03: MVC 모델을 이용한 웹 애플리케이션 작성

학습 목표

- 웹 애플리케이션 개발 순서를 알아본다
- 웹 애플리케이션의 실행 순서를 이해한다.
- 웹 애플리케이션의 반복적인 개발 및 테스트를 이해한다.

1.웹 애플리케이션 개발순서

2. 웹 애플리케이션 실행순서

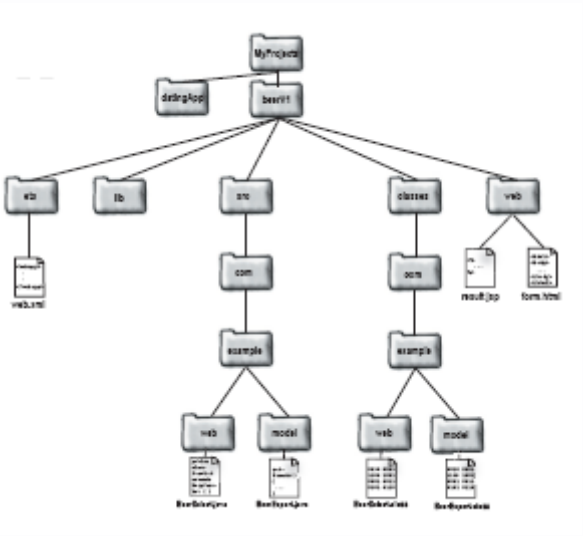
3. 웹 애플리케이션 배포환경 구성

4. 웹 애플리케이션 개발 및 테스트

웹 애플리케이션 >> 개발 순서

■ 사용자 화면 설계

■ 개발 환경 구성



