Chapter

12

서블릿(Servlet)

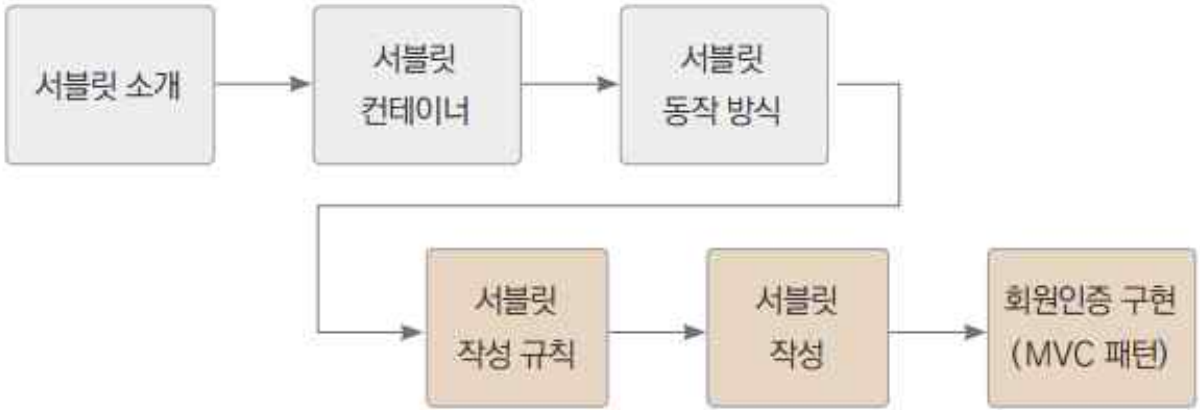
학습 목표 및 순서

▣ 학습 목표

. MVC 패턴을 적용한 모델2 방식의 게시판을 제작하기 위해 필요한 기술인 서블릿을 학습합니다.

. 서블릿의 개념과 동작 방식을 이해하고, 작성 규칙에 따라 URL 요청명과 서블릿 클래스를 매핑한 후 클라이언트의 요청을 처리해 볼 것입니다.

▣ 학습 순서



12.1 서블릿이란?

▣ **서블릿이란..??**

- 서블릿(Servlet)은 JSP가 나오기 전, 자바로 웹 애플리케이션을 개발할 수 있도록 만든 기술
- 서블릿은 서버 단에서 클라이언트의 요청을 받아 처리한 후 응답하는 역할을 함
- 서블릿의 특징
  - 클라이언트의 요청에 대해 동적으로 작동하는 웹 애플리케이션 컴포넌트
  - MVC 모델에서 컨트롤러Controller 역할
  - 모든 메서드는 스레드로 동작됩니다.
  - jakarta.servlet.http 패키지의 HttpServlet 클래스를 상속

