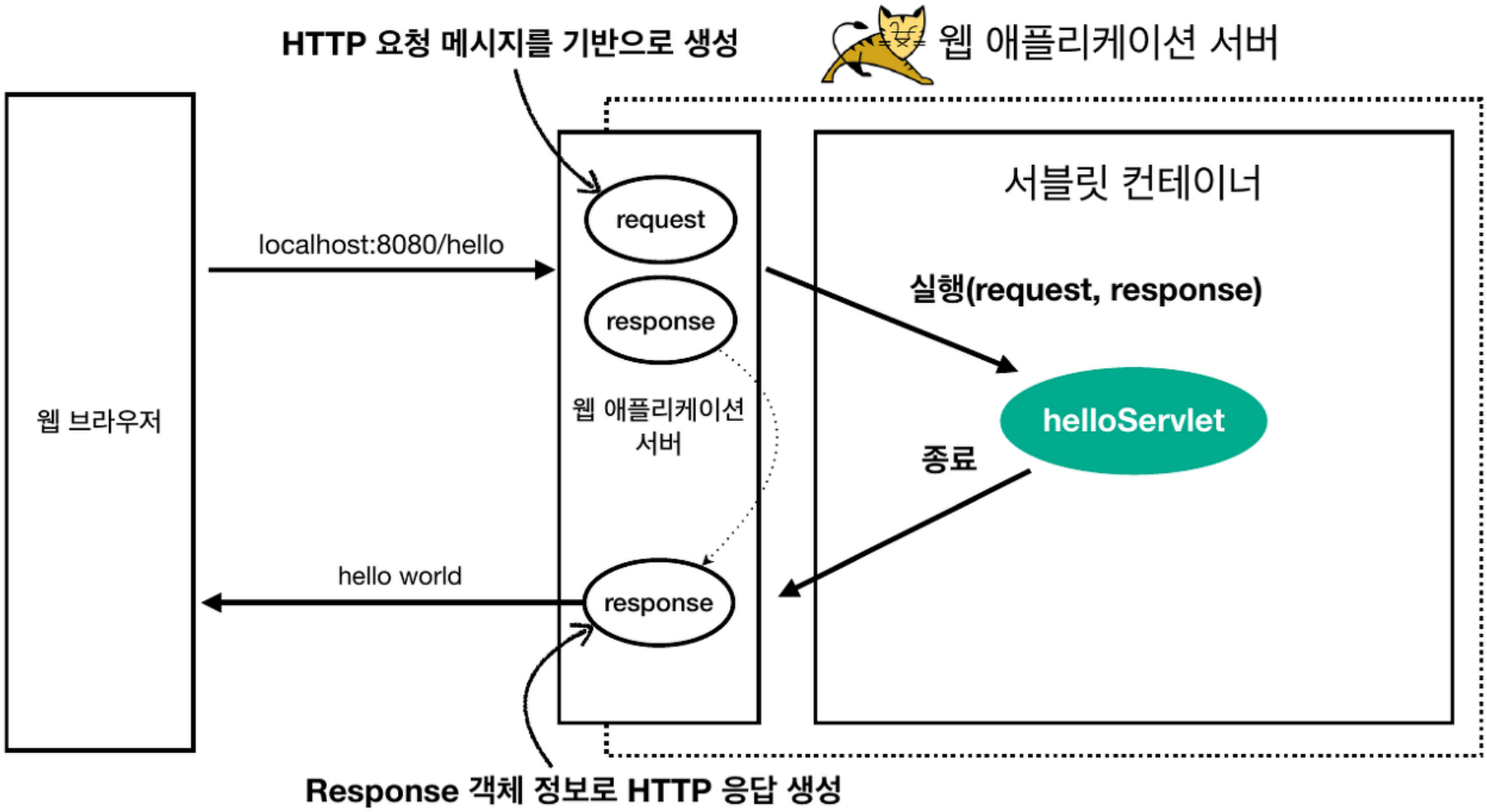
Servlet, JSP

524730
2024년 봄학기
3/20/2024
박경신

Web Architecture

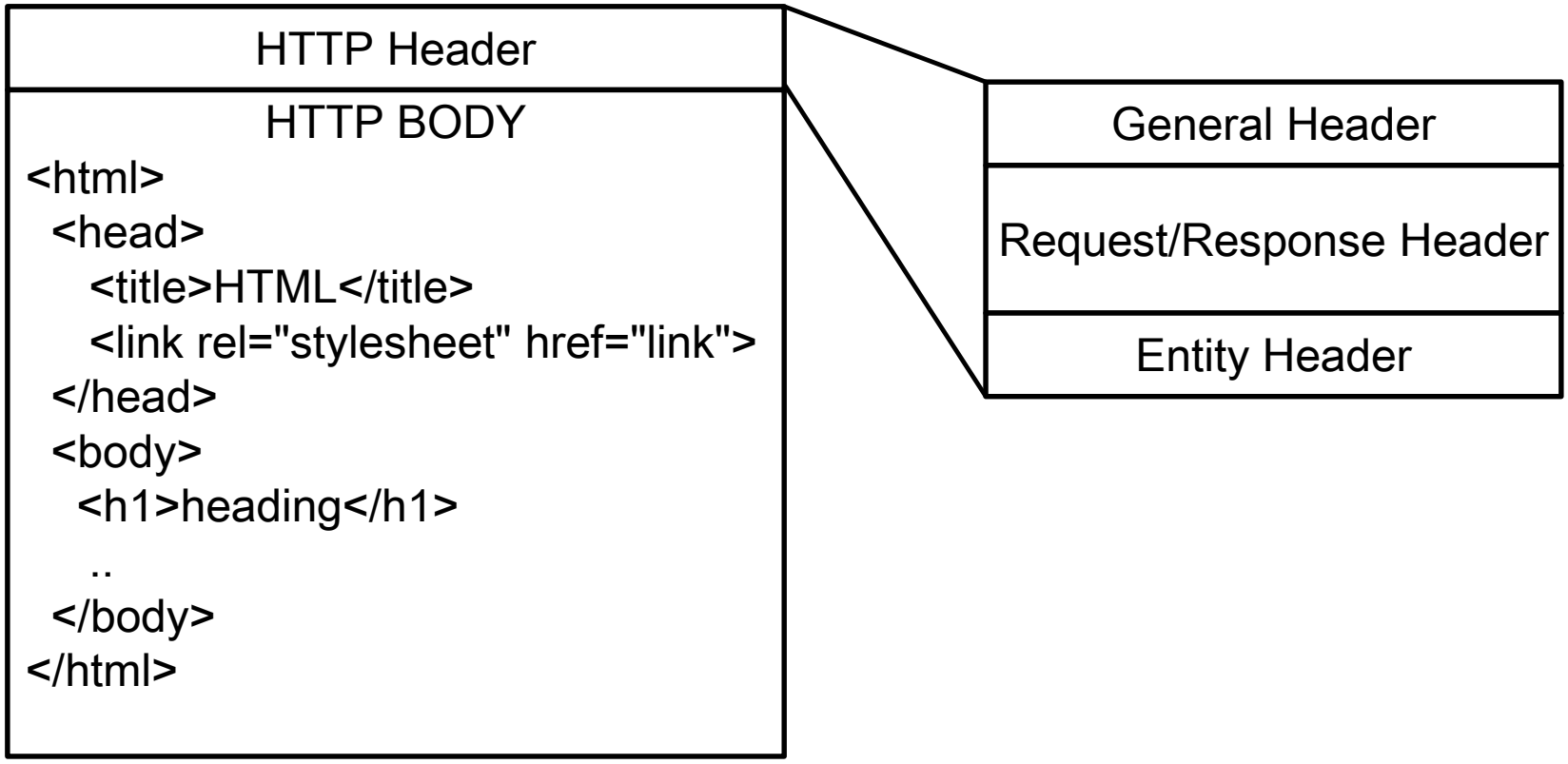


HTTP (Hyper Text Transfer Protocol)

- HTTP (Hyper Text Transfer Protocol)
  - 웹 서버와 클라이언트는 HTTP를 이용해 통신
  - Request & Response 구조 - 웹 클라이언트가 서버에 요청을 보내면, 서버가 요청에 대한 응답을 보내는 구조
  - Stateless - HTTP에서는 서버가 클라이언트의 상태를 보존하지 않음. 클라이언트의 상태를 유지하기위해 쿠키, 세션 등을 이용.
  - HTTP 1.0 Connectionless - HTTP는 하나의 요청과 응답을 처리한 후 연결을 종료. 웹 특성상 여러 명의 사용자가 브라우저를 통해서 서버를 호출하는 구조이기 때문에, 비연결성 방식을 사용하여 최소한의 자원으로 서버를 유지할 수 있음.
  - HTTP 1.1 Persistent Connection - HTTP 1.0 비연결성의 한계 (각각의 자원을 다운로드하기 위해 연결과 종료를 반복해야 했음)로, HTTP 1.1 지속 연결에서는 연결이 이루어지고 난 뒤 각각의 자원들을 요청하고, **모든 자원에 대한 응답이 돌아온 후에 연결을 종료.**

