

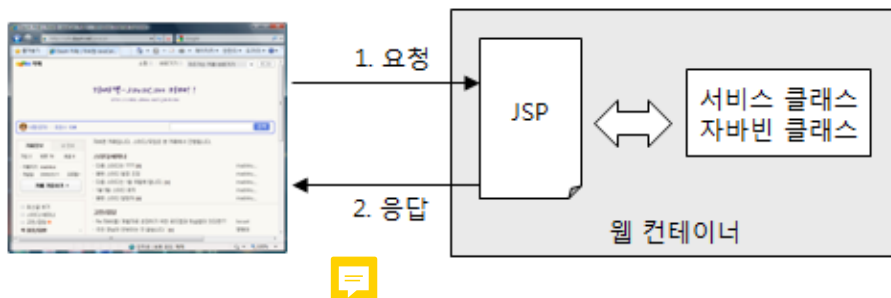


18장 MVC 패턴 구현

18.1 모델 2 구조와 MVC 패턴

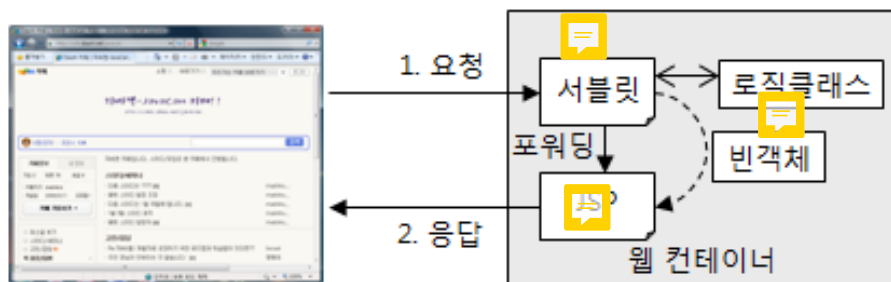
18.1.1 모델 1 구조

- JSP를 이용한 단순한 모델
- JSP에서 요청 처리 및 뷰 생성 처리
 - 구현이 쉬움
 - 요청 처리 및 뷰 생성 코드가 뒤섞여 코드가 복잡함



18.1.2 모델 2 구조

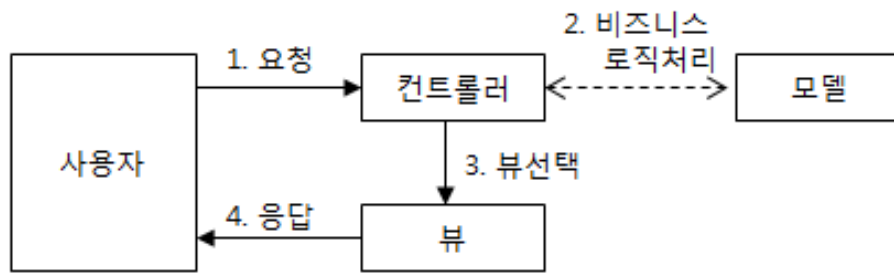
- 서블릿이 요청을 처리하고 JSP가 뷰를 생성
- 모든 요청을 단일 서블릿에서 처리
 - 요청 처리 후 결과를 보여줄 JSP로 이동



18.1.3 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.