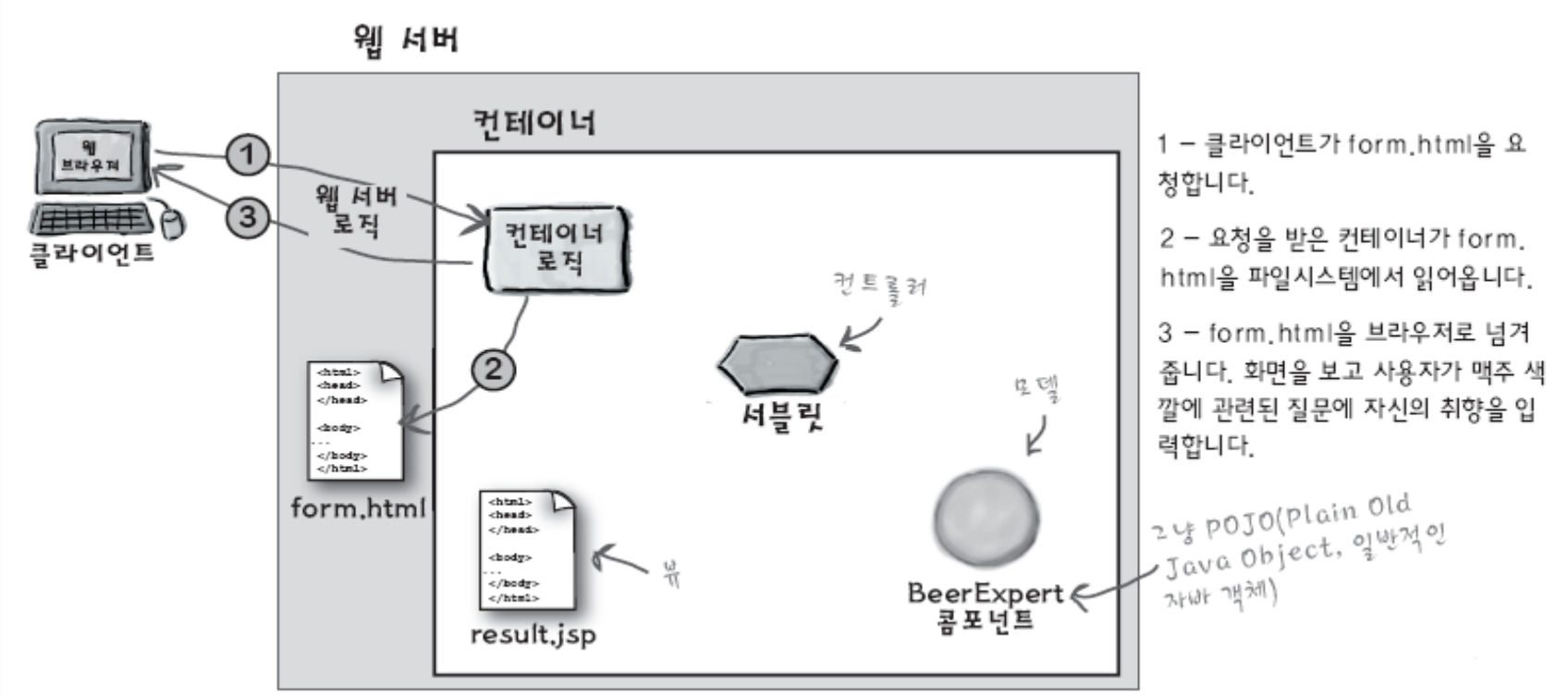
웹 애플리케이션 >> 개발 순서

■ 배포 환경 구성

■ 반복적인 개발과 테스트

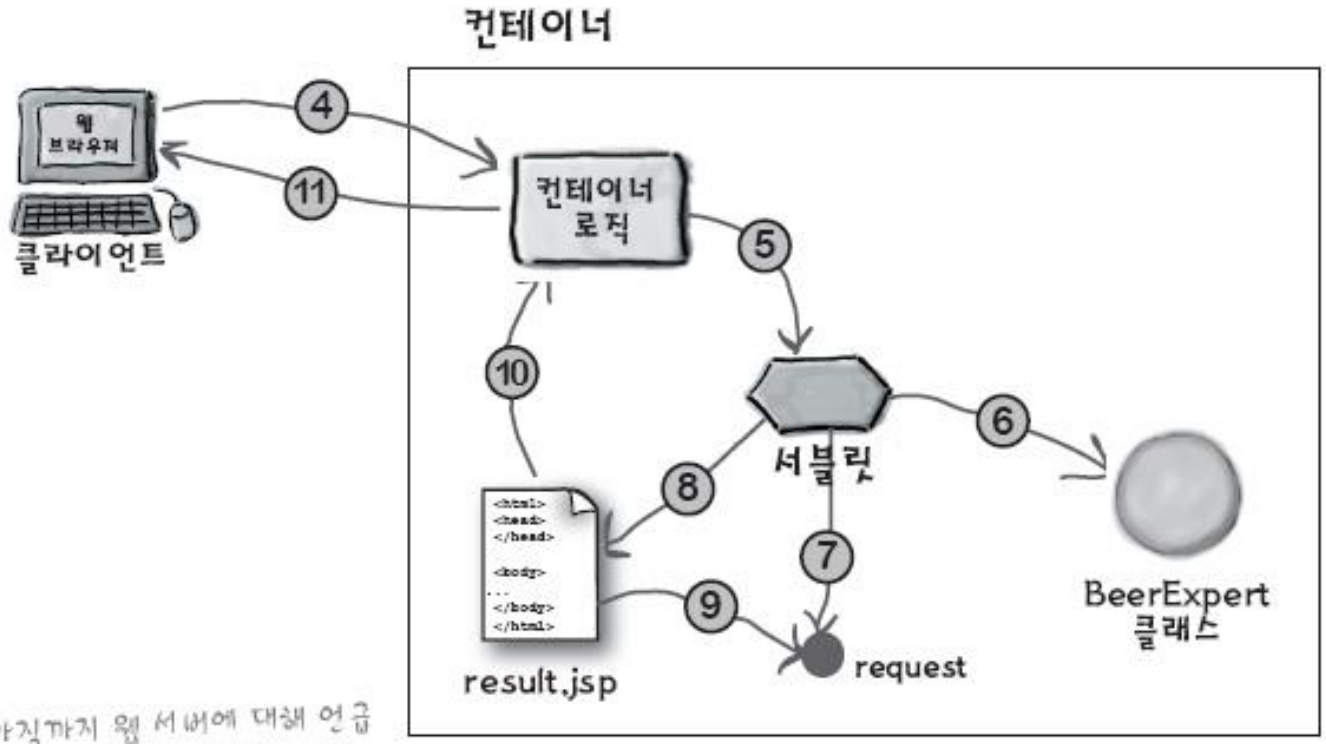
웹 애플리케이션 >> 실행 순서

■ HTML(정적 컨텐츠) 요청/응답 순서



웹 애플리케이션 >> 실행 순서

■ 서블릿 요청/응답 순서



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

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.