HTTP 구조

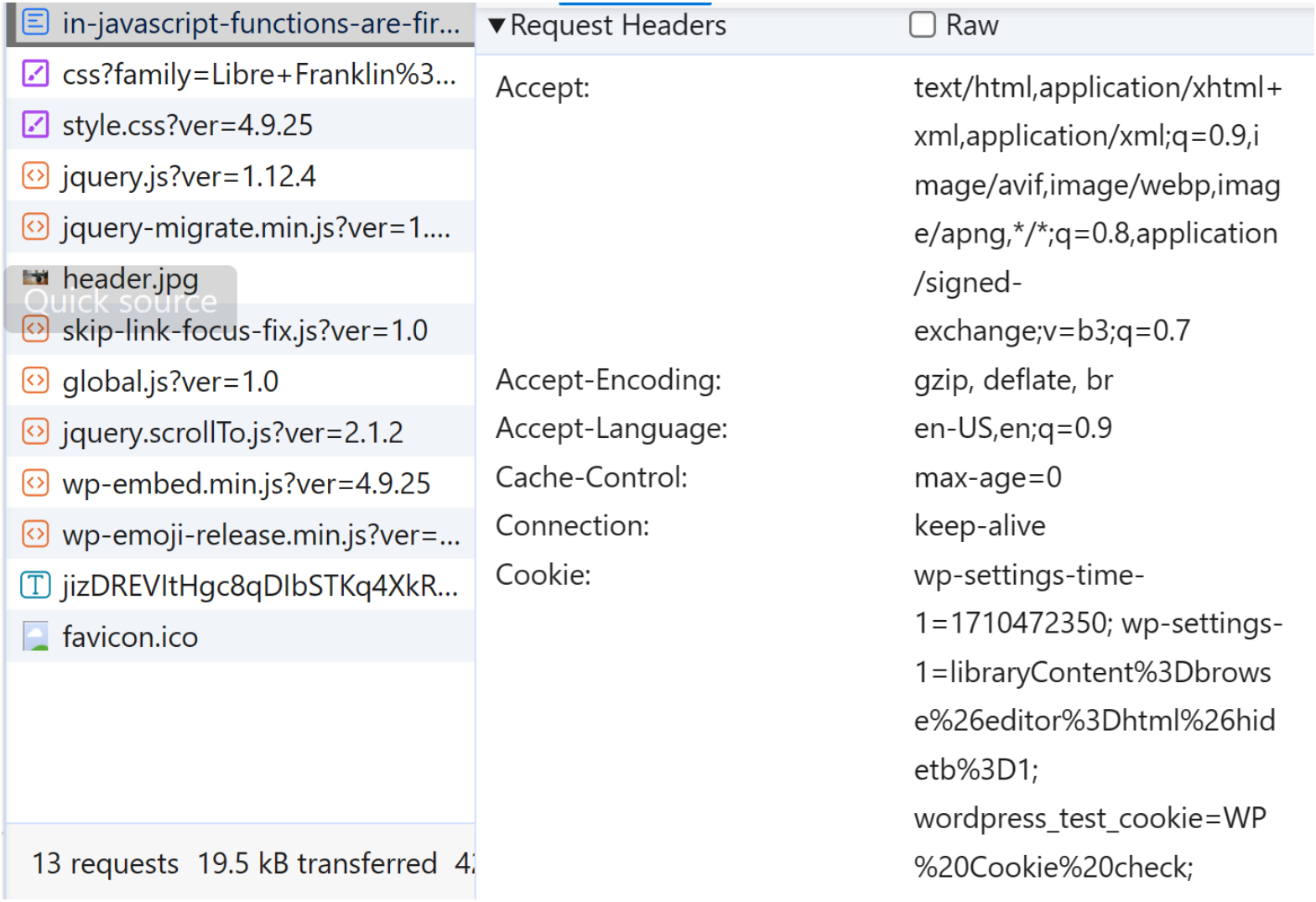
- HTTP는 Header와 Body로 구성
- HTTP Header는 HTTP 메시지(요청/응답)과 본문에 대한 정보를 포함



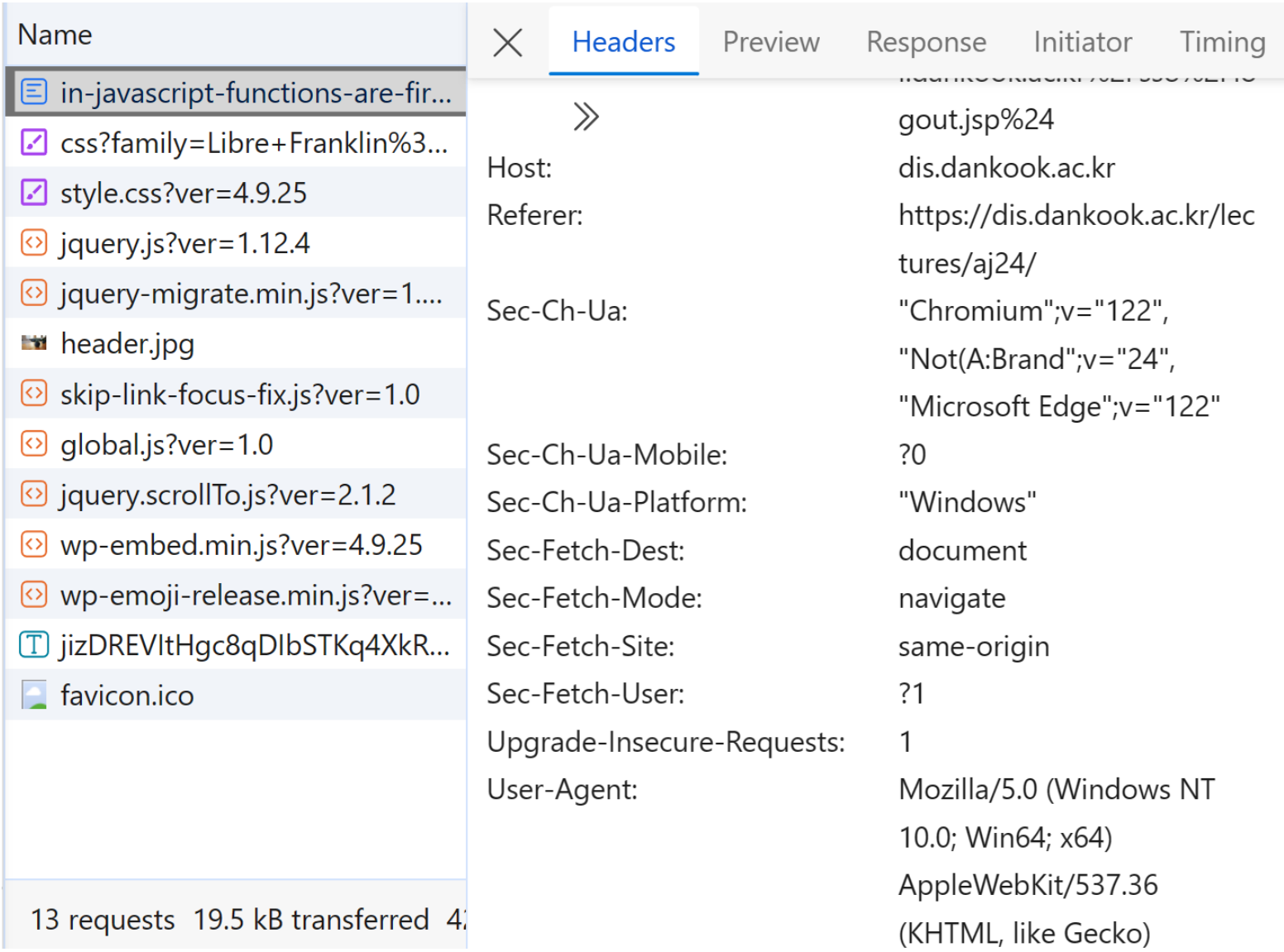
HTTP Message

- Request Message
  - Request Line, Request Headers, A blank line separates header & body, Request Message Body로 구성
  - Request Line은 데이터 처리 방식 (**HTTP Method**)과 기본 페이지 그리고 프로토콜 버전이 포함
  - Request Headers는 User-Agent, Accept, Cookie, Referer, host 정보가 포함

