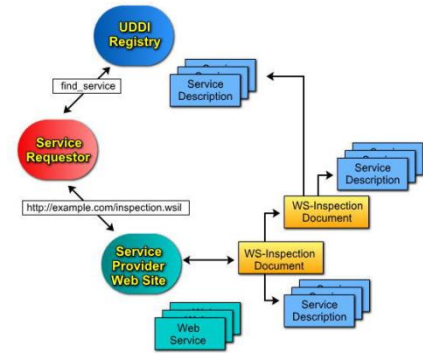


# Web Application & Service

Servlet



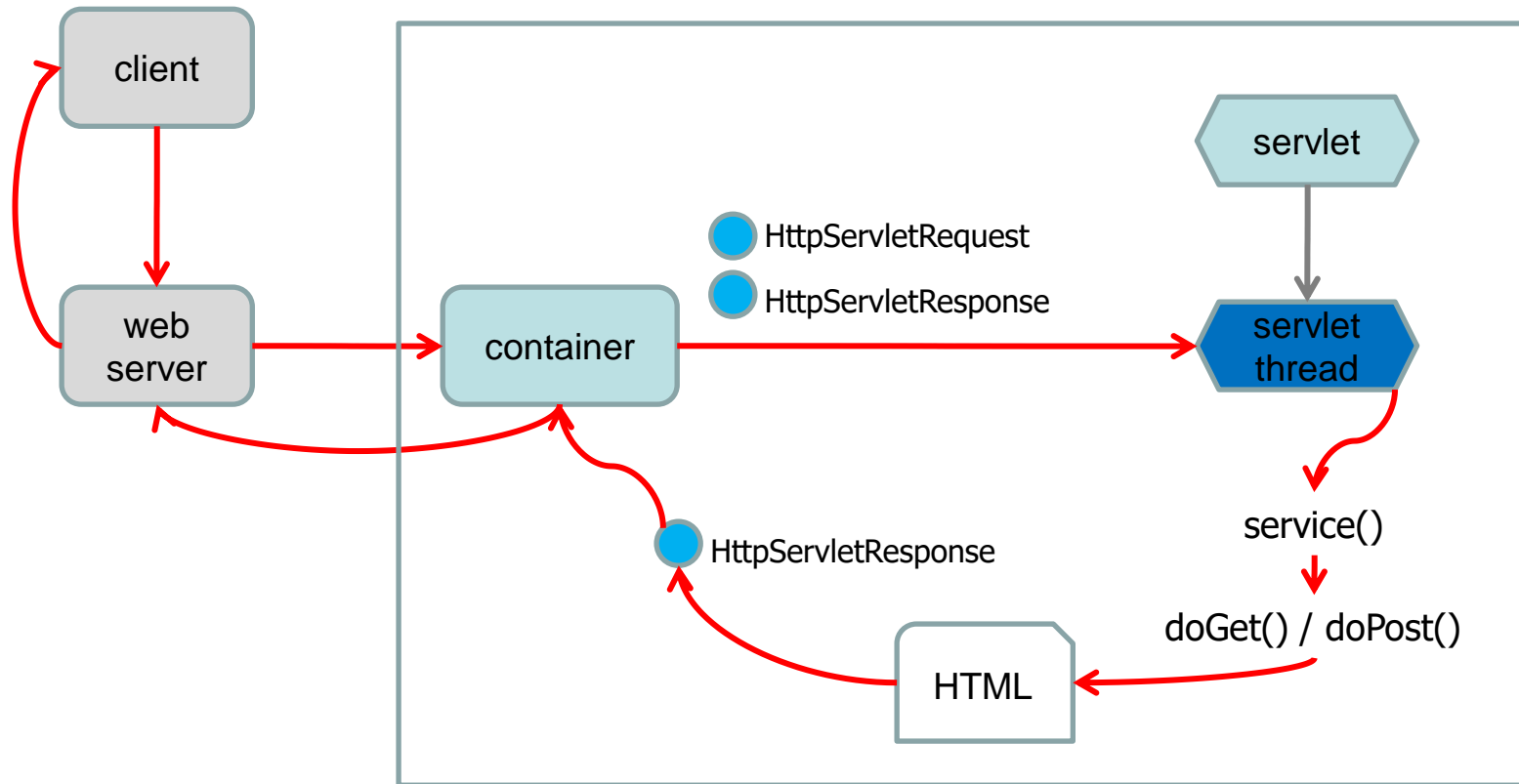
컴퓨터 과학과  
김희열

# Servlet

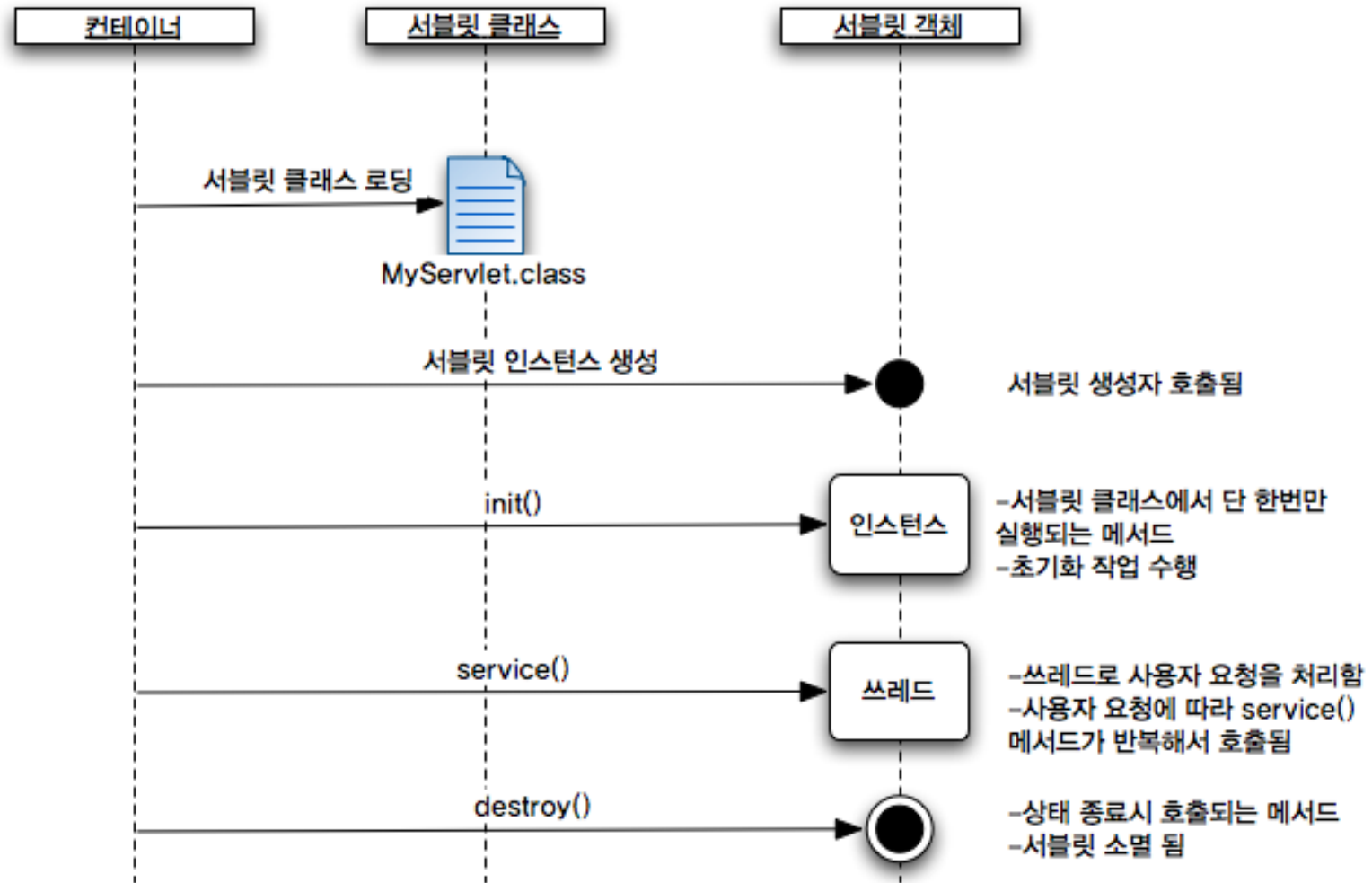
---

- Servlet
  - Client에 의해 요청된 request를 동적으로 처리하고 이에 대한 response를 생성해 내는 Java object
  - Java 플랫폼에서 동적인 contents를 생성하기 위해 사용
  - 주로 HTML문서를 생성해서 client에게 돌려줌
- Servlet container (Web container)
  - Servlet을 관리하고 실행하는 component
  - 자체적으로 JVM과 JRE를 포함
  - 웹서버의 URL요청을 받아 매핑되는 servlet을 실행
  - Servlet의 전반적인 lifecycle (생성, 실행, 종료)를 관리
  - Apache Tomcat, BEA WebLogic, ...
  - JSP도 내부적으로 servlet으로 변환되어 실행

# Container와 Servlet 동작 방식



# Container와 Servlet 동작 방식



# 요청 URL과 Servlet 매핑

- web.xml에 매핑 정보를 작성
  - 유연성과 보안성 증가

```
<servlet>
  <description>
  </description>
  <display-name>CalcServlet</display-name>
  <servlet-name>CalcServlet</servlet-name>
  <servlet-class>jspbook.ch04.CalcServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>CalcServlet</servlet-name>