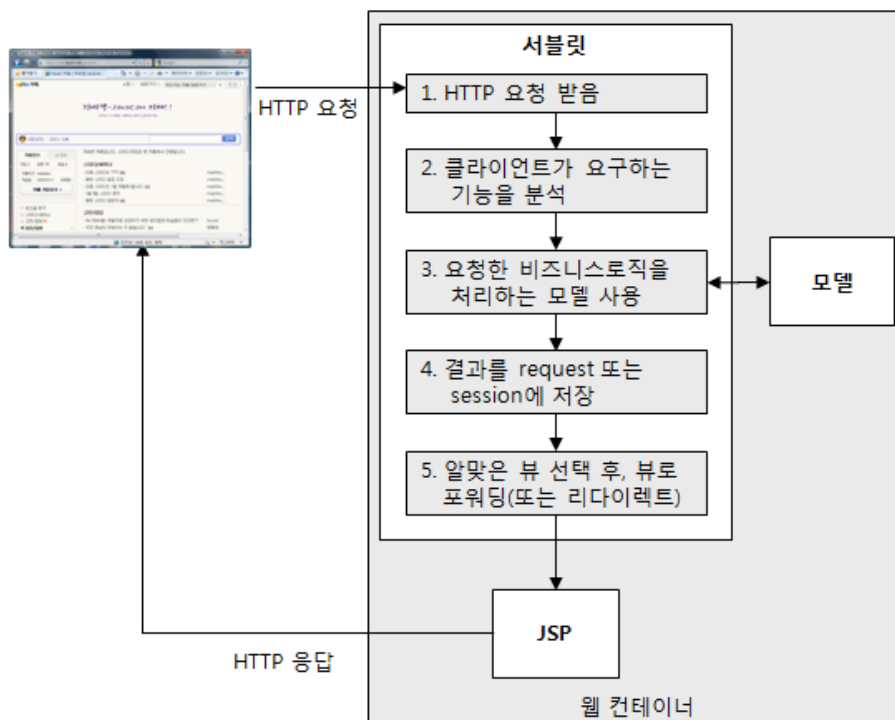


- 특징
 - 로직을 처리하는 모델과 결과 화면을 보여주는 뷰가 분리
 - 흐름 제어나 사용자의 처리 요청은 컨트롤러에 집중

18.1.4 MVC 패턴과 모델 2 구조의 매핑

- 컨트롤러 = 서블릿
- 모델 = 로직 처리 클래스, 자바빈
- 뷰 = JSP

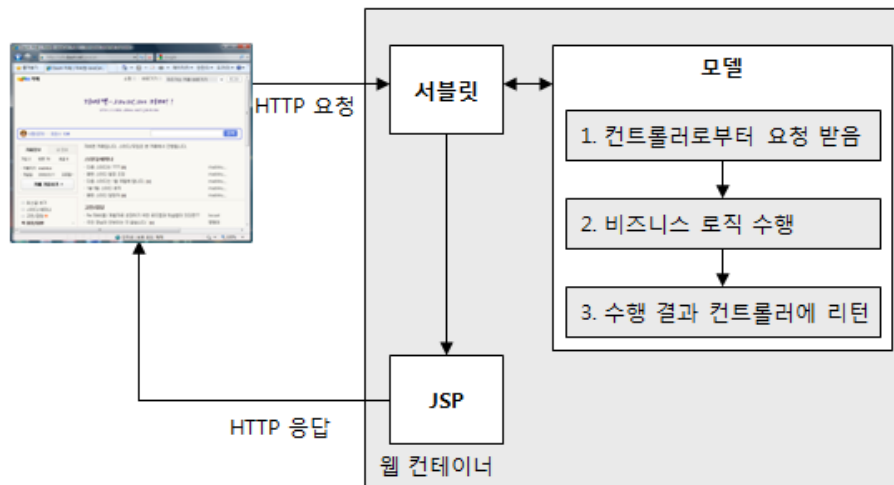
18.1.5 MVC의 컨트롤러 : 서블릿



18.1.6 MVC의 뷰 : JSP

- 모델 2 구조에서 JSP는 뷰 역할을 담당한다.

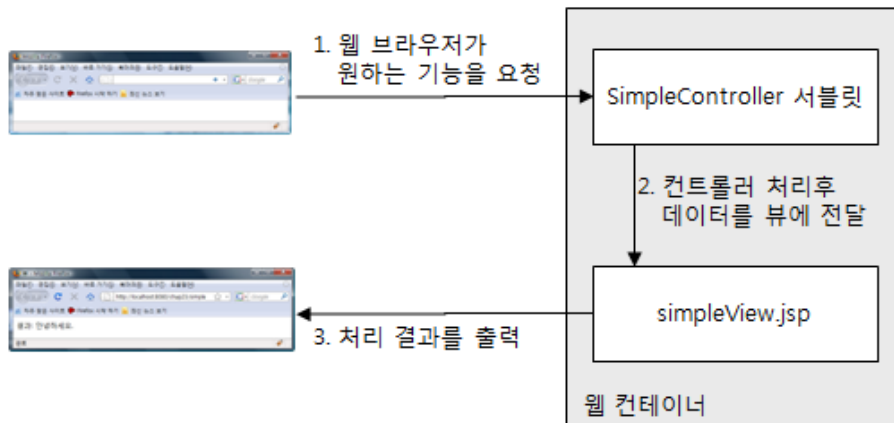
18.1.7 MVC의 모델



18.2 모델 2 구조를 이용한 MVC 패턴 구현

18.2.1 모델 2 구조의 구현 방법 : 기본 MVC 패턴 구현 기법

- 서블릿은 화면에 출력할 메시지를 생성해서 JSP에 전달한다.
- JSP는 서블릿으로부터 전달받은 메시지를 화면에 출력한다.



