v1/SelectBeer.do`

↑ 서버의 루트 디렉터리
 ↑ 웹 애플리케이션 컨텍스트* 루트
 ↑ 자원에 대한 논리적인 이름



```
POST /Beer-v1/SelectBeer.do
HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U;
PPC Mac OS X Mach-O; en-US; rv:1.4)
Gecko/20030624 Netscape/7.1
Accept: text/xml,application/
xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,video/x-
mng,image/png,image/jpeg,image/
gif;q=0.2,*/*;q=0.1
```



*역지주: 보통 컨텍스트라는 영어 단어에 꼭 맞는 말이 없는데 대부분 환경이라는 뜻으로 이해하면 무난합니다.

HTML 안에는 `/Beer-v1/`을 표기하지 않습니다. 단지

`<form method="POST" action="SelectBeer.do">`라고만 코딩합니다.

이렇게만 하면 브라우저가 요청을 보낼 때 자동으로 `SelectBeer.do` 앞에 `/Beer-v1/`을 붙입니다. 어떻게 그것이 가능하냐고요? 원래 `form.html`을 호출할 때 루트가 `/Beer-v1/`이기 때문이죠. `SelectBeer.do`를 상대 경로로 인식하고 절대 경로로 바꿔주는 것입니다.

MVC 구현(버전 1): Servlet 구현

□ 실행 설계

■ 서블릿 매핑: 논리적인 이름을 서블릿 클래스 파일에 매핑하기

- ② 컨테이너는 DD의 `<servlet-mapping>` 항목에서 `/SelectBeer.do`라는 값을 가진 `<url-pattern>`을 찾습니다. 여기서 `/`는 컨텍스트 루트를 의미하며, `SelectBeer.do`는 자원의 논리적인 이름입니다.



```
<web-app>
  <servlet>
    <servlet-name>
      Ch3 Beer
    </servlet-name>
    <servlet-class>
      com.example.web.BeerSelect
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>
      Ch3 Beer
    </servlet-name>
    <url-pattern>
      /SelectBeer.do
    </url-pattern>
  </servlet-mapping>
</web-app>
```

- ③ 컨테이너는 DD에서 `<url-pattern>`과 쌍을 이루는 `<servlet-name>`의 값 `Ch3 Beer`를 기억해둡니다. 물론 이 이름도 실제 서블릿 클래스 파일명이 아님을 알고 있죠? `Ch3 Beer`는 실제 서블릿 클래스 파일명이 아닌 DD 내에서만 사용하는 서블릿명입니다.

컨테이너는 내부 서블릿명과 여기에 일치하는 `<servlet>` 항목을 매핑합니다.



MVC 구현(버전 1): Servlet 구현

□ 실행 설계

- 서블릿 매핑: 논리적인 이름을 서블릿 클래스 파일에 매핑하기

④ 컨테이너는 <servlet-name> 항목 값 Ch3 Beer와 일치하는 <servlet> 항목이 있는지 검색합니다.



```
<web-app>
  <servlet>
    <servlet-name>
      Ch3 Beer
    </servlet-name>
    <servlet-class>
      com.example.web.BeerSelect
    </servlet-class>
  </servlet>
```

⑥ <servlet> 항목을 찾았으면 <servlet-class> 항목 값을 읽어옵니다. 이 값이 바로 요청을 처리할 서블릿 클래스입니다. 서블릿이 초기화된 적이 없다면, 컨테이너는 클래스를 로드하고 초기화합니다.



```
<servlet-mapping>
  <servlet-name>
    Ch3 Beer
  </servlet-name>
  <url-pattern>
    /SelectBeer.do
  </url-pattern>
</servlet-mapping>
</web-app>
```

MVC 구현(버전 1): Servlet 구현

□ 서블릿 구현

- 응답 페이지(UI-02)를 Servlet 내에서 구현하여 리턴

서블릿 코드

```
package com.example.web;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class BeerSelect extends HttpServlet {

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("Beer Selection Advice<br>");
        String c = request.getParameter("color");
        out.println("<br>Got beer color " + c);
    }
}
```

앞에서 만든 개발 환경 및 배포 환경과 일치
해야 함을 잊지 마세요.