  <url-pattern>/CalcServlet</url-pattern>
</servlet-mapping>
```

Servlet 이름

Servlet 클래스 이름

클라이언트가 사용  
하는 servlet 이름

- 최근에는 annotation 기반으로 매핑을 설정

```
@WebServlet("/CalcServlet")
public class CalcServlet extends HttpServlet {
```

# Servlet

---

- Servlet의 장점
  - Thread를 기반으로 해서 성능 우수
  - Java API를 직접 사용할 수 있음
  - OS나 하드웨어의 영향을 받지 않는다
- 웹 애플리케이션 개발시 **servlet**의 장점
  - 뷰와 비즈니스 로직을 분리
  - 기능 확장 및 유지 보수 용이
  - 리스너, 필터 등 고급 기법을 사용할 수 있음
- Servlet만 사용할 때의 단점
  - Servlet내에서 **print**를 통해 **HTML**문서를 출력
  - 유지 보수 어려움

# Servlet

① 서블릿 문제점 대두

프로그램에서 HTML 핸들링  
개발과 관리의 어려움

② JSP 등장

JSP를 통해 HTML에서 프로그램 핸들링

③ JSP 스크립트  
기술의 한계

뷰와 비즈니스 로직이 분리되지 않음

④ MVC 패턴  
주목받기 시작

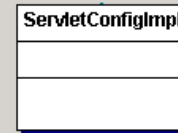
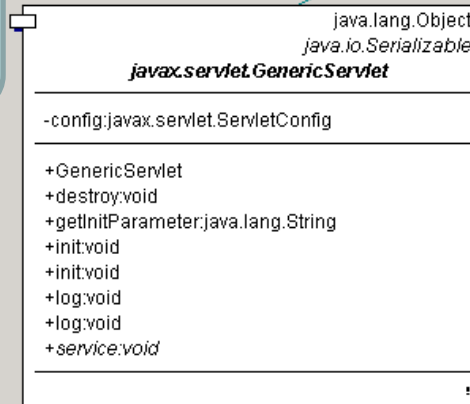
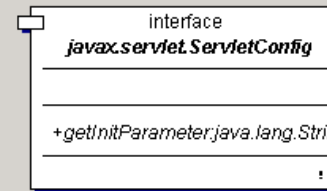
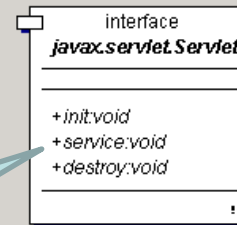
애플리케이션 구성 요소 단위로 역할 분담  
모델: 자바 클래스(DAO, DO)  
뷰: JSP, JSTL  
컨트롤러: 서블릿

