

MVC 패턴

- HTTP Method

- HttpServletRequest/HttpServletResponse

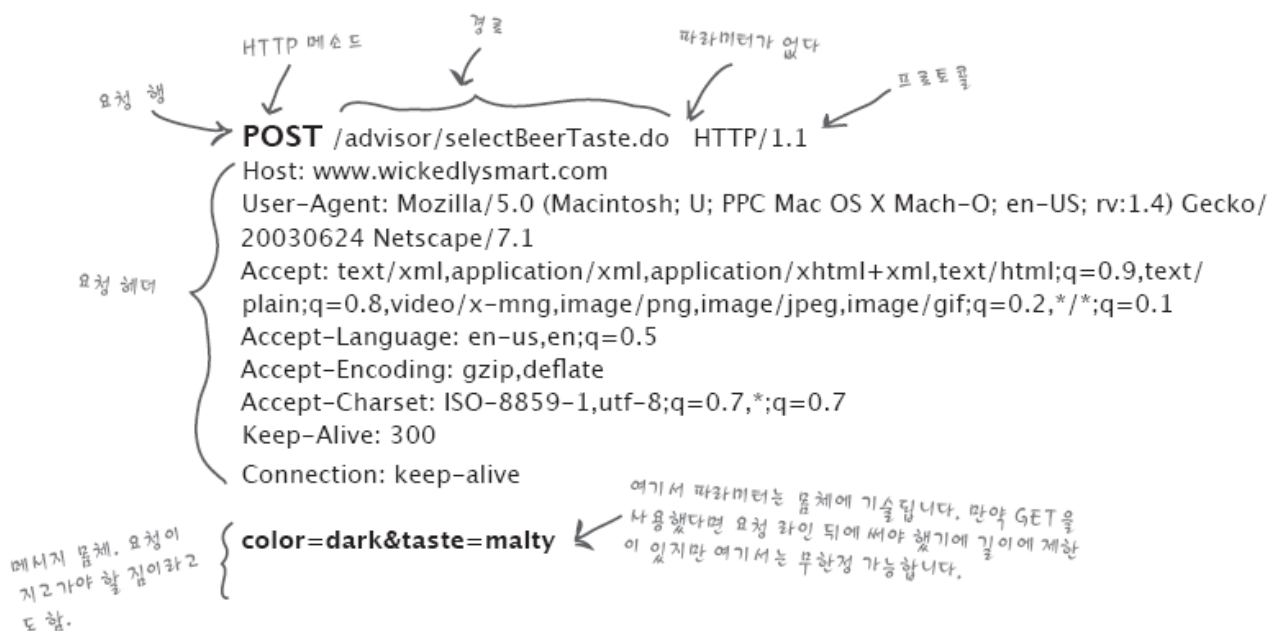
- MVC 모델(디자인)

- MVC 모델을 통한 웹 개발

HTTP Method revisit

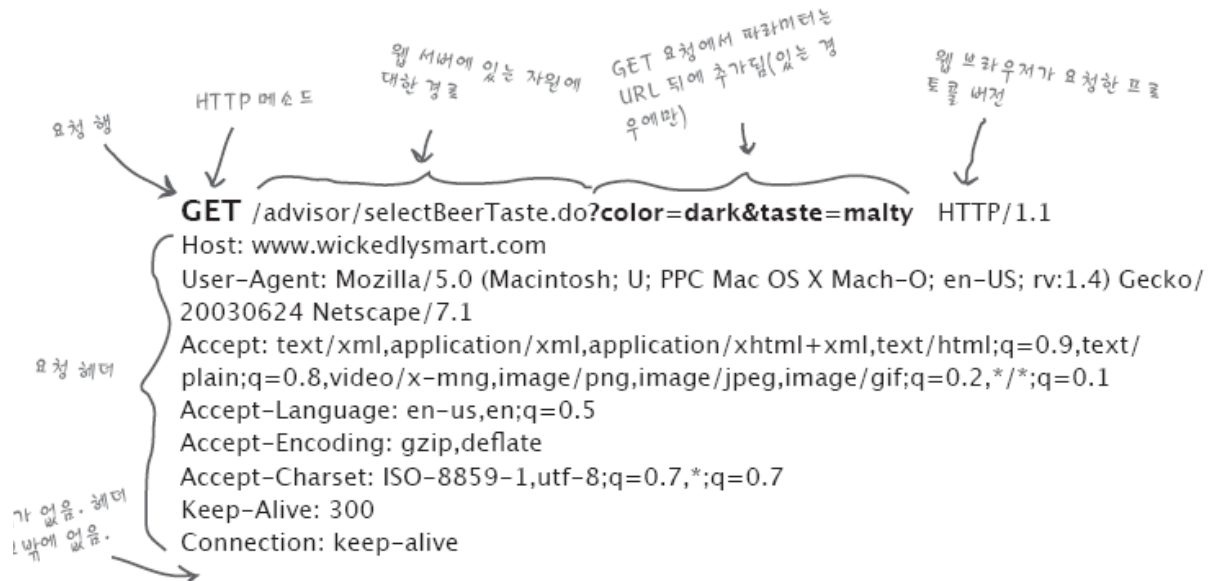


• POST



HTTP Method revisit

- GET



3

HttpServletRequest

- 쿠키, 헤더, 세션 등 HTTP 에 대한 것들에 대한 처리 관련 메소드 포함
- HTTP 프로토콜에 관련된 메소드들이 추가 되어 있음

ServletRequest interface
(javax.servlet.ServletRequest)