HttpServlet은 GenericServlet을 상속받
았고 GenericServlet은 Servlet 인터페이스
를 구현하였습니다.

HTML 폼에서
method=POST라고 했
기에 doPost()를 재정의
하겠습니다.

이 메소드는 ServletResponse 인
터페이스에 있는 메소드입니다.

이 메소드는 ServletRequest 인
터페이스에 있는 메소드입니다. 인
자가 HTML (select) 항목에 있는
name 값과 일치해야 합니다.

여기서는 단지 파라미터값만 화면에 보
여주는 것으로 테스트하겠습니다.

MVC 구현(버전 1): Servlet 구현

□ 서블릿 컴파일 및 deployment

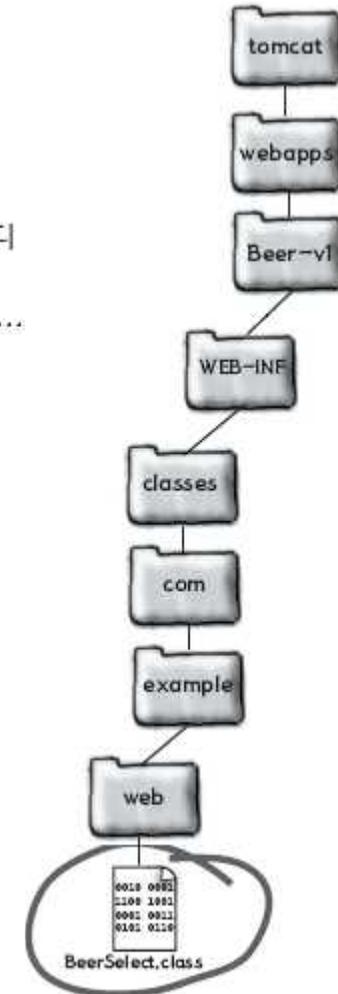
서블릿 배포하기

서블릿을 배포하려면 .class 파일을 /Beer-v1/WEB-INF/classes/com/example/web/ 디렉토리로 옮기면 됩니다.

서블릿 테스트하기

- 1 - 톰캣을 다시 실행합니다.
- 2 - 브라우저를 하나 띄워서 주소 창에 `http://localhost:8080/Beer-v1/form.html`을 입력합니다.
- 3 - 취향에 맞는 맥주 색깔을 선택하고 Submit 버튼을 클릭합니다.
- 4 - 서블릿이 제대로 동작한다면 브라우저에서 아래 글씨가 보일 것입니다:
Beer Selection Advice
Got beer color brown

```
File Edit Window Help SlashdotMe
% cd tomcat
% bin/shutdown.sh
% bin/startup.sh
```



MVC 구현(버전 2): Servlet-POJO 구현

- 모델 클래스 작성, 컴파일
 - 대부분 평이한 일반 자바 클래스(POJO, Plain Old Java Object)로 코딩

```
package com.example.model;
import java.util.*;

public class BeerExpert {
    public List getBrands(String color) {
        List brands = new ArrayList();
        if (color.equals("amber")) {
            brands.add("Jack Amber");
            brands.add("Red Moose");
        }
        else {
            brands.add("Jail Pale Ale");
            brands.add("Gout Stout");
        }
        return (brands);
    }
}
```

맥주 선택에 관한 복잡하고, 전문적인 지식을 가장 간단한 조건 표현식으로 구현하였음을 보세요!

설마 이 부분을 그대로 코딩할 생각은 아니겠지요, 여러분의 톱캣 디렉토리에 맞춰 이 부분을 수정하십시오.

