

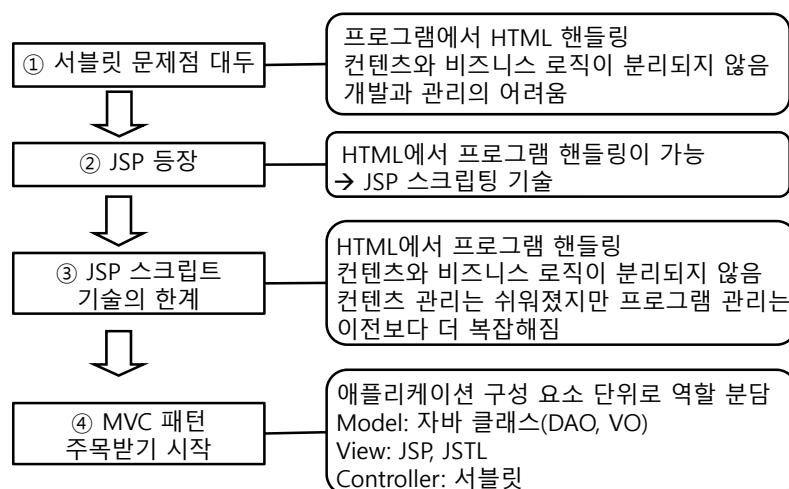
# 웹 MVC

524730-1  
2020년 봄학기  
5/13/2020  
박경신

## Servlet 이해하기

- ▣ Servlet
  - 자바 플랫폼에서 컴포넌트 기반의 웹애플리케이션 개발기술
- ▣ JSP는 서블릿 기술에 기반함
  - Servlet의 프리젠테이션 문제를 해결하기 위해 JSP가 등장
  - 이로 인해 웹 애플리케이션의 유지보수 어려움 심각.
- ▣ JSP 모델2가 주목받으며 다시 서블릿에 대한 중요성 부각

## Servlet 변천

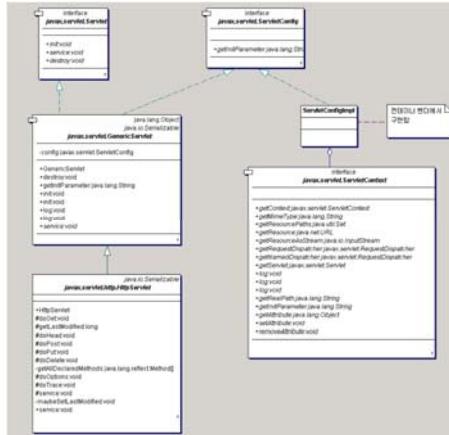


## Servlet 이점

- ▣ 웹 애플리케이션 개발 시 Servlet 이점
  - 컨텐츠와 비즈니스 로직을 분리할 수 있음
  - 컨트롤러와 뷰의 역할 분담 : 웹 디자이너와 개발자 간의 원활한 공동작업이 가능해짐
  - 유지보수가 수월함
  - 기능의 확장이 용이함
  - 개발자가 HTML, 자바스크립트 스타일시트 등 복잡한 기술을 모두 알아야 할 필요가 없음

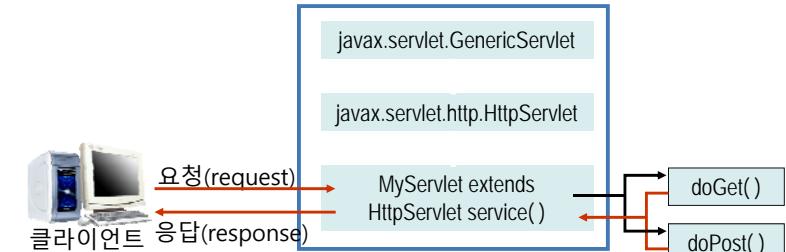
## Servlet API

- Servlet은 자바 클래스로 제작됨
  - javax.servlet.GenericServlet과 javax.servlet.http.HttpServlet
  - API 구조



## Servlet 구조

- ▣ HttpServlet 구조
    - 서블릿은 javax.servlet.http.HttpServlet을 상속
    - service() 메소드는 컨테이너에서 호출
    - doGet(), doPost() 메소드를 **오버라이드**해서 처리에 필요한 기능을 구현