HTTP Message



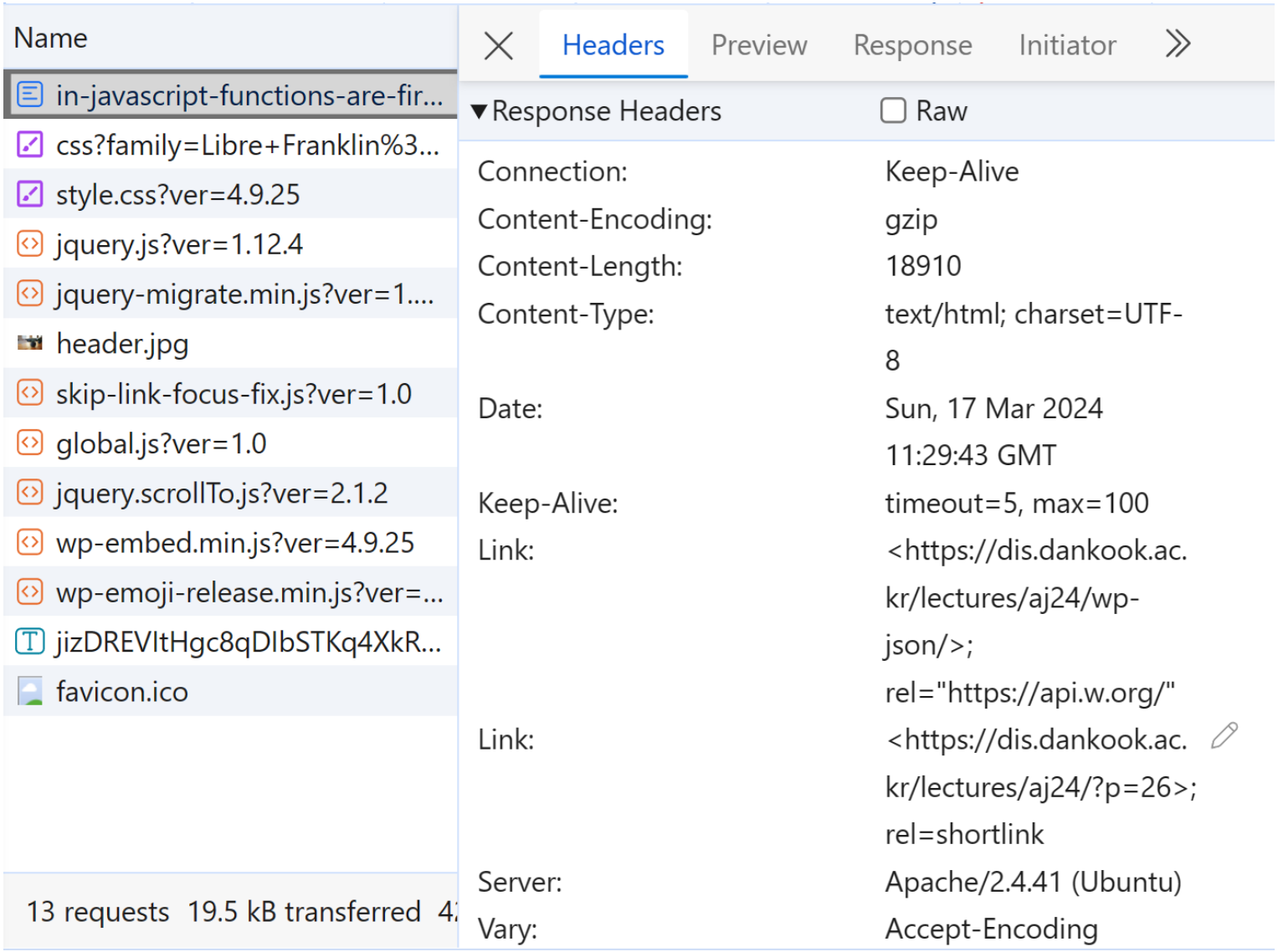
HTTP Message



HTTP Message

- Response Message
  - Status Line, Response Headers, A blank line separates header & body, Response Message Body로 구성
  - Status Line은 HTTP 버전, 상태코드(Status code), Reason-phrase가 포함
  - Response Headers는 Date, Server, Content-type, Last-Modified 정보가 포함