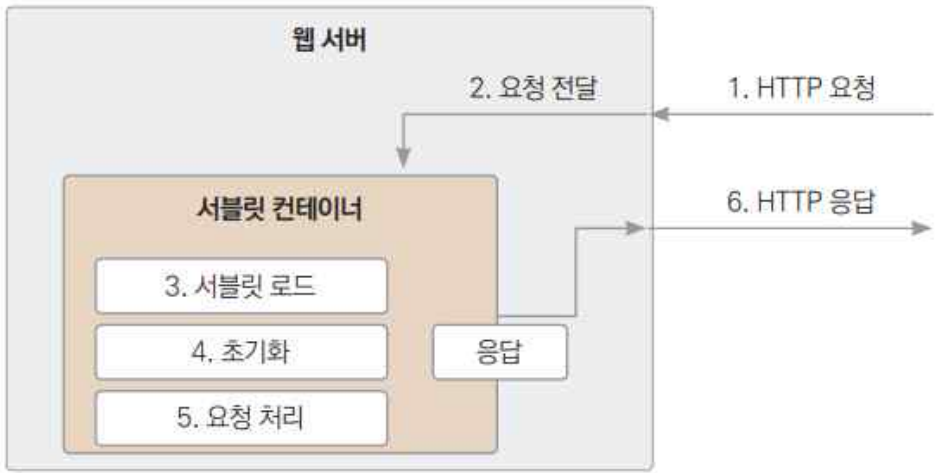
12.2 서블릿 컨테이너

▣ 서블릿 컨테이너

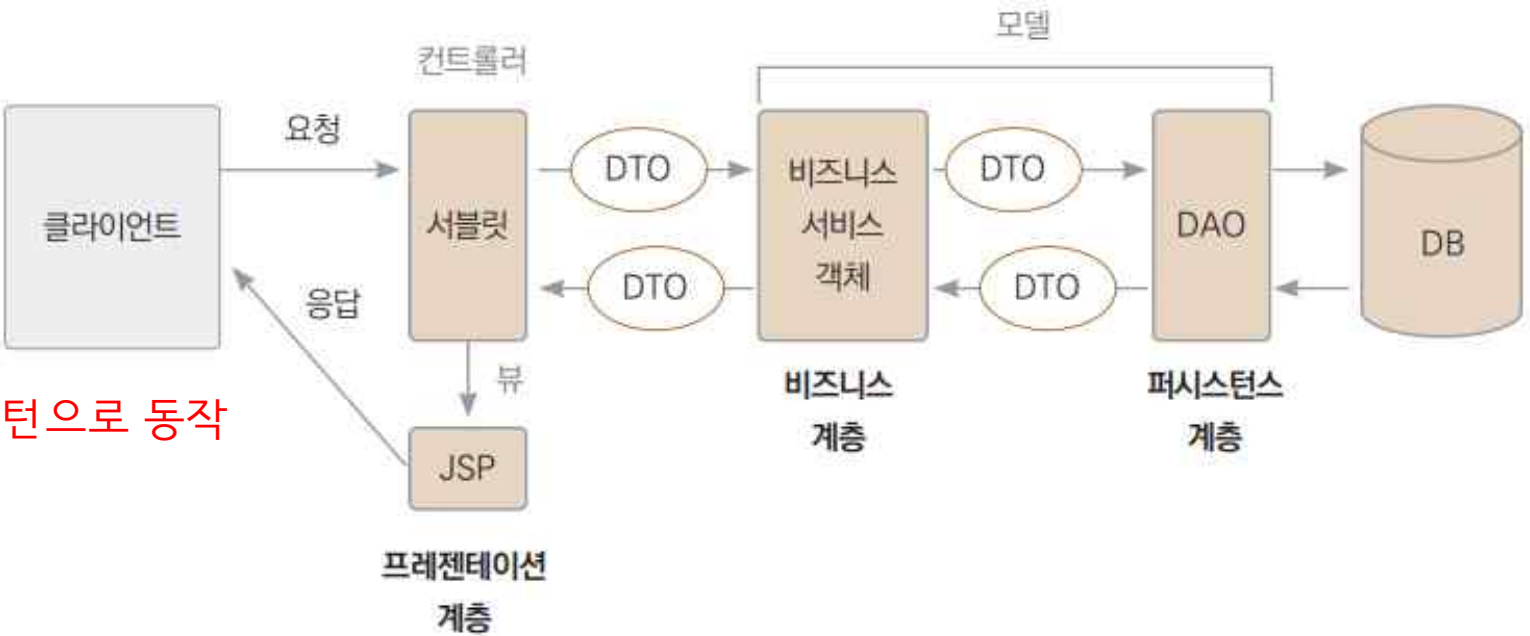
- 서블릿을 관리하는 컨테이너로 톰캣(Tomcat)을 사용
- 서블릿의 수명주기를 관리하고, 요청이 오면 스레드를 생성해 처리해줌

▣ 서블릿 컨테이너의 역할

- 통신 지원 : 특정 포트port로 소켓Socket을 열고 I/O 스트림을 생성하는 등 복잡한 과정을 간단히 처리
- 수명주기 관리 : 서블릿의 초기화, 요청처리, 가비지 컬렉션을 통해 객체를 소멸
- 멀티스레딩 관리 : 멀티스레드 방식으로 여러 요청을 동시에 처리
- 선언적인 보안 관리 및 JSP 지원 등



12.3 서블릿의 동작 방식



MVC 패턴으로 동작

❶ 클라이언트의 요청을 분석

❷ 요청을 처리할 서블릿(Controller)으로 전달

❸ 비즈니스 서비스 로직 호출

❹ 모델(Model)로부터 그 결괏값 받음

❺ 결괏값을 출력할 적절한 뷰(View) 선택

❻ 출력하여 클라이언트에 응답

12.4 서블릿 작성 규칙

1. 기본적으로 jakarta.servlet, jakarta.servlet.http, java.io 패키지를 임포트
2. 서블릿 클래스는 반드시 public으로 선언해야 하고, HttpServlet을 상속
3. 사용자의 요청을 처리하기 위해 doGet() 혹은 doPost() 를 반드시 오버라이딩
4. 해당 메서드는 ServletException과 IOException 예외를 throws
5. 또한 메서드를 호출할 때의 매개변수는 HttpServletRequest와 HttpServletResponse를 사용

규칙에 따라 작성한 서블릿 클래스

```
package 패키지명;

import java.io.IOException;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class 서블릿클래스명 extends HttpServlet {  // HttpServlet 상속
    @Override  // doGet() 오버라이딩
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        // 메서드의 실행부
    }
}
```

