Servlet

Back-end Programming

김기정 (bangry313@gmail.com)

목차 (Table of Contents)

1. Web 기본 개념

1.1 HTTP 프로토콜 (1/5) – 개요

✓ Web Programming

- Web 상에서 HTTP 프로토콜을 사용하여 다양한 데이터를 송수신하는 네트워크 프로그램을 말한다
- 웹 클라이언트 프로그래밍과 웹 서버 프로그래밍으로 구분할 수 있다

✓ HTTP(HyperText Transfer Protocol)

- TCP/IP 기반 대표적인 응용 프로토콜의 하나로 웹 클라이언트의 요청과 웹 서버의 응답 데이터를 주고 받기 위한 통신 규약이다
- 다양한 텍스트 데이터(HTML, CSS, JavaScript, XML, JSON 등) 를 송수신 한다
- HTTP 프로토콜은 비 연결지향으로 웹 클라이언트가 응답을 받으면 웹 서버와 연결이 끊어진다(Connectionless)
- 이러한 특징으로 인해 웹 서버는 클라이언트의 이전 상태를 알 수 없다(Stateless)

1.1 HTTP 프로토콜 (2/5) – HTTP 메소드

- ✓ HTTP 메소드는 요청의 종류를 서버에 알려주기 위해 사용하는 것으로 여러 방식이 존재한다
- ✓ 대부분의 웹 사이트에서, 서버에 정보를 요청할 때는 GET 방식을 폼 데이터를 전송할 때는 POST 방식을 사용한다
- ✓ RESTful 웹 서비스에서는 HTTP 메소드의 목적에 맞게 사용할 것을 권장한다
- ✓ HEAD, TRACE, OPTIONS 등의 메소드는 서버의 상태를 확인하기 위해 사용한다

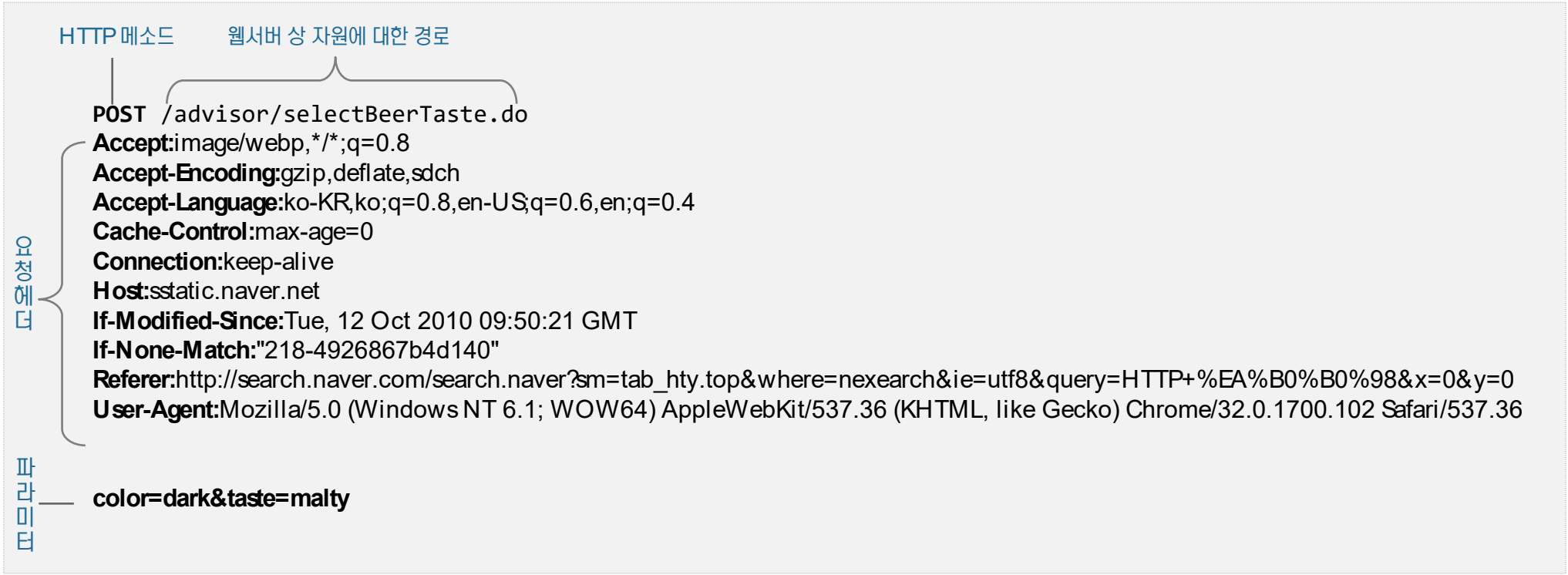
GET	URL을 이용하여 자원을 요청 (Retrieve)
POST	Request에 첨부한 Body 정보를 서버로 전송 (Create)
PUT	Body정보를 요청한 URL로 저장 (Update)
DELETE	URL에 해당하는 자원을 삭제 (Delete)
HEAD	헤더정보만 요청. 해당 자원이 존재하는지 또는 서버의 문제가 없는지를 확인하기 위해 사용
TRACE	요청을 그대로 반환. 테스트 목적으로 서버에서 무엇을 받았는지 알고 싶을 때 사용
OPTIONS	요청 URL에 응답할 수 있는 HTTP Method 종류들이 무엇인지 요청

1.1 HTTP 프로토콜 (3/5) – GET 방식

- ✓ GET 방식은 가장 단순한 HTTP 요청 방식으로 서버에게 자원을 요청할 때 사용한다
- ✓ URL 뒤에 요청 파라미터를 붙이는 방식으로 서버에 데이터를 전달한다
- ✓ 파라미터를 포함한 URL은 최대길이(2KB)가 제한되어 있으며 브라우저마다 차이가 있다

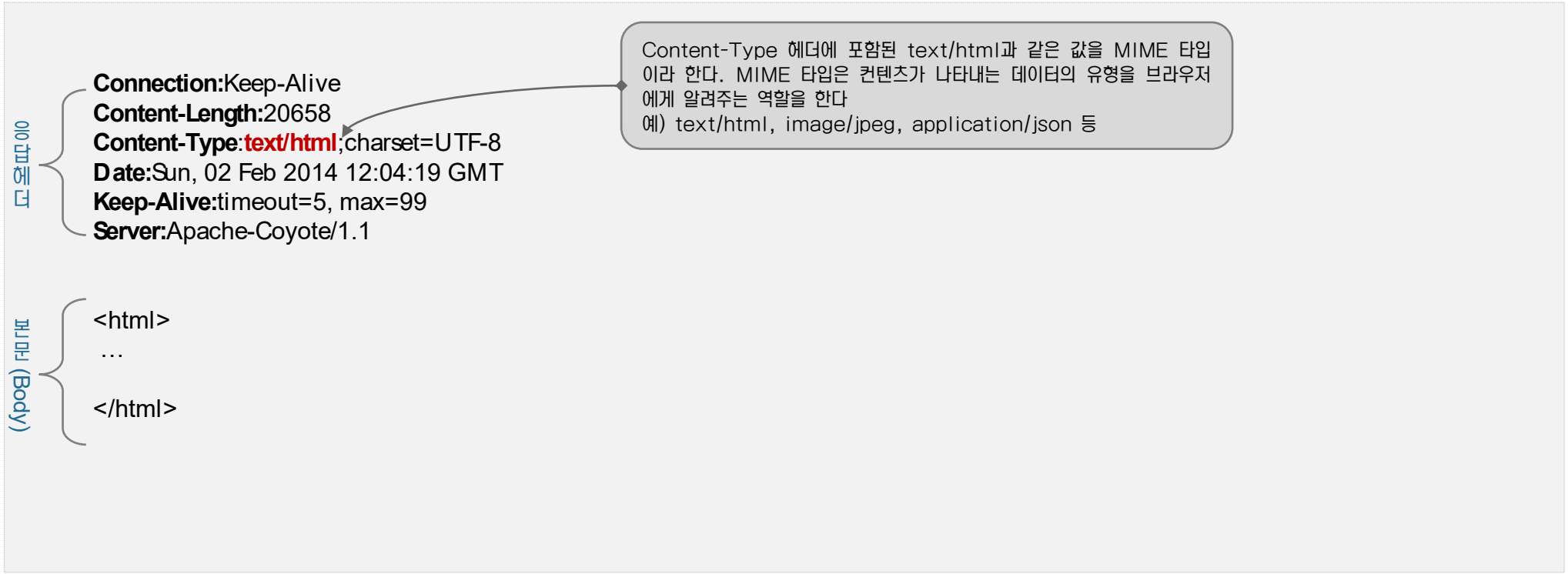
1.1 HTTP 프로토콜 (4/5) – POST 방식

- ✓ **POST** 방식은 서버에 전송할 데이터를 전달하기 위하여 사용한다
- ✓ 서버에 전달하는 데이터는 요청 본문에 포함되며 **URL**에 파라미터 값들이 노출되지 않는다
- ✓ **GET**과는 달리 전달하는 파라미터 길이의 제약이 없다
- ✓ 웹 페이지에서 서버로 파일을 업로드 하거나 폼을 전송할 때 주로 사용한다