# Servlet 구조

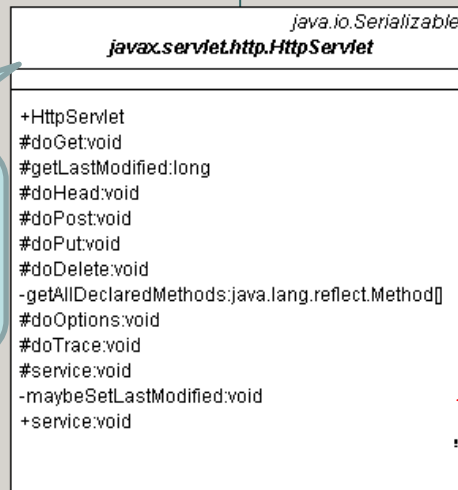
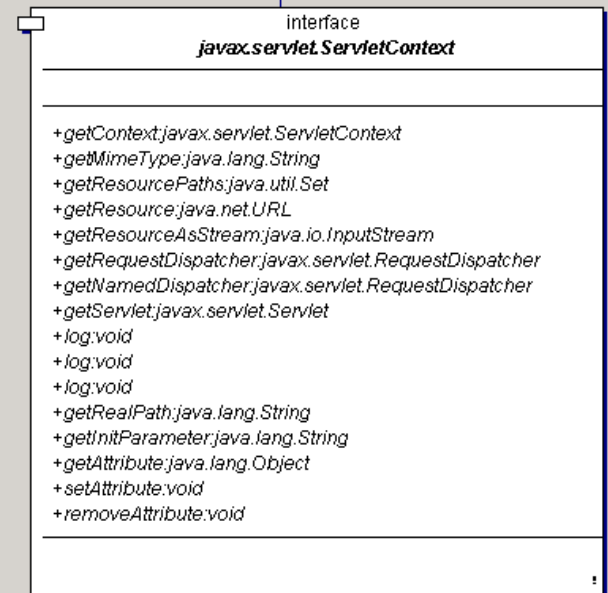
- API 구조

init(ServletConfig)  
service(ServletRequest,  
ServletResponse)  
destroy()

service(HttpServletRequest,  
HttpServletResponse)



컨테이너 벤더에서 구현함



MyServlet

doPost()  
doGet()  
myMethod()