기본적으로 필요한 패키지(클래스)

12.5 서블릿 작성

▣ 서블릿 매핑

. web.xml에 기술하는 방법
. @WebServlet 애너테이션을 사용하여 코드에 직접 명시하는 방법

▣ web.xml에서 매핑


```
<servlet>  <!-- 서블릿 등록 -->
    <servlet-name>서블릿명</servlet-name>
    <servlet-class>패키지를 포함한 서블릿 클래스명</servlet-class>
</servlet>
<servlet-mapping>  <!-- 서블릿과 요청명(요청 URL) 매핑 -->
    <servlet-name>서블릿명</servlet-name>
    <url-pattern>클라이언트 요청 URL</url-pattern>
</servlet-mapping>
```

12.5 서블릿 작성

▣ web.xml에서 매핑(계속)

HelloServlet.jsp

```
<body>
    <h2>web.xml에서 매핑 후 JSP에서 출력하기</h2>
    <p>
        <strong><%= request.getAttribute("message") %></strong> ❶
        <br />
        <a href="./HelloServlet.do">바로가기</a> ❷
    </p>
</body>
```

❶ request영역의 속성 출력
❷ 요청을 위한 바로가기 링크

❶❸ 서블릿명
❷ 요청을 처리할 서블릿 클래스
❹ 컨텍스트 루트를 제외한 슬러쉬로 시작하는 요청명 URL

즉 ❹로 들어온 요청을 ❷에서 처리하는 형태로 매핑한다.

web.xml

```
<servlet>  <!-- 서블릿 등록 -->
    <servlet-name>HelloServlet</servlet-name> ❶
    <servlet-class>servlet.HelloServlet</servlet-class> ❷
</servlet>
<servlet-mapping>  <!-- 서블릿과 요청명 매핑 -->
    <servlet-name>HelloServlet</servlet-name> ❸
    <url-pattern>/12Servlet/HelloServlet.do</url-pattern> ❹
</servlet-mapping>
```

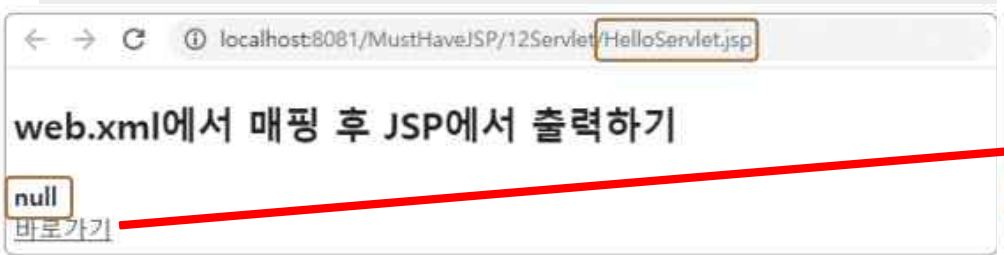
12.5 서블릿 작성

▣ web.xml에서 매핑(계속)

```
public class HelloServlet extends HttpServlet {  ❶
    private static final long serialVersionUID = 1L; ❷
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {  ❸
        req.setAttribute("message", "Hello Servlet..!!");  ❹
        req.getRequestDispatcher("/12Servlet/HelloServlet.jsp")
           .forward(req, resp);  ❺
    }
}
```

HelloServlet.java

❶ HttpServlet 클래스를 상속
❷ 직렬화된 클래스의 버전 관리에 사용하는 식별자
❸ doGet() 메서드 오버라이딩
❹ request 영역에 속성 저장
❺ View역할의 JSP로 포워드



12.5 서블릿 작성

▣ @WebServlet 애너테이션으로 매핑

AnnoMapping.jsp

```
    <p>
        <strong>${ message }</strong> ❶
        <br />
        <a href="<%= request.getContextPath() %>/12Servlet/AnnoMapping.do">바로
가기</a> ❷
    </p>
```

AnnoMapping.java

```
@WebServlet("/12Servlet/AnnoMapping.do") ❶
public class AnnoMapping extends HttpServlet { ❷
    private static final long serialVersionUID = 1L;
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) ❸
            throws ServletException, IOException {
        req.setAttribute("message", "@WebServlet으로 매핑"); ❹
        req.getRequestDispatcher("/12Servlet/AnnoMapping.jsp")
           .forward(req, resp); ❺
    }
}
```

❶ @WebServlet 애너테이션을 이용해 매핑
❷ HttpServlet을 상속하여 서블릿 정의
❸ doGet() 메서드 오버라이딩
❹ request 영역에 속성 저장
❺ View역할의 JSP로 포워드