File Edit Window Help Skareboard

```
% cd beerV1
% javac -classpath /Users/bert/Applications2/tomcat/common/lib/
servlet-api.jar:classes:. -d classes src/com/example/model/BeerExpert.java
```

MVC 구현(버전 2): Servlet-POJO 구현

□ 서블릿 구현

```
package com.example.web;

import com.example.model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class BeerSelect extends HttpServlet {

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

BeerExpert 클래스의 패키지를 import하는 것. 잊지 마세요.

HttpServlet을 상속받아 필요한 메소드만 재 정의합니다.

MVC 구현(버전 2): Servlet-POJO 구현

□ 서블릿 구현

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("Beer Selection Advice<br>");
String c = request.getParameter("color");

BeerExpert be = new BeerExpert();
List result = be.getBrands(c);
Iterator it = result.iterator();
while(it.hasNext()) {
    out.print("<br>try: " + it.next());
}
}
}
```

BeerExpert 클래스 인스턴스를 하나 만들고 getBrands() 메소드를 호출합니다.

getBrands()가 리턴한 ArrayList를 루핑하며 하나씩 출력합니다(최종 버전(버전 3)에서 서블릿이 아니라 JSP에서 출력하도록 수정할 것임).

MVC 구현(버전 2): Servlet-POJO 구현

□ 배치 및 실행

■ 서블릿(버전 2)과 모델 클래스 배포, 테스트

서블릿 뿐만 아니라 모델도 배포해야 합니다.
순서는 다음과 같습니다:

1 - 서블릿 .class 파일을 아래 디렉토리로 복사합니다:

```
../Beer-v1/WEB-INF/classes/com/example/web/
```

아마 이전에 서블릿 버전 1 .class 파일이 있을 겁니다. 이를 대체하는 거죠.

2 - 모델 .class 파일을 아래 디렉토리로 복사합니다:

```
../Beer-v1/WEB-INF/classes/com/example/model/
```

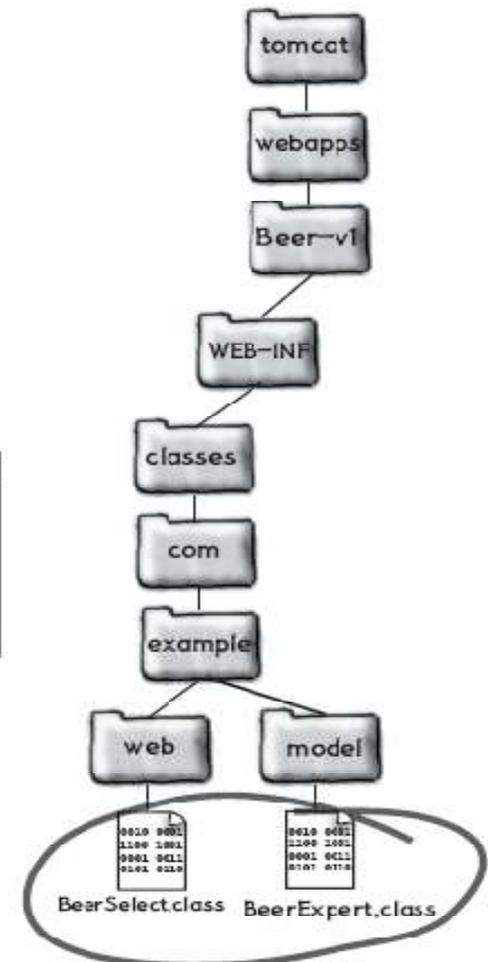
3 - 톰캣을 재시작합니다.

4 - 브라우저에서 form.html을 띄워
애플리케이션이 제대로 동작하는지 테스트
합니다.

아래와 같은 출력이 화면에 나와야 합니다.

```
Beer Selection Advice  
try: Jack Amber  
try: Red Moose
```

```
File Edit Window Help Shell  
% cd tomcat  
% bin/shutdown.sh  
% bin/startup.sh
```



MVC 구현(버전 3): JSP-Servlet-POJO

□ JSP 구현

```
<%@ page import="java.util.*" %>
<html>
<body>
<h1 align="center">Beer Recommendations JSP</h1>
<p>
<%
    List styles = (List)request.getAttribute("styles");
    Iterator it = styles.iterator();
    while(it.hasNext()) {
        out.print("<br>try: " + it.next());
    }
%>
</body>
</html>
```

← "page 지시자(directive)"라고 부르는 부분입니다. 이름만 들어도 무엇을 의미하는지 알 수 있겠죠?