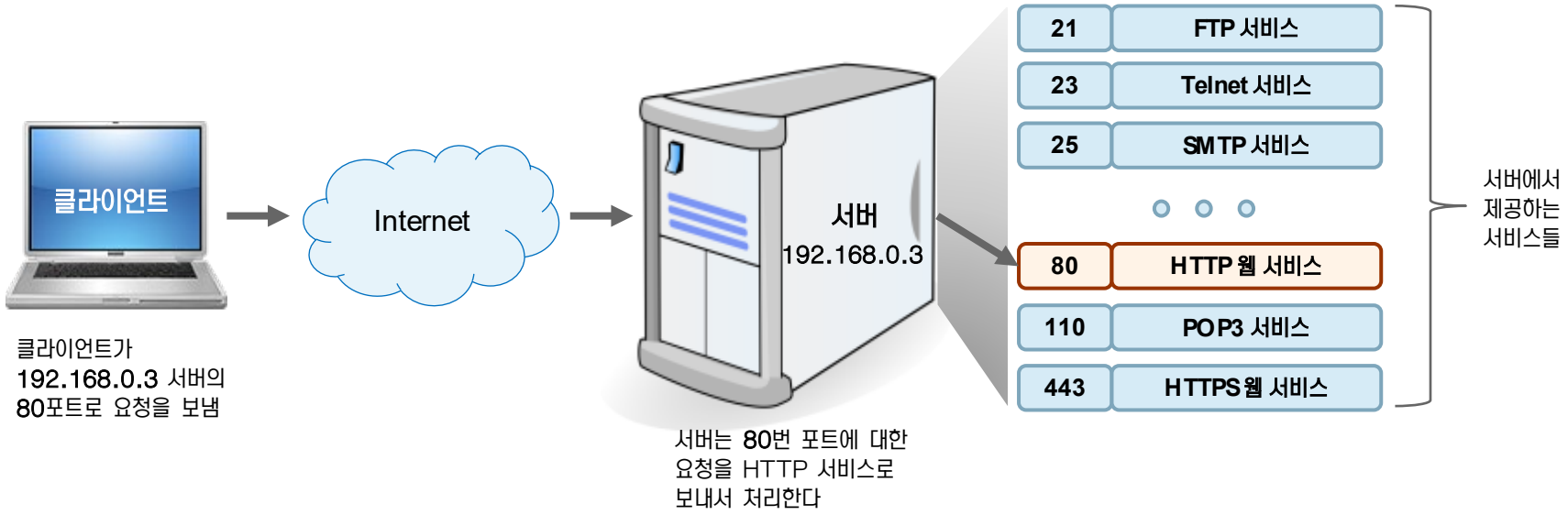
1.1 HTTP 프로토콜 (5/5) – Response

- ✓ 응답(Response)이란 웹 서버가 웹 클라이언트로 보내는 메시지를 말한다
- ✓ 응답 메시지는 헤더와 본문(Body)으로 구성된다
- ✓ 헤더에는 사용 프로토콜, 요청 성공 여부, 본문에 포함된 컨텐츠의 종류 등이 있다
- ✓ 본문은 HTML, 이미지와 같은 클라이언트에서 사용할 컨텐츠가 들어있다



1.2 TCP 포트와 서비스

- ✓ TCP 포트는 서버(하드웨어)에서 구동되는 소프트웨어(서비스)를 구별하기 위한 번호이다(0~65535)
- ✓ 포트번호는 서버(하드웨어)에서 구동되는 특정 서비스에 대한 논리적인 연결을 나타낸다
- ✓ TCP 포트번호 0 ~1024까지는 널리 알려진 서비스를 위하여 예약 되어 있는 번호이다
- ✓ 포트번호를 통해 클라이언트가 어느 애플리케이션에 접속하기를 원하는지 알 수 있다

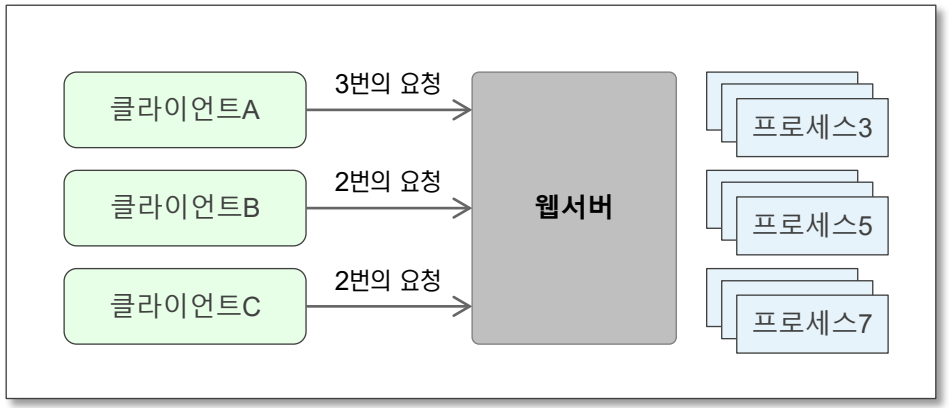


1.3 Web Server

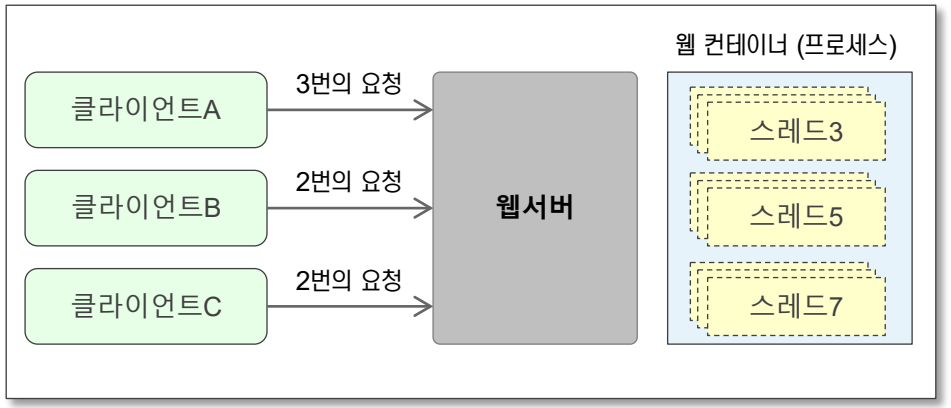
- ✓ 웹 서버는 **HTTP** 프로토콜을 통해 클라이언트가 요청한 자원을 찾아 제공하는 서버이다
- ✓ 웹 서버는 단지 정적 리소스(**HTML** 문서, 이미지 등)만 찾아서 그대로 클라이언트에게 전송한다
- ✓ 클라이언트가 요청한 자료가 서버에 없는 경우, 서버는 **404 Not Found** 응답메시지를 보낸다
- ✓ 웹 서버는 동적으로 컨텐츠를 생성할 수 없기 때문에 대안으로 **CGI**와 같은 기술이 등장하였다

1.4 CGI (Common Gateway Interfaces)

- ✓ CGI는 웹 서버에서 동적인 컨텐츠를 생성하여 클라이언트에 제공할 수 있는 인터페이스이다
- ✓ 전통적인 CGI 방식은 클라이언트 요청마다 개별 프로세스가 생성되어 시스템 부하가 많이 생기는 단점이 있었다
- ✓ 또한 플랫폼에 종속적이어서 윈도우에서 C언어 등으로 만들어진 CGI 애플리케이션은 리눅스에서 사용할 수 없었다
- ✓ 이러한 단점을 해결하기 위하여 개별 스레드가 요청을 처리하는 방식으로 발전하였다(Servlet, ASP, PHP 등)