<<인터페이스>> ServletRequest
<code>getAttribute(String)</code> <code>getContentTypeLength()</code> <code>getInputStream()</code> <code>getLocalPort()</code> <code>getParameter()</code> <code>getParameterNames()</code> <code>// 기타 메소드 ...</code>

HttpServletRequest interface
(javax.servlet.http.HttpServletRequest)

<<인터페이스>> HttpServletRequest
<code>getContextPath()</code> <code>getCookies()</code> <code>getHeader(String)</code> <code>getQueryString()</code> <code>getSession()</code> <code>getMethod()</code> <code>// 기타 메소드 ...</code>

HttpServletRequest - 예

- Form Parameter

HTTP POST 요청

```
POST /advisor/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
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

옆에 코드는 개발자가 작성하는 부분이 아니라, 브라우저가 자동으로 생성하는 것입니다. 단지 서버로 넘어가는 데이터가 이런 내용이다 정도는 봐 두어야 합니다.

color=dark

서블릿 클래스

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    String colorParam = request.getParameter("color");
    // 여기에는 좀더 화려한 코드를 집어넣으면 되겠죠...
}
// 여기서 colorParam 문자 변수에는 dark라는
// 값이 들어 가겠죠.
```

↑
폼에 있는 이름과 짝이
맞아야 합니다.

HttpServletRequest - 예

- Form Parameters

HTTP POST 요청

```
<form method=POST
  action="SelectBeer.do">
  Select beer characteristics<p>
  Can Sizes: <p>
  <input type=checkbox name=sizes value="12oz"> 12 oz.<br>
  <input type=checkbox name=sizes value="16oz"> 16 oz.<br>
  <input type=checkbox name=sizes value="22oz"> 22 oz.<br>
  <br><br>

  <center>
    <input type="SUBMIT">
  </center>
</form>
```

서블릿 클래스

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    String [] sizes = request.getParameterValues("sizes");
}
```

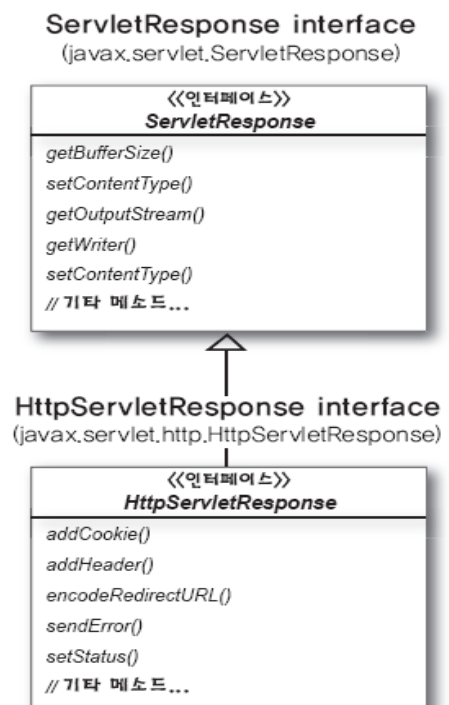
HttpServletRequest - 자주 사용하는 메소드

- 클라이언트 플랫폼 정보 및 브라우저 정보
 - `String client = request.getHeader("User-Agent");`
- Request 에 관련된 쿠키
 - `Cookie[] cookies = request.getCookies();`
- 클라이언트 세션 정보
 - `HttpSession session = request.getSession();`
- Request 의 HTTP 메소드
 - `String theMethod = request.getMethod();`
- Request 의 입력 스트림
 - `InputStream input = request.getInputStream();`

7

HttpServletResponse

- HTTP에 관련된 오류, 쿠키, 헤더 정보에 대한 메소드들이 추가되어 있음



HttpServletResponse - 자주 사용하는 메소드

- 응답 데이터의 데이터 타입 정의
 - response.setContentType("text/html");
 - MIME Type
 - text/html
 - application/pdf
 - video/quicktime
 - image/jpeg
 - application/octet-stream
 - application/x-zip
- 출력 스트림 사용 : 가급적 JSP 사용 권장
 - Writer writer = response.getWriter();
 - OutputStream output = response.getOutputStream();
- 헤더 정보 설정, 쿠키 추가 등
 - response.addCookie();
 - response.addHeader();

9

HttpServletResponse - 예

- 바이너리 데이터 보내기

```
// import 구문 한 무더기
```

```
public class CodeReturn extends HttpServlet {
```

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

```
        response.setContentType("application/jar");
```

이 코딩의 목적은 브라우저에게 지금 우리가 내려 보내는 것이 JAR라는 사실을 알려주는 것입니다. HTML 페이지가 아니기에 application/jar라고 타입을 설정하는 것입니다.

```
        ServletContext ctx = getServletContext();
```

```
        InputStream is = ctx.getResourceAsStream("/bookCode.jar");
```

```
        int read = 0;
```

```
        byte[] bytes = new byte[1024];
```

이 코드를 뜯어서 말해보면 "자원 (/bookcode.jar 파일)을 입력 스트림으로 주세요"라는 뜻입니다.