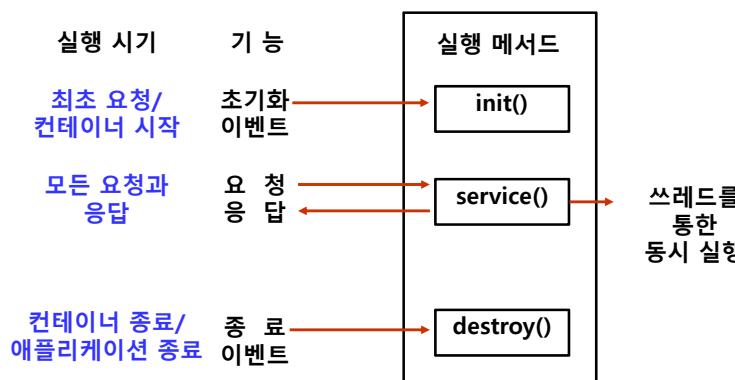
## Servlet 구조

- ▣ GET 방식
    - 서버에 있는 정보를 가져오기 위해 설계됨
    - 240바이트까지 전달할 수 있음
    - QUERY\_STRING 환경변수를 통해 전달
    - 형식 : <http://xxx.xxx.co.kr/servlet/login?id=hj&name=hong>
    - URL노출로 보안성이 요구되는 경우엔 사용할 수 없음
    - 검색엔진에서 검색단어 전송에 많이 이용함
  - ▣ POST 방식
    - 서버로 정보를 올리기 위해 설계됨
    - 데이터크기의 제한은 없음
    - URL에 parameter가 표시되지 않음

## Servlet 구조

- ▣ 서블릿 로딩
    - 최초 클라이언트 요청 시 init() 메소드가 호출되며 메모리에 적재됨
  - ▣ 요청 처리
    - service() 메소드가 컨테이너에 의해 호출해서 클라이언트의 요청을 처리함
  - ▣ 처리 수행
    - service() 메소드는 특정 HTTP 요청(GET, POST, 등)을 처리하는 doGet(), doPost() 메소드를 호출하여 처리함
  - ▣ 서블릿 종료
    - destroy() 메소드를 호출하여 서블릿을 제거함

## Servlet 생명주기



## Servlet 이해하기

### □ web.xml에 다음 내용 추가

```
<servlet-mapping>
  <servlet-name>ApplicationController</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

## HttpServletRequest 클래스

### □ HttpServletRequest 클래스

- HttpServletRequest 클래스의 doGet(), doPost() 메소드 호출 시 파라미터로 전달됨.
- 사용자 **요청**과 관련된 정보를 제공.
- HTML 폼 입력 값을 가져옴
- 쿠키, 세션정보에 접근할 수 있음
- 클라이언트 IP주소를 알 수 있음

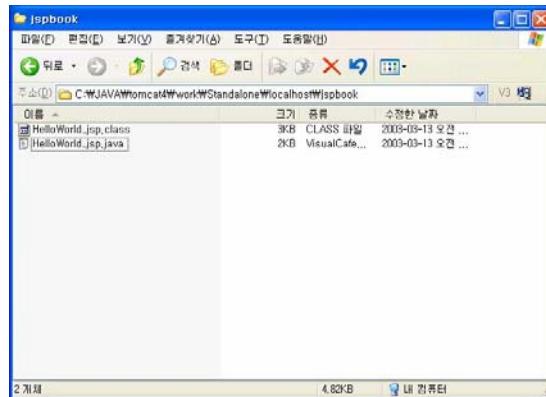
## HttpServletResponse 클래스

### □ HttpServletResponse 클래스

- HttpServlet 클래스의 doGet(), doPost() 메소드 호출 시 파라미터로 전달
- 사용자 **응답**을 처리하기 위한 클래스
- MIME Type 설정
- HTTP 헤더 정보 설정
- 페이지 전환

## Servlet 코드로 생성된 JSP

### □ 서블릿 코드로 생성된 JSP



## Servlet 코드로 생성된 JSP

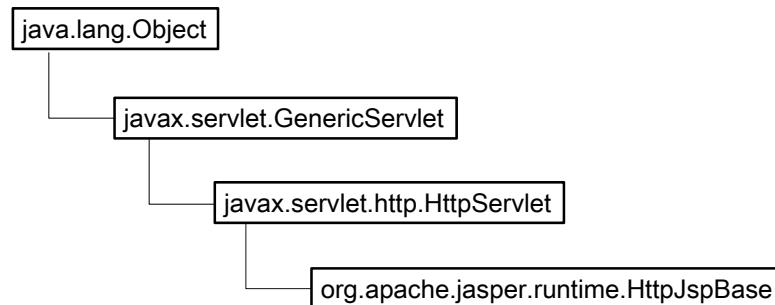
### □ 서블릿 코드로 생성된 JSP

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;

public class HelloWorld_jsp extends HttpJspBase {
```