← 표준 HTML 코드군요(JSP에서는 이런 것을 "템플릿 텍스트(Template Text)"라고 부릅니다).

← Request 객체로부터 속성(Attribute)을 읽어오는 부분이군요. 조금 있다가 속성이 무엇인지에 대해 속 시원히 말하겠습니다. 어떻게 값을 설정하고, 다시 읽어오는 지까지도 말입니다.

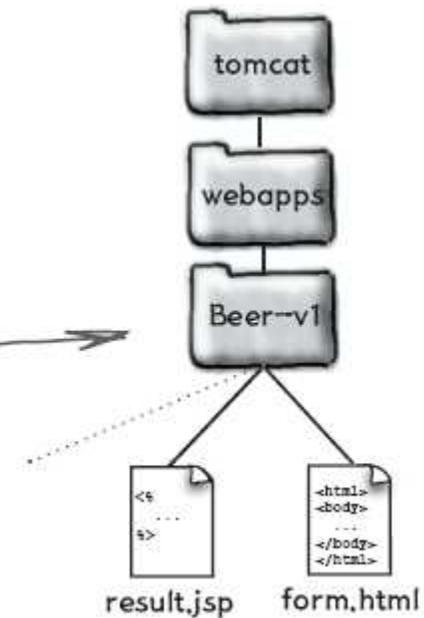
<%%> 항목 안에 표준 자바 언어를 코딩합니다. 이런 걸 스크립틀릿 코드라고 부릅니다.

MVC 구현(버전 3): JSP-Servlet-POJO

□ JSP 뷰 Deployment

JSP는 컴파일하지 않습니다(JSP 최초 호출 시 컴파일러가 이 일을 합니다).
우리가 해야 할 일은:

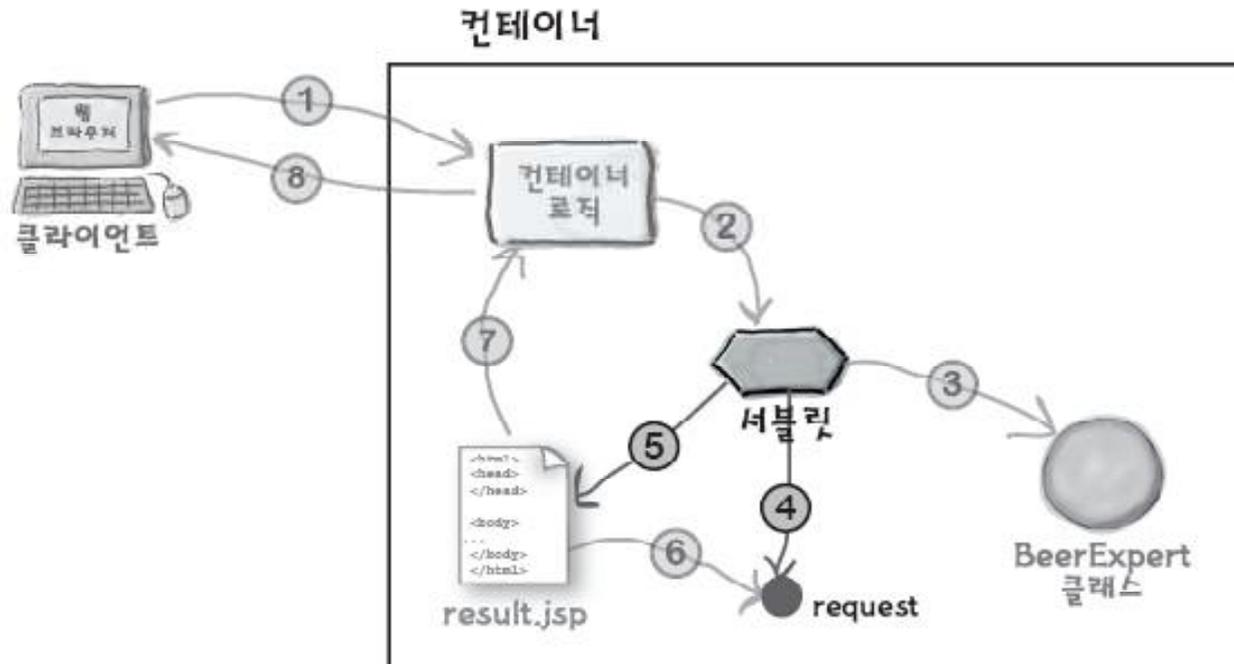
- 1 - 이름을 지어줘야 겠지요. "result.jsp"
- 2 - 개발 환경에 저장해야 합니다. "/web/"
- 3 - 배포 환경에 복사해야 합니다. "/Beer-v1/"