```
        OutputStream os = response.getOutputStream();
```

```
        while ((read = is.read(bytes)) != -1) {
```

```
            os.write(bytes, 0, read);
```

```
        }
```

```
        os.flush();
```

```
        os.close();
```

```
    }
```

여기가 바로 핵심입니다. 핵심치고는 코딩이 너무 단순한가요? 진짜 특별한 것이 없네요. 늘 보던 전형적인 I/O 코딩이죠. 먼저 JAR 바이트를 읽어, 출력 스트림에 바이트를 기록하는 것뿐입니다.

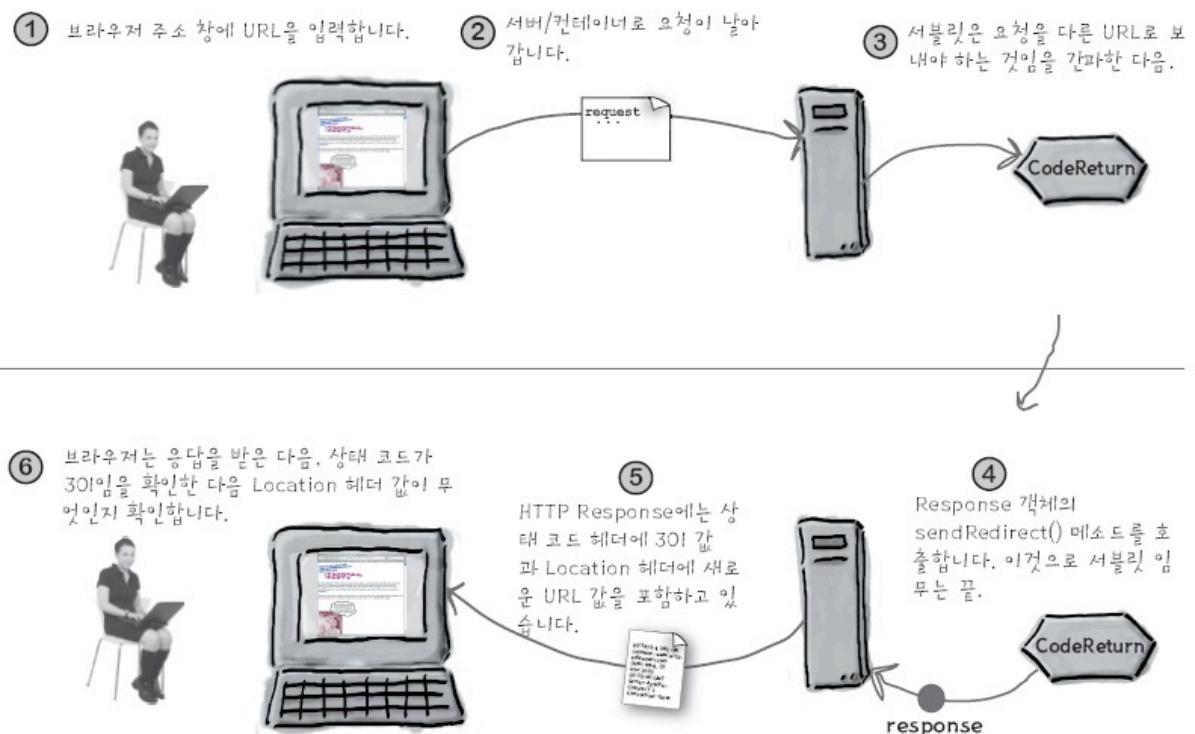