## Servlet 코드로 생성된 JSP

### □ 서블릿 코드로 생성된 JSP



## Servlet 코드로 생성된 JSP

### □ JSP 파일의 모든 내용은 \_jspService() 메서드에 위치

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {
    ....
    ....
    out.write("\r\n\r\n");
    out.write("<HTML>\r\n");
    out.write("<HEAD>");
    out.write("<TITLE>Hello World");
    out.write("</TITLE>");
    out.write("</HEAD>\r\n\r\n");
    out.write("<BODY>");
    out.write("<H2>Hello World : 헬로월드");
    out.write("</H2>\r\n오늘의 날짜와 시간은 : ");
    out.print( new java.util.Date() );
    ...
    ... 이하종략
}
```

## JSP 개발 방법

### □ 디자인 패턴

디자인 패턴은 기존 환경에서 반복적으로 일어나는 문제를 설명하고, 그 문제 해법의 핵심을 설명하는 것이다. 이렇게 하면 같은 방법을 두 번 반복하지 않고, 이 해법을 백만 번 이상 재사용할 수 있다. - 크리스토퍼알렉산더

- 객체 지향 소프트웨어 개발에 필요한 기법을 정리
- 소프트웨어 유지보수에 유리
- 개발 표준화와 유지보수의 효율성

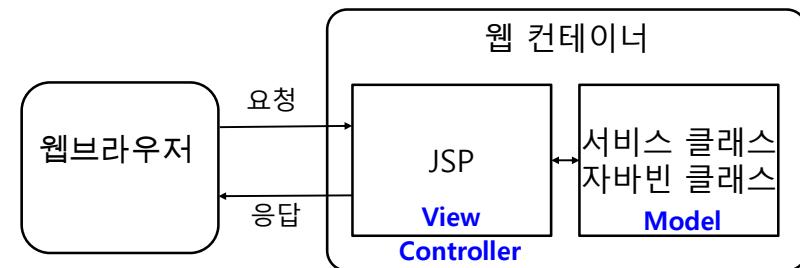
## 모델 1

### □ 모델1 소스코드 예시