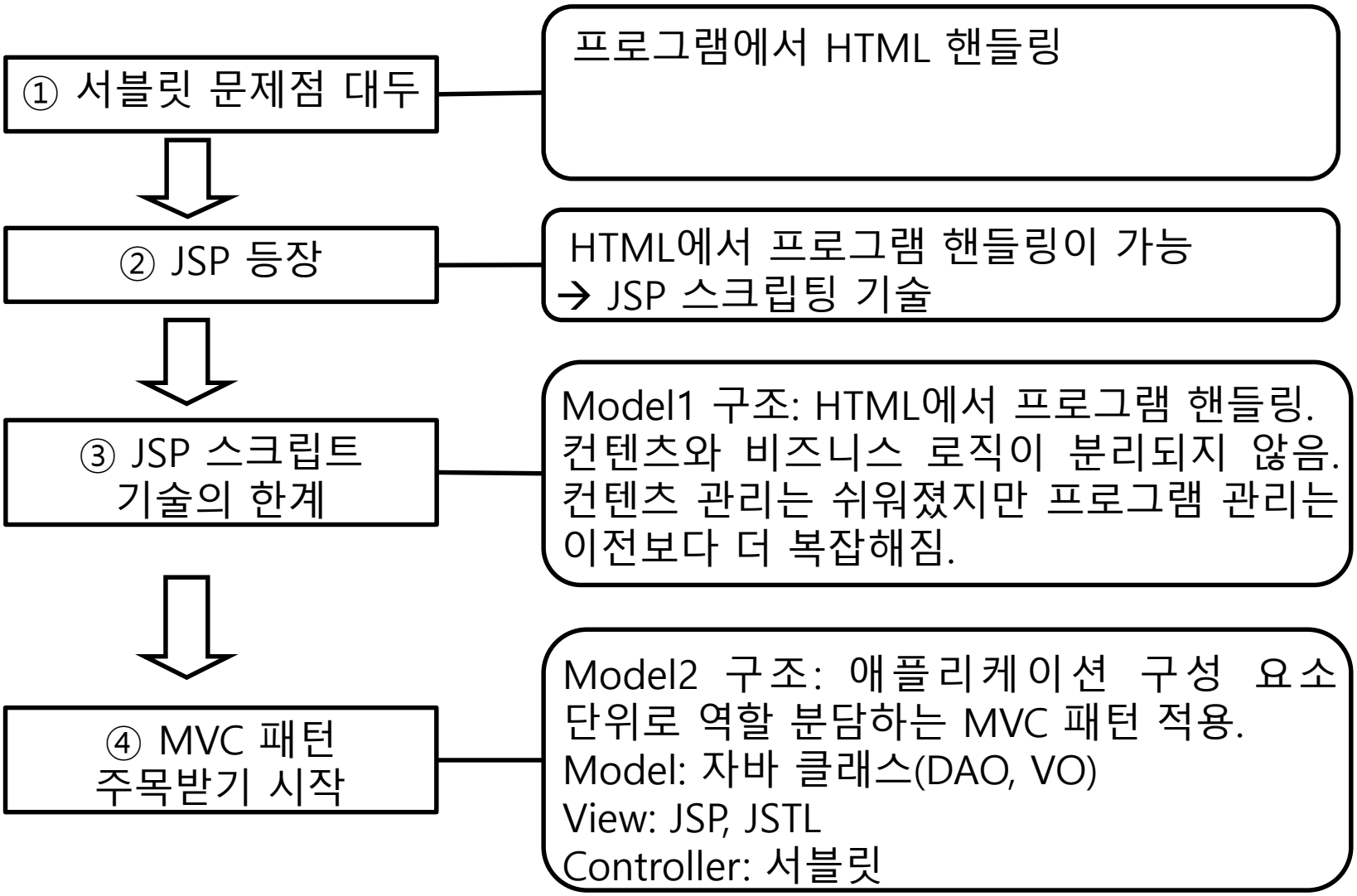
HTTP Message



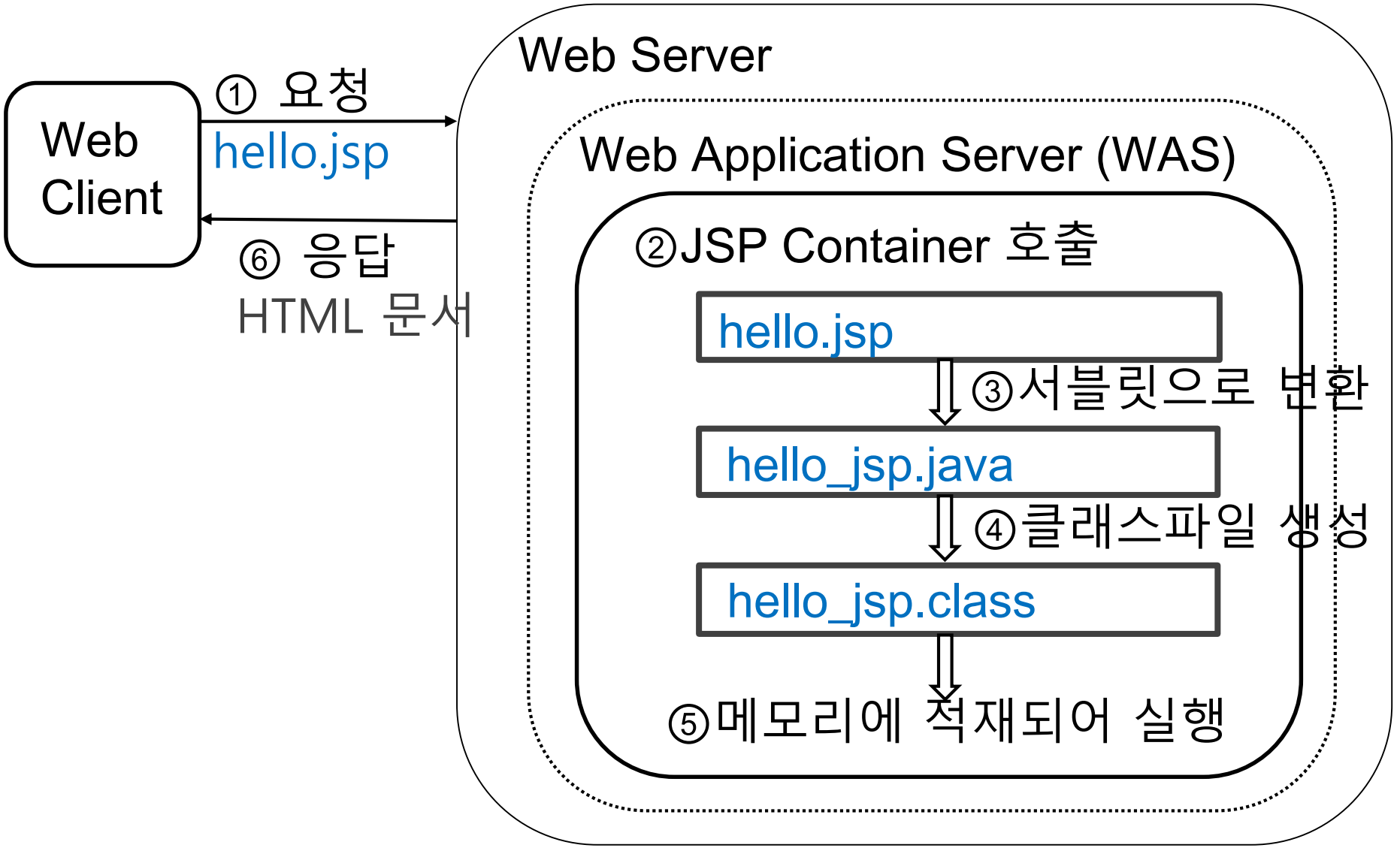
Servlet, JSP

- Servlet
  - 자바 플랫폼에서 컴포넌트 기반의 웹애플리케이션 개발 기술 클래스. 즉, 서버에 실행되면서 클라이언트의 요청에 따라 동적으로 서비스를 제공하는 자바 클래스.
  - 동적 웹페이지를 만들 때 Java 코드 안에 HTML 태그가 삽입되는 구조 HTML 태그를 문자열로 처리해야 함.
  - 서블릿을 이용하면 웹 프로그래밍이 가능하지만, 자바에 대한 지식이 필요하고, **화면 인터페이스 구현**에 너무 많은 코드를 필요로 하는 등 비효율적인 측면이 있었음.
- JSP(Java Server Pages)는 Servlet 기술에 기반함
  - Servlet의 프리젠테이션 문제를 해결하기 위해 JSP가 등장.
  - HTML 속에 자바 코드를 동적 웹페이지를 생성하는 스크립트 언어.
  - 이로 인해 웹 애플리케이션의 유지보수 어려움 심각.

Servlet 변천

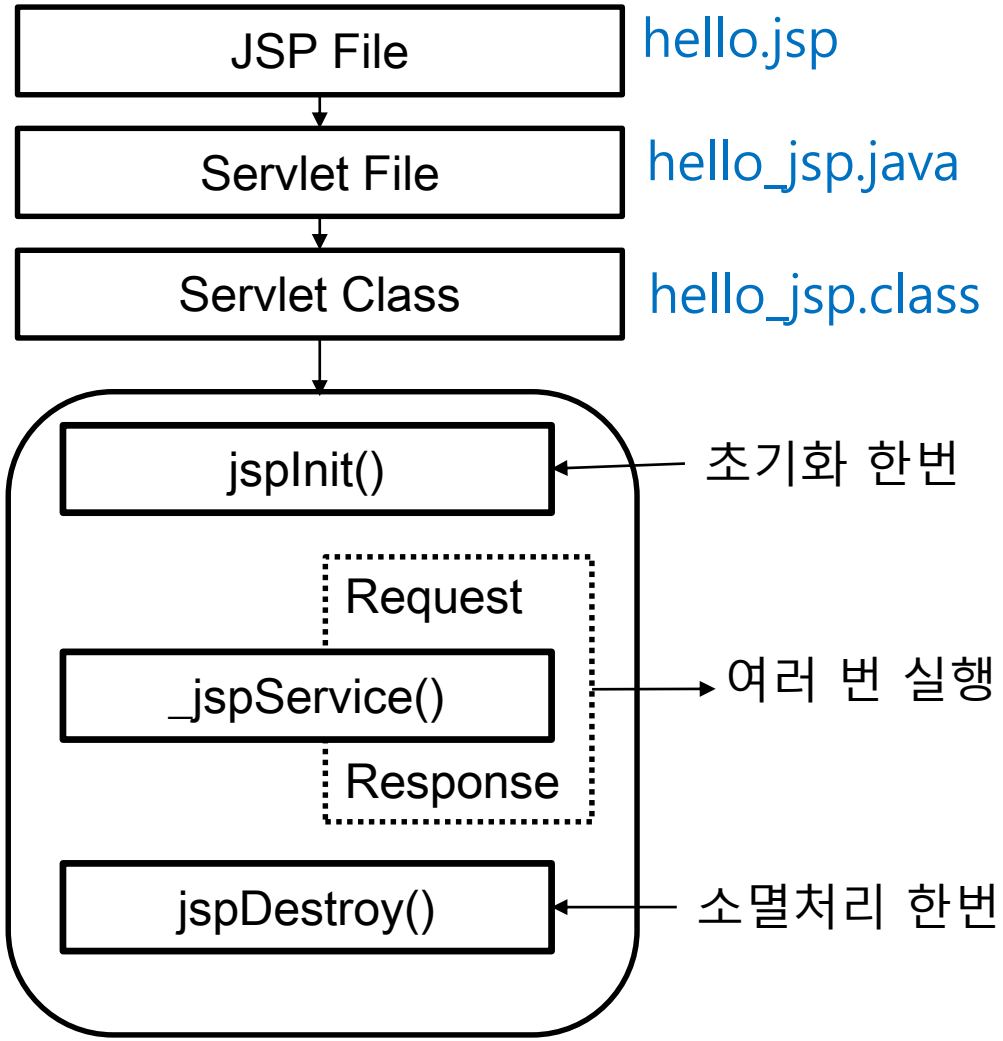

① 서블릿 문제점 대두
프로그램에서 HTML 핸들링
② JSP 등장
HTML에서 프로그램 핸들링이 가능
→ JSP 스크립팅 기술
③ JSP 스크립트 기술의 한계
Model1 구조: HTML에서 프로그램 핸들링. 컨텐츠와 비즈니스 로직이 분리되지 않음. 컨텐츠 관리는 쉬워졌지만 프로그램 관리는 이전보다 더 복잡해짐.
④ MVC 패턴 주목받기 시작
Model2 구조: 애플리케이션 구성 요소 단위로 역할 분담하는 MVC 패턴 적용.
Model: 자바 클래스(DAO, VO)
View: JSP, JSTL
Controller: 서블릿

JSP 동작 원리


Web Client
① 요청
hello.jsp
⑥ 응답
HTML 문서
Web Server
Web Application Server (WAS)
②JSP Container 호출
hello.jsp
③서블릿으로 변환
hello_jsp.java
④클래스파일 생성
hello_jsp.class
⑤메모리에 적재되어 실행