[chap18/src/mvc/simple/SimpleController.java]

```
01 package mvc.simple;
02
03 import java.io.IOException;
04
05 import javax.servlet.RequestDispatcher;
06 import javax.servlet.ServletException;
07 import javax.servlet.http.HttpServlet;
08 import javax.servlet.http.HttpServletRequest;
09 import javax.servlet.http.HttpServletResponse;
10
```

```

11 public class SimpleController extends HttpServlet {
12     // 1단계, HTTP 요청 받음
13     public void doGet(HttpServletRequest request,
14                       HttpServletResponse response)
15         throws ServletException, IOException {
16         processRequest(request, response);
17     }
18
19     public void doPost(HttpServletRequest request,
20                       HttpServletResponse response)
21         throws ServletException, IOException {
22         processRequest(request, response);
23     }
24
25     private void processRequest(HttpServletRequest request,
26                                HttpServletResponse response)
27         throws IOException, ServletException {
28         // 2단계, 요청 파악
29         // request 객체로부터 사용자의 요청을 파악하는 코드
30         String type = request.getParameter("type");
31
32         // 3단계, 요청한 기능을 수행한다.
33         // 사용자에게 요청에 따라 알맞은 코드
34         Object resultObject = null;
35         if (type == null || type.equals("greeting")) {
36             resultObject = "안녕하세요.";
37         } else if (type.equals("date")) {
38             resultObject = new java.util.Date();
39         } else {
40             resultObject = "Invalid Type";
41         }
42
43         // 4단계, request나 session에 처리 결과를 저장
44         request.setAttribute("result", resultObject);
45
46         // 5단계, RequestDispatcher를 사용하여 알맞은 뷰로 포워딩
47         RequestDispatcher dispatcher =
48             request.getRequestDispatcher("/simpleView.jsp");
49         dispatcher.forward(request, response);
50     }
51 }

```

[chap18/WebContent/simpleView.jsp]

```

01 <%@ page contentType="text/html; charset=utf-8" %>
02
03 <html>
04 <head><title>뷰</title></head>
05 <body>
06
07 결과: ${result}
08
09 </body>
10 </html>

```

[chap18/WebContent/simpleView.jsp]

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
03          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04          xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
05                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
06          version="3.1">

```

```

07
08     <servlet>
09         <servlet-name>SimpleController</servlet-name>
10         <servlet-class>mvc.simple.SimpleController</servlet-class>
11     </servlet>
12
13     <servlet-mapping>
14         <servlet-name>SimpleController</servlet-name>
15         <url-pattern>/simple</url-pattern>
16     </servlet-mapping>
17
18 </web-app>

```

18.2.2 커맨드 패턴 기반의 코드

- 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?...>
- 커맨드 패턴 : 클라이언트의 각 요청을 처리하는 별도 클래스를 제공하는 구현 패턴
 - 하나의 명령어를 하나의 클래스가 처리한다.
 - 명령어를 처리하는 클래스들은 동일한 인터페이스를 상속해서 구현한다.

```

// 1. 특정 이름의 파라미터에 명령어 정보를 전달하는 방식
String command = request.getParameter("cmd");
String viewPage = null;
if (command == null) {
    // 명령어 오류 처리
    viewPage = "/error/invalidCommand.jsp";
} else if (command.equals("BoardList")) {
    // 글 목록 읽기 요청 처리
    ...
    viewPage = "/board/list.jsp";
} else if (command.equals("BoardWriteForm")) {
    // 글쓰기 입력 폼 요청 처리
    ...
    viewPage = "/board/writeForm.jsp";
}
RequestDispatcher dispatcher = request.getRequestDispatcher(viewPage);
dispatcher.forward(request, response);