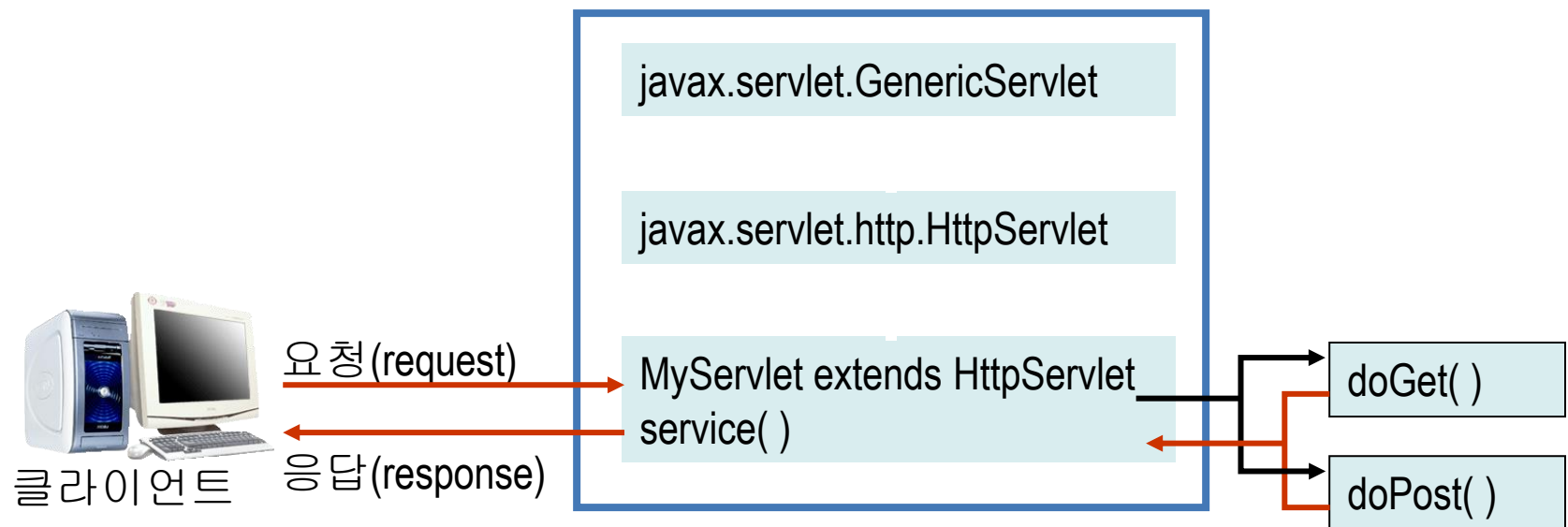
# Servlet Lifecycle

---

- Servlet loading
  - Container가 시작되거나 요청을 처리하기 전에 container에 의해 관리
  - 새로운 servlet 인스턴스를 생성 후 `init()`을 호출
  - `init()` 안에서 각종 초기화 작업을 수행
  - DB 접속 등 해당 servlet 내에서 수행할 작업이 있으면 `init()`을 override
- 요청 처리
  - 새로운 요청이 들어오면 container가 servlet의 새로운 thread를 생성
  - 해당 thread의 `service()`를 호출
  - `service()` 내에서 요청이 GET방식인지 POST방식인지 구분해서 `doGet()/doPost()`를 호출
  - 이때, `request` 객체와 `response` 객체가 파라미터로 전송
  - `doGet()/doPost()`에서 해당 요청을 처리
  - `service()`가 종료되면 thread 종료
- Servlet 종료
  - 보통 container가 종료되는 시점에 `destroy()` 호출
  - 특정 servlet 로드/언로드 시에도 사용

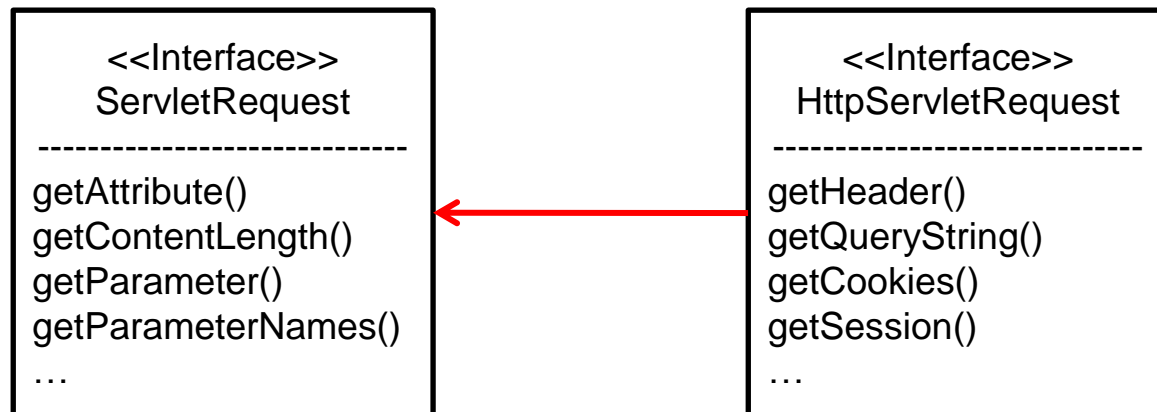
# Servlet Lifecycle

- Servlet API reference site
  - <http://www.javadoc.io/doc/javax.servlet/javax.servlet-api/4.0.1>



# HttpServletRequest

- 사용자의 요청 정보와 쿠키, 세션 등의 정보를 제공하는 interface
- Container가 이 interface를 구현한 객체를 servlet에게 제공



# HttpServletRequest

---

- Methods
  - `getParameter()`
    - 클라이언트가 입력한 파라미터의 값을 제공

GET request

`select.do?color=dark&body=heavy`

Servlet

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    String colorParam = request.getParameter("color");
    String bodyParam = request.getParameter("body");
    ...
}
```

# HttpServletRequest

- Methods
  - `getParameterValues()`
    - Checkbox나 select등으로 하나의 이름으로 여러 값을 보낸 경우 사용

## HTML

```
<form method="post" action="select.do">
    <input type="checkbox" name="sizes" value="12">12 <br>
    <input type="checkbox" name="sizes" value="14">14 <br>
    <input type="checkbox" name="sizes" value="16">16 <br>
    <input type="submit">
</form>
```

## GET request

```
select.do?sizes=14&sizes=16
```

## Servlet

```
String[] sizes = request.getParameterValues("sizes");
For (int i=0; i<sizes.length; i++) {
    String param = sizes[i];
}
```

# HttpServletRequest

---

- Methods

- `getParameterNames()`
  - 요청시 입력된 Name 값의 Enumeration을 제공
- `getHeader()`
  - 요청의 헤더 정보를 제공

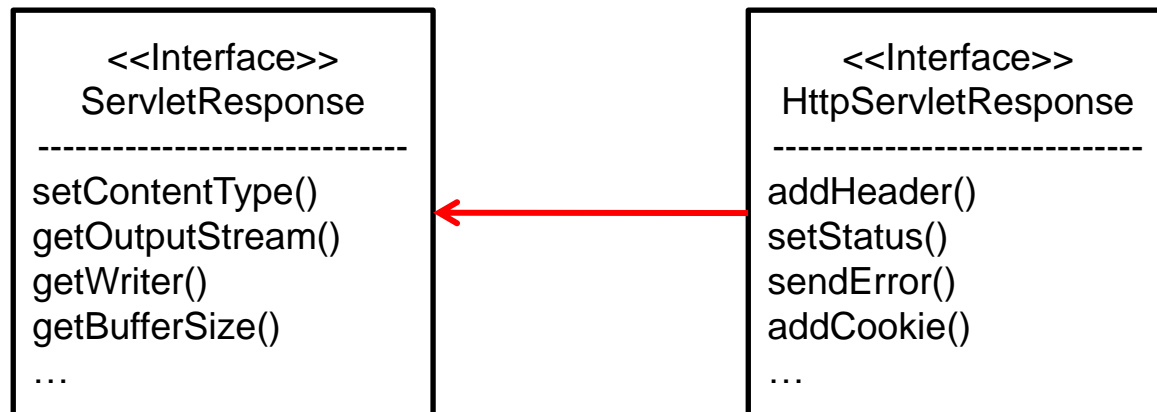
[Servlet](#)