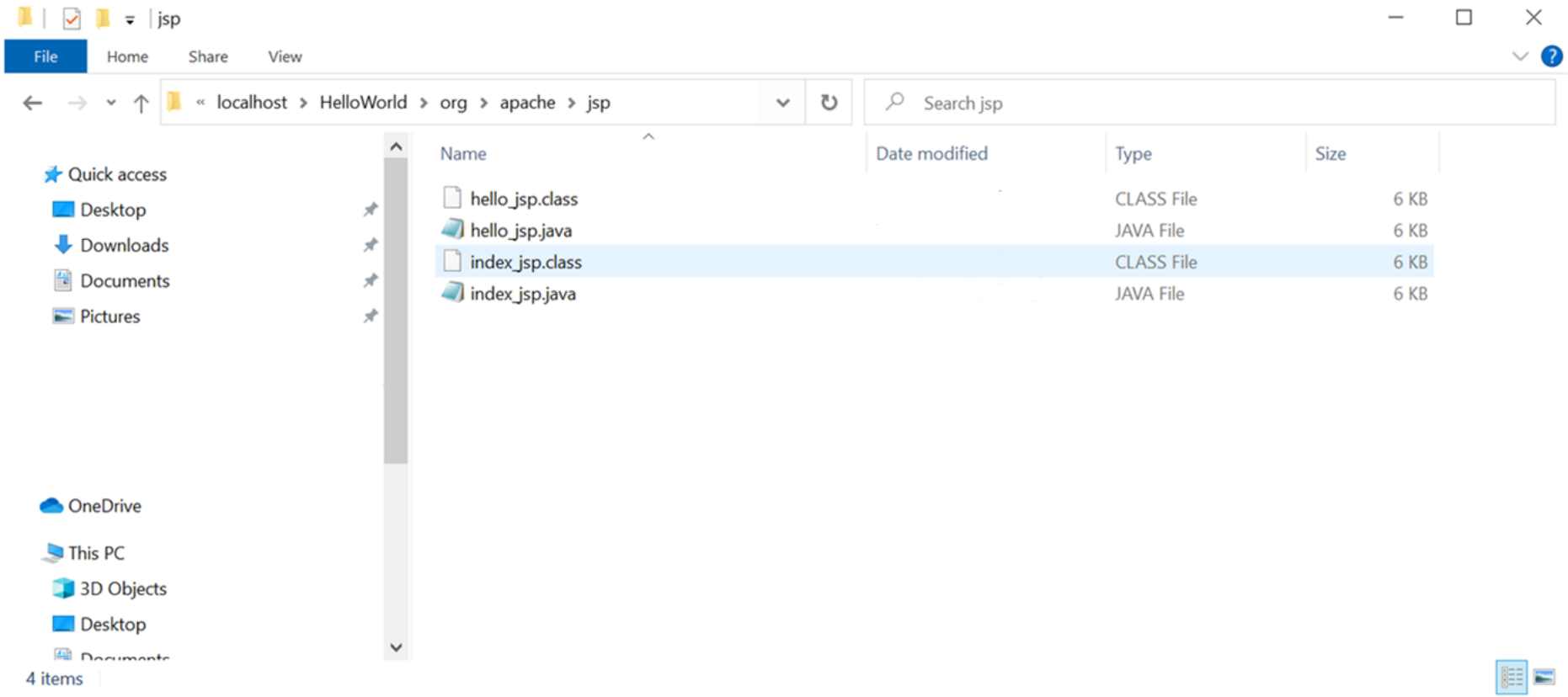
JSP Lifecycle

- JSP 생명주기
  - JSP Page Translation
  - JSP Page Compilation
  - Class Loading
  - Initialization
  - Execution
  - Cleanup