HttpServletResponse의 응답

- 리다이렉트 (Redirect)
 - 브라우저에서 작업
 - Response 객체의 쓰기 작업을 한 후에 sendRedirect() 를 할 수 없음
 - sendRedirect()의 매개변수로 URL(String 객체) 를 사용
 - 브라우저의 URL 이 변화가 됨
- 디스패치 (Dispatch)
 - 서버에서 작업
 - 브라우저의 URL 변화가 없음

11

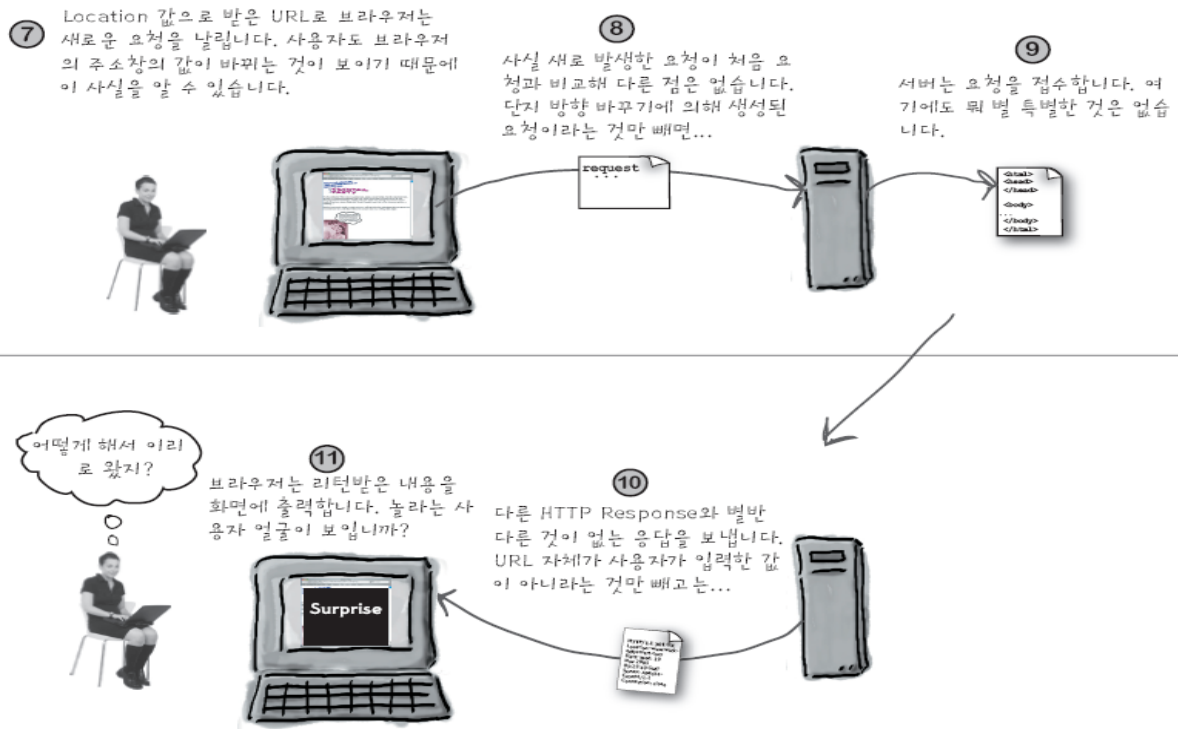
HttpServletResponse의 응답

• Redirect



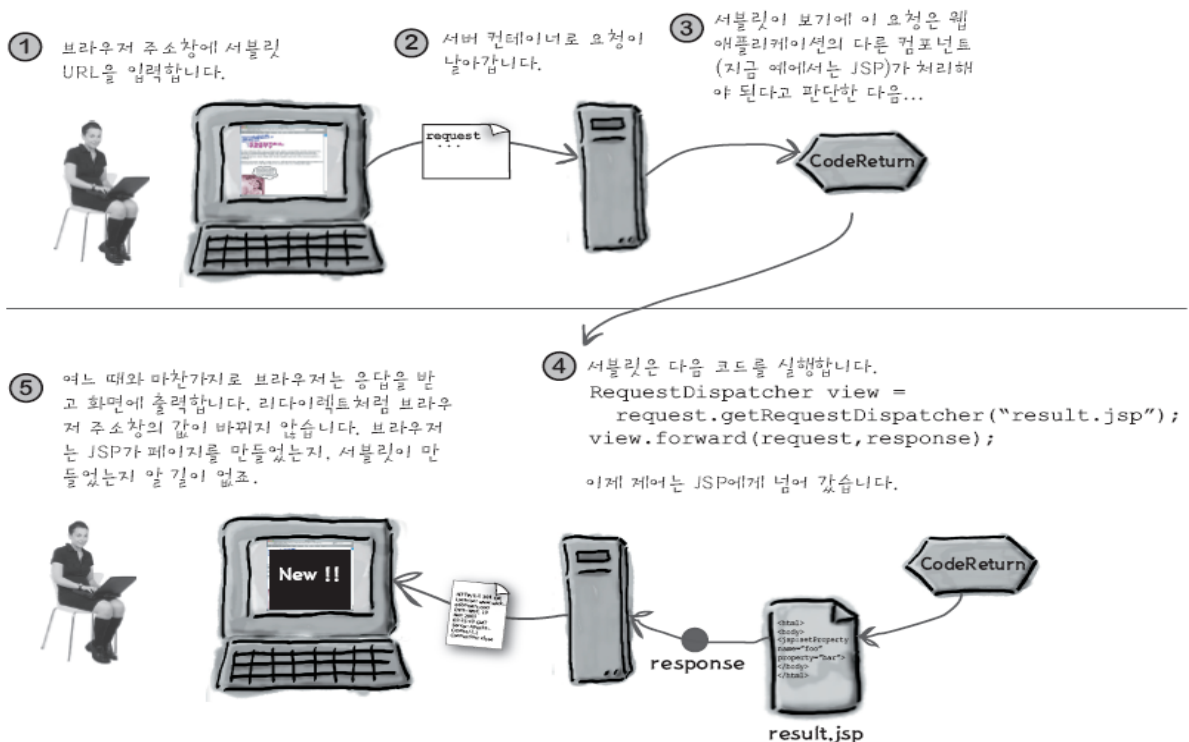
HttpServletResponse의 응답

• Redirect



HttpServletResponse의 응답