```
String client = request.getHeader("User-Agent");
```

- `getCookies()`
  - 요청자의 쿠키값을 Cookie 객체의 array로 제공
- `getSession()`
  - 현재 요청자와 연결되어 사용되는 HttpSession 객체를 제공
- `getMethod()`
  - 요청자의 method (GET/POST)값을 제공
- `getRemoteAddr()`
  - 요청자의 IP address값을 제공
- ...

# HttpServletResponse

- 요청 처리 결과를 생성/전달하기 위한 정보를 제공하는 interface
- Container가 이 interface를 구현한 객체를 servlet에게 제공
- 주로 response로부터 Writer객체를 얻어서 HTML문서를 출력



# HttpServletResponse

---

- Methods

- `setContentType()`

- 요청에 대해 클라이언트에게 돌려줄 `content`의 타입을 결정

Servlet

```
response.setContentType("text/html");
```

- `setHeader()` / `addHeader()`

- 요청에 대해 클라이언트에게 돌려줄 `content`의 헤더 값을 설정

Servlet

```
response.setHeader("content-type", "text/html");
```

- `sendError()`

- 에러가 발생했음을 알려줌

Servlet

```
response.sendError(SC_NOT_FOUND);
```

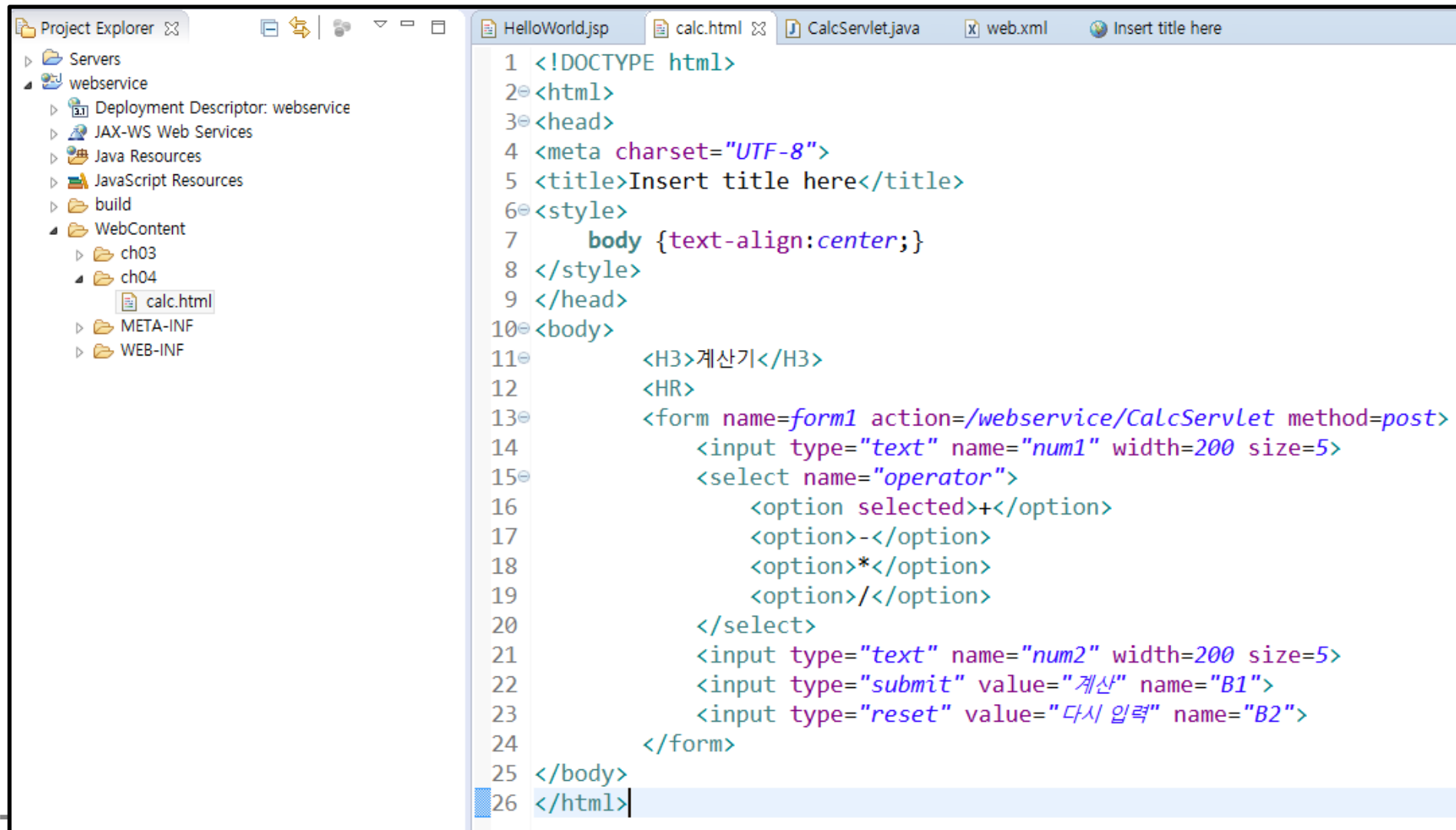
**Not Found**

The requested URL /oreilly/java-ent/servlet/ch05\_05.htm3 was not found on this server.



# Calculator Servlet

- HTML 문서 작성

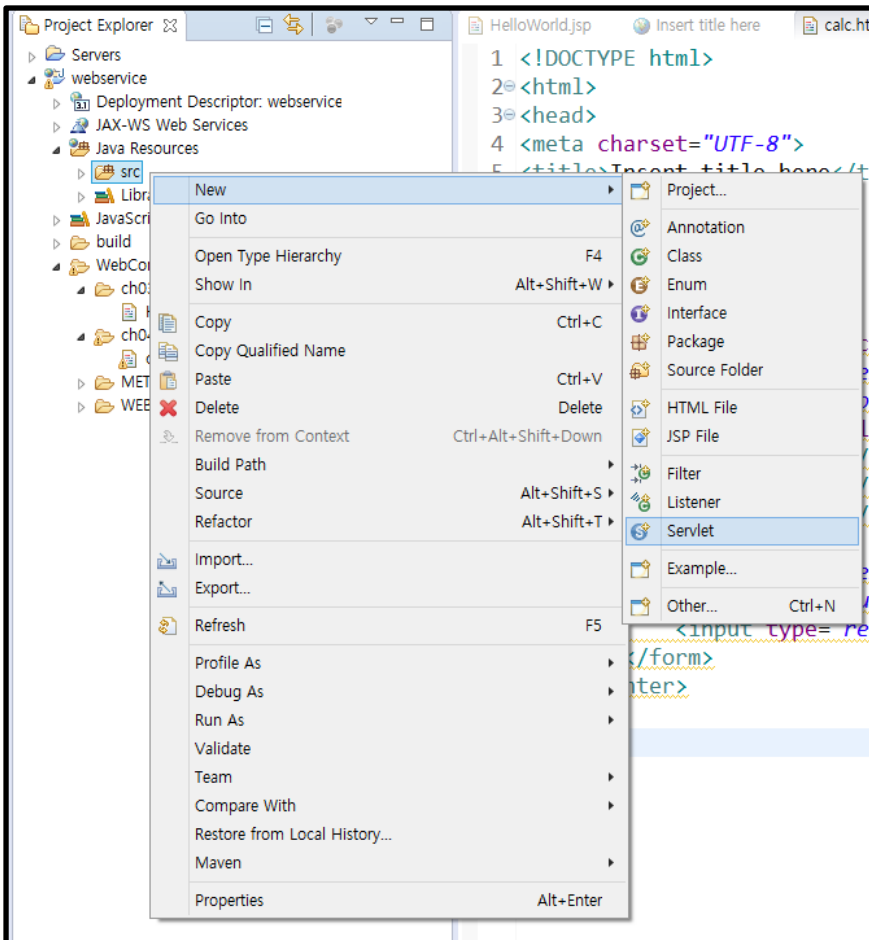


The screenshot shows an IDE with a Project Explorer on the left and a code editor on the right. The Project Explorer shows a web service project with a 'WebContent' folder containing 'ch03' and 'ch04' subfolders. The 'ch04' folder contains a 'calc.html' file. The code editor shows the content of 'calc.html', which is an HTML document for a calculator. The document includes a title 'Insert title here', a centered body, and a form with two text input fields, a dropdown menu for operators, and two submit buttons labeled '계산' (Calculate) and '다시 입력' (Re-enter).

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Insert title here</title>
6 <style>
7     body {text-align:center;}
8 </style>
9 </head>
10 <body>
11     <H3>계산기</H3>
12     <HR>
13     <form name=form1 action=/webservice/CalcServlet method=post>
14         <input type="text" name="num1" width=200 size=5>
15         <select name="operator">
16             <option selected>+</option>
17             <option>-</option>
18             <option>*</option>
19             <option>/</option>
20         </select>
21         <input type="text" name="num2" width=200 size=5>
22         <input type="submit" value="계산" name="B1">
23         <input type="reset" value="다시 입력" name="B2">
24     </form>
25 </body>
26 </html>
```