// 2. 커맨드 패턴을 이용한 명령어 처리기의 분리 방식
String command = request.getParameter("cmd");
CommandHandler handler = null;
if (command == null) {
    handler = new NullHandler();
} else if (command.equals("BoardList")) {
    handler = new BoardListHandler();
} else if (command.equals("BoardWriteForm")) {
    handler = new BoardWriteFormHandler();
}
String viewPage = handler.process(request, response);
RequestDispatcher dispatcher = request.getRequestDispatcher(viewPage);

```

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

18.2.3 설정 파일에 커맨드와 클래스의 관계 명시하기

- <커맨드, 요청 처리 클래스> 매핑 정보를 코드가 아닌 별도 설정 파일에 저장
 - 새로운 커맨드 추가나 클래스 변경 시 편리

// 로직 처리 코드를 컨트롤러 서블릿에서 핸들러 클래스로 옮겼지만, 여전히 컨트롤러 서블릿은 명령어에 따른 알맞은 처리를 하기 위해 중첩된 if-else 구문을 사용해야 한다. 이 코드는 새로운 명령어가 추가되면 컨트롤러 서블릿 클래스의 코드를 직접 변경해야 하는 단점이 있다.

```
String command = request.getParameter("cmd");
CommandHandler handler = null;
```

```
if (command == null) {
    handler = new NullHandler();
} else if (command.equals("BoardList")) {
    handler = new BoardListHandler();
} else if (command.equals("BoardWriteForm")) {
    handler = new BoardWriteFormHandler();
}
```

// 단점을 해결하는 방법은 <명령어, 핸들러 클래스>의 매핑 정보를 설정 파일에 저장하는 것이다.

```
BoardList=mvjsp.command.BoardListHandler
BoardWriteForm=mvjsp.command.BoardWriteFormHandler
...
```

(1) 컨트롤러 서블릿 (ControllerUsingFile.java)

- 컨트롤러 서블릿은 설정 파일에서 명령어와 핸들러 클래스의 매핑 정보를 읽어와 명령어에 해당하는 핸들러 클래스 객체를 미리 생성해두었다가 process() 메서드에서 사용하면 된다.

[chap18/src/mvc/controller/ControllerUsingFile.java]

```
01 package mvc.controller;
02
03 import java.io.FileReader;
04 import java.io.IOException;
05 import java.util.HashMap;
06 import java.util.Iterator;
07 import java.util.Map;
08 import java.util.Properties;
09
10 import javax.servlet.RequestDispatcher;
11 import javax.servlet.ServletException;
12 import javax.servlet.http.HttpServlet;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
15
16 import mvc.command.CommandHandler;
17 import mvc.command.NullHandler;
18
19 public class ControllerUsingFile extends HttpServlet {
20
21     // <커맨드, 핸들러인스턴스> 매핑 정보 저장
22     private Map<String, CommandHandler> commandHandlerMap =
23         new HashMap<>();
```

```

24
25     public void init() throws ServletException {
26         String configFile = getInitParameter("configFile");
27         Properties prop = new Properties();
28         String configFileFullPath = getServletContext().getRealPath(configFile);
29         try (FileReader fis = new FileReader(configFileFullPath)) {
30             prop.load(fis);
31         } catch (IOException e) {
32             throw new ServletException(e);
33         }
34         Iterator keyIter = prop.keySet().iterator();
35         while (keyIter.hasNext()) {
36             String command = (String) keyIter.next();
37             String handlerClassName = prop.getProperty(command);
38             try {
39                 Class<?> handlerClass = Class.forName(handlerClassName);
40                 CommandHandler handlerInstance =
41                     (CommandHandler) handlerClass.newInstance();
42                 commandHandlerMap.put(command, handlerInstance);
43             } catch (ClassNotFoundException | InstantiationException
44                     | IllegalAccessException e) {
45                 throw new ServletException(e);
46             }
47         }
48     }
49
50     public void doGet(HttpServletRequest request, HttpServletResponse response)
51         throws ServletException, IOException {
52         process(request, response);
53     }
54
55     protected void doPost(HttpServletRequest request,
56         HttpServletResponse response) throws ServletException, IOException {
57         process(request, response);
58     }
59
60     private void process(HttpServletRequest request,
61         HttpServletResponse response) throws ServletException, IOException {
62         String command = request.getParameter("cmd");
63         CommandHandler handler = commandHandlerMap.get(command);
64         if (handler == null) {
65             handler = new NullHandler();
66         }
67         String viewPage = null;
68         try {
69             viewPage = handler.process(request, response);
70         } catch (Throwable e) {
71             throw new ServletException(e);
72         }
73         if (viewPage != null) {
74             RequestDispatcher dispatcher = request.getRequestDispatcher(viewPage);
75             dispatcher.forward(request, response);
76         }
77     }
78 }

```

(2) web.xml

- configFile 초기화 파라미터를 설정 파일 경로로 사용하므로, web.xml 파일에 <init-param> 태그를 통해서 설정 파일 경로를 지정한다.

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
03         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
05                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
06         version="3.1">
07
08     <servlet>
09         <servlet-name>ControllerUsingURI</servlet-name>
10         <servlet-class>mvc.controller.ControllerUsingURI</servlet-class>
11         <init-param>
12             <param-name>configFile</param-name>
13             <param-value>/WEB-INF/commandHandlerURI.properties</param-value>
14         </init-param>
15         <load-on-startup>1</load-on-startup>
16     </servlet>
17
18     <servlet-mapping>
19         <servlet-name>ControllerUsingURI</servlet-name>
20         <url-pattern>*.do</url-pattern>
21     </servlet-mapping>
22
23     <servlet>
24         <servlet-name>ControllerUsingFile</servlet-name>
25         <servlet-class>mvc.controller.ControllerUsingFile</servlet-class>
26         <init-param>
27             <param-name>configFile</param-name>
28             <param-value>/WEB-INF/commandHandler.properties</param-value>
29         </init-param>
30         <load-on-startup>1</load-on-startup>
31     </servlet>
32
33     <servlet-mapping>
34         <servlet-name>ControllerUsingFile</servlet-name>
35         <url-pattern>/controllerUsingFile</url-pattern>
36     </servlet-mapping>
37
38     <servlet>
39         <servlet-name>SimpleController</servlet-name>
40         <servlet-class>mvc.simple.SimpleController</servlet-class>
41     </servlet>
42
43     <servlet-mapping>
44         <servlet-name>SimpleController</servlet-name>
45         <url-pattern>/simple</url-pattern>
46     </servlet-mapping>
47
48 </web-app>