• Dispatch



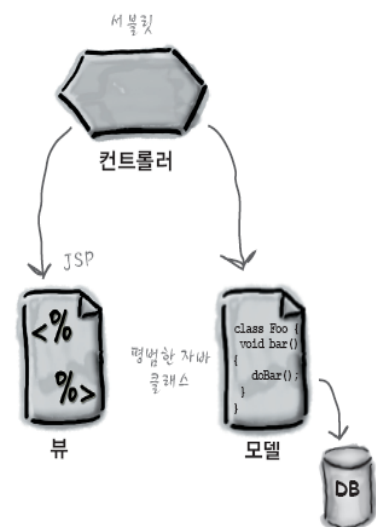
MVC 모델

- 모델 (Model) – 컨트롤러 (Controller) – 뷰 (View) 로 구성
- 비즈니스 로직과 프리젠테이션 로직의 분리
- 비즈니스 로직의 재사용
 - 프리젠테이션 로직의 변경에 영향을 받지 않음
- 서블릿과 비즈니스 로직의 분리가 필요

15

MVC 모델 구성요소

- 구성요소
- Model
 - 비즈니스 로직 위치
 - 모델 정보(state) 읽어오거나(getter) 수정(setter)하는 로직 포함
 - 데이터베이스와 통신하는 곳
- View
 - 프리젠테이션 파트
 - Controller에서 Model 정보 읽음
 - 사용자가 입력한 정보를 Controller에서 넘겨줌
- Controller