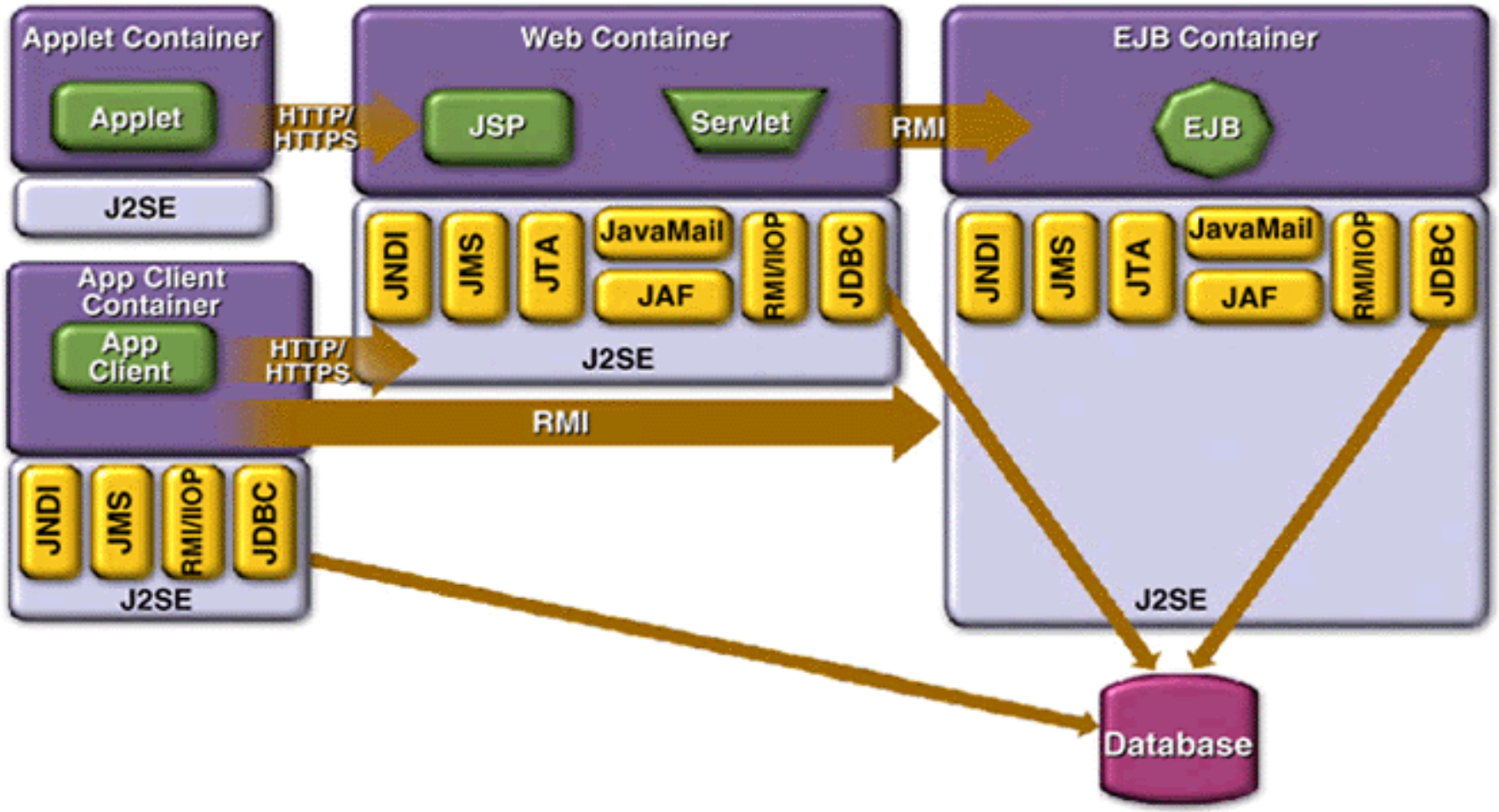


전통적인 CGI 방식



요청을 스레드로 처리하는 방식

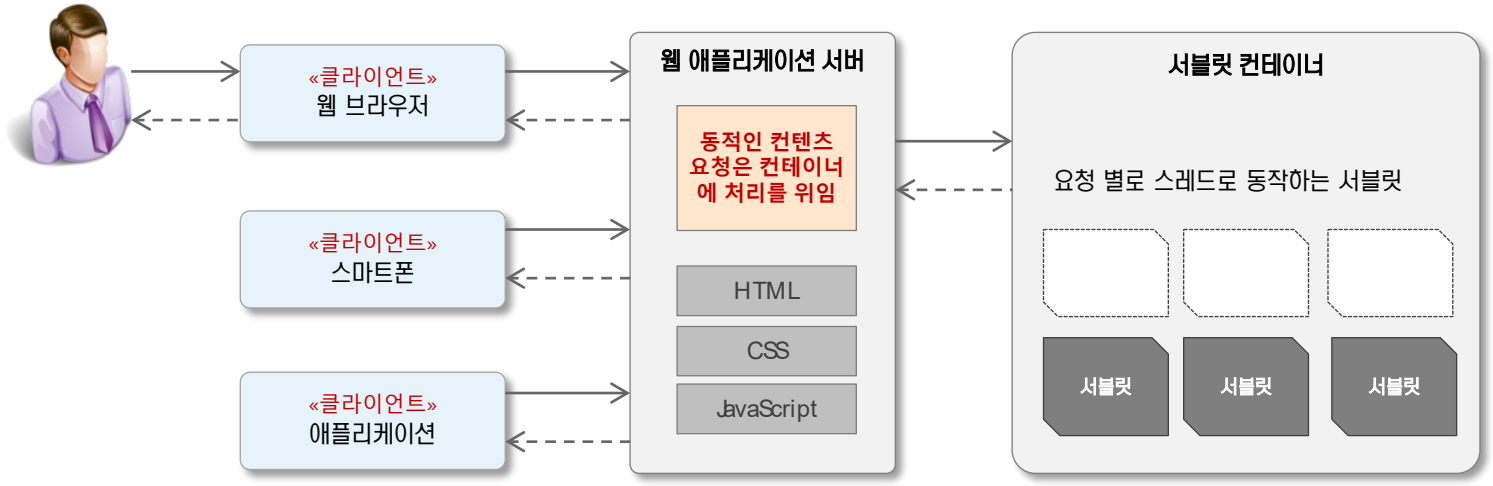
1.5 JavaEE 기반 표준 기술



2. Servlet Programming

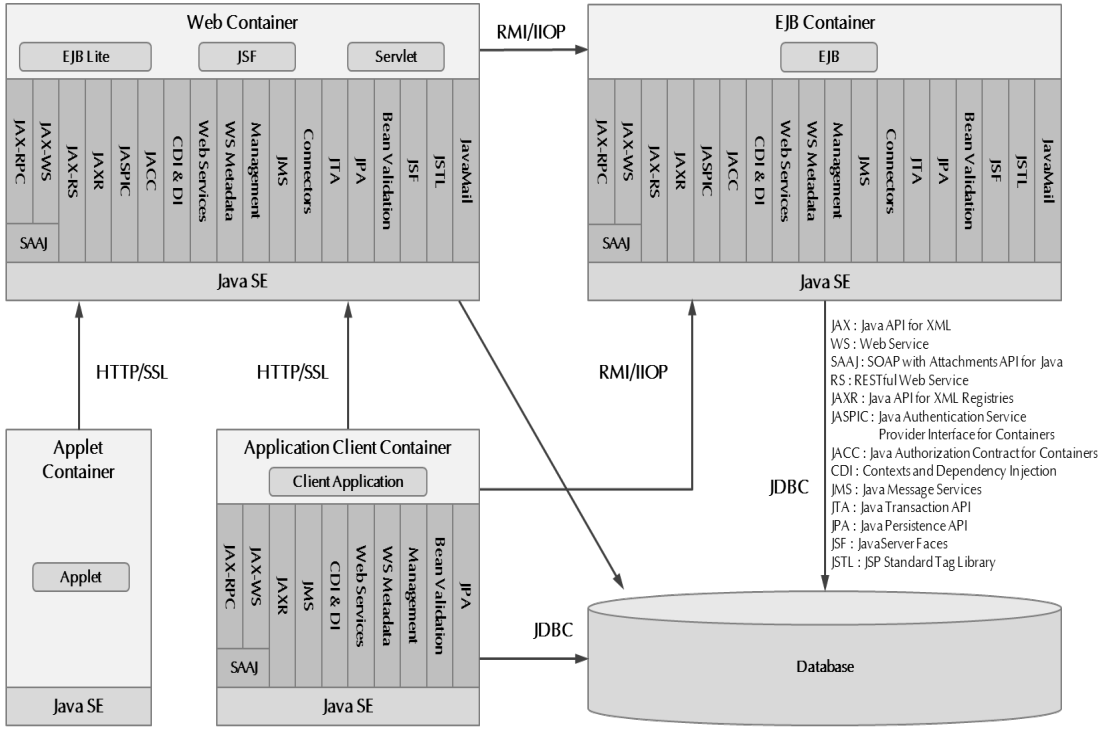
2.1 Servelt 기본 (1/6) – 개요

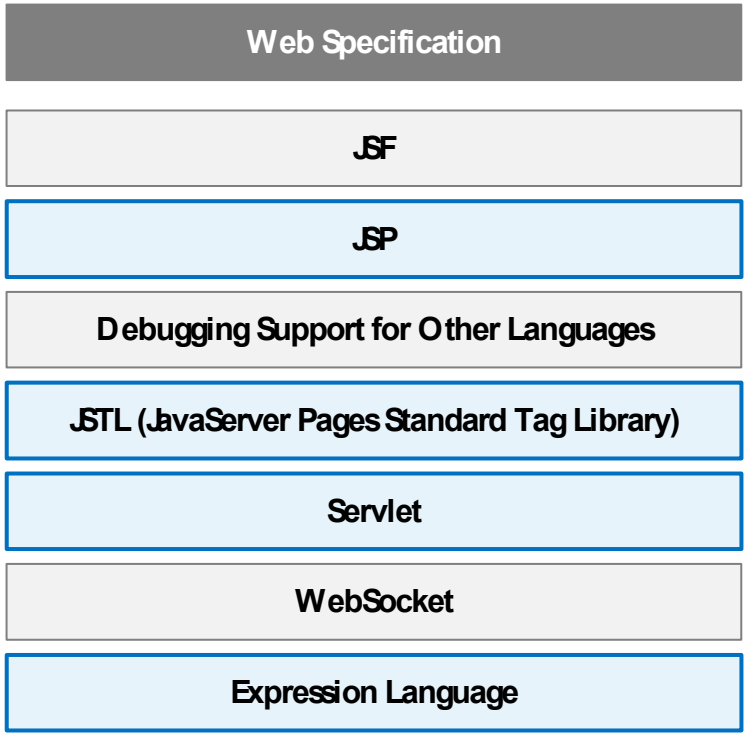
- ✓ 서블릿(Servlet)은 Java 언어를 사용하여 웹 페이지를 동적으로 생성할 수 있는 서버 측 웹 프로그램(CGI)이다
- ✓ WAS (Web Application Server)의 서블릿 컨테이너에 의해 배치되고 관리되는 JavaEE 기반 표준 웹 기술이다
- ✓ Java를 사용하므로 플랫폼에 독립적이고, 스레드 기반으로 좀 더 효율적인 멀티 태스킹을 지원한다
- ✓ Servlet API 는 서블릿 프로그램 개발에 필요한 다양한 클래스 및 인터페이스를 제공한다
- ✓ Servlet 컨테이너는 서블릿이 동작하는 실행환경으로 서블릿의 생명주기(생성, 실행, 삭제)를 관리한다



2.1 Servelt 기본 (2/6) – JavaEE 명세

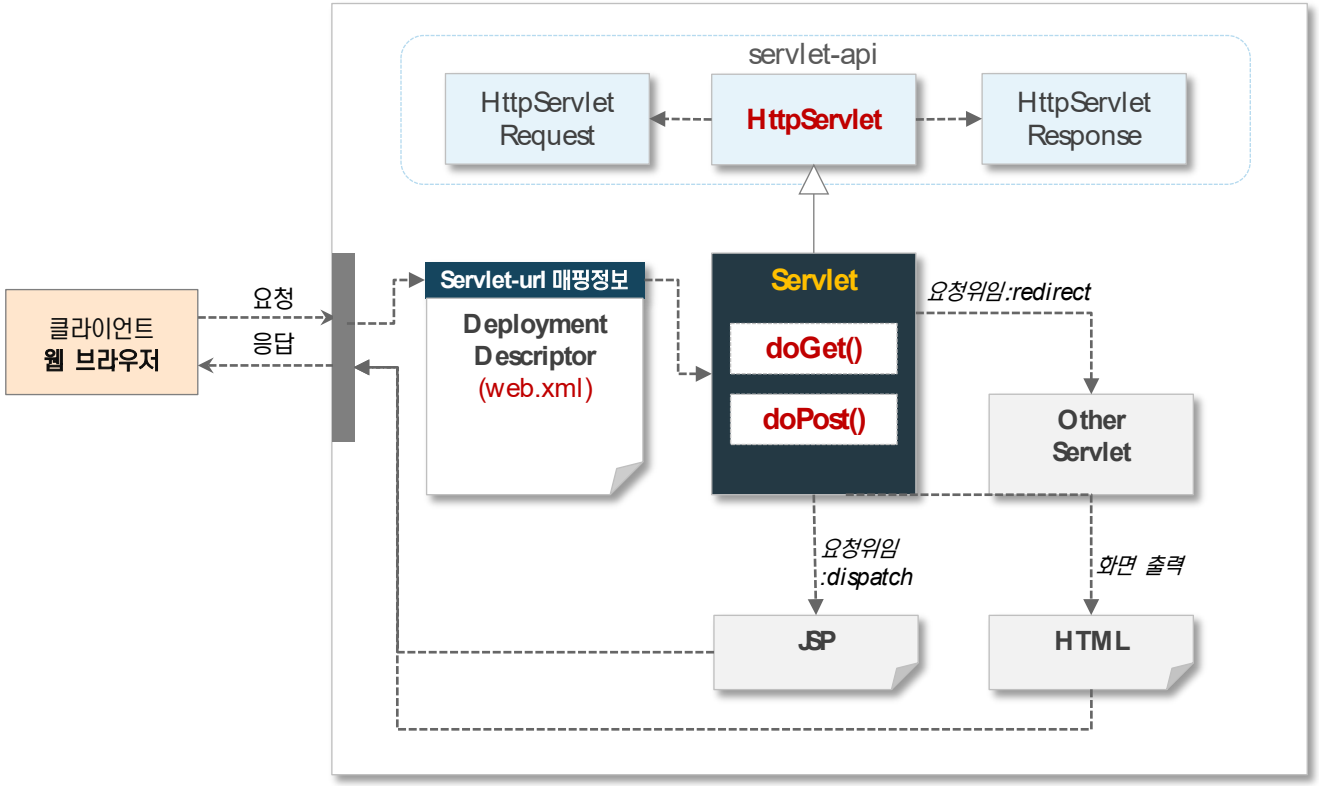
- ✓ JavaEE 스펙은 자바 엔터프라이즈 애플리케이션 개발에 필요한 기술들에 대한 표준이다
- ✓ JavaEE 스펙 중 웹 애플리케이션 개발과 관련된 기술들을 모아둔 것을 **Web Specification** 이라고 한다
- ✓ JavaEE 모든 스펙을 구현한 서버를 **WAS(Web Application Server)**라고 하며, 이 중 웹과 관련된 기술만을 처리하는 서버를 웹 컨테이너라 한다 (예: WebLogic, Websphere, JEUS, Apache Tomcat 등)
- ✓ 본 과정에서는 웹과 관련된 기술인 Servlet, JSP, JSTL, EL 등을 다룬다