# Calculator Servlet

- CalcServlet 작성



Create Servlet

Specify class file destination.

Project: webservice

Source folder: /webservice/src/main/java [Browse...](#)

Java package: myapp [Browse...](#)

Class name: CalcServlet

Superclass: javax.servlet.http.HttpServlet [Browse...](#)

☐ Use an existing Servlet class or JSP

Class name: CalcServlet [Browse...](#)

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

# Calculator Servlet

**Create Servlet**

Enter servlet deployment descriptor specific information.

Name:

Description:

Initialization Parameters:

Add... Edit... Remove

URL Mappings:

Add... Edit... Remove

< Back Next > Finish Cancel

**Create Servlet**

Specify modifiers, interfaces to implement, and method stubs to generate.

Modifiers: ☒ Public ☐ Abstract ☐ Final

Interfaces:

Add... Remove

Which method stubs would you like to create?

☐ Constructors from superclass

☒ Inherited abstract methods

<input type="checkbox"/> init	<input type="checkbox"/> toString	<input type="checkbox"/> getServletInfo
<input checked="" type="checkbox"/> doPost	<input type="checkbox"/> doPut	<input type="checkbox"/> doDelete
<input type="checkbox"/> destroy	<input checked="" type="checkbox"/> doGet	

< Back Next > Finish Cancel

# Calculator Servlet

- CalcServlet.java 작성

```
1 package myapp;
2
3 import java.io.IOException;
4 import java.io.PrintWriter;
5
6 import javax.servlet.ServletException;
7 import javax.servlet.annotation.WebServlet;
8 import javax.servlet.http.HttpServlet;
9 import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11
12 /**
13  * Servlet implementation class CalcServlet
14  */
15 @WebServlet("/Calculator")
16 public class CalcServlet extends HttpServlet {
17     private static final long serialVersionUID = 1L;
18
19     /**
20      * @see HttpServlet#HttpServlet()
21      */
22     public CalcServlet() {
23         super();
24         // TODO Auto-generated constructor stub
25     }
```