 - Request 객체에서 사용자 입력한 정보를 읽어, 모델이 어떤작업을 해야하는지 알아냄
 - 모델의 정보 수정, 뷰에게 넘겨줄 새로운 모델 작성



16

MVC 모델 - J2EE 어플리케이션 서버

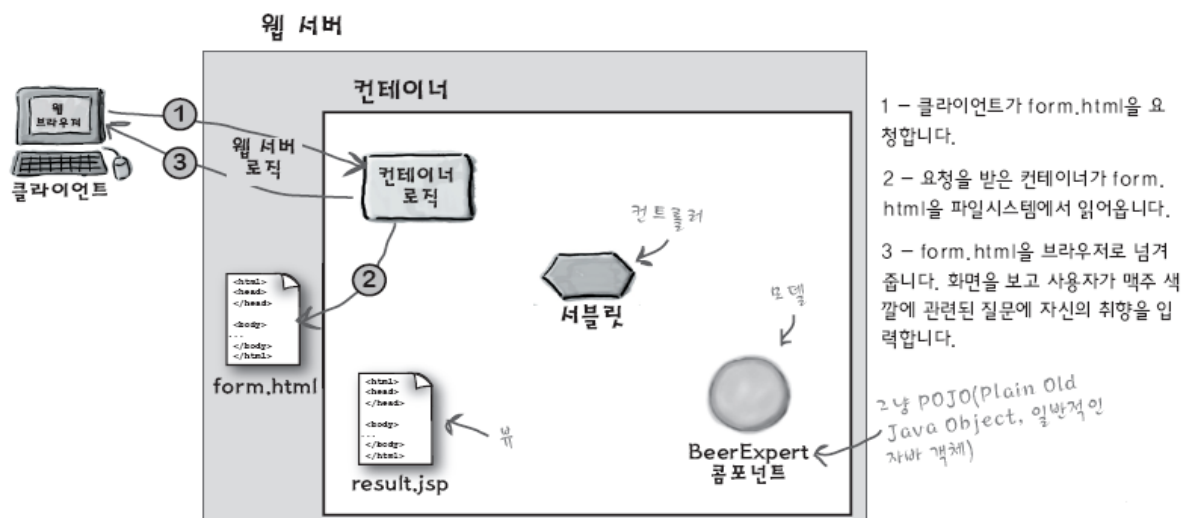
- J2EE 스펙
 - 서블릿, JSP, EJB 스펙들 포함



17

MVC 모델 - 웹 어플리케이션 실행

- 실행순서 - 1
 - HTML(정적 콘텐츠) Request/Response 순서

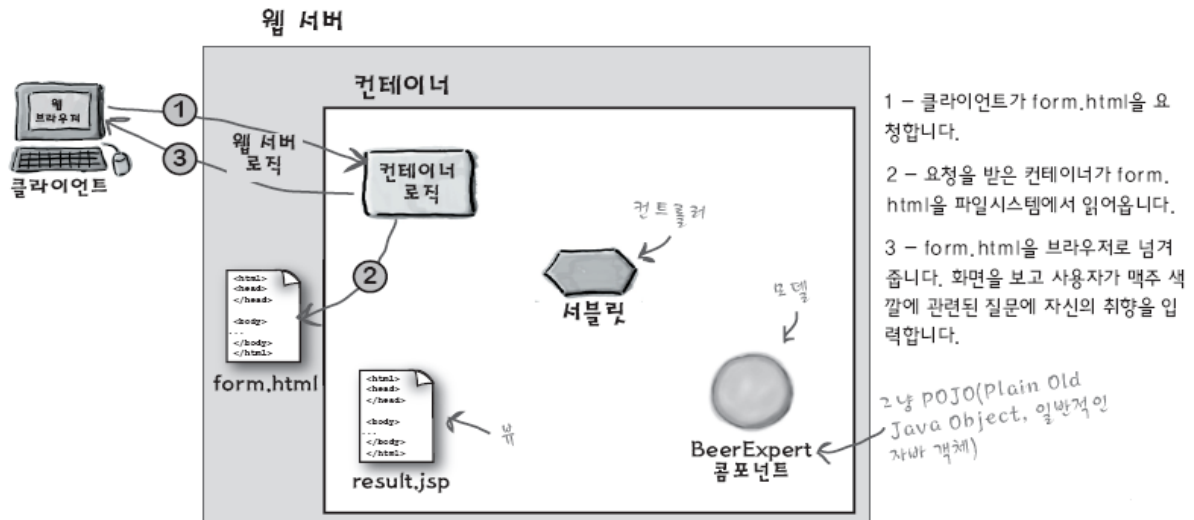


18

MVC 모델 - 웹 어플리케이션 실행

• 실행순서 - 2

- HTML(동적 콘텐츠) Request/Response 순서

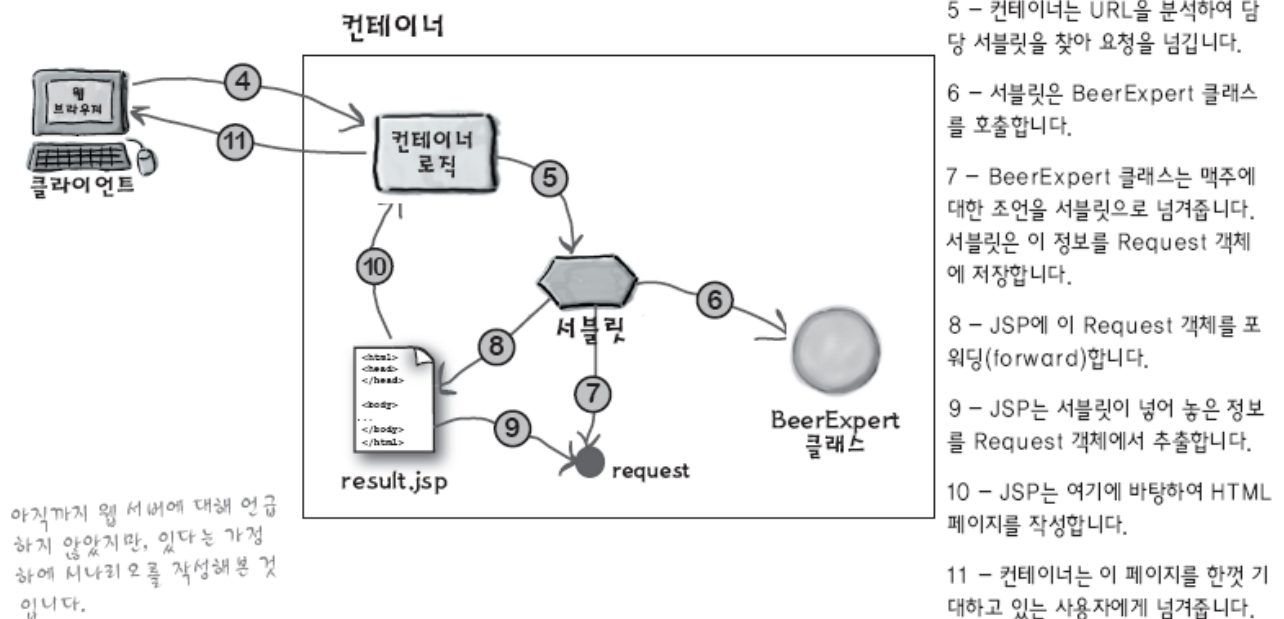