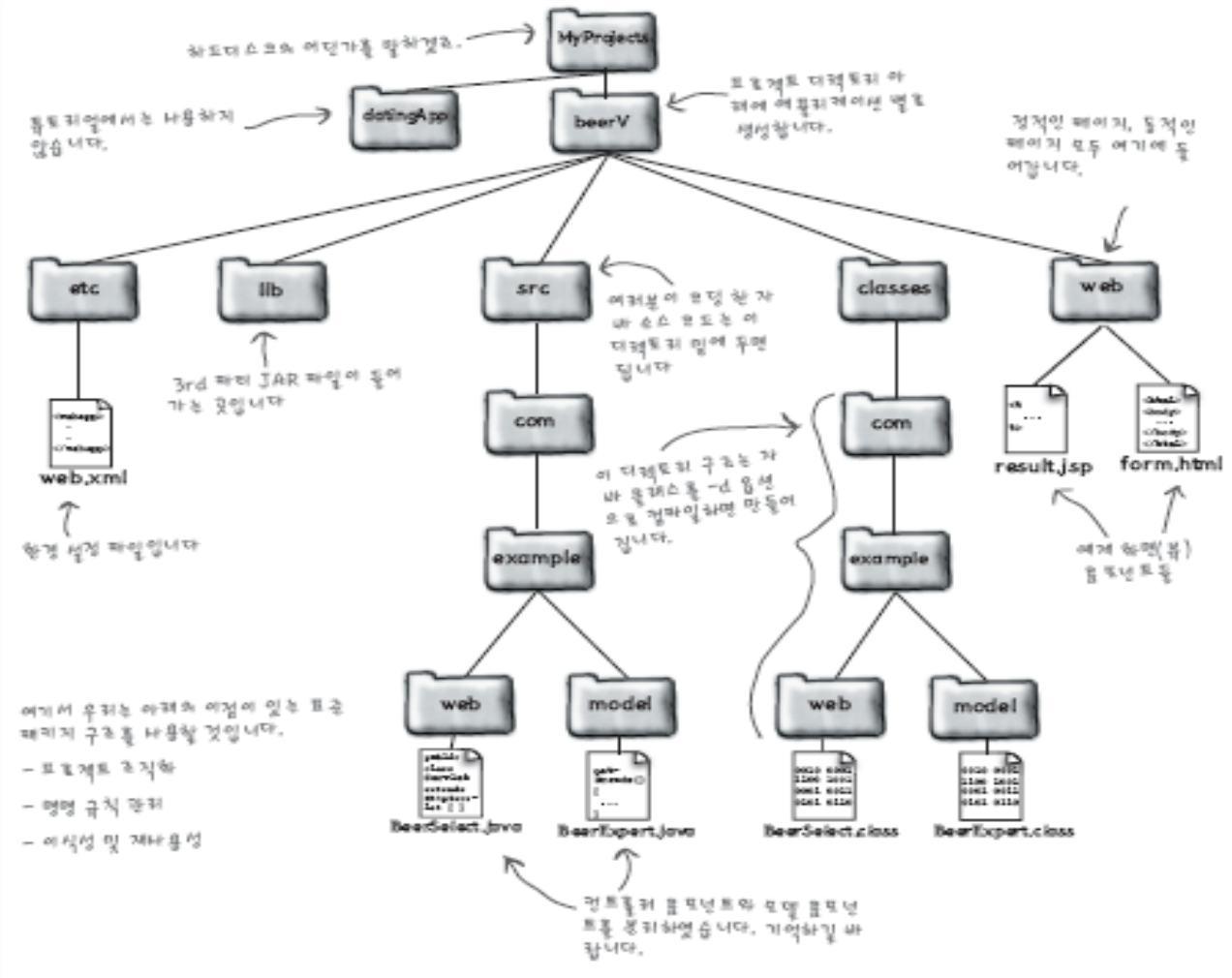
8 – JSP에 이 Request 객체를 포워딩(forward)합니다.

9 – JSP는 서블릿이 넣어 놓은 정보를 Request 객체에서 추출합니다.