Servlet 코드로 생성된 JSP

- 서블릿 코드로 생성된 JSP



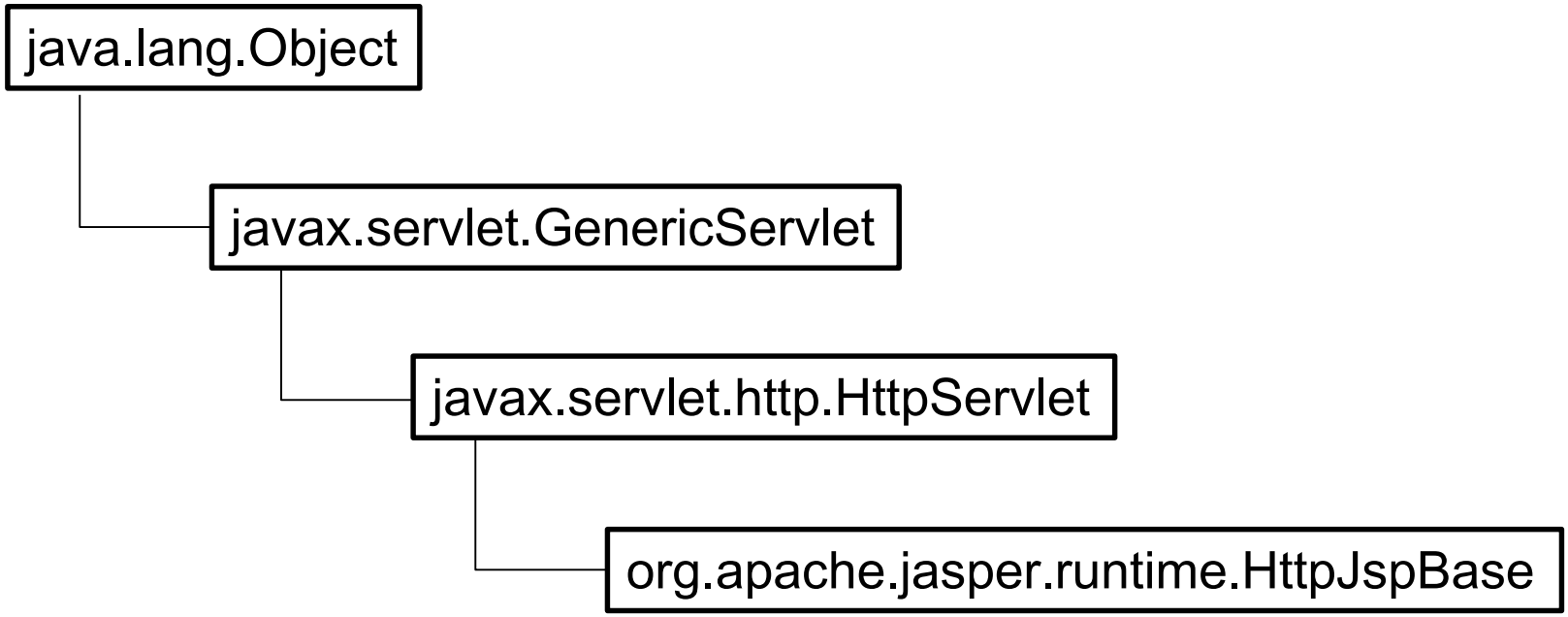
Servlet 코드로 생성된 JSP

- 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



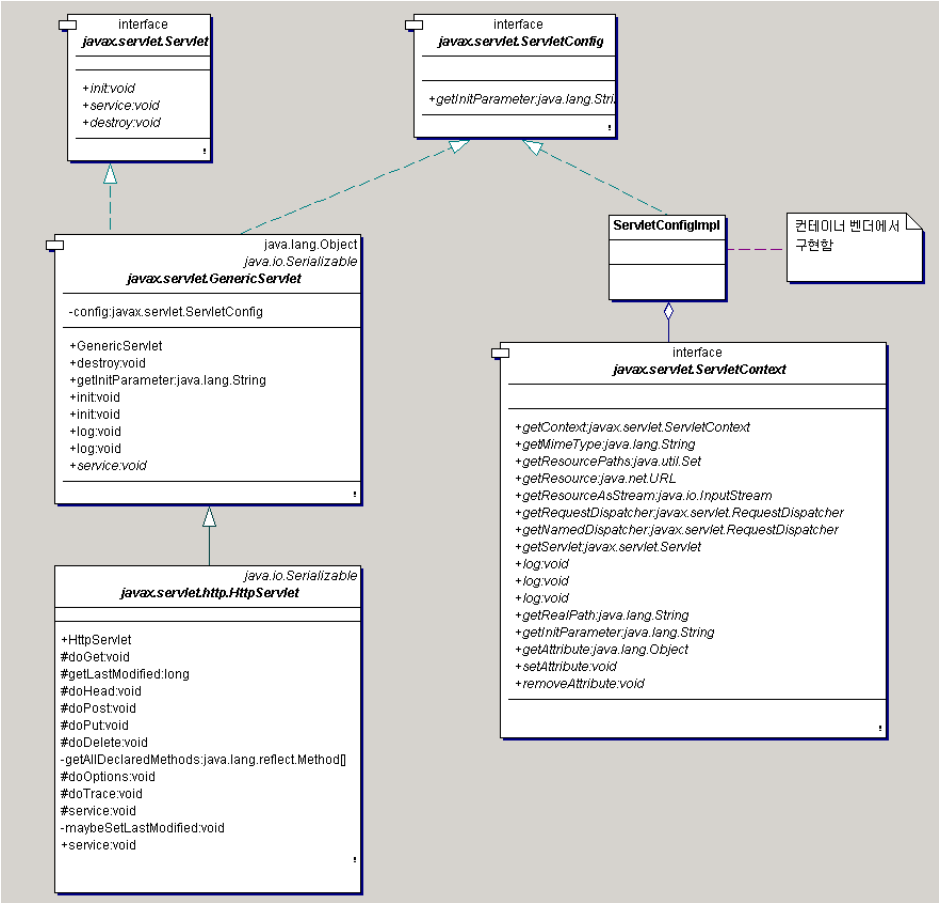
Servlet 코드로 생성된 JSP

- 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 ");
    out.write("</H2>\r\n오늘의 날짜와 시간은 : ");
    out.print( new java.util.Date() );
   … 이하중략
}
```

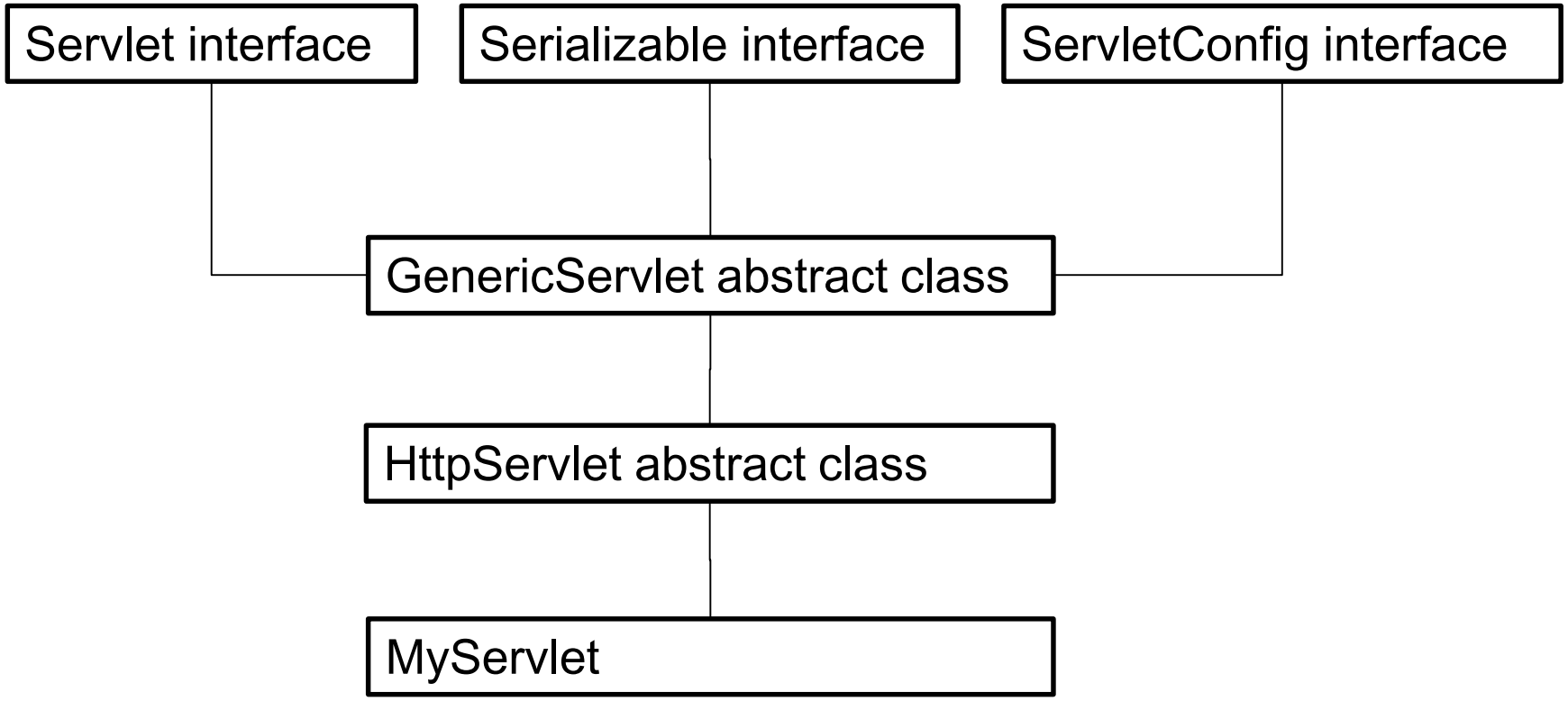
Servlet API

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



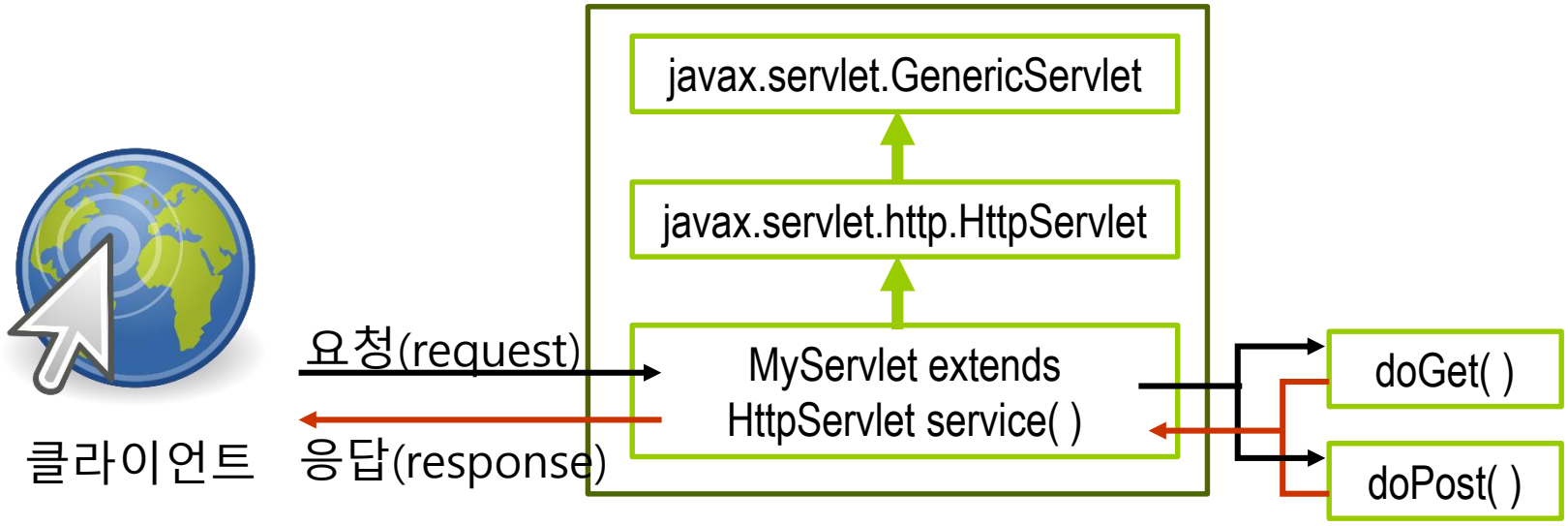
User Defined Servlet

- 사용자 정의 Servlet 클래스 상속 구조



Servlet 구조

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

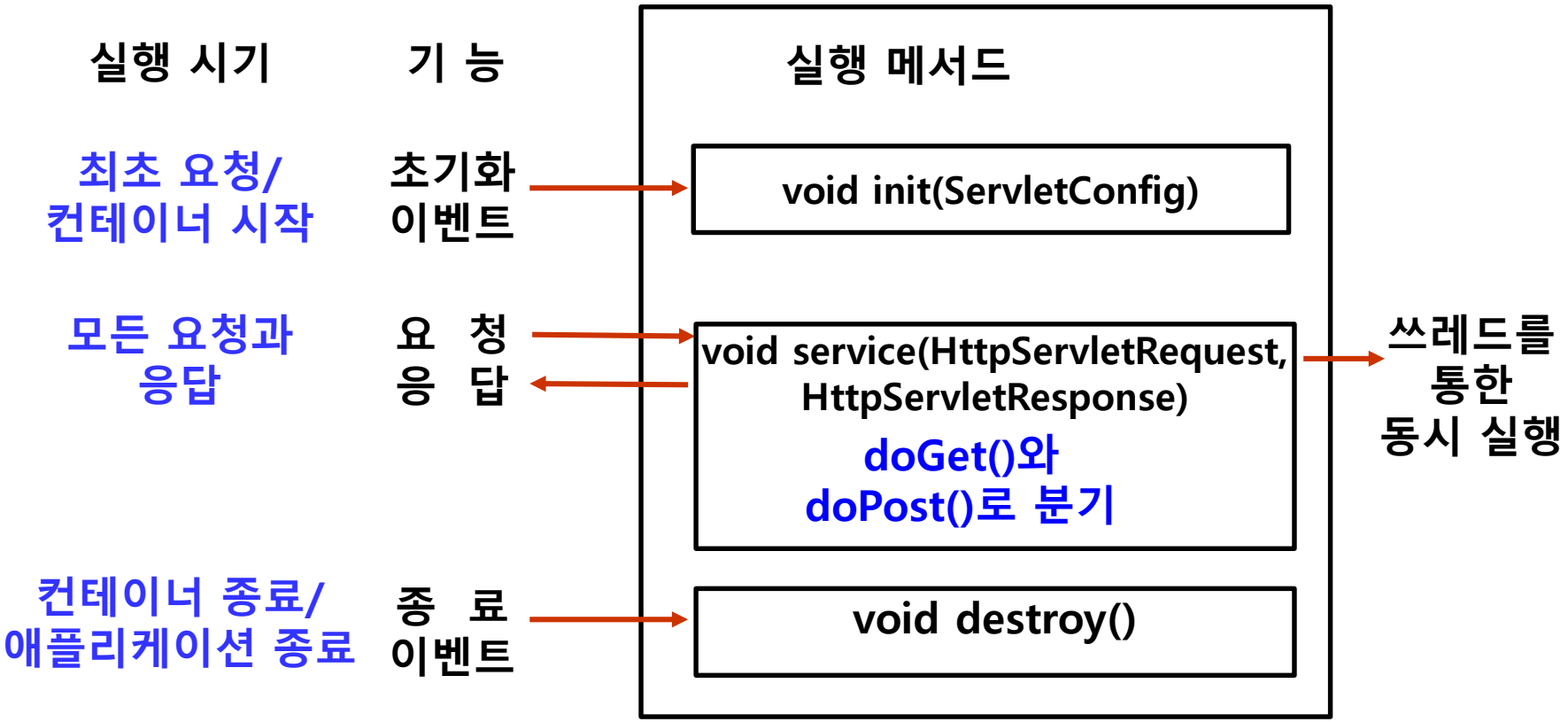


Servlet 구조

- 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
  - URL 노출로 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

HTTP Method	설명
GET	요청받은 URI의 정보를 검색하여 응답함.
HEAD	GET방식과 동일하지만, 응답에 BODY가 없고 응답코드와 HEAD만 응답함. 웹서버 정보확인, 헬스체크, 버전확인, 최종 수정일자 확인 등의 용도로 사용함
POST	요청된 자원을 생성(CREATE). 새로 작성된 리소스인 경우 HTTP헤더 항목 Location : URI주소를 포함하여 응답.
PUT	요청된 자원을 수정(UPDATE). 내용 갱신을 위주로 Location : URI를 보내지 않아도 됨. 클라이언트측은 요청된 URI를 그대로 사용하는 것으로 간주함.
PATCH	PUT과 유사하게 요청된 자원을 수정(UPDATE)할 때 사용함. PUT의 경우 자원 전체를 갱신하는 의미지만, PATCH는 해당자원의 일부를 교체하는 의미로 사용함.
DELETE	요청된 자원을 삭제할 것을 요청함. (안전성 문제로 대부분의 서버에서 비활성)
CONNECT	동적으로 터널 모드를 교환, 프락시 기능을 요청시 사용함.
TRACE	원격지 서버에 루프백 메시지 호출하기 위해 테스트용으로 사용.
OPTIONS	웹서버에서 지원되는 메소드의 종류를 확인할 경우 사용.

Servlet Lifecycle



Servlet

- 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

ServletConfig

- **javax.servlet.ServletConfig**
  - 개별 서블릿이 실행하기 위해 필요한 설정 정보를 제공
  - 개별 서블릿 객체 당 하나씩 생성 (web container에 의해 init() 메서드 실행 시 전달됨)
  - ServletConfig 객체는 Servlet이 초기화될 때 생성되며, Servlet의 init() 메서드에서 사용할 수 있음
  - ServletConfig 객체는 Servlet이 종료될 때까지 사용할 수 있음
  - 초기 파라미터 (init-param), ServletContext 객체의 주소값 등이 ServletConfig 객체에 저장됨
  - 코드와 설정 정보를 분리해서, 유지보수성을 향상시킬 수 있음 (코드 - Servlet / 설정 정보 - xml / ServletConfig로 연결)

ServletConfig

- ServletConfig 메서드

메소드	설명
getInitParameterNames()	초기 파라미터 값의 설정 이름을 Enumeration 객체로 반환
getInitParameter(name)	문자열 name에 해당하는 초기화 파라미터 값을 반환
getServletContext()	현재 애플리케이션의 ServletContext 객체를 반환 내장 객체인 application과 동일한 객체 참조
getServletName()	Servlet의 이름을 반환

HttpServletRequest

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

HttpServletRequest

- 요청 HTTP 헤더 관련 메서드
  - 웹 브라우저는 HTTP 헤더에 부가적인 정보를 담아 서버로 전송

메소드	설명
getHeader(name)	설정한 name의 헤더 값을 가져옴
getHeaders(name)	설정한 name의 헤더 목록 값을 가져옴
getHeaderNames()	모든 헤더 이름을 가져옴
getIntHeader(name)	설정한 name의 헤더 값을 정수로 가져옴
getDateHeader(name)	설정한 name의 헤더 값을 시간 값으로 가져옴
getCookies()	모든 쿠키 값을 javax.servlet.http.Cookie의 배열 형태로 가져 옴

HttpServletRequest

- 웹브라우저/서버 관련 메소드

메소드	설명
getRemoteAddress()	웹브라우저 IP 주소를 알려줌
getMethod()	현재 요청이 **GET, POST**인지 가져 옴
getSession()	현재 요청관련 세션 객체를 가져 옴. 만약 세션이 없으면 새로 만들어서 반환함.
getProtocol()	현재 서버의 프로토콜을 문자열 형태로 알려줌
getRequestURI()	웹브라우저가 요청한 URI 경로를 가져 옴