19

MVC 모델 - 웹 어플리케이션 실행

• 실행순서 - 3

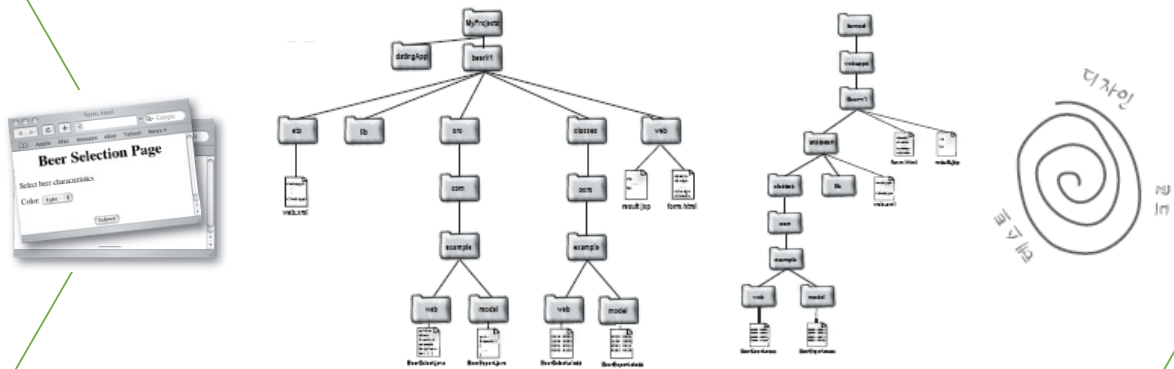
- Servlet Request/Response 순서



MVC 모델 - 웹 어플리케이션 개발

• 개발순서

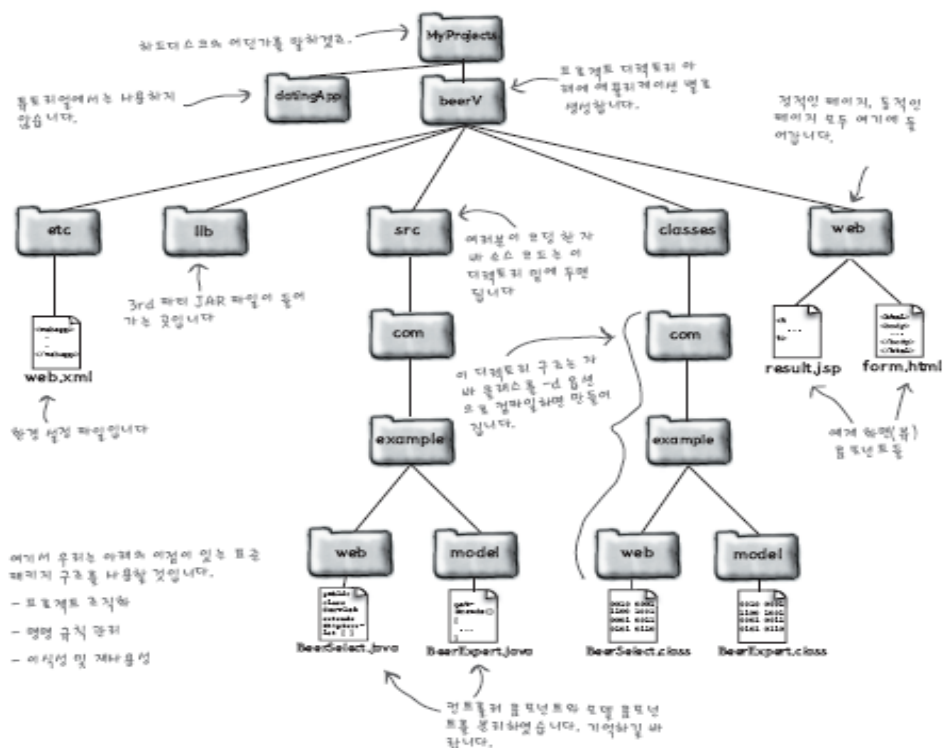
- 사용자 화면 설계
- 개발 환경 구성
- 배포환경구성
- 개발 및 테스트



21

MVC 모델 - 웹 어플리케이션 개발

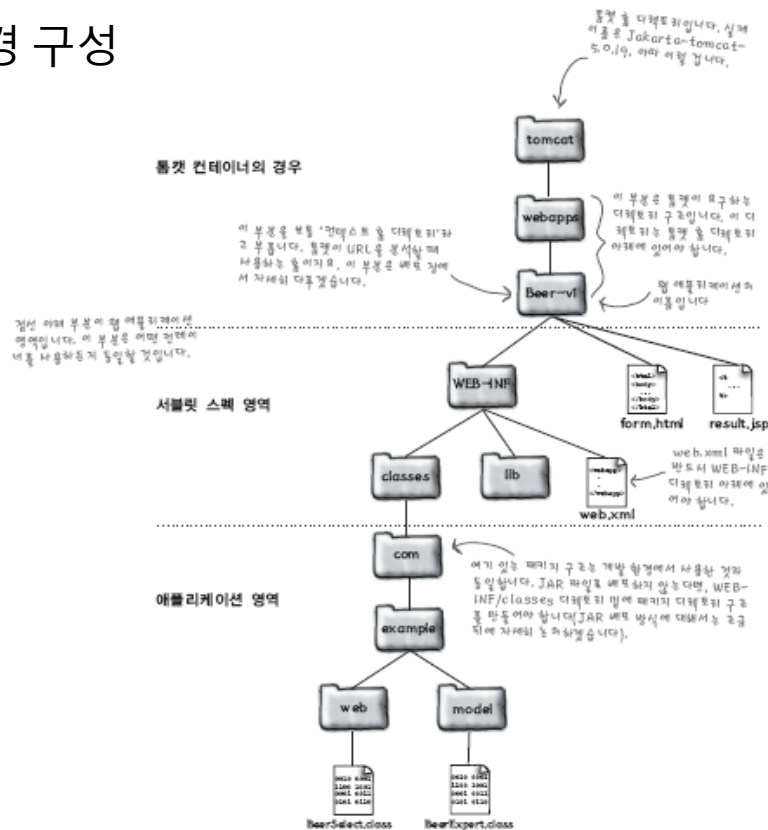
• 개발 환경 구성



22

MVC 모델 - 웹 어플리케이션 개발

• 배포 환경 구성



23

MVC 모델 - 웹 어플리케이션 개발

• 화면 생성(html form)