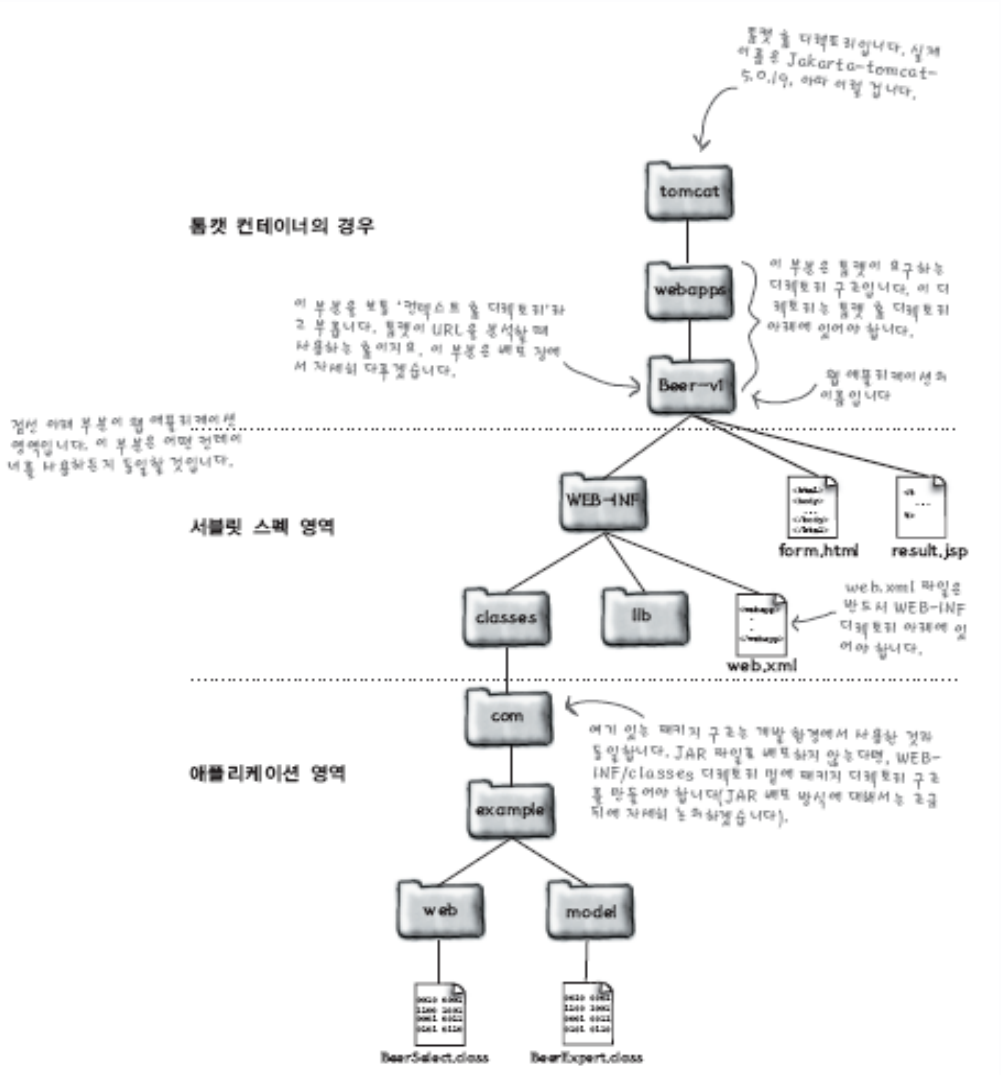
10 – JSP는 여기에 바탕하여 HTML 페이지를 작성합니다.

11 – 컨테이너는 이 페이지를 한껏 기대하고 있는 사용자에게 넘겨줍니다.

웹 애플리케이션 >> 개발 환경 구성



웹 애플리케이션 >> 배포 환경 구성



웹 애플리케이션 >> 개발과 테스트

■ 화면(HTML) 생성 및 테스트

```
<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 사용했을까요?

호출할 서블릿이 무엇인지를 지시하는 HTML 코드죠. SelectBeer.do라고밖에 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?

웹 애플리케이션 >> 개발과 테스트

■ 배포 서술자(DD) 생성 : web.xml 작성

```
<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.

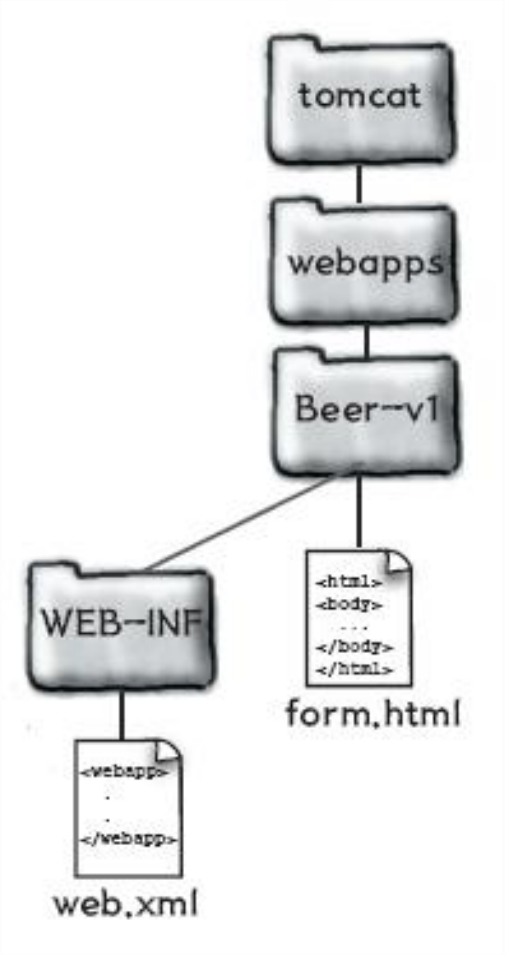
클라이언트가 사용할 이름입니다. .do는 큰 의미가 없습니다.

/를 빠뜨리지 마세요.