getRequestURL()	웹브라우저가 요청한 URL 경로를 가져 옴
getServletPath()	웹브라우저가 요청한 URL에서 서블릿이나 JSP 이름을 가져옴
getContextPath()	현재 요청한 context를 가리키는 URI를 가져옴
getCharacterEncoding()	웹브라우저의 문자 인코딩을 가져옴
setCharacterEncoding()	지정한 캐릭터셋으로 변환. html form에서 한글 입력 시 정상적으로 처리하려면 반드시 필요.

HttpServletRequest

- 웹브라우저/서버 관련 메소드

메소드	설명
getContentType()	웹브라우저의 콘텐츠 유형을 가져 옴
getContentLength()	웹브라우저의 요청 파라미터 길이를 가져 옴
getQueryString()	웹브라우저의 전체 요청 파라미터 문자열(?다음 URL에 할당된 문자열)을 가져옴
getParameter(name)	쿼리스트링에서 name를 이용해서 value를 얻음
getParameterValues(name)	쿼리스티링의 동일한 이름(name)으로 파라메터가 여러개 있는 경우에 배열로 values를 받음
getParameterNames()	쿼리스트링 중 name 값을 받음
setAttribute(key, value)	키(key), 값(value) 형태로 데이터 저장함. 키는 String, 값은 Object.
getRequestDispatcher()	RequestDispatcher 타입의 객체를 얻음. RequestDispatcher는 현재의 요청을 다른 서버의 자원에 전달(forward)하는 용도로 사용함.

HttpServletResponse 클래스

- **javax.servlet.http.HttpServletResponse**
  - HttpServlet 클래스의 doGet(), doPost() 메서드 호출 시 파라미터로 전달
  - 사용자 **응답**을 처리하기 위한 클래스
  - MIME Type 설정
  - HTTP 헤더 정보 설정
  - 페이지 전환

HttpServletResponse

- 응답 HTTP 헤더 관련 메서드
  - 응답 HTTP 헤더 관련 메소드는 서버가 웹 브라우저에 응답하는 정보에 헤더를 추가하는 기능을 제공
  - 헤더 정보에는 주로 서버에 대한 정보가 저장되어 있음

메소드	설명
getHeader(name)	헤더 이름 name 값을 가져옴
containsHeader(name)	헤더 이름 name이 HTTP 헤더에 포함되었는지 여부를 확인
setHeader(name, value)	헤더 이름 name에 String값 value를 설정
setIntHeader(name, value)	헤더 이름 name에 정수값 value를 설정
setDateHeader(name, date)	헤더 이름 name에 날짜/시간 date을 설정
addHeader(name, value)	헤더 이름 name에 String값 value를 추가
addIntHeader(name, value)	헤더 이름 name에 정수값 value를 추가
addDateHeader(name, date)	헤더 이름 name에 날짜/시간 date을 추가
addCookie(cookie)	쿠키를 추가

HttpServletResponse

- 응답 콘텐츠 관련 메소드
  - response 내장 객체는 웹 브라우저로 응답하기 위해 MIME 유형, 문자 인코딩, 오류 메시지, 상태 코드 등을 설정하고 가져오는 응답 콘텐츠 관련 메소드를 제공

메소드	설명
setContentType(type)	웹브라우저에 응답할 MIME 유형을 설정
getContentType()	웹브라우저에 응답할 MIME 유형을 가져옴
setCharacterEncoding(charset)	웹브라우저에 응답할 문자 인코딩을 설정
getCharacterEncoding()	웹브라우저에 응답할 문자 인코딩을 가져옴
sendError(statusCode, msg)	웹브라우저에 **응답할 오류**(코드 및 오류메시지)를 설정해서 보냄
setStatus(statusCode)	웹브라우저에 응답할 HTTP 코드를 설정

HttpServletResponse

- 페이지 이동 관련 메소드
  - 페이지 이동 = 리다이렉션(redirection)
    - 사용자가 새로운 페이지를 요청할 때와 같이 페이지를 강제로 이동하는 것
  - 서버는 웹 브라우저에 다른 페이지로 강제 이동하도록 response 내장 객체의 리다이렉션 메소드를 제공
  - 페이지 이동 시에는 문자 인코딩을 알맞게 설정해야 함

메소드	설명
sendRedirect(url)	설정한 URL 페이지로 강제 이동. 브라우저의 주소가 아예 변경되기 때문에 사용자의 '새로고침 ' 같은 요청을 미리 방지할 수 있고, 특정 작업이 완전히 끝나고 새로 시작하는 흐름을 만들 수 있음.

@WebServlet

- @WebServlet
  - 여러 서블릿을 web.xml에 설정할 경우 복잡해진다는 단점이 존재
  - 톰캣7 버전부터는 서블릿 매핑을 web.xml 외에 annotation을 이용해 서블릿 클래스에 직접 설정할 수 있는 기능이 추가됨
  - 서블릿 클래스에 annotation을 이용해서 직접 서블릿 표시를 해주면 가독성이 좋아짐
  - Annotation이 적용되는 클래스는 반드시 HttpServlet 클래스를 상속받아야 함

@ServletComponentScan

- @ServletComponentScan
  - 스프링부트 환경에서 basePackages 하위 서블릿 컴포넌트 (서블릿, 필터, 리스너)를 스캔해서 빈으로 등록함
  - 서블릿 컴포넌트는 아래와 같이 별도의 annotation이 추가되어 있어야 함
    - 서블릿: @WebServlet
    - 필터: @WebFilter
    - 리스너: @WebListner

ServletApplication