MVC 구현(버전 3): JSP-Servlet-POJO

□ MVC 실행 설계

- 1 - Request 객체에 모델 컴포넌트로부터 받은 정보를 저장하는 것. 그래야 JSP가 이걸 꺼내 볼 수 있으니까요(아래 그림의 4번째 단계).
- 2 - 컨테이너에게 요청을 result.jsp로 넘겨줄(forward) 것을 요청하는 것(5번째 단계).



1 - 브라우저가 컨테이너에게 요청을 보냅니다.

2 - 컨테이너는 URL이 올바른 서블릿을 호출한 것인지를 판단한 다음, 요청을 서블릿으로 넘깁니다.

3 - 서블릿은 BeerExpert에게 도움을 청하겠지요.

4 - BeerExpert 클래스가 값(추천할 맥주 목록)을 리턴합니다. 서블릿은 이 내용을 Request 객체에 기억시켜둡니다.

5 - 서블릿은 요청을 JSP 파일에게 넘깁니다(forward).

6 - JSP는 Request 객체에서 서블릿이 넣어 놓은 값을 끄집어 냅니다.

7 - JSP는 페이지를 생성합니다.

8 - 컨테이너는 페이지를 클라이언트로 보냅니다.

MVC 구현(버전 3): JSP-Servlet-POJO

□ 서블릿 구현

```
package com.example.web;

import com.example.model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class BeerSelect extends HttpServlet {

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

        // response.setContentType("text/html");

        // remove the old test output
        // PrintWriter out = response.getWriter();
        // out.println("Beer Selection Advice<br>");

        String c = request.getParameter("color");

        // out.println("<br>Got beer color " + c);

        BeerExpert be = new BeerExpert();
        List result = be.getBrands(c);
    }
}
```

이제 HTML 화면 출력은 JSP에게 일임했으니, 서블릿에 있는 것은 지우겠습니다. 지우지 말라구요. 알겠습니다. 참고로 하기 위해 주석 처리만 하지요.

MVC 구현(버전 3): JSP-Servlet-POJO

- 서블릿 구현
 - JSP Forwarding의 구현

```
request.setAttribute("styles", result);
```

JSP가 나중에 읽을 수 있게 Request 객체의 속성(Attribute)에 값을 설정합니다. 나중에 styles란 값으로 JSP에서 이 객체를 읽어 올 것입니다.

```
RequestDispatcher view =  
    request.getRequestDispatcher("result.jsp");
```

JSP로 작업을 부탁할 RequestDispatcher를 인스턴스화 합니다.

```
view.forward(request, response);
```

RequestDispatcher는 컨테이너에게 JSP를 준비하라고 요청합니다. 그 다음 JSP에게 request/response 객체를 넘깁니다.

MVC 구현(버전 3): JSP-Servlet-POJO

□ 배치 및 실행

1 - 서블릿 .class 파일을 ../Beer-v1/WEB-INF/classes/com/example/web/ 디렉토리로 복사합니다(다시 말하지만 이전 버전을 대체하는 것입니다).

2 - 톰캣을 다시 실행합니다.

3 - form.html로 애플리케이션이 제대로 동작하는지 테스트합니다.

```
File Edit Window Help SaveYourself
% cd tomcat
% bin/shutdown.sh
% bin/startup.sh
```



이것이 보이나요? 보인다고요...
축하합니다. →