12.5 서블릿 작성

▣ JSP 없이 서블릿에서 바로 응답 출력

DirectServletPrint.jsp

```
<h2>web.xml에서 매핑 후 Servlet에서 직접 출력하기</h2>
<form method="post" action="../12Servlet/DirectServletPrint.do">  ❶
    <input type="submit" value="바로가기" />
</form>
```

❶ post 방식의 전송을 위한 <form> 태그. action 속성에 요청명 지정.

web.xml

```
<servlet>
  <servlet-name>DirectServletPrint</servlet-name>
  <servlet-class>servlet.DirectServletPrint</servlet-class>  ❶
</servlet>
<servlet-mapping>
  <servlet-name>DirectServletPrint</servlet-name>
  <url-pattern>/12Servlet/DirectServletPrint.do</url-pattern>  ❷
</servlet-mapping>
```

❶ 요청을 처리할 서블릿 클래스

❷ 요청명

12.5 서블릿 작성

▣ JSP 없이 서블릿에서 바로 응답 출력(계속)

DirectServletPrint.java

```
public class DirectServletPrint extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)  ❶
            throws ServletException, IOException {
        resp.setContentType("text/html;charset=UTF-8");  ❷
        PrintWriter writer = resp.getWriter();  ❸
        writer.println("<html>");
        writer.println("<head><title>DirectServletPrint</title></head>");
        writer.println("<body>");
        writer.println("<h2>서블릿에서 직접 출력합니다.</h2>");
        writer.println("<p>jsp로 포워드하지 않습니다.</p>");
        writer.println("</body>");
        writer.println("</html>");
        writer.close();  ❺
    }
}
```

❶ post 방식의 요청처를 위한 doPost() 메서드 오버라이딩

❷ 콘텐츠 타입 지정

❸ 응답결과 출력을 위해 PrintWriter 객체 생성

❹ println() 메서드로 응답 내용 출력

❺ 사용한 PrintWriter 객체 닫기

12.5 서블릿 작성

▣ JSP 없이 서블릿에서 바로 응답 출력(계속)

- JSP없이 서블릿에서 직접 출력하면 소스가 굉장히 복잡하고 지저분해지는 단점이 있음
- 대부분의 경우는 JSP를 통해 출력하는게 좋음
- 하지만 외부 서버와 통신을 위해 JSON 혹은 XML 데이터가 필요한 경우 직접 출력이 편리함

12.5 서블릿 작성

▣ 한 번의 매핑으로 여러 가지 요청 처리

- 요청명이 추가되면 이에 따른 매핑도 함께 추가되야 하므로 번거로움
- FrontController 패턴으로 한번의 매핑으로 여러개의 요청을 처리할 수 있음

```
<h3>한 번의 매핑으로 여러 가지 요청 처리하기</h3>
${ resultValue } ❶
<ol>
    <li>URI : ${ uri }</li> ❷
    <li>요청명 : ${ commandStr }</li> ❸
</ol>
<ul>
    <li><a href="../12Servlet/regist.one">회원가입</a></li>
    <li><a href="../12Servlet/login.one">로그인</a></li>
    <li><a href="../12Servlet/freeboard.one">자유게시판</a></li>
</ul>
```

❹

FrontController.jsp

❶ 서블릿에서 request 영역에 저장할 결괏값
❷ 요청명의 전체 경로
❸ 전체 경로에서 마지막의 xxx.one 부분을 추출한 문자열

❹ 각 페이지 바로가기 링크

12.5 서블릿 작성

▣ 한 번의 매핑으로 여러 가지 요청 처리(계속)

- 요청명이 추가되면 이에 따른 매핑도 함께 추가되야 하므로 번거로움
- FrontController 패턴으로 한번의 매핑으로 여러개의 요청을 처리할 수 있음

```
@WebServlet("*.one")  ❶
public class FrontController extends HttpServlet {
    private static final long serialVersionUID = 1L;
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        String uri = req.getRequestURI();  ❷
        int lastSlash = uri.lastIndexOf("/");  ❸
        String commandStr = uri.substring(lastSlash);  ❹
```

FrontController.java

❶ 와일드카드(*)로 .one으로 끝나는 모든 요청명에 대한 매핑

❷~❹ 현재 경로명을 통해 마지막 문자열을 잘라냄

12.5 서블릿 작성

▣ 한 번의 매핑으로 여러 가지 요청 처리(계속)

```
        if (commandStr.equals("/regist.one"))
            registFunc(req);
        else if (commandStr.equals("/login.one"))
            loginFunc(req);
        else if (commandStr.equals("/freeboard.one"))
            freeboardFunc(req);
        req.setAttribute("uri", uri);
        req.setAttribute("commandStr", commandStr);
        req.getRequestDispatcher("/12Servlet/FrontController.jsp")
           .forward(req, resp);
    }
    // 페이지별 처리 메서드
    void registFunc(HttpServletRequest req) {
        req.setAttribute("resultValue", "<h4>회원가입</h4>");
    }
```