웹 애플리케이션 >> 개발과 테스트

■ 배포 서술자(DD)를 배포 환경에 복사

tomcat / webapps / Beer-v1 / WEB-INF / 에 복사



웹 애플리케이션 >> 개발과 테스트

■ 서블릿 매핑: 논리적인 이름을 서블릿 클래스 파일에 매핑하기

*역자주: 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.html을 호출할 때 루트가 /Beer-v1/이기 때문이죠. SelectBeer.do을 상대 경로로 인식하고 절대 경로로 바꿔주는 것입니다.

웹 애플리케이션 >> 개발과 테스트

■ 서블릿 매핑: 논리적인 이름을 서블릿 클래스 파일에 매핑하기

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

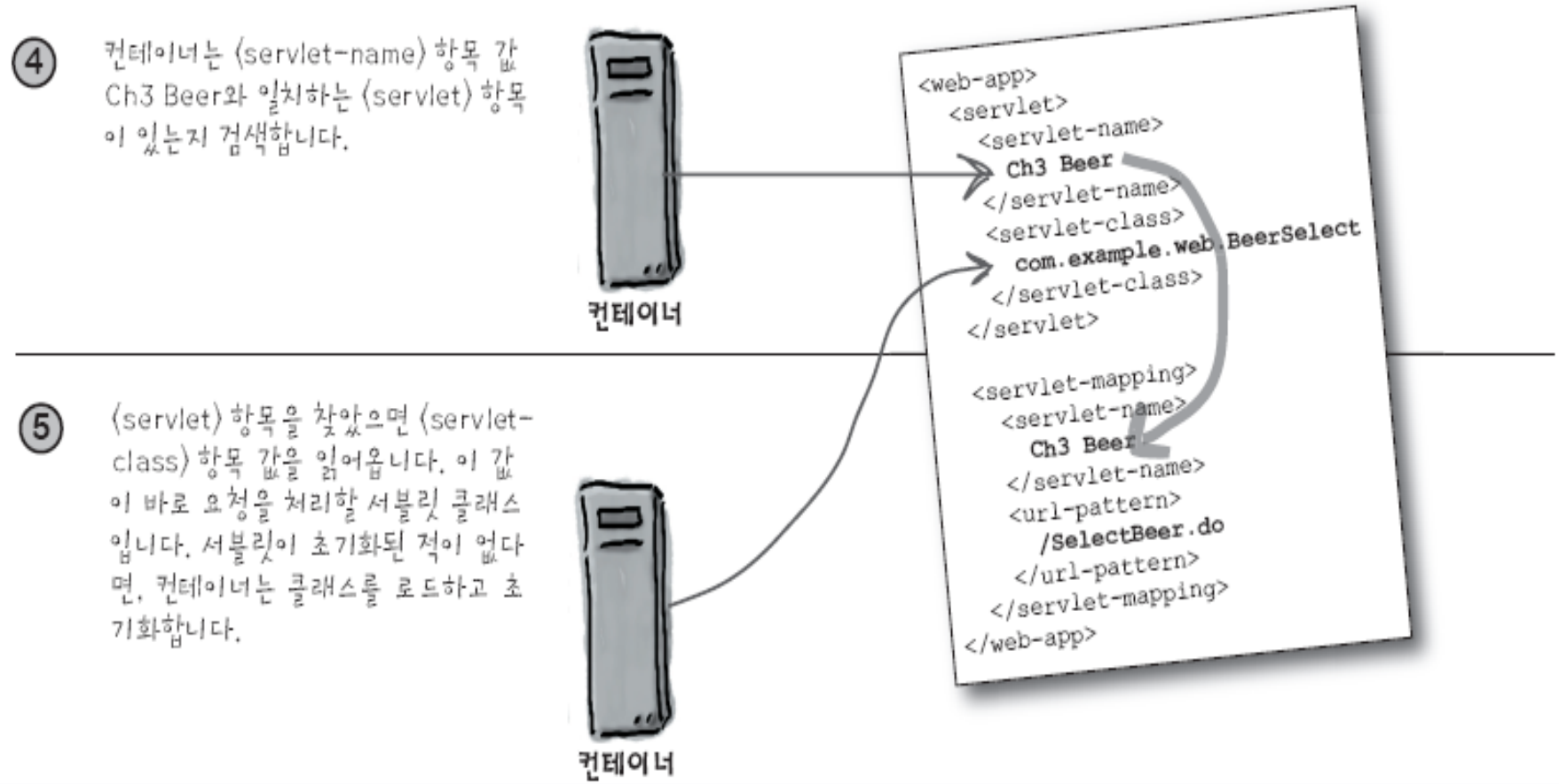

```
<web-app>
  <servlet>
    <servlet-name>
      Ch3 Beer
    </servlet-name>
    <servlet-class>
      com.example.web.BeerSelect
    </servlet-class>
  </servlet>

  <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 Beer를 기억해둡니다. 물론 이 이름도 실제 서블릿 클래스 파일명이 아님은 알고 있죠? Ch3 Beer는 실제 서블릿 클래스 파일명이 아닌 DD 내에서만 사용하는 서블릿명입니다.

컨테이너는 내부 서블릿명과 여기에 일치하는 <servlet> 항목을 매핑합니다.

웹 애플리케이션 >> 개발과 테스트

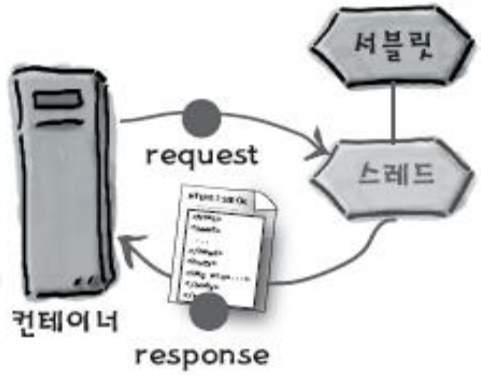
■ 서블릿 매핑: 논리적인 이름을 서블릿 클래스 파일에 매핑하기



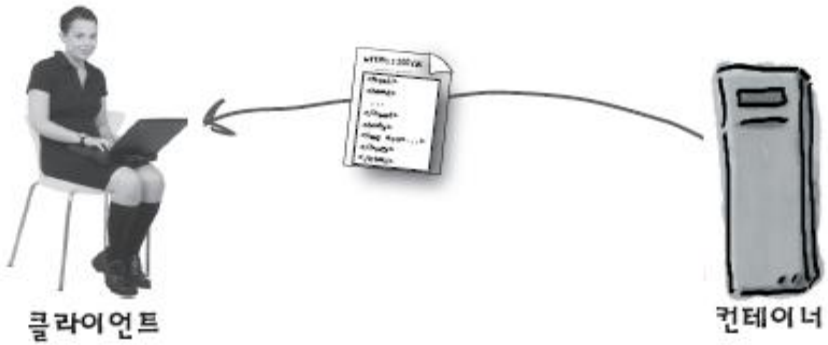
웹 애플리케이션 >> 개발과 테스트

■ 서블릿 매핑: 논리적인 이름을 서블릿 클래스 파일에 매핑하기

⑥ 컨테이너는 요청을 처리할 새로운 스레드를 시작하고 서블릿의 service() 메소드에 Request 객체 참조를 인자로 넘깁니다.



⑦ 스레드가 완료되면 클라이언트에게 응답을 보냅니다. 물론 웹 서버를 통해서 보내겠지요.



웹 애플리케이션 >> 개발과 테스트

■ 서블릿 작성 : 버전 1

서블릿 코드

앞에서 만든 개발 환경 및 배포 환경과 일치해야 함을 잊지 마세요.