2.1 Servelt 기본 (3/6) – 서블릿 프로그래밍 핵심 구조

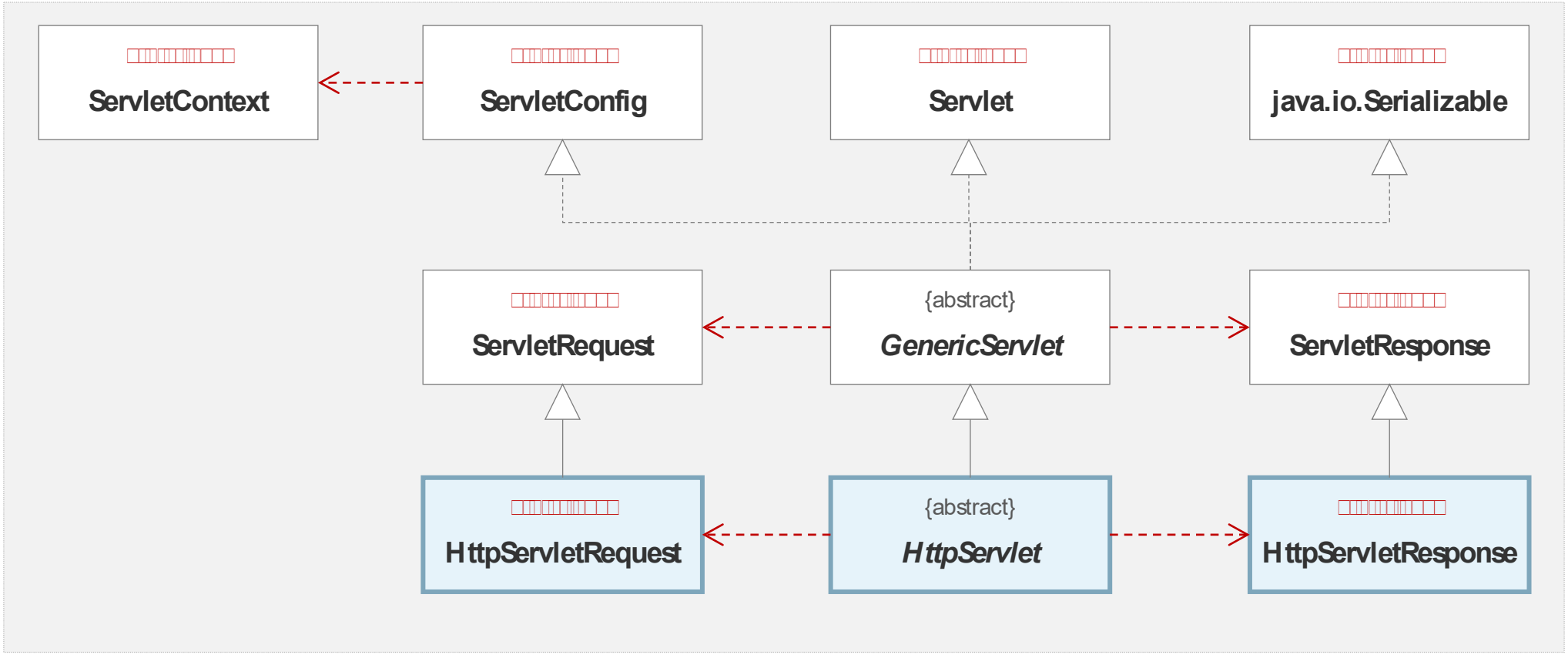
- ✓ 웹 클라이언트의 요청은 **Deployment Descriptor(배포서술자, DD)**의 **URL**에 매핑 된 서블릿으로 연결된다
- ✓ **Servlet Spec 3.0** 부터는 **@WebServlet** 어노테이션으로 매핑 가능하다
- ✓ 서블릿은 **servlet-api**의 **HttpServlet**을 상속받아 구현하고, **HTTP** 요청방식에 따른 처리를 구현할 수 있다
- ✓ 매핑된 서블릿이 요청을 직접 처리하여 출력하거나, 다른 서블릿이나 **JSP**로 요청을 위임할 수도 있다



2.1 Servelt 기본 (4/6) – 서블릿 API

✓ 서블릿 **API**는 서블릿 프로그램을 작성을 위한 인터페이스 및 클래스로 **javax.servlet** 패키지 하위에 존재한다

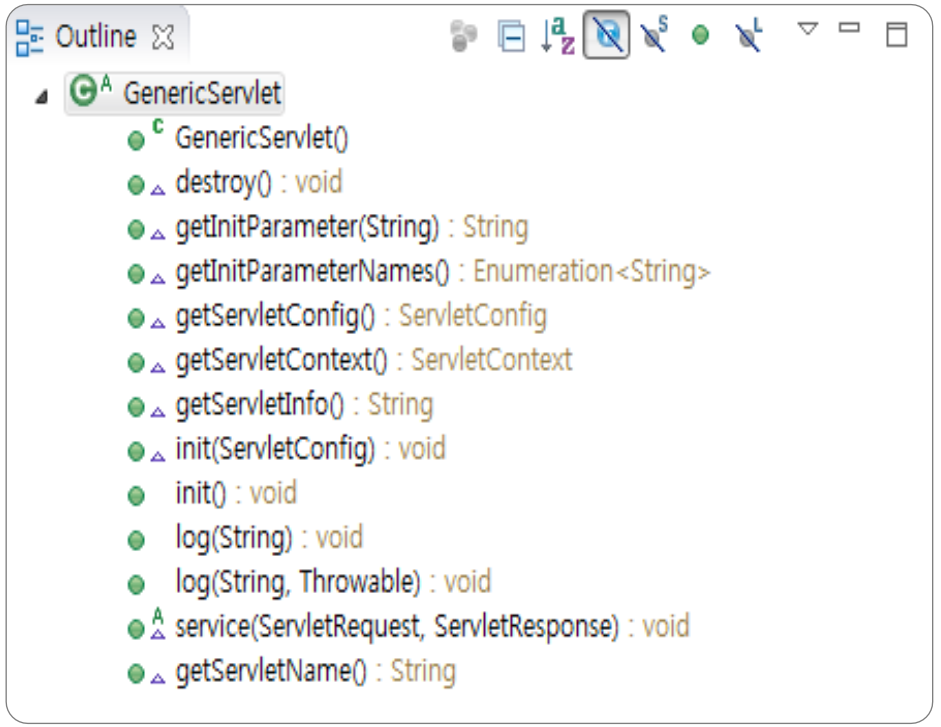
✓ 서블릿을 사용한 **Java** 웹 프로그래밍을 할 때는 거의 대부분 **HttpServlet** 을 상속하여 개발한다

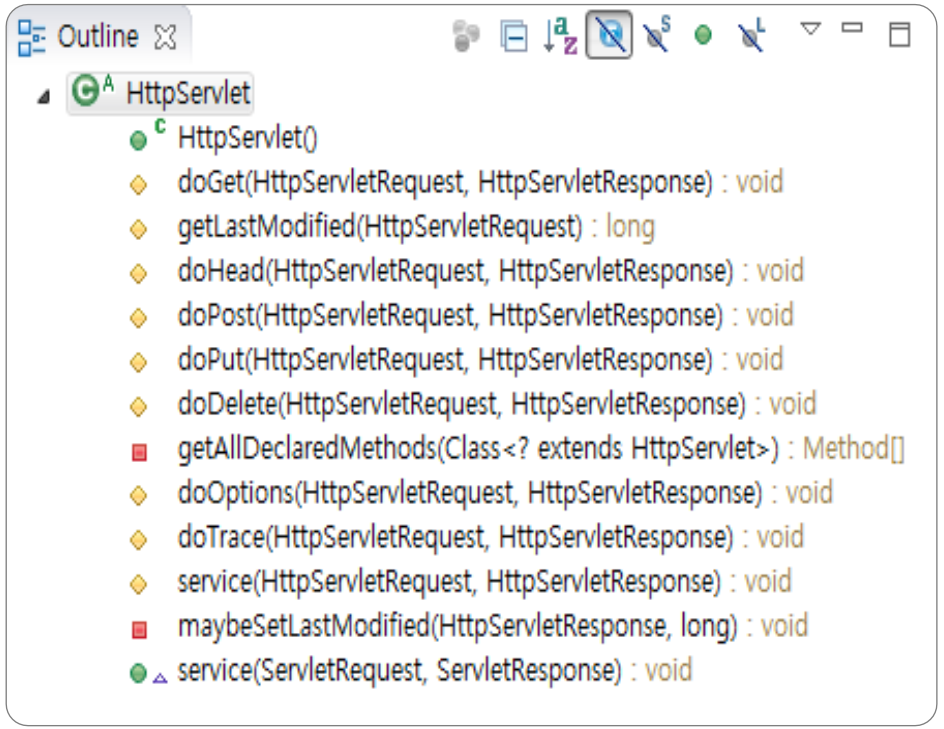


서블릿 관련 클래스 및 인터페이스간의 관계

2.1 Servelt 기본 (5/6) – HttpServlet 추상클래스

- ✓ **GenericServlet**은 서블릿 관리에 필요한 기능들이 미리 구현되어 있는 추상 클래스이다
- ✓ **HttpServlet**는 **GenericServlet**을 상속하며 **HTTP** 요청 메소드 유형별로 메소드가 정의되어 있다
- ✓ 대부분의 웹 애플리케이션에서는 **HttpServlet** 추상클래스를 상속받아 요청방식에 해당하는 메소드를 재정의한다
- ✓ **HttpServlet**의 **service()** 메소드에서는 요청방식에 따라 적합한 메소드를 호출한다(예, **GET**방식은 **doGet()** 호출)