FrontController.java(계속)

❺ 잘라낸 마지막 문자열을 통해 요청 분석

❻ request 영역에 속성 저장

❼ JSP 페이지 포워드

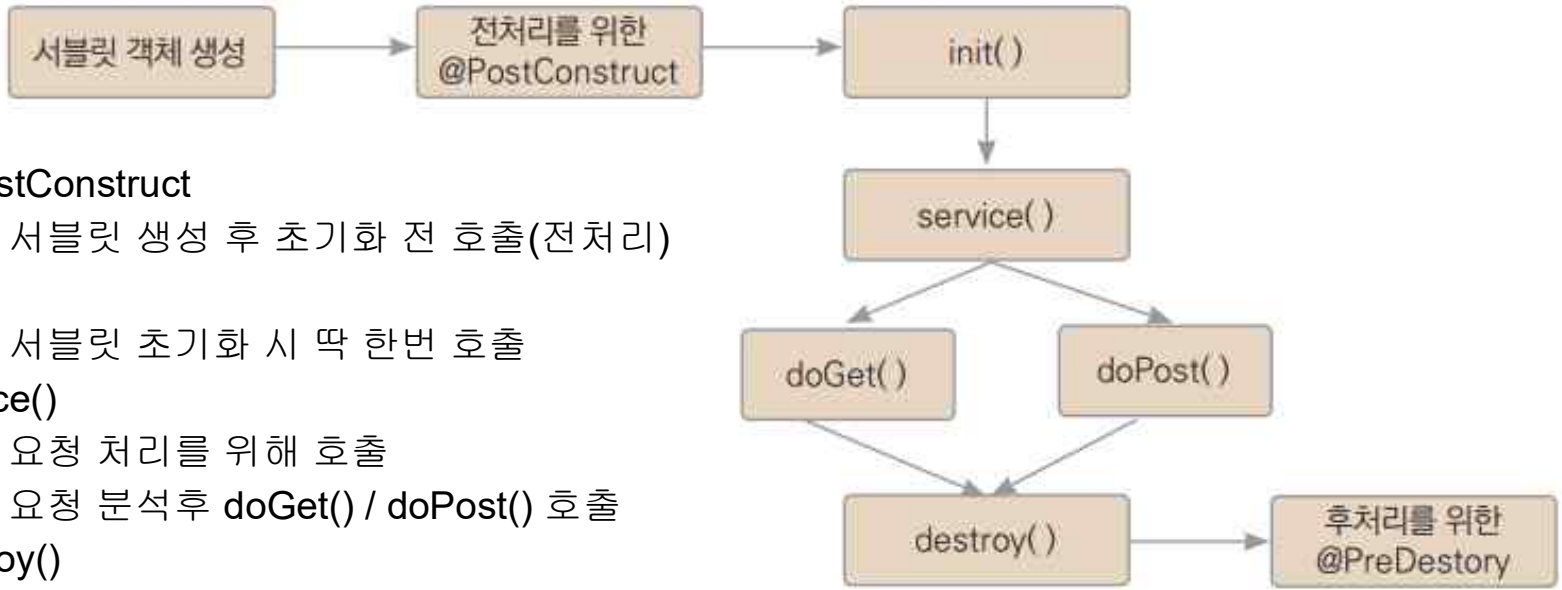
❽ 각 요청별 처리 메서드

12.5 서블릿 작성

▣ 한 번의 매핑으로 여러 가지 요청 처리(계속)

12.5 서블릿 작성

▣ 서블릿의 수명주기 메서드



- @PostConstruct
  - 서블릿 생성 후 초기화 전 호출(전처리)
- init()
  - 서블릿 초기화 시 딱 한번 호출
- service()
  - 요청 처리를 위해 호출
  - 요청 분석후 doGet() / doPost() 호출
- destroy()
  - 서버 종료시 호출됨
- @PreDestroy
  - 제일 마지막에 호출(후처리)

12.5 서블릿 작성

▣ 서블릿의 수명주기 메서드(계속)

. 서블릿은 클라이언트의 요청이 들어오면 앞의 그림과 같은 과정을 거치게 됨
. 해당 과정내에서 자동으로 호출되는 메서드를 "수명주기 메서드"라고 함

LifeCycle.jsp

```
<script>
function requestAction(frm, met) {  ❶
    if (met == 1) {  ❷
        frm.method = 'get';
    }
    else {
        frm.method = 'post';
    }
    frm.submit();  ❸
}
</script>
```

```
<h2>서블릿 수명주기(Life Cycle) 메서드</h2>
<form action="./LifeCycle.do">  ❹
    <input type="button" value="Get 방식 요청하기"
           onclick="requestAction(this.form, 1)" />
    <input type="button" value="Post 방식 요청하기"
           onclick="requestAction(this.form, 2)" />
```