```
<%
String user_passwd = request.getParameter("passwd");
if(user_passwd.equals("123456")) {
%>
<H2> 관리자 메뉴 </H2>
<HR>
1. <a href=>사용자 추가</a><p>
2. <a href=>비밀번호 변경</a><p>
3. <a href=>리포트 보기</a><p>
<%
}
else {
    out.println("<H2>비밀번호가 잘못되었습니다.</H2>");
    out.println("<a href=check_form.jsp>돌아가기</a>");
}
%>
```

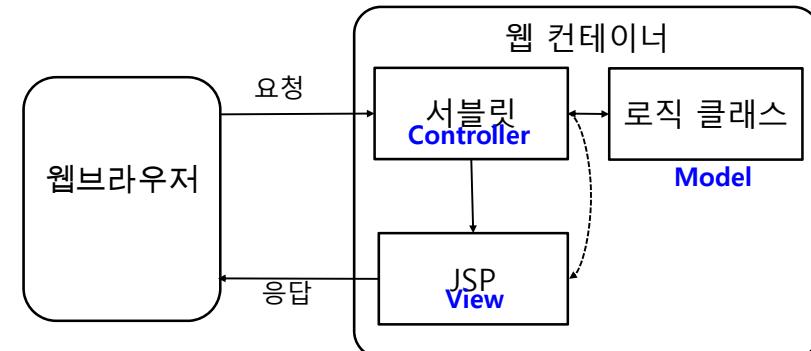
## 모델 1

- 모델1은 초기 JSP개발에 사용된 모델
- 일반적으로 사용되는 JSP개발 방법
- 유지보수의 어려움으로 인해 모델2가 권장됨



## 모델 2

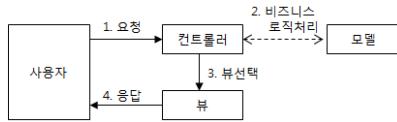
- 서블릿이 요청을 처리하고 JSP가 뷰를 생성
- 모든 요청을 단일 서블릿에서 처리
  - 요청 처리 후 결과를 보여줄 JSP로 이동



## MVC 패턴

### ▣ Model-View-Controller

- 모델 - 비즈니스 영역의 상태 정보를 처리
- 뷰 - 비즈니스 영역에 대한 프레젠테이션 뷰(즉, 사용자가 보게 될 결과 화면)를 담당
- 컨트롤러 - 사용자의 입력 및 흐름 제어를 담당



### ▣ 특징

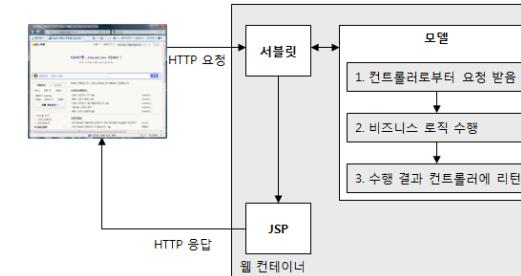
- 로직을 처리하는 모델과 결과 화면을 보여주는 뷔가 분리됨
- 흐름 제어나 사용자의 처리 요청은 컨트롤러에 집중

### ▣ 모델 2 구조와 매팅: 컨트롤러-서블릿, 뷔-JSP

## MVC 모델

### ▣ 모델 (Model)

- 데이터 처리부분 담당.
- Value Object(VO), Data Access Object(DAO)으로 분리 하기도 함.
- 데이터베이스와 연동
- 빈즈, 자바 클래스로 구현



## MVC 뷔

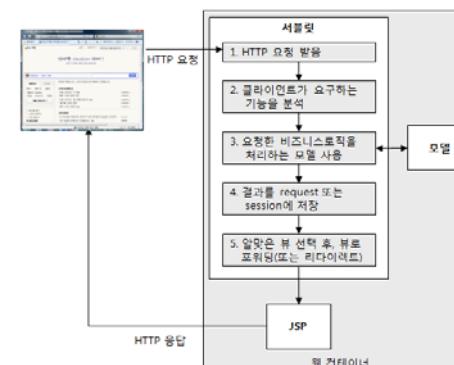
### ▣ 뷔 (View)

- 프리젠테이션 영역 즉 사용자에게 보여지는 화면을 담당.
- 스크립트릿을 통한 로직제어(for, if, while)에 따라 화면을 구성하면 안됨.
- HTML, JSP로 구현
- 커스텀 태그 라이브러리 사용 권장(JSTL 등)

## MVC 컨트롤러

### ▣ 컨트롤러 (Controller)

- 뷔와 뷔를 연결하고 데이터처리를 위해 모델 영역과 연동
- 비즈니스 로직이 복잡하면 BO(Business Object) 클래스를 두어 처리
- 서블릿을 권장하며 간단한 작업에는 JSP로도 가능



## 모델 2

### 모델2 소스코드 예시

- 서블릿 컨트롤러에서는 일반적으로 forward 를 이용

```
RequestDispatcher view = request.getRequestDispatcher("admin_menu.jsp");
view.forward(request, response);
```

```
<%
String user_passwd = request.getParameter("passwd");
if(user_passwd.equals("123456")) {
    response.sendRedirect("admin_menu.jsp");
}
else {
    response.sendRedirect("error.jsp");
}
%>
```

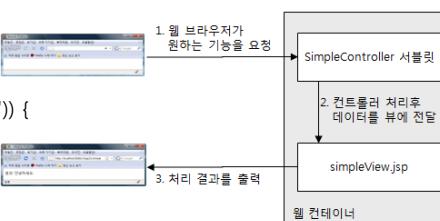
## MVC 패턴 컨트롤러 기본 구현