2.1 Servelt 기본 (6/6) – 서블릿 등록

✓ 웹 애플리케이션에 서블릿을 등록하는 방법에는 두 가지가 있다

✓ 첫번째는 **web.xml** 에 **<servlet>** 과 **<servlet-mapping>** 요소로 등록하는 방법이다

✓ 두번째는 서블릿 **3.0** 부터 **@WebServlet** 어노테이션을 사용하여 서블릿을 등록할 수 있다

```
<servlet>
  <description>처음 작성하는 서블릿</description>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>namoo.servlet.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/hello.do</url-pattern>
</servlet-mapping>
```

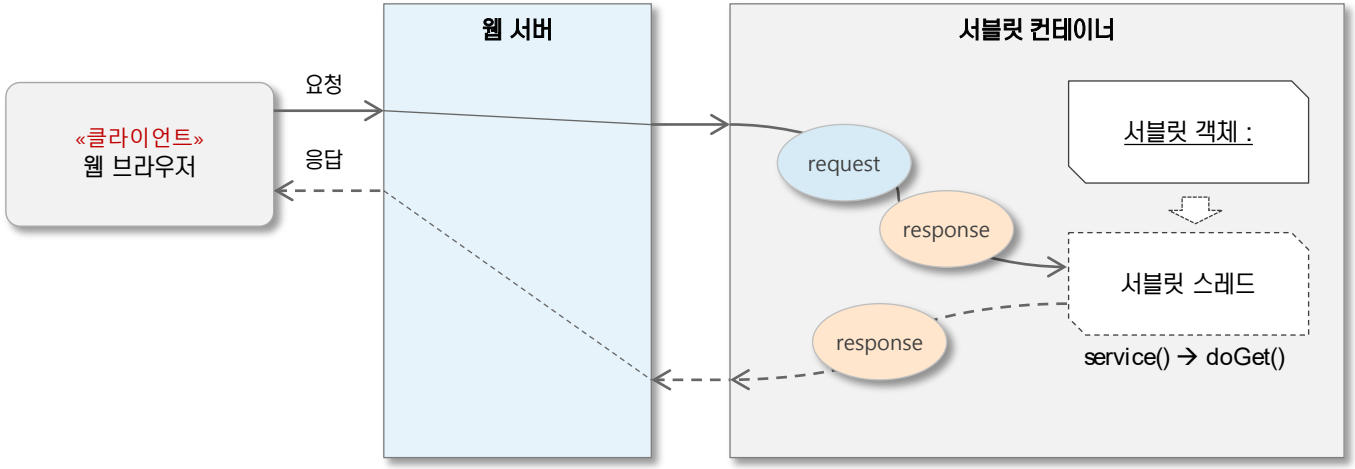
```
@WebServlet("/hello.do")
public class HelloServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        ... 중략 ...
    }
}
```

2.2 Servelt Container (1/4)

- ✓ 서블릿 컨테이너는 소켓 생성, 포트 리스닝 등과 같은 웹 서버와 서블릿의 커뮤니케이션을 지원한다
- ✓ 서블릿 컨테이너는 서블릿 객체의 생성, 초기화, 호출, 소멸과 같은 서블릿의 라이프사이클을 관리한다
- ✓ 서블릿 컨테이너는 다중 요청에 대한 스레드 생성 및 운영에 대해 관리한다
- ✓ 서블릿 컨테이너는 **XML** 설정을 통한 선언적인 보안관리를 제공한다
- ✓ 서블릿 컨테이너는 **JSP**를 지원한다

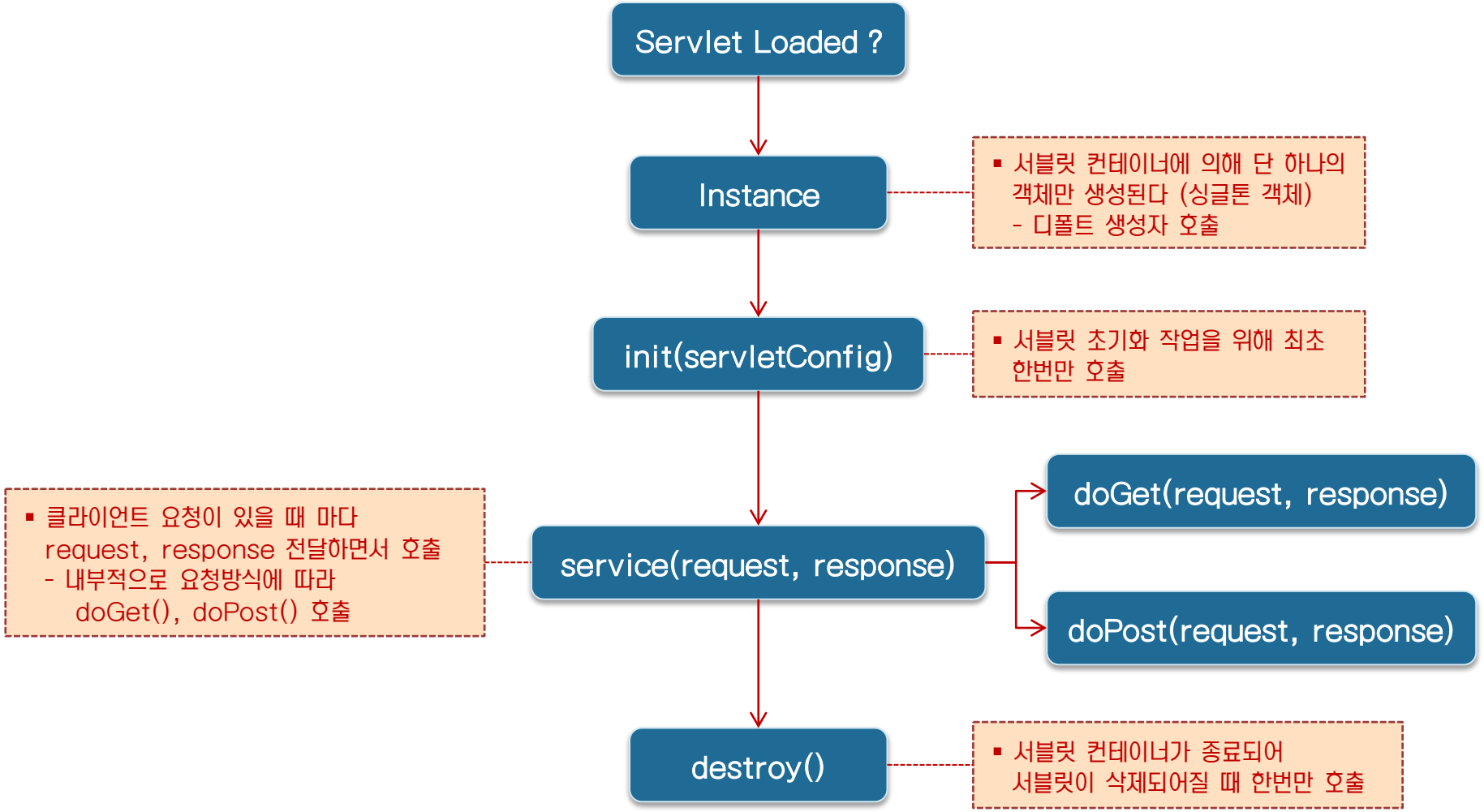
2.2 Servelt Container (2/4) – Servlet 요청 처리 과정

- ✓ 클라이언트(브라우저)가 웹 서버로 HTTP 요청을 보낸다
  - 요청 메시지 전송
- ✓ 웹 서버는 클라이언트의 요청을 받아들여 서블릿 컨테이너에게 전달한다
  - 요청 메시지 분석 및 요청 위임
- ✓ 서블릿 컨테이너는 HTTP Request, HTTP Response 객체를 생성하여 서블릿에 전달한다
  - HttpServletRequest, HttpServletResponse 생성 및 전달
- ✓ 서블릿은 요청을 처리하고 결과인 응답객체를 요청 역순으로 클라이언트(브라우저) 에게 전송한다
  - 웹 서버와의 출력스트림 생성 및 전송



2.2 Servelt Container (3/4) – Servlet 라이프사이클 (1/2)

✓ 서블릿 컨테이너는 서블릿의 라이프사이클을 관리한다



2.2 Servelt Container (3/4) – Servlet 라이프사이클 (2/2)

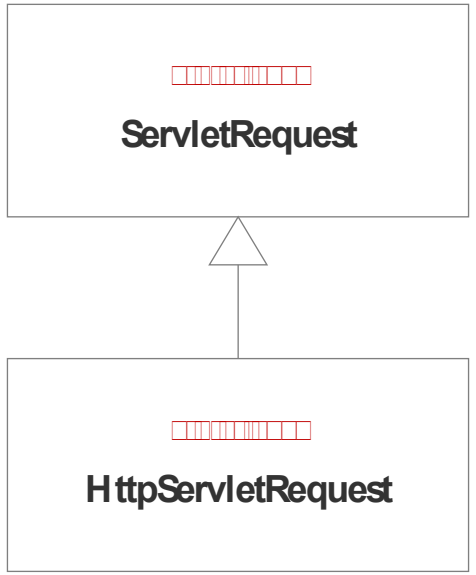
- ✓ **init()**
  - 클라이언트의 요청을 처리하기 전에 서블릿을 초기화하는 메소드
  - 컨테이너가 서블릿 인스턴스를 생성한 후 호출한다
  - 필요한 경우 재정의 가능 (예 : 데이터 베이스 접속)
- ✓ **Service()**
  - 클라이언트 요청을 받아 처리하는 메소드
  - 요청의 HTTP 메소드(GET/POST 등)를 참조하여 doXXX() 메소드를 호출한다
  - 재정의하지 않음 (상속받은 상위 클래스의 service() 메소드를 그대로 사용)
- ✓ **doGet() / doPost()**
  - 개발이 필요한 부분
  - 요청 메소드 유형에 따라 메소드를 재정의하여 구현한다
- ✓ **Destory()**
  - 요청처리가 끝나면 컨테이너가 호출하는 메소드
  - 재정의하지 않는다