# Calculator Servlet

```
27  /**
28   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
29   */
30  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
31      int num1, num2;
32      int result;
33      String op;
34
35      response.setContentType("text/html; charset=UTF-8");
36      PrintWriter out = response.getWriter();
37
38      num1 = Integer.parseInt(request.getParameter("num1"));
39      num2 = Integer.parseInt(request.getParameter("num2"));
40      op = request.getParameter("operator");
41
42      result = calc(num1, num2, op);
43
44      out.println("<HTML>");
45      out.println("<HEAD><TITLE>계산기</TITLE></HEAD>");
46      out.println("<BODY><center>");
47      out.println("<H2>계산 결과</H2>");
48      out.println("<HR>");
49      out.println(num1 + " " + op + " " + num2 + " = " + result);
50      out.println("</BODY></HTML>");
51  }
52  }
```

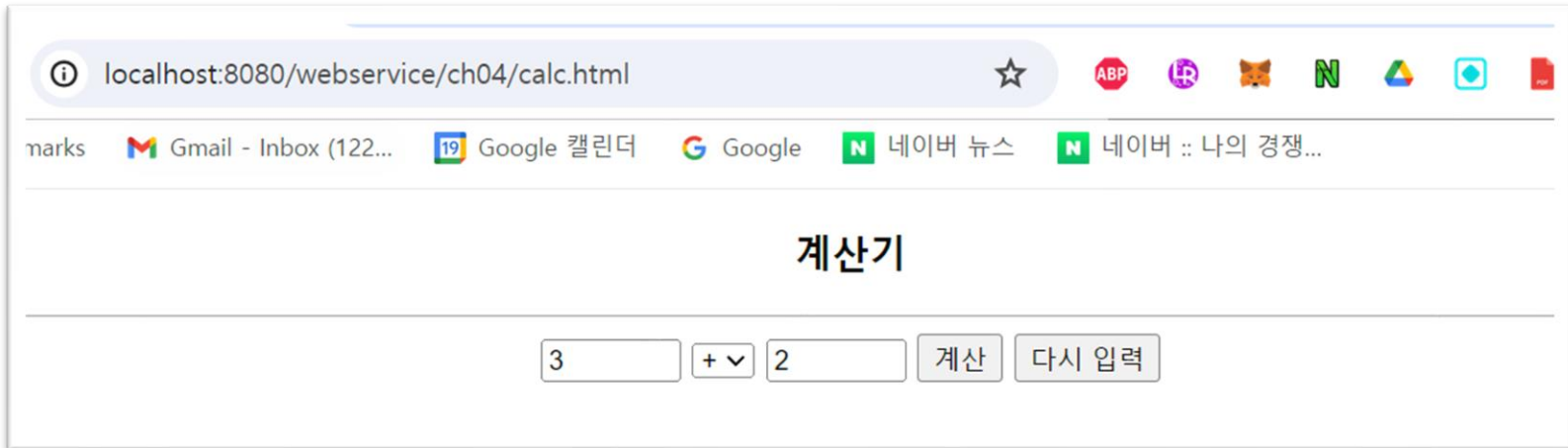
# Calculator Servlet

---

```
54 private int calc(int num1, int num2, String op) {  
55     int result = 0;  
56  
57     if (op.equals("+")) result = num1 + num2;  
58     else if (op.equals("-")) result = num1 - num2;  
59     else if (op.equals("*")) result = num1 * num2;  
60     else if (op.equals("/")) result = num1 / num2;  
61  
62     return result;  
63 }  
64  
65  
66 /**  
67  * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  
68  */  
69 protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
70     // TODO Auto-generated method stub  
71     doGet(request, response);  
72 }  
73  
74  
75  
76 }
```

# Calculator Servlet

- Run
  - calc.html > Run as > Run on Server

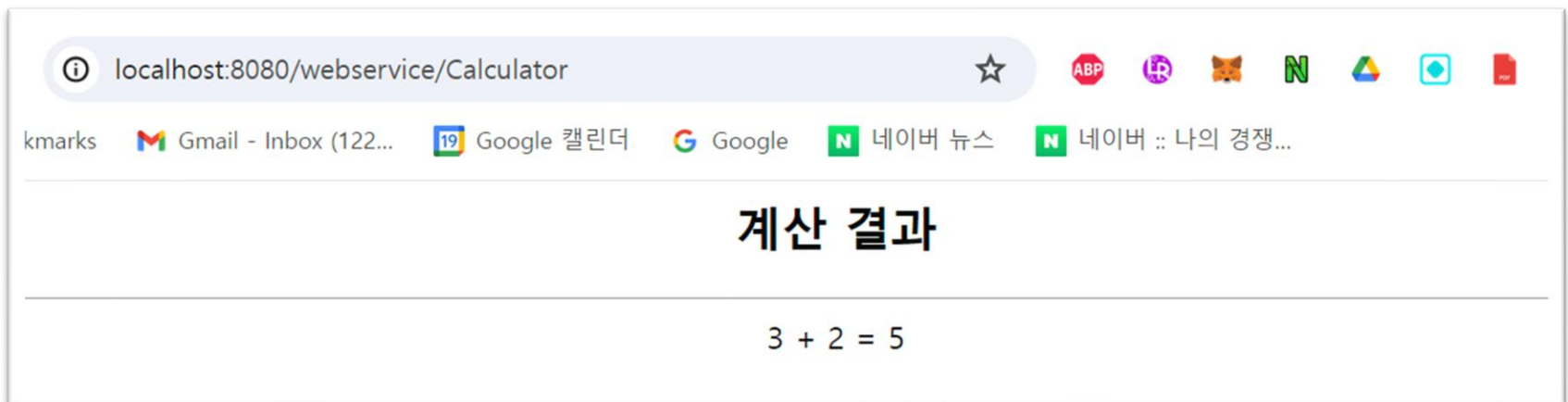


localhost:8080/webservice/ch04/calc.html

marks Gmail - Inbox (122... 19 Google 캘린더 Google 네이버 뉴스 네이버 :: 나의 경쟁...