```
<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>
```

← GET을 안 쓰고 POST를 사용 했을까요?

← 호출할 서블릿이 무엇인지를 지시하는 HTML 코드죠. SelectBeer.do라 밖에 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 모르겠는 않겠군요?

24

MVC 모델 - 웹 어플리케이션 개발

• 서블릿 작성

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);
    }
}
```

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

HttpServlet은 GenericServlet을 상속받
았2 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.

25

MVC 모델 - 웹 어플리케이션 개발

• 서블릿 컴파일, 배포, 테스트

```
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
```

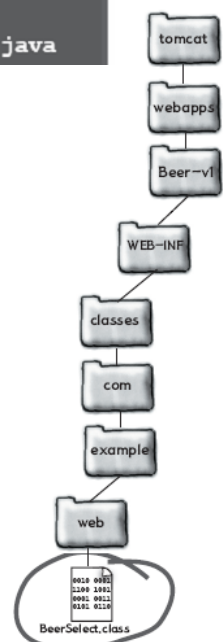
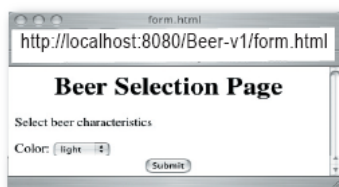
서블릿 배포하기

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
```



MVC 모델 - 웹 어플리케이션 개발

• 개발 및 테스트

• 배포서술자(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.

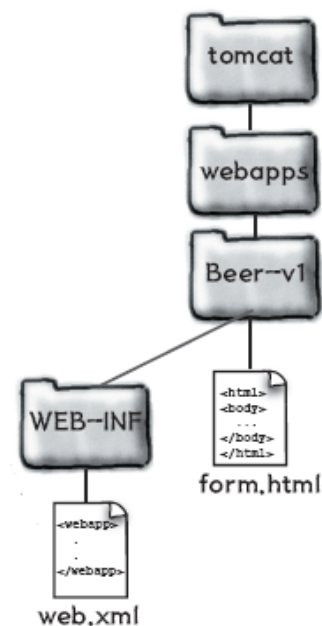
클라이언트가 사용할 이름입니다. .do는 큰 의미가
없습니다.

/를 빼뜨리지 마세요.

MVC 모델 - 웹 어플리케이션 개발

• 개발 및 테스트

- 배포서술자(web.xml) 복사
- 톰캣 운영환경에 복사
- tomcat/webapps/Beer-v1/WEB-INF/



MVC 모델 개발 - 버전 2

• 모델 클래스

- 대부분 일반 자바 클래스로 코딩(POJO)

```
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 긴
보통 조건 표현식으로 구현하였음을 보세요!

설마 이 부분을 그대로 코딩할 생각은 아니겠지요,
여러분의 톱캣 디렉토리에 맞춰 이 부분을 수정하
길 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
```

29

MVC 모델 개발 - 버전 2

• 서블릿

```
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.

30

MVC 모델 개발 - 버전 2

• 서블릿

```
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 만들거나 getBrands() 메소드를 호출합니다.

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

31

MVC 모델 개발 - 버전 2

• 배포 및 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:

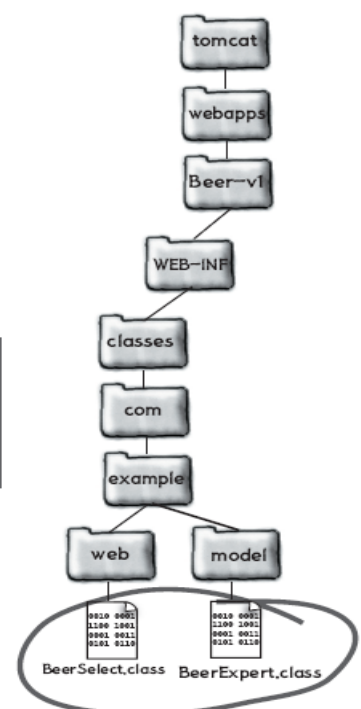
../Beer-v1/WEB-INF/classes/com/example/model/

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

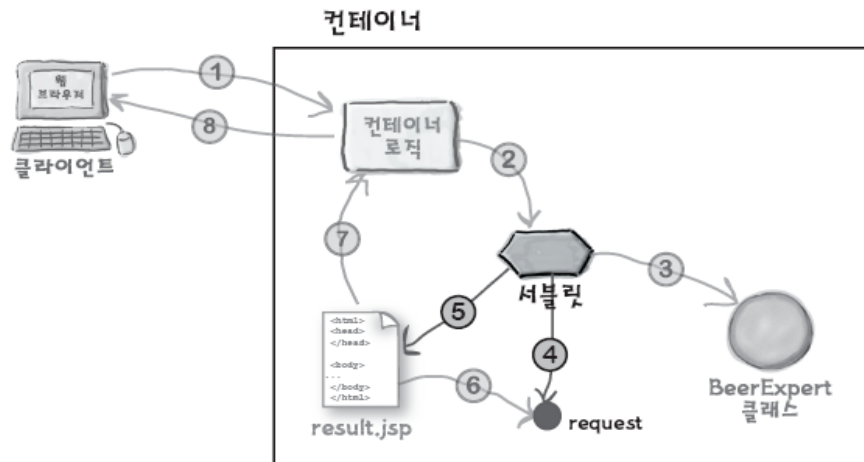
```
File Edit Window Help Shell-High
% cd tomcat
% bin/shutdown.sh
% bin/startup.sh
```



MVC 모델 개발 - 버전 3

• JSP 뷰 화면 이용

- 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.

33

MVC 모델 개발 - 버전 3

• 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 있겠지요?

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 설정하냐, 다시 읽어오는 지까지도 말입니다.

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

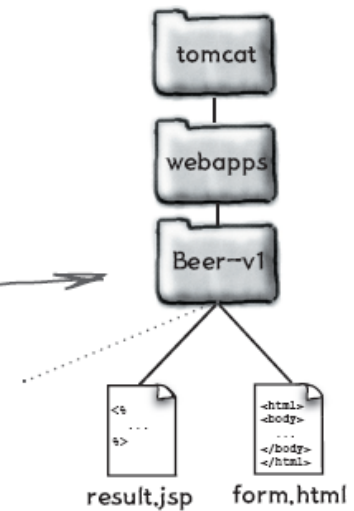
34

MVC 모델 개발 - 버전 2

• JSP 뷰 복사(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/"



35

MVC 모델 개발 - 버전 3

• 서블릿 작성

```
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 말라고요. 알겠습니다. 참고로 하기 위해 주석 처리만 하지요.

36

MVC 모델 개발 - 버전 3

• 서블릿 작성

```
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.

37

MVC 모델 개발 - 버전 3

• 배포 및 테스트

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.



38