2.3 요청과 응답 (1/5) – HttpServletRequest

✓ **HttpServletRequest** 객체는 클라이언트의 요청정보를 관리한다

✓ **HttpServletRequest** 인터페이스의 메소드

- getHeader() : 요청 정보의 헤더정보를 반환
- getMethod() : 요청 정보의 HTTP Method 정보 반환
- getParameter() : 파라미터명으로 요청 값 반환
- getParameterValues() : 파라미터명으로 요청 값 배열 반환
- setCharacterEncoding() : 요청정보의 인코딩 설정

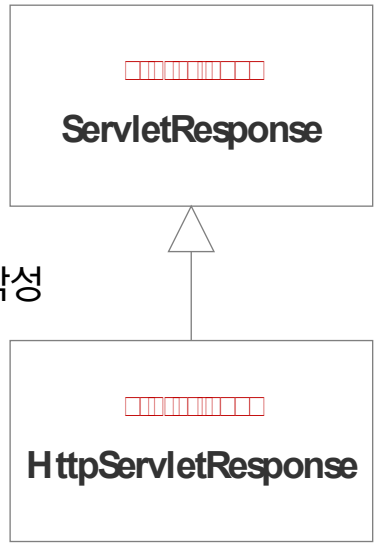


2.3 요청과 응답 (2/5) – HttpServletResponse

✓ **HttpServletResponse 객체는 클라이언트에게 제공하는 응답정보를 관리한다**

✓ **HttpServletResponse 인터페이스의 메소드**

- setContentType() : 클라이언트로 보내는 컨텐츠 타입 설정
- getWriter() : PrintWriter 객체반환, 클라이언트에게 전달할 텍스트 데이터 작성
- getOutputStream() : ServletOutputStream 객체반환, 클라이언트에게 전달할 바이너리 데이터 작성
- setStatus() : 응답에 대한 상태코드 지정



2.3 요청과 응답 (3/5) – ContentType (1/2)

✓ **ContentType**은 웹 클라이언트에게 전송하는 데이터가 무엇인지를 미리 알려주는 응답헤더이다

✓ 웹 클라이언트는 응답헤더의 **ContentType**을 보고 반환된 컨텐츠를 적절한 형태로 렌더링한다

✓ 출력 스트림에 데이터를 기록하기 전에 **setContentType()** 메소드를 호출하여 **ContentType**을 설정해야 한다

✓ **MIME TYPE**

- Multipurpose Internet Mail Extension (다목적 인터넷 메일 확장 규약)
- 인터넷 메일을 통해 문자코드로 구성된 텍스트 파일 뿐만 아니라 멀티미디어 파일도 주고 받을 수 있도록 인터넷 메일 표준 규약을 확장한 규약이다
- 현재는 메일 뿐만 아니라 HTTP 통신에서 파일 시스템 내에 존재하는 파일을 구분하기 위해 유용하게 사용되고 있다
- 웹 서버가 웹 클라이언트에 전송하는 콘텐츠 형식을 구분하기 위해 응답 헤더의 Content-type에 MIME 타입 및 문자 인코딩 설정
- 예) Content-type: text/html; charset=utf-8
- MIME 타입에는 8가지 형식이 존재한다
  - ‘application’, ‘audio’, ‘image’, ‘message’, ‘model’, multipart', ‘text’, ‘video’

2.3 요청과 응답 (4/5) – ContentType (2/2)

✓ 주요 MIME 타입 및 SUB 타입

MIM TYPE	File Extension	MIM TYPE	File Extension
application/msword	doc	audio/mpeg	mp3
application/vnd.ms-excel	xls	image/gif	gif
application/vnd.ms-powerpoint	ppt	image/jpeg	jpeg, jpg
application/octet-stream	bin	text/css	css
application/pdf	pdf	text/html	html, htm
application/x-zip	zip	text/plain	txt
application/jar	jar	text/xml	xml
application/java	java	video/mpeg	mpeg, mpg
audio/x-wav	wav	video/x-msvideo	avi

- http://www.iana.org/assignments/media-types/media-types.xhtml

2.4 Servlet API (1/16) – 개요 (1/2)

✓ **서블릿 컨테이너에 관리되는 서블릿의 개발과 실행을 가능하게 하는 인터페이스와 클래스들의 집합**

- javax.servlet 패키지 - 모든 프로토콜에 사용 가능한 서블릿 작성을 위한 인터페이스와 클래스로 구성
- javax.servlet.http 패키지 - HTTP 프로토콜에 특화된 서블릿 작성을 위한 인터페이스와 클래스로 구성

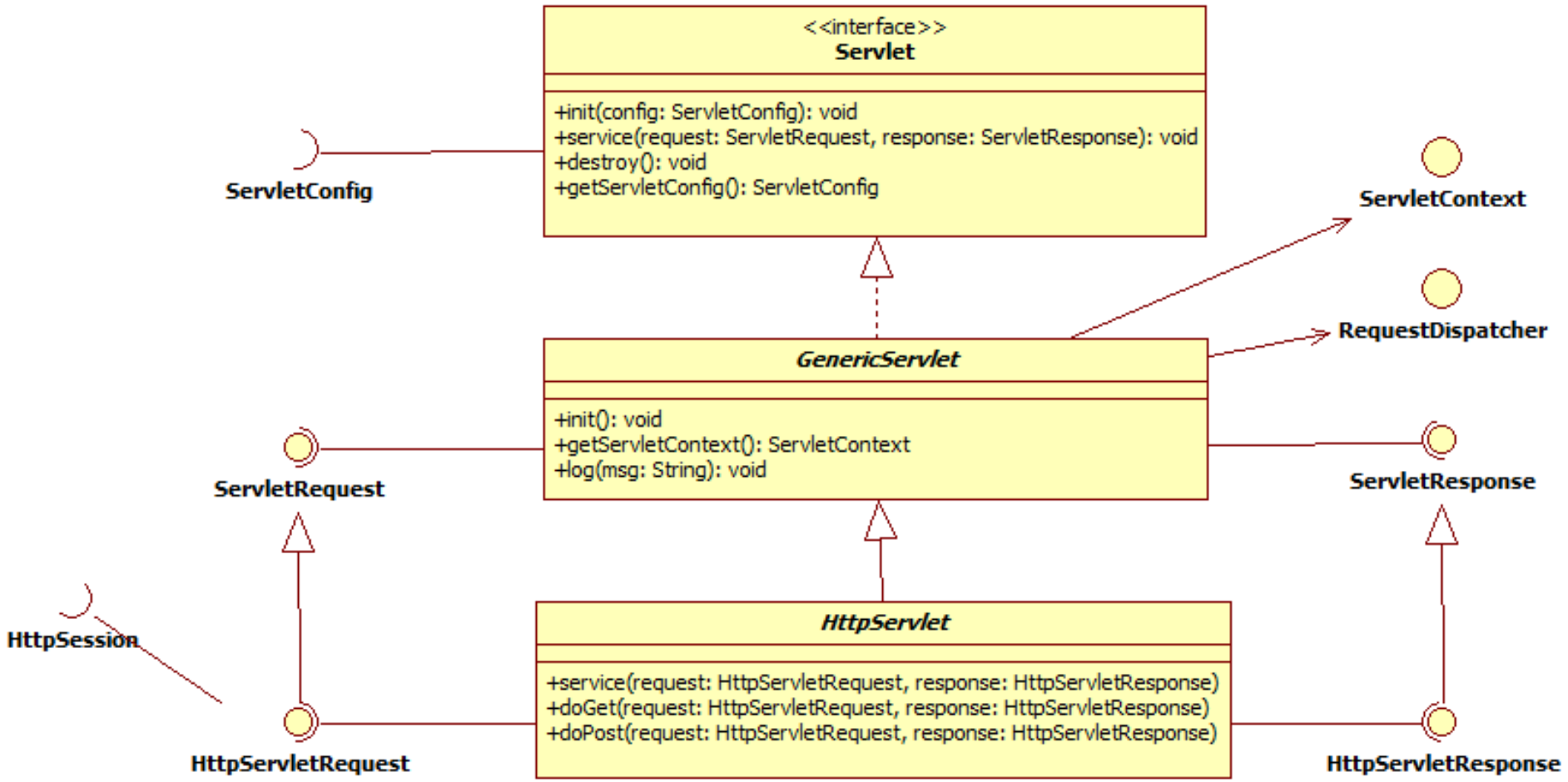
✓ **모든 서블릿은 javax.servlet.Servlet 인터페이스를 구현하여야 하며, 일반적으로 Servlet 인터페이스를 구현한 javax.servlet.http.HttpServlet을 사용한다**

- HttpServlet은 HTTP 요청 방식에 따른 doGet(), doPost() 메소드를 제공한다
- 요청 메시지 정보를 담은 HttpServletRequest 객체와 응답 메시지 저장을 위한 HttpServletResponse 객체를 전달받는다

✓ **기타 웹 관련 다양한 서비스 제공을 위해 인터페이스와 클래스를 제공한다**

- ServletConfig, HttpSession, ServletContext, RequestDispatcher, Cookie 등

2.4 Servlet API (2/16) – 개요 (2/2)



2.4 Servlet API (3/16) – HttpServletRequest (1/3)

✓ ServletRequest & HttpServletRequest

- 서블릿 컨테이너는 서블릿에 클라이언트 요청 메시지 전달을 위해 HttpServletRequest 객체를 생성하여 전달한다
- 웹 클라이언트 요청 메시지가 적절히 파싱되어 HttpServletRequest 객체에 설정되므로 개발자는 getter 메소드를 이용하여 원하는 요청 정보를 추출할 수 있다

✓ HttpServletRequest 주요 메소드

- getServletContext() : ServletContext
- getRemoteHost() : String
- getRemoteAddr() : String
- getProtocol() : String
- getMethod() : String
- getRequestURL() : StringBuffer
- getRequestURI() : String
- getContextPath() : String
- getQueryString() : String
- getHeader(name: String) : String
- getHeaderNames() : Enumeration
- getContentType() : String
- getContentLength() : int