```
@ServletComponentScan // servlet auto register
@SpringBootApplication
public class SpringBootServletApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootServletApplication.class, args);
    }

}
```

HelloServlet

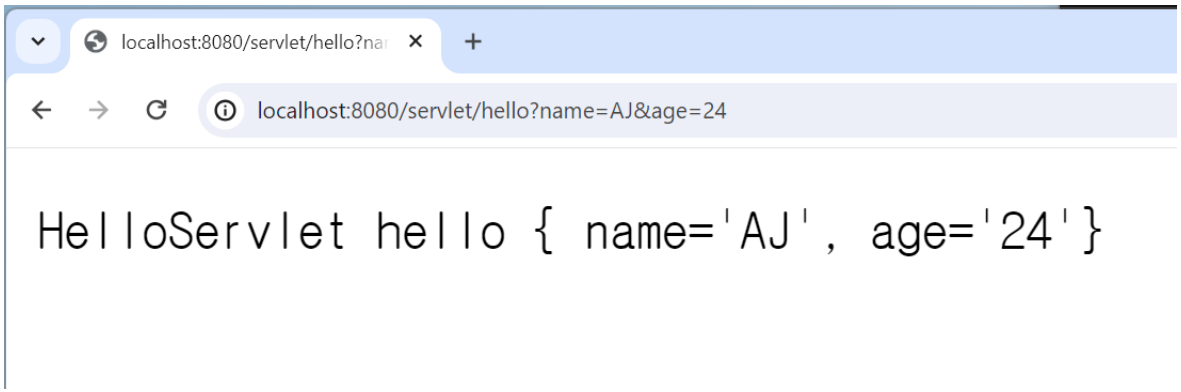
```
// using initParams & @WebInitParam annotations to set init parameter
@WebServlet(name = "helloServlet", urlPatterns = "/servlet/hello",
initParams= {@WebInitParam(name="id", value="abc"),
@WebInitParam(name="passwd", value="12345")})
public class HelloServlet extends HttpServlet {
    @Override
    public void init() {
        // load init-param using ServletConfig().getInitParamater("")
        String id = getInitParameter("id");
        String passwd = getInitParameter("passwd");
        System.out.println("HelloServlet init() id=" + id + " passwd=" +
passwd);
    }
```

HelloServlet

```
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String username = request.getParameter("name");
        int age = Integer.parseInt(request.getParameter("age"));
        Person person = new Person(username, age);
        response.setContentType("text/plain");
        response.setCharacterEncoding("utf-8");
        response.getWriter().write("HelloServlet hello " + person);
    }

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

HelloServlet & GET Method

- http://localhost:8080/servlet/**hello?name=AJ&age=24**



NewFormServlet

```
@WebServlet(name = "newFormServlet", urlPatterns = "/servlet/new-form")
public class NewFormServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");
        response.getWriter().write(
            "<form action=₩"/servlet/addperson₩" method=₩"post₩"><br/>"+
                " user name: <input type=₩"text₩" name=₩"name₩" /> <br/>" +
                " user age: <input type=₩"text₩" name=₩"age₩" /> <br/>" +
                " <button type=₩"submit₩">Submit</button>"
        );
    }
}
```

AddPersonServlet

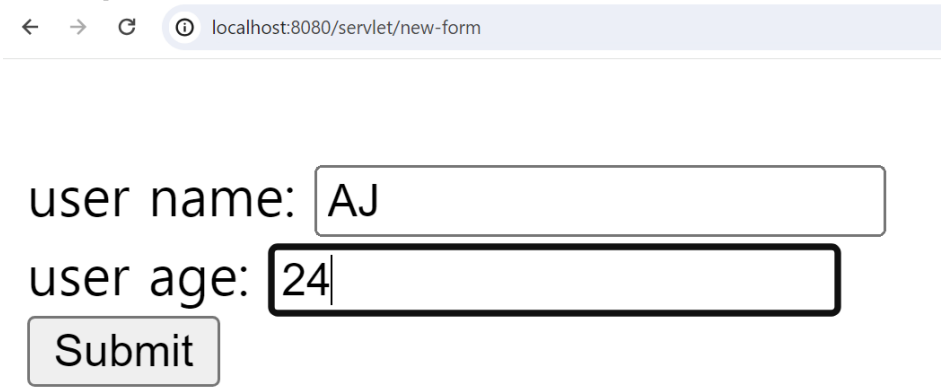
```
@WebServlet(name = "addPersonServlet", urlPatterns =
"/servlet/addperson")
public class AddPersonServlet extends HttpServlet {
    List<Person> people = null;

    public AddPersonServlet() {
        people = new ArrayList<>();
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        String username = request.getParameter("name");
        int age = Integer.parseInt(request.getParameter("age"));
        Person person = new Person(username, age);
        people.add(person);
```

AddPersonServlet

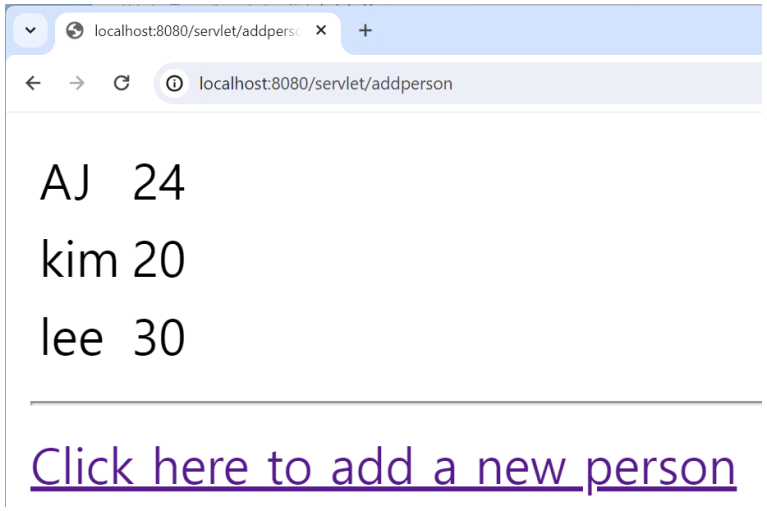
```
        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");
        response.getWriter().write("<table>");
        for (Person p: people) {
            response.getWriter().write(" <tr>");
            response.getWriter().write(" <td>" + p.getName() + "</td>");
            response.getWriter().write(" <td>" + p.getAge() + "</td>");
            response.getWriter().write(" </tr>");
        }
        response.getWriter().write("</table>");
        response.getWriter().write(
            "<hr><a href = ₩"/servlet/new-form₩">Click here to add a new
person</a>"
        );
    }
}
```

NewFormServlet & POST Method

- http://localhost:8080/servlet/**new-form**



- http://localhost:8080/servlet/**addperson**



JSP Syntax

- 주석(Comment) <%-- 주석 --%>
- 스크립트릿(Scriptlet) <% 스크립트 %>
- 선언(Declaration) <%! 선언 %>
- 표현식(Expression) <%= 표현식 %>
- 지시어(Directive)
  - <%@ page 속성1="속성값1" 속성2="속성값2"..%>
  - <%@ include file="포함할 파일" %>
  - <%@ taglib uri="태그라이브러리 위치" prefix="이름" %>
- 액션 태그(Actions)
  - <jsp:include .. /> 다른 페이지의 실행 결과를 현재 페이지에 포함
  - <jsp:forward .. /> 현재 JSP 페이지의 제어를 다른 페이지로 전달
  - <jsp:useBean .. /> 자바 빈즈(JavaBeans) 등의 다양한 기능을 제공
  - 그 외 param, setProperty, getProperty, plugin, element, attribute, body, text

JSP Syntax

- 내장 객체(Implicit Objects)
  - request, response, out, session, application, config, pageContext, page, exception
- JSP Java Beans & Form Processing
- JSP Database Access
- JSP Filters, Cookies, Session
- JSP Exception Handling
- JSP Expression Language ${표현1.표현2}
- JSTL (JSP Standard Tag Library)
  - Core tags, Formatting tags, SQL tags, XML tags, JSTL Functions
- JSP Custom tags
  - Custom tags with body in JSP, Custom tags with attributes