```
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);
    }
}
```

HttpServlet은 GenericServlet을 상속받았고 GenericServlet은 Servlet 인터페이스를 구현하였습니다.

HTML 폼에서 method=POST라고 했기에 doPost()를 재정의하겠습니다.

이 메소드는 ServletResponse 인터페이스에 있는 메소드입니다.

이 메소드는 ServletRequest 인터페이스에 있는 메소드입니다. 인자가 HTML <select> 항목에 있는 name 값과 일치해야 합니다.

여기서는 단지 파라미터값만 화면에 보여주는 것으로 테스트하겠습니다.

웹 애플리케이션 >> 개발과 테스트

■ 서블릿 컴파일, 배포, 테스트

서블릿 컴파일하기

개발 환경에서 -d 옵션을 줘서 서블릿을 컴파일합니다.

여러분 환경에 맞게 디렉토리 경로를 수정하세요. tomcat/ 도 동일합니다.

File Edit Window Help UpdateBrain

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

-d 옵션은 컴파일러에게 .class 파일을 클래스 디렉토리에 올바른 패키지 구조대로 생성하라라는 명령입니다. .class 파일이 /beerV1/classes/com/example/web/에 생성될 것입니다.

웹 애플리케이션 >> 개발과 테스트

■ 서블릿 컴파일, 배포, 테스트

서블릿 배포하기

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

서블릿 테스트하기

1 – 톰캣을 다시 실행합니다.

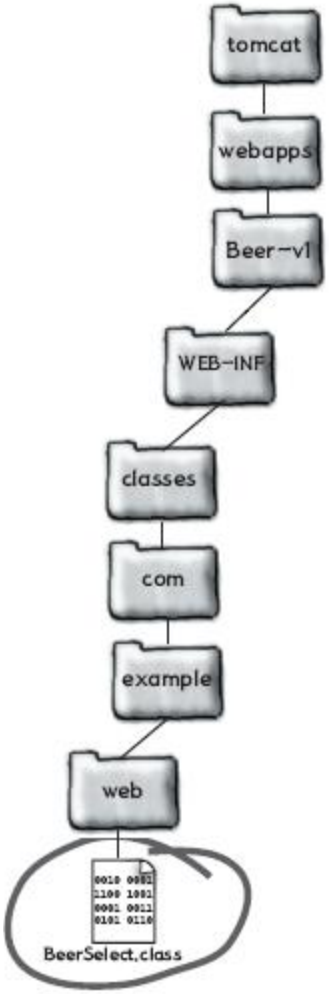
2 – 브라우저를 하나 띄워서 주소 창에 http://localhost:8080/Beer-v1/form.html을 입력합니다.

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
```