2.4 Servlet API (4/16) – HttpServletRequest (2/3)

✓ **웹 클라이언트 FORM 데이터 처리**

- 웹 클라이언트는 HTML FORM 태그를 이용하여 사용자 입력 정보를 서블릿에 전달한다

✓ **FORM 태그의 3가지 속성**

- METHOD
  - GET : 디폴트 요청방식으로 200바이트 이하 데이터를 URL 쿼리스트링을 통해 정보 전달(예: /someServlet?id=bangry)
  - POST : 응답메시지의 바디에 Data Stream 형태로 보내진다. 보안 처리 및 많은 양의 데이터 전송 시 사용
  - HEAD, PUT, DELET, TRACE, OPTIONS : HTML에 지원하지 않음
- ACTION
  - URL 절대경로와 상대경로를 이용하여 FORM 태그의 정보를 전달받을 서블릿 설정
  - 생략 시 현재 웹 페이지 URL
- ENCTYPE
  - 데이터의 인코딩 방식을 설정하며 요청방식이 POST 방식일 경우만 사용 가능
  - 생략 시 application/x-www-form-urlencoded(예 : id=bangry)
  - 파일 업로드 처리시 multipart/form-data
- 사용자가 입력한 정보 전달을 위한 서브 태그
  - INPUT, SELECT, TEXTAREA 등

2.4 Servlet API (5/16) – HttpServletRequest (3/3)

✓ **HttpServletRequest의 메소드를 이용한 FORM 파라메터 수신**

- String parameterValue = request.getParameter(“parameterName”);
- String[] parameterValues = request.getParameterValues(“parameterName”);
- Enumeration parameterNames = request.getParameterNames();

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
     // 한글 인코딩 처리
     request.setCharacterEncoding(“utf-8”);

     String id = request.getParameter(“id”);
     String[] hobbys = request.getParameterValues(“hobby”);
     Enumeration<String> paramNames = request.getParameterNames();
     while(paramNames.hasMoreElements()){
          String paramName = paramNames.nextElement();
          String paramValue = request.getParameter(paramName);
     }
}
```

2.4 Servlet API (6/16) – HttpServletResponse

✓ ServletResponse & HttpServletResponse

- 서블릿 컨테이너는 응답 메시지 처리를 위해 HttpServletResponse 객체를 생성하여 전달한다
- 개발자는 setter 메소드를 이용하여 클라이언트에 전송할 응답 메시지를 가공하면 된다

✓ HttpServletResponse 주요 메소드

- setContentType() : void
- getWriter() : PrintWriter
- getOutputStream() : ServletOutputStream
- setStatus(statusCode:int) : void
  - 상태코드 상수 : SC_OK(200), SC_MOVED_PERMANENTLY(301), SC_BAD_REQUEST(400), SC_FORBIDDEN(403), SC_NOT_FOUND(404), INTERNAL_SERVER_ERROR(500), SC_SERVICE_UNAVAILABLE(503)
- setHeader(headerName: String, headerValue: String) : void
- sendRedirect(url: String) : void
  - 웹 브라우저 자동 요청 처리

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
     // response.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
     // response.setHeader("Location", "/someURL");
     response.sendRedirect("/someURL");
     // HTML META 태그와 동일 기능
     // <meta http-equiv="refresh" content="0; URL=/someURL">
}
```

2.4 Servlet API (7/16) – ServletConfig

✓ 서블릿 컨테이너 설정 파일(/WEB-INF/web.xml)에 등록된 서블릿 설정 정보를 제공한다

✓ **ServletConfig** 주요 메소드

- getInitParameter(parameterName: String): String
- getInitParameterNames(): Enumeration

```
<servlet>
     <servlet-name>SomeServlet</servlet-name>
     <servlet-class>xxx.yyy.SomeServlet</servlet-class>
     <init-param>
          <param-name>paramName</param-name>
          <param-value>paramValue</param-value>
     </init-param>
</servlet>
```

```
public void init(ServletConfig config) throws ServletException {
     // 서블릿 초기화 시 한번만
     config.getInitParameter("paramName");
}
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
     // 클라이언트 요청 시 마다
     // ServletConfig config = getServletConfig();
     // config.getInitParameter("paramName");
     // getInitParameter("paramName");
}
```

2.4 Servlet API (8/16) – ServletContext

✓ 서블릿 컨테이너 환경 정보를 제공하며, 컨테이너에 의해 관리되는 서블릿들 간의 데이터 공유를 위해 제공된다

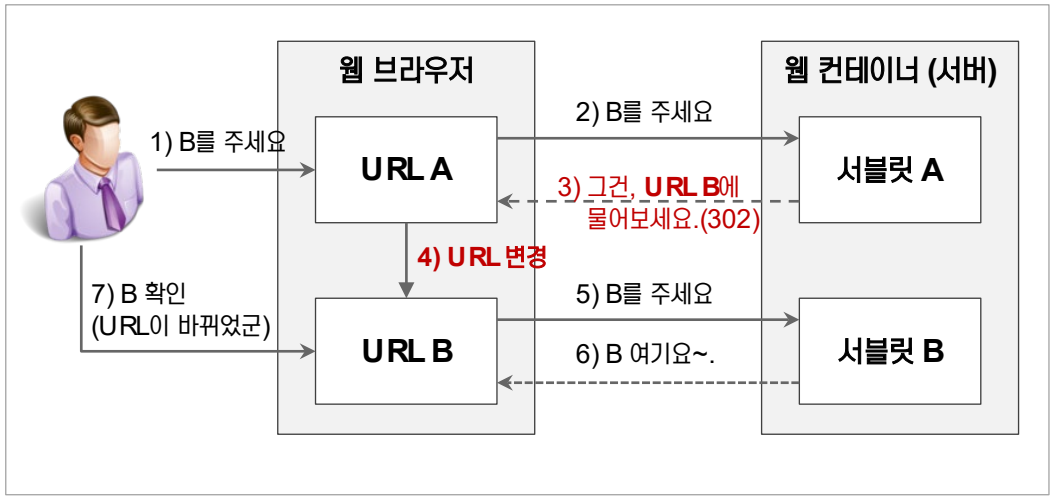
✓ **ServletContext** 주요 메소드

- getContextPath() : String
- getServletContextName() : String
- getServlet(name: String): String
- getInitParameter(paramName: String): String
- getInitParameterName(): Enumeration
- getRequestDispatcher(url: String): RequestDispatcher
- getAttribute(attName: String): String
- setAttribute(attName: String, attValue: String): void
- removeAttribute(attName: String): void

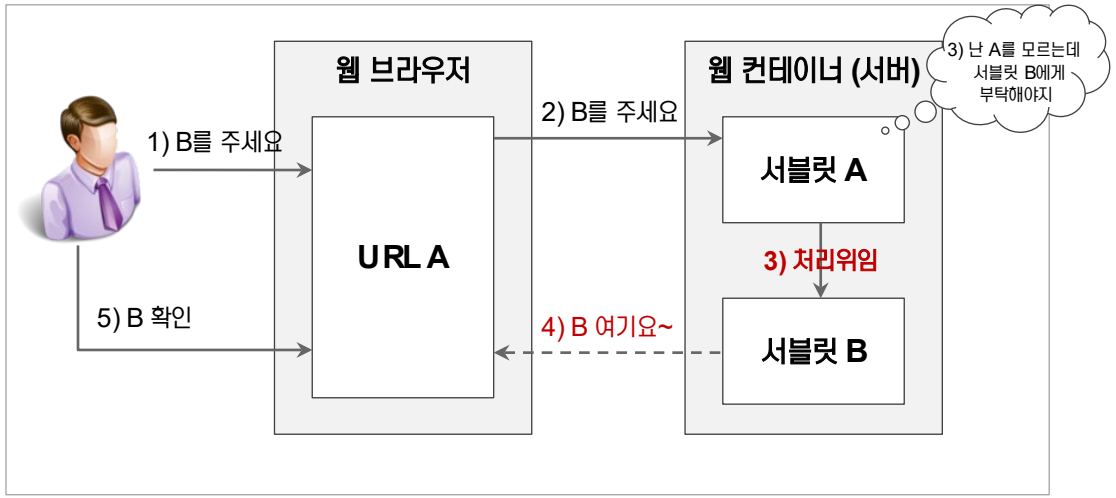
```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
     ServletContext context = getServletContext();
}
```

2.4 Servlet API (9/16) – Request를 위임하는 두 가지 방식

- ✓ 클라이언트 요청 위임이란 요청을 받은 서블릿이 다른 **URL**에 요청을 위임하는 것을 말한다
- ✓ 요청의 위임처리가 서버에서 일어나는지, 클라이언트(브라우저)에서 일어나는지에 따라 두 가지 방식이 있다
- ✓ **Redirect** 방식은 클라이언트 즉, 브라우저에서 위임할 **URL**로 다시 요청하는 방식이다
- ✓ **Request Dispatch** 방식은 서버 내부에서 다른 **URL**로 요청처리가 일어난다



Redirect 방식



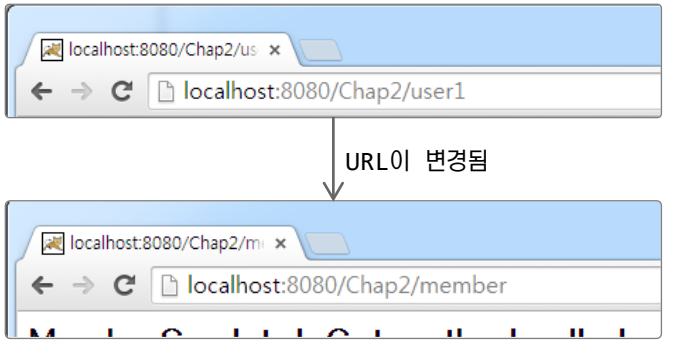
Request Dispatch 방식

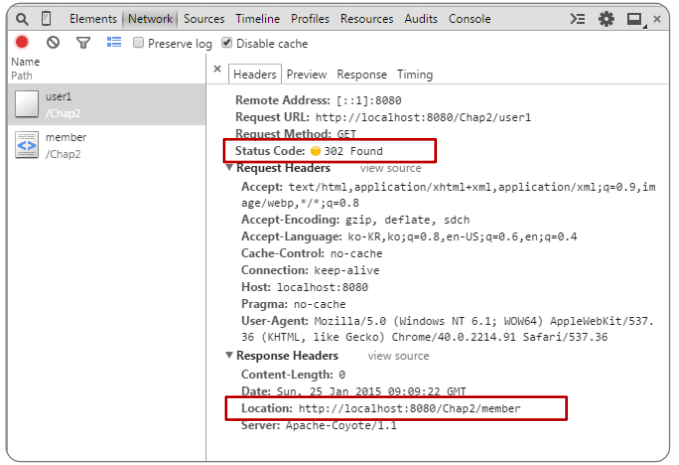
2.4 Servlet API (10/16) – Redirect 방식

- ✓ 요청을 받은 서블릿은 다른 URL에서 처리해야 할 대상인 경우 Response객체의 sendRedirect()를 호출한다
- ✓ sendRedirect() 메소드는 HTTP 응답 메시지에 상태코드 302와 Location 헤더에 다른 URL을 설정한다
- ✓ 브라우저는 응답을 받은 후 상태코드가 302 임을 확인하고 Location 헤더에 설정된 URL로 다시 요청한다
- ✓ 사용자도 브라우저에 변경된 URL을 확인할 수 있다

```java
@WebServlet("/user1")
public class RedirectUserServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

        res.sendRedirect("member");
    }
}
```

```java
@WebServlet("/member")
public class MemberServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

        PrintWriter writer = res.getWriter();
        writer.append("<HTML><BODY>");
        writer.append("<H2>MemberServlet doGet method called.</H2>");
        writer.append("</HTML></BODY>");
    }
}
```