```

(3) 설정 파일 (commandHandler.properties)

- 명령어 매핑 정보는 commandHandler.properties 파일에 작성한다.

[chap18/WebContent/WEB-INF/commandHandler.properties]

```


01 hello=mvc.hello.HelloHandler
02 #someCommand=any.SomeHandler

```


(4) 요청 URL

- <http://localhost:8080/chap18/controllerUsingFile?cmd=hello>

18.2.4 요청 URI를 명령어로 사용하기

- 앞서 살펴본 예제는 cmd 파라미터를 명령어로 사용했다. 명령어 기반의 파라미터는 한가지 단점이 있는데 그것이 바로 컨트롤러의 URL이 사용자에게 노출된다는 점이다.
 - <http://localhost:8080/chap18/controllerUsingFile?cmd=hello>
- 이런 단점을 보완하는 것이 URL의 일부를 명령어로 사용하는 것이다. 
 - <http://localhost:8080/chap18/hello.do>

(1) 컨트롤러 서블릿 (ControllerUsingURI.java)

- 요청 URI를 명령어로 사용한다.

[chap18/src/mvc/controller/ControllerUsingURI.java]

```
01 package mvc.controller;
02
03 import java.io.FileReader;
04 import java.io.IOException;
05 import java.util.HashMap;
06 import java.util.Iterator;
07 import java.util.Map;
08 import java.util.Properties;
09
10 import javax.servlet.RequestDispatcher;
11 import javax.servlet.ServletException;
12 import javax.servlet.http.HttpServlet;
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
15
16 import mvc.command.CommandHandler;
17 import mvc.command.NullHandler;
18
19 public class ControllerUsingURI extends HttpServlet {
20
21     // <커맨드, 핸들러인스턴스> 매핑 정보 저장
22     private Map<String, CommandHandler> commandHandlerMap =
23         new HashMap<>();
24
25     public void init() throws ServletException {
26         String configFile = getInitParameter("configFile");
27         Properties prop = new Properties();
28         String configFilePath = getServletContext().getRealPath(configFile);
29         try (FileReader fis = new FileReader(configFilePath)) {
30             prop.load(fis);
31         } catch (IOException e) {
32             throw new ServletException(e);
33         }
34         Iterator keyIter = prop.keySet().iterator();
35         while (keyIter.hasNext()) {
36             String command = (String) keyIter.next();
37             String handlerClassName = prop.getProperty(command);
38             try {
39                 Class<?> handlerClass = Class.forName(handlerClassName);
```

```

40         CommandHandler handlerInstance =
41             (CommandHandler) handlerClass.newInstance();
42         commandHandlerMap.put(command, handlerInstance);
43     } catch (ClassNotFoundException | InstantiationException
44             | IllegalAccessException e) {
45         throw new ServletException(e);
46     }
47 }
48 }
49
50 public void doGet(HttpServletRequest request, HttpServletResponse response)
51     throws ServletException, IOException {
52     process(request, response);
53 }
54
55 protected void doPost(HttpServletRequest request,
56     HttpServletResponse response) throws ServletException, IOException {
57     process(request, response);
58 }
59
60 private void process(HttpServletRequest request,
61     HttpServletResponse response) throws ServletException, IOException {
62     String command = request.getRequestURI();
63     if (command.indexOf(request.getContextPath()) == 0) {
64         command = command.substring(request.getContextPath().length());
65     }
66     CommandHandler handler = commandHandlerMap.get(command);
67     if (handler == null) {
68         handler = new NullHandler();
69     }
70     String viewPage = null;
71     try {
72         viewPage = handler.process(request, response);
73     } catch (Throwable e) {
74         throw new ServletException(e);
75     }
76     if (viewPage != null) {
77         RequestDispatcher dispatcher = request.getRequestDispatcher(viewPage);
78         dispatcher.forward(request, response);
79     }
80 }
81 }