웹 애플리케이션 >> 개발과 테스트

■ 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 톰캣 디렉토리에 맞춰 이 부분을 수정하길 바랍니다.

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

웹 애플리케이션 >> 개발과 테스트

■ 서블릿 작성 : 버전 2 (모델 클래스 사용)

```
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 재정의합니다.

웹 애플리케이션 >> 개발과 테스트

■ 서블릿 작성 : 버전 2 (모델 클래스 사용)

```
        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)에서는 서블릿이 아니라 JSP에서 출력하도록 수정할 것임).

웹 애플리케이션 >> 개발과 테스트

■ 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.

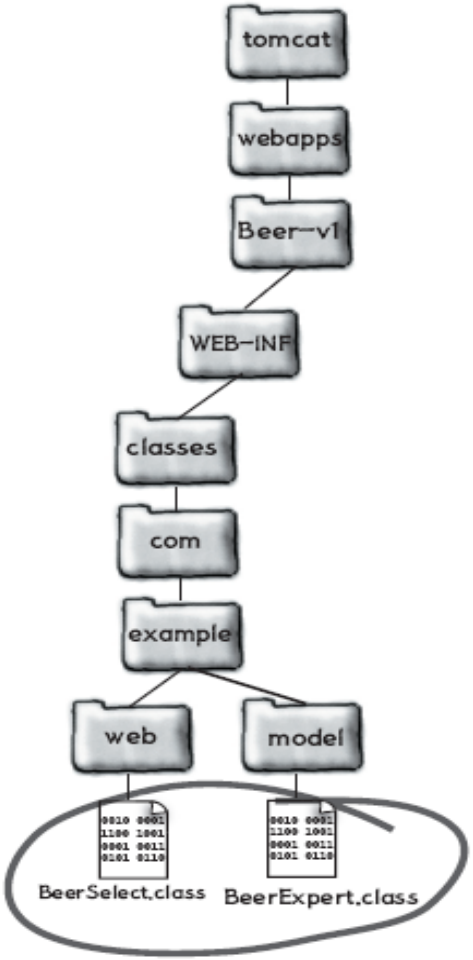
2 – 모델 .class 파일을 아래 디렉토리로 복사합니다:
../Beer-v1/WEB-INF/classes/com/example/model/

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  SellHigh
% cd tomcat
% bin/shutdown.sh
% bin/startup.sh
```



웹 애플리케이션 >> 개발과 테스트

■ JSP 뷰 작성

```
<%@ 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 좀 더 자세히 말하겠습니다. 어떻게 값을 설정하고, 다시 읽어오는지까지도 말입니다.

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

웹 애플리케이션 >> 개발과 테스트

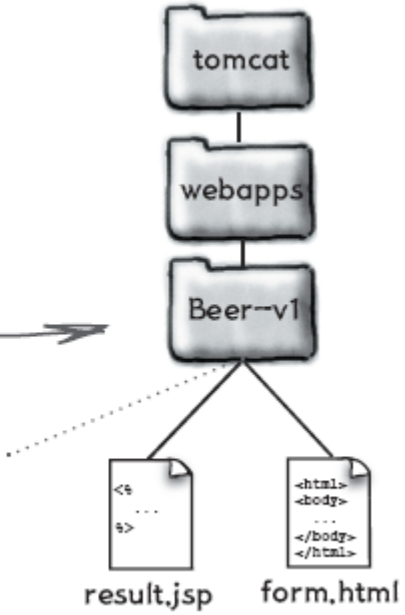
JSP 뷰 배포

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/"