2.4 Servlet API (11/16) – RequestDispatch 방식 (1/2)

- ✓ **RequestDispatch** 방식은 요청을 다른 방향으로 위임하는 작업이 서버 내에서 일어난다
- ✓ 요청을 받은 서블릿은 해당 요청이 다른 컴포넌트에서 처리해야 할 대상이라면 **dispatch forward**를 호출한다
- ✓ 브라우저의 **URL**은 바뀌지 않으므로 사용자는 **Dispatch** 된 응답인지를 알 수 없다
- ✓ 클라이언트의 요청을 받은 서블릿이 **View**를 구성하기 위해 **JSP**로 요청을 위임할 때 가장 많이 사용된다

```java
@WebServlet("/user2")
public class DispatchUserServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

        RequestDispatcher dispatcher = req.getRequestDispatcher("member");
        dispatcher.forward(req, resp);
    }
}
```

```java
@WebServlet("/member")
public class MemberServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

        PrintWriter writer = res.getWriter();
        writer.append("<HTML><BODY>");
        writer.append("<H2>MemberServlet doGet method called.</H2>");
        writer.append("</HTML></BODY>");
    }
}
```

2.4 Servlet API (12/16) – RequestDispatch 방식 (2/2)

- ✓ **RequestDispatcher**는 클라이언트 요청을 컨테이너에 의해 관리되는 다른 자원(**Servlet, JSP, HTML** 등)으로 포워드(**forward**) 시키거나, 자원의 실행 결과를 현재 **Servlet**으로 포함(**include**)시키고자 할 때 사용된다
- ✓ **RequestDispatcher** 주요 메소드
  - forward(request: HttpServletRequest, response: HttpServletResponse): void
  - include(request: HttpServletRequest, response: HttpServletResponse): void

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
     // 디스패치 대상에게 데이터 전달을 위해
     // request.setAttribute(attName: String, attValue: Object);
     // request.getAttribute(attName: String) : Object
     // request.removeAttribute(attName: String): void

     getServletContext().getRequestDispatcher("/someServlet").forward(request, response);
     // request.getRequestDispatcher("/someServlet").forward(request, response);

     // getServletContext().getRequestDispatcher("/someServlet").include(request,
response);
     // request.getRequestDispatcher("/someServlet").include(request, response);
}
```

2.4 Servlet API (13/16) – 클라이언트 상태 정보 유지 (HttpSession) (1/4)

✓ **서블릿은 HTTP 프로토콜을 사용하기 때문에 클라이언트와 서버와의 연결 관계가 지속적이지 않다(Connectionless)**

- 동일한 사용자가 요청을 여러 번 하더라도 서버는 request가 같은 사용자가 보낸 것인지 알 수 없기 때문에 클라이언트 상태 정보를 유지해야 하는 웹 애플리케이션 개발 시 많은 어려움이 따른다

✓ **클라이언트 상태 정보 유지 방법**

- 세션(HttpSession) & 쿠키(Cookie)

✓ **HttpSession**

- 웹 컨테이너에 개별 클라이언트 상태 정보 저장을 위해 제공된다.

✓ **HttpSession 주요 메소드**

- setAttribute(attName: String, attValue: Object): void, getAttribue(attName: String) : Object
- removeAttribute(attName: String) : void
- isNew(): boolean, invalidate(): void 등

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
      // 클라이언트에 해당하는 HttpSession 객체 존재 시 HttpSession 객체 반환하고,
      // 존재하지 않을 경우 새로운 HttpSession 생성하여 반환
      // HttpSession session = request.getSession(true);
      HttpSession session = request.getSession();

      // 클라이언트에 해당하는 HttpSession 객체 존재 시 HttpSession 객체 반환하고,
      // 존재하지 않을 경우 null 반환
      // HttpSession session = request.getSession(false);
}
```

2.4 Servlet API (14/16) – 클라이언트 상태 정보 유지 (Cookie) (2/4)

- ✓ 클라이언트의 상태 정보를 클라이언트의 메모리에 '일정한 형식의 텍스트 데이터'로 저장하고, HTTP 요청 시 요청 메시지 헤더에 쿠키를 포함시켜 전송한다
- ✓ 쿠키에는 쿠키 이름과 값, 효력을 가지는 도메인과 패스, 유효시간이 포함된다
- ✓ 클라이언트마다 최대 300개의 쿠키를 생성할 수 있으며, 쿠키 별로 4KB까지 저장할 수 있다

- ✓ 응답 메시지 헤더를 통해 웹 클라이언트로 전송하는 쿠키 구조
  - Set-Cookie : name=value; expires=date; path=path; domain=domain

Set-Cookie	설 명
name=value	쿠키 이름과 값
expires=date	쿠키가 삭제되는 날짜, 생략 시 현재 브라우저의 세션 동안에만 유효
path=path	쿠키가 유효하게 사용될 수 있는 URL 패스, 생략 시 쿠키를 설정한 자원의 패스
domain=domain_name	쿠키가 유효하게 사용될 수 있는 URL 도메인, 생략 시 쿠키를 설정한 도메인

- ✓ 요청 메시지 헤더를 통해 웹 서버로 전송하는 쿠키 구조
  - Cookie : name1=value1; name2=value2

2.4 Servlet API (15/16) – 클라이언트 상태 정보 유지 (Cookie) (3/4)

✓ 쿠키 생성 및 응답 헤더에 쿠키 설정

```
Cookie idCookie = new Cookie("loginId", "bangry");
// 유효기간 설정(초단위)
// idCookie.setMaxAge(500);
// 유효 도메인 설정
// idCookie.setDomain("www.some.co.kr");
// 유효 패스 설정
// idCookie.setPath("/");

// 응답 헤더에 쿠키 설정
response.addCookie(idCookie);
```

✓ 요청 헤더의 쿠키 정보 읽기

```
Cookie[] cookies = request.getCookies();
if(cookies != null){
     for(Cookie cookie : cookies){
          String cookieName = cookie.getName();
          String cookieValue = cookie.getValue();
     }
}
```

2.4 Servlet API (16/16) – 클라이언트 상태 정보 유지 (Cookie) (4/4)

✓ 쿠키 삭제

```
Cookie[] cookies = request.getCookies();
if(cookies != null){
     for(Cookie cookie : cookies){
          String cookieName = cookie.getName();
          if(cookieName.equals(“loginId”){
               // 유효기간 설정
               cookie.setMaxAge(0);
               // 응답헤더에 쿠키 설정
               response.addCookie(cookie);
               break;
          }
     }
}
```

2.5 파일 업로드 (1/6)

✓ 파일 업로드를 위한 HTML FORM 작성

- METHOD
  - GET : 생략 시 디폴트 요청방식으로 200바이트 이하 데이터를 URL 쿼리스트링을 통해 정보 전달한다
  - POST : 응답메시지의 본문에 파일 데이터를 포함하여 전달한다 (파일 업로드 시 사용)
- ACTION
  - URL 절대경로 또는 상대경로를 이용하여 파일 데이터를 전달받을 서블릿을 지정한다
- ENCTYPE
  - 데이터의 인코딩 방식을 설정하며 요청방식이 POST 방식일 경우만 사용 가능하다
  - application/x-www-form-urlencoded : 디폴트 값
  - multipart/form-data : 파일 업로드 시 사용

```
<form action="servlet-name" method="post" enctype="multipart/form-data">
     <input type="file" name="upfile1" />
     <input type="file" name="upfile2" />
     <input type="submit" value="파일업로드">
</form>
```

2.5 파일 업로드 (2/6)

✓ **multipart/form-data 처리를 위한 Servlet API를 지원하지 않는다**

✓ **Apache 재단에서 무료로 배포하는 파일 업로드 API 활용**

- 다운로드 : https://commons.apache.org
  - commons-fileupload-1.4.x.jar
  - commons-io-2.11.x.jar
- 웹 애플리케이션 루트 디렉터리/WEB-INF/lib 디렉터리에 복사

2.5 파일 업로드 (3/6)

✓ multipart/form-data 로 업로드 된 데이터 구조 확인하기

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    request.setCharacterEncoding("utf-8");
    String writer = request.getParameter("writer");
    System.out.println("Writer : " + writer);
    String file = request.getParameter("upfile1");
    System.out.println("File : " + file);
    // 서블릿 API를 이용하여 업로드 파일 데이터 읽기
    InputStream in = request.getInputStream();
    byte[] buffer = new byte[1024];
    int count = 0;
    while((count=in.read(buffer)) != -1){
        String data = new String(buffer, 0, count);
        System.out.println(data);
    }
    in.close();
}
```

2.5 파일 업로드 (4/6)

✓ File Upload API 활용

```
public class FileUploadServlet extends HttpServlet {
    //String fileStorage = “d:/…/…”;
    private String fileStorage;
    private int limitFileSize = 2 * 1024 * 1024;

    @Override
    public void init() throws ServletException {
        fileStorage = getServletContext().getInitParameter("fileStorage");
        String size = getServletContext().getInitParameter(“limitFileSize”);
        if( size != null){
            limitFileSize = Integer.parseInt(size);
        }
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
```

2.5 파일 업로드 (5/6)

```
// FileUpload API 활용
File storageDir = new File(fileStorage);
DiskFileItemFactory fileItemFactory = new DiskFileItemFactory();
// 파일 저장 위치
fileItemFactory.setRepository(storageDir);
// 파일 사이즈 제한
fileItemFactory.setSizeThreshold(limitFileSize);
ServletFileUpload fileUpload = new ServletFileUpload(fileItemFactory );
try {
   List<FileItem> items = fileUpload.parseRequest(request);
   for (FileItem item : items) {
      if (item.isFormField()) {
         System.out.