❶ 각 버튼 클릭시 매개변수로 1 혹은 2가 전달됨
❷ 1인 경우 get 방식, 2인 경우 post 방식으로 전송방식을 변경
❸ 즉, 하나의 <form>에서 2가지 방식으로 전송하는 것을 Javascript로 구현

12.5 서블릿 작성

▣ 서블릿의 수명주기 메서드(계속) **LifeCycle.java**

```
@WebServlet("/12Servlet/LifeCycle.do")
public class LifeCycle extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @PostConstruct
    public void myPostConstruct() {
        System.out.println("myPostConstruct() 호출");
    }
    @Override
    public void init() throws ServletException {
        System.out.println("init() 호출");
    }

    @Override
    protected void service(HttpServletRequest req, HttpServletResp
            throws ServletException, IOException {
        System.out.println("service() 호출");
        // 전송 방식을 확인해 doGet() 또는 doPost() 호출
        super.service(req, resp);
    }
```

```
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        System.out.println("doGet() 호출");
        req.getRequestDispatcher("/12Servlet/LifeCycle.jsp").forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        System.out.println("doPost() 호출");
        req.getRequestDispatcher("/12Servlet/LifeCycle.jsp").forward(req, resp);
    }

    @Override
    public void destroy() {
        System.out.println("destroy() 호출");
    }
```

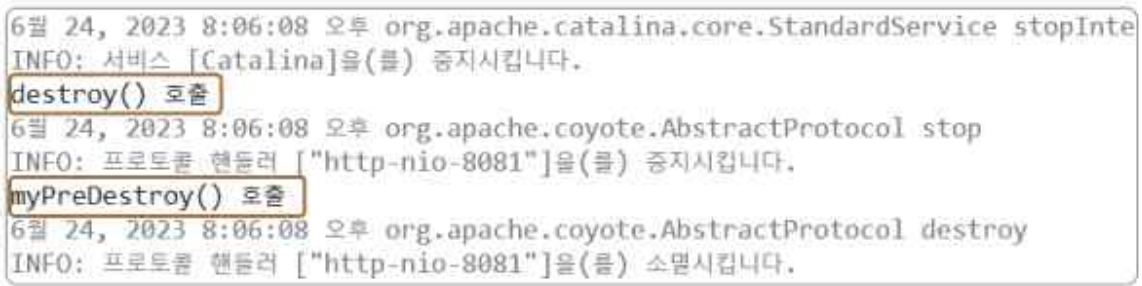
12.5 서블릿 작성

▣ 서블릿의 수명주기 메서드(계속)