```

(2) web.xml

- configFile 초기화 파라미터를 설정 파일 경로로 사용하므로, web.xml 파일에 <init-param> 태그를 통해서 설정 파일 경로를 지정한다.

[chap18/WebContent/WEB-INF/web.xml]

```

01 <?xml version="1.0" encoding="UTF-8"?>
02 <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
03     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
04     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
05         http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
06     version="3.1">
07
08     <servlet>
09         <servlet-name>ControllerUsingURI</servlet-name>
10         <servlet-class>mvc.controller.ControllerUsingURI</servlet-class>
11         <init-param>
12             <param-name>configFile</param-name>

```

```

13         <param-value>/WEB-INF/commandHandlerURI.properties</param-value>
14     </init-param>
15     <load-on-startup>1</load-on-startup>
16 </servlet>
17
18 <servlet-mapping>
19     <servlet-name>ControllerUsingURI</servlet-name>
20     <url-pattern>*.do</url-pattern>
21 </servlet-mapping>
22
23 <servlet>
24     <servlet-name>ControllerUsingFile</servlet-name>
25     <servlet-class>mvc.controller.ControllerUsingFile</servlet-class>
26     <init-param>
27         <param-name>configFile</param-name>
28         <param-value>/WEB-INF/commandHandler.properties</param-value>
29     </init-param>
30     <load-on-startup>1</load-on-startup>
31 </servlet>
32
33 <servlet-mapping>
34     <servlet-name>ControllerUsingFile</servlet-name>
35     <url-pattern>/controllerUsingFile</url-pattern>
36 </servlet-mapping>
37
38 <servlet>
39     <servlet-name>SimpleController</servlet-name>
40     <servlet-class>mvc.simple.SimpleController</servlet-class>
41 </servlet>
42
43 <servlet-mapping>
44     <servlet-name>SimpleController</servlet-name>
45     <url-pattern>/simple</url-pattern>
46 </servlet-mapping>
47
48 </web-app>

```

(3) 설정 파일 (commandHandlerURI.properties)

[chap18/WebContent/WEB-INF/commandHandlerURI.properties]

```
01 /hello.do=mvc.hello.HelloHandler
```

(4) 요청 URL

- <http://localhost:8080/chap18/hello.do>

18.3 모델 1 구조와 모델 2 구조의 선택

모델	장점	단점
모델 1	<ul style="list-style-type: none"> -배우기 쉬움 -자바 언어를 몰라도 구현 가능 -기능과 JSP의 직관적인 연결. 하나의 JSP가 하나의 기능과 연결 	<ul style="list-style-type: none"> -로직 코드와 뷰 코드가 혼합되어 JSP 코드가 복잡해 짐 -뷰 변경 시 논리코드의 빈번한 복사 즉, 유지보수 작업이 불편함
모델 2	<ul style="list-style-type: none"> -로직 코드와 뷰 코드의 분리에 따른 유지 보수의 편리함 	<ul style="list-style-type: none"> -자바 언어에 친숙하지 않으면 접근하기가 쉽지 않음 -작업량이 많음(커맨드클래스+뷰JSP)

	<ul style="list-style-type: none"> -컨트롤러 서블릿에서 집중적인 작업 처리 가능.(권한/인증 등) -확장의 용이함 	
--	---	--