```
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(HttpServletRequest request,
                               HttpServletResponse response)
        throws IOException, ServletException {
        // 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);
    }
}
```

## 컨트롤러 구현 예

```
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);
}
```

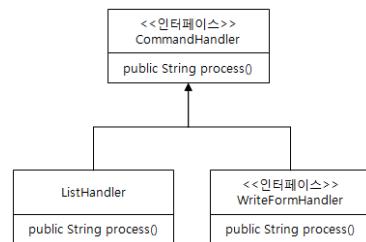


## 클라이언트의 요청 명령을 구분하는 방식

- 컨트롤러가 알맞은 로직을 수행하려면 클라이언트가 어떤 기능을 요청하는지 구분할 수 있어야 함
- 요청 기능 구분 방식
  - 특정 이름의 파라미터에 명령어 정보를 전달
    - http://host/board/controller/ControllerServlet?cmd=BoardList&...
  - 요청 URI를 명령어로 사용
    - http://host/board/boardList?...

## 커맨드 패턴을 이용한 요청 처리

- 커맨드 패턴 - 클라이언트의 각 요청을 처리하는 별도 클래스를 제공하는 구현 패턴
  - 하나의 명령어를 하나의 클래스가 처리
- 클래스 구성
  - 명령어를 처리하는 클래스들은 동일한 인터페이스를 구현



## 설정 파일 사용 컨트롤러 구현 - 설정 파일로딩

```
public class ControllerUsingFile extends HttpServlet {
    // <커맨드, 핸들러인스턴스> 매핑 정보 저장
    private Map commandHandlerMap = new java.util.HashMap();
    public void init(ServletConfig config) throws ServletException {
        String configfile = config.getInitParameter("configFile");
        Properties prop = new Properties();
        FileInputStream fis = null;
        try {
            String configFilePath = config.getServletContext().getRealPath(configfile);
            fis = new FileInputStream(configFilePath);
            prop.load(fis);
        } catch (IOException e) {
            throw new ServletException(e);
        } finally {
            if (fis != null) try { fis.close(); } catch (IOException ex) { }
        }
        Iterator keyIter = prop.keySet().iterator();
        while (keyIter.hasNext()) {
            String command = (String) keyIter.next();
            String handlerClassName = prop.getProperty(command);
            try {
                Class handlerClass = Class.forName(handlerClassName);
                Object handlerInstance = handlerClass.newInstance();
                commandHandlerMap.put(command, handlerInstance);
            } catch (Exception e) {
                throw new ServletException(e);
            }
        }
    }
}
```

## 커맨드와 처리 클래스의 매핑 정보를 담은 설정 파일

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

```
BoardList = action.BoardListHandler
BoardWriteForm = action.BoardWriteFormHandler
```

## 설정 파일 사용 컨트롤러 구현 - 요청 처리

```
private void processRequest(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    String command = request.getParameter("cmd");
    CommandHandler handler = (CommandHandler)commandHandlerMap.get(command);
    if (handler == null) {
        handler = new NullHandler();
    }
    String viewPage = null;
    try {
        viewPage = handler.process(request, response);
    } catch (Throwable e) {
        throw new ServletException(e);
    }
    RequestDispatcher dispatcher = request.getRequestDispatcher(viewPage);
    dispatcher.forward(request, response);
}
```

## 설정 파일 사용 컨트롤러 구현 - web.xml

```
<servlet>
  <servlet-name>ControllerUsingFile</servlet-name>
  <servlet-class>mvc.controller.ControllerUsingFile</servlet-class>
  <init-param>
    <param-name>configFile</param-name>
    <param-value>/WEB-INF/commandHandler.properties</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>ControllerUsingFile</servlet-name>
  <url-pattern>/controllerUsingFile</url-pattern>
</servlet-mapping>
```

<http://localhost:8080/board/controllerUsingFile?cmd=Hello>

## 요청 URI 사용시

```
private void processRequest(HttpServletRequest request,
    HttpServletResponse response) throws ServletException, IOException {
    String command = request.getRequestURI();
    if (command.indexOf(request.getContextPath()) == 0) {
        command = command.substring(request.getContextPath().length());
    }
    CommandHandler handler
        = (CommandHandler) commandHandlerMap.get(command);
    ...
}
```

## 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 -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)