LifeCycle.java

서블릿 수명주기(Life Cycle) 메서드

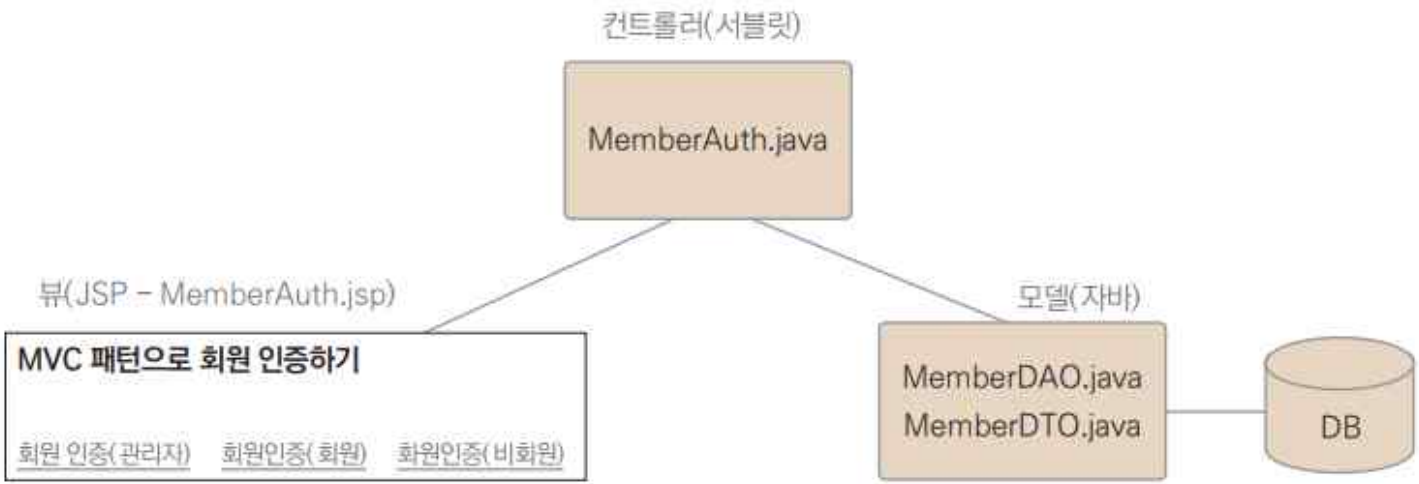
Get 방식 요청하기 | Post 방식 요청하기



- 첫 실행 시 myPostConstruct( ) → init( ) → service( ) 순서로 호출
- 2번째 버튼 부터는 service()부터 호출됨
- 서버 종료시 destroy( ) 메서드와 myPreDestroy( ) 메서드가 차례대로 호출됨

12.6 MVC 패턴을 적용한 회원인증 구현

▣ **MVC 패턴 회원인증 구성도**



서블릿의 수명주기 메서드와 MVC 패턴을 적용해서 회원인증 프로그램 제작

12.6 MVC 패턴을 적용한 회원인증 구현

▣ 뷰(JSP)

MemberAuth.jsp

```
<body>
    <h2>MVC 패턴으로 회원인증하기</h2>
    <p>
        <strong>${ authMessage }</strong> ❶
        <br />
        <a href="./MemberAuth.mvc?id=nakja&pass=1234">회원인증(관리자)</a> ❷
          
        <a href="./MemberAuth.mvc?id=musthave&pass=1234">회원인증(회원)</a> ❸
          
        <a href="./MemberAuth.mvc?id=stranger&pass=1234">회원인증(비회원)</a> ❹
    </p>
</body>
```

12.6 MVC 패턴을 적용한 회원인증 구현

▣ 컨트롤러(서블릿) web.xml

```
<servlet>
  <servlet-name>MemberAuth</servlet-name> ❶
  <servlet-class>servlet.MemberAuth</servlet-class> ❷
  <init-param> ❸
    <param-name>admin_id</param-name>
    <param-value>nakja</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>MemberAuth</servlet-name>
  <url-pattern>/12Servlet/MemberAuth.mvc</url-pattern> ❹
</servlet-mapping>
```

❶❹ 요청명에 대한 서블릿 매핑

❸ 해당 서블릿 내에서만 사용할 수 있는 초기화 파라미터
<context-param> 은 웹애플리케이션 전체에서 사용할 수 있음

12.6 MVC 패턴을 적용한 회원인증 구현

▣ 컨트롤러(서블릿) **MemberAuth.java**

```
public class MemberAuth extends HttpServlet {
    private static final long serialVersionUID = 1L;
    MemberDAO dao;  ❶

    @Override
    public void init() throws ServletException {  ❷
        // application 내장 객체 얻기 ❸
        ServletContext application = this.getServletContext();

        // web.xml에서 DB 연결 정보 얻기 ❹
        String driver = application.getInitParameter("OracleDriver");
        String connectUrl = application.getInitParameter("OracleURL");
        String oId = application.getInitParameter("OracleId");
        String oPass = application.getInitParameter("OraclePwd");

        // DAO 생성 ❺
        dao = new MemberDAO(driver, connectUrl, oId, oPass);
    }
```

❶ JDBC 프로그래밍을 위한 객체 선언

❷ 서블릿 초기화에 사용되는 init()메서드. 접속 정보를 가져온 후 DB연결

12.6 MVC 패턴을 적용한 회원인증 구현

▣ 컨트롤러(서블릿)

MemberAuth.java(계속)

```
@Override
protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {  ❻
    // 서블릿 초기화 매개변수에서 관리자 ID 받기 ❼
    String admin_id = this.getInitParameter("admin_id");
```

```
    // 회원 테이블에서 인증 요청한 ID/패스워드에 해당하는 회원 찾기 ❾
    MemberDTO memberDTO = dao.getMemberDTO(id, pass);
```

```
    String memberName = memberDTO.getName();
    if (memberName != null) {  // 일치하는 회원 찾음 ❿
        req.setAttribute("authMessage", memberName + " 회원님 방가방가^^*");
    }
    else {  // 일치하는 회원 없음
        if (admin_id.equals(id))  // 관리자 ⓫
            req.setAttribute("authMessage", admin_id + "는 최고 관리자입니다.");
```

❻ 요청 처리를 위해 service() 메서드 오버라이딩.