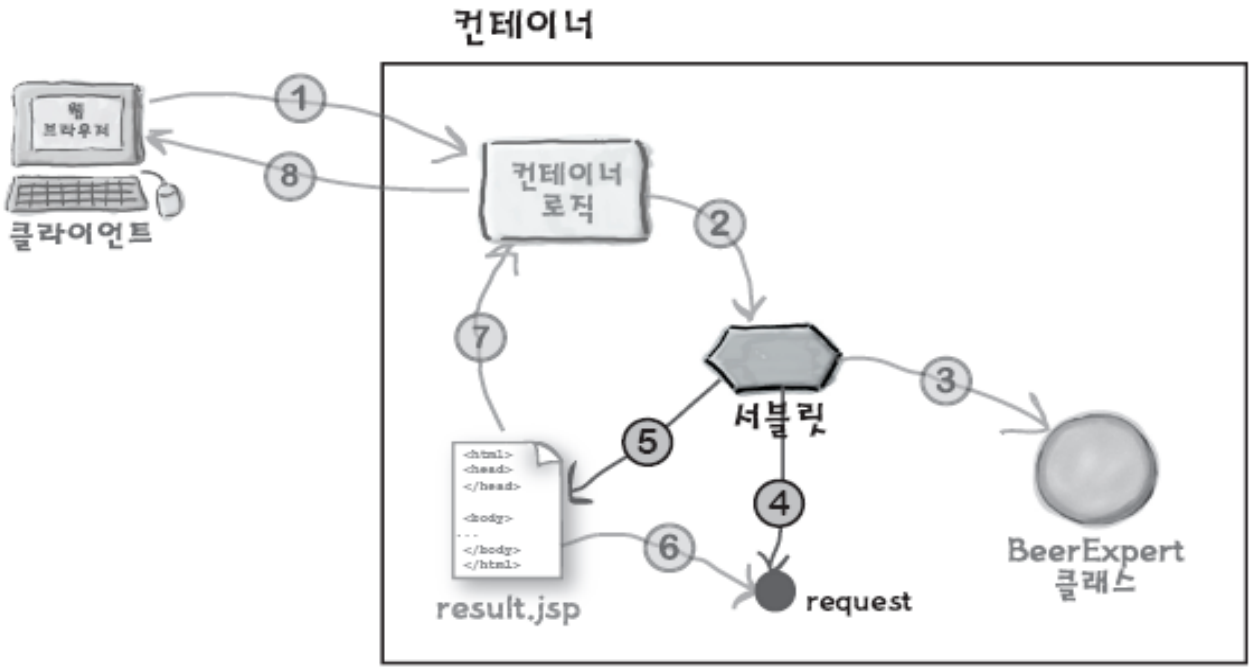


웹 애플리케이션 >> 개발과 테스트

■ 서블릿 작성 : 버전 3 (JSP 호출)

1 – Request 객체에 모델 콤포넌트로부터 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.

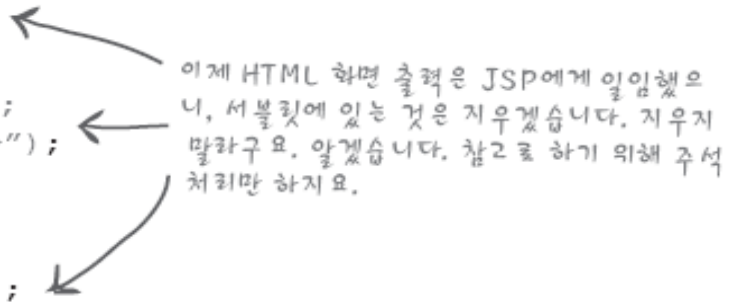
7 – JSP는 페이지를 생성합니다.

8 – 컨테이너는 페이지를 클라이언트로 보냅니다.

웹 애플리케이션 >> 개발과 테스트

■ 서블릿 작성 : 버전 3 (JSP 호출)

```
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);
```



웹 애플리케이션 >> 개발과 테스트

■ 서블릿 작성 : 버전 3 (JSP 호출)

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

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

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

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

JSP로 작업을 부탁할 RequestDispatcher를 인스턴스화 합니다.

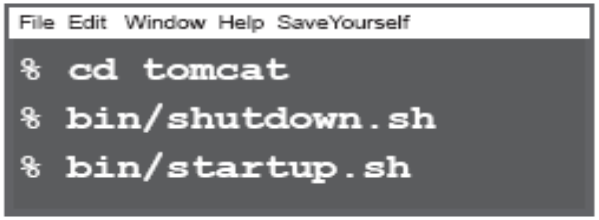
RequestDispatcher는 컨테이너에게 JSP를 준비하라고 요청합니다. 그 다음 JSP에게 request/response 객체를 넘깁니다.

웹 애플리케이션 >> 개발과 테스트

■ 서블릿(버전 3) 배포, 테스트

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

2 – 톰캣을 다시 실행합니다.



3 – form.html로 애플리케이션이 제대로 동작하는지 테스트합니다.

이것이 보이나요? 보인다고요... 축하합니다.