## 계산기

3 + 2 계산 다시 입력



localhost:8080/webservice/Calculator

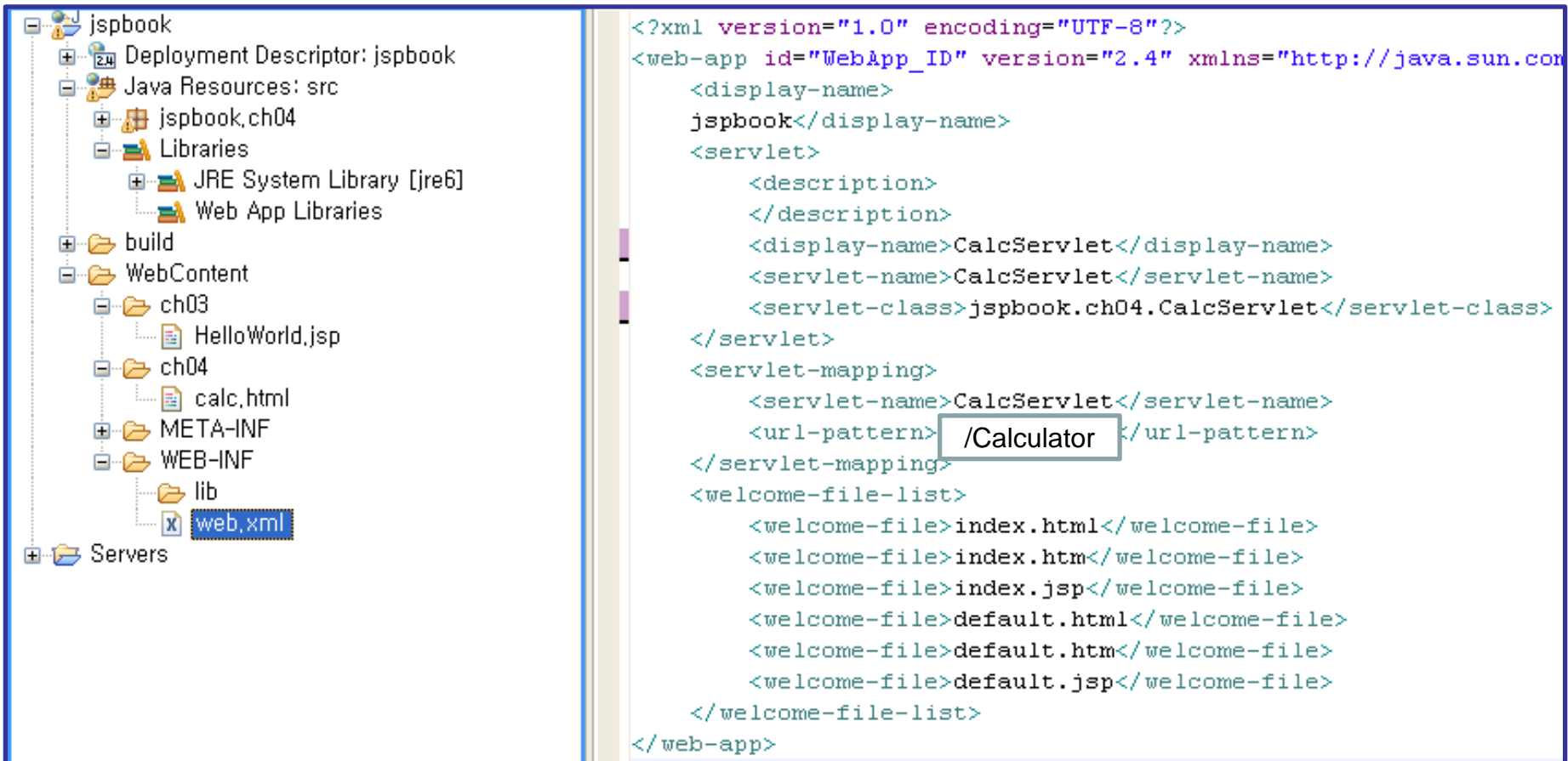
marks Gmail - Inbox (122... 19 Google 캘린더 Google 네이버 뉴스 네이버 :: 나의 경쟁...

## 계산 결과

3 + 2 = 5

# Calculator Servlet

- webservice\WebContent\WEB-INF\web.xml



The screenshot shows an IDE with a project named 'jspbook'. The project structure on the left includes 'Deployment Descriptor: jspbook', 'Java Resources: src', 'jspbook.ch04', 'Libraries' (JRE System Library [jre6], Web App Libraries), 'build', 'WebContent' (ch03 with HelloWorld.jsp, ch04 with calc.html), 'META-INF', 'WEB-INF' (lib with web.xml), and 'Servers'. The 'web.xml' file is selected and its content is displayed on the right. The XML content defines a web application with a display name of 'jspbook' and a single servlet named 'CalcServlet' that maps the URL pattern '/Calculator' to the class 'jspbook.ch04.CalcServlet'. The servlet has a list of welcome files including 'index.html', 'index.htm', 'index.jsp', 'default.html', 'default.htm', and 'default.jsp'.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/javaee">
  <display-name>
    jspbook</display-name>
  <servlet>
    <description>
    </description>
    <display-name>CalcServlet</display-name>
    <servlet-name>CalcServlet</servlet-name>
    <servlet-class>jspbook.ch04.CalcServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CalcServlet</servlet-name>
    <url-pattern>/Calculator</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

```
@WebServlet("/Calculator")
public class CalcServlet extends HttpServlet {
```