❼ 서블릿 초기화 파라미터 가져옴

❾ 파라미터를 통해 회원 정보 확인

12.6 MVC 패턴을 적용한 회원인증 구현

▣ 컨트롤러(서블릿)

MemberAuth.java(계속)

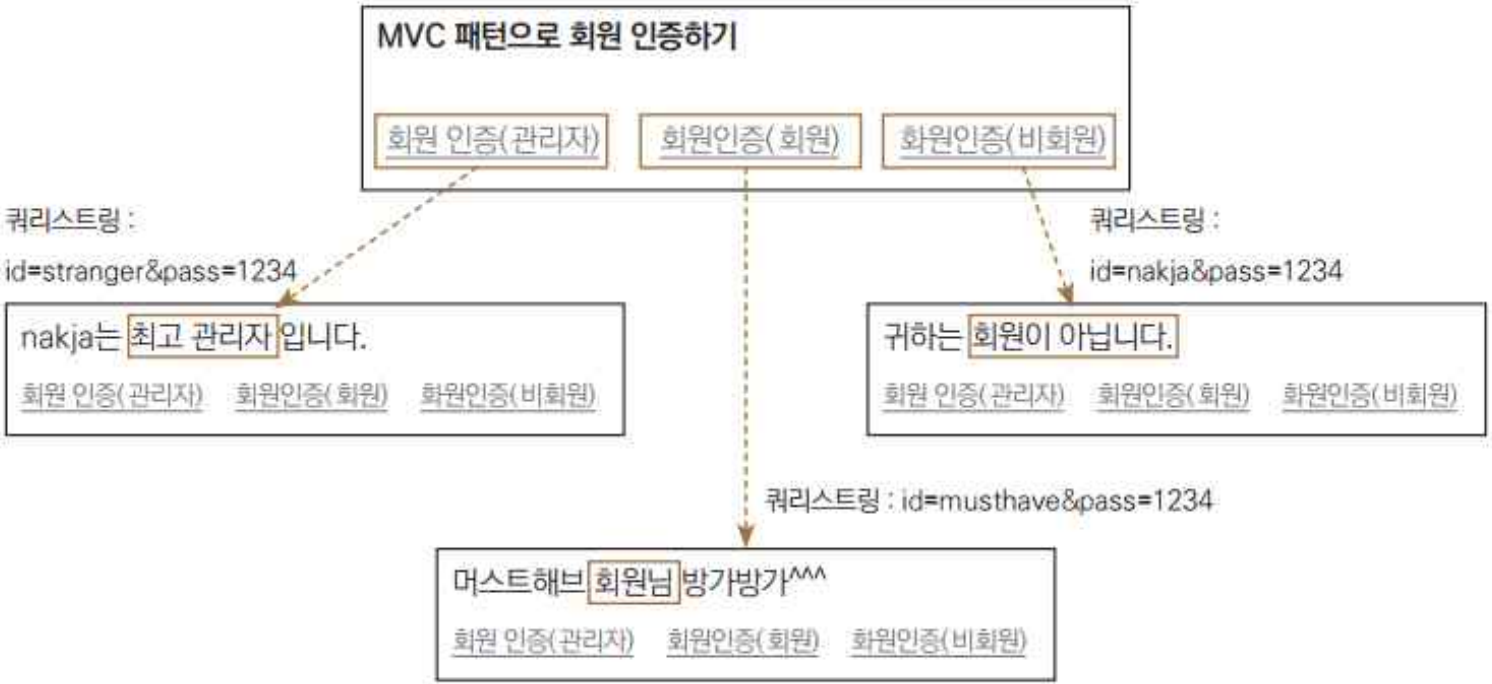
```
        else  // 비회원 ⑫
            req.setAttribute("authMessage", "귀하는 회원이 아닙니다.");
    }
    req.getRequestDispatcher("/12Servlet/MemberAuth.jsp").forward(req, resp); ⑬
}

@Override
public void destroy() {  ⑭
    dao.close();
}
```

⑬ JSP로 포워드

⑭ 서블릿 객체 소멸 시 DB 자원반납

12.6 MVC 패턴을 적용한 회원인증 구현

▣ 컨트롤러(서블릿)



학습 마무리

▣ 핵심요약

- 서블릿을 사용하면 MVC 패턴을 적용한 모델2 방식으로 웹 애플리케이션을 개발할 수 있음
- 요청명(요청 URL)과 이를 처리할 파일(서블릿)이 분리되어 있어서 둘을 매핑해야 함
- 요청명과의 매핑은 web.xml 혹은 @WebServlet 애너테이션을 이용하는 방식을 제공
- 서블릿은 HttpServlet 클래스를 상속받은 후 요청을 처리할 doGet( ) 혹은 doPost( ) 메서드를 오버라이딩
- 와일드카드(*)를 사용하여 여러 가지 요청을 하나의 서블릿에서 처리하도록 매핑할 수 있음
- 수명주기 메서드에서 확인했듯이 두 번째 요청부터는 첫 번째 요청 때 만들어둔 객체를 재사용하므로 처리 속도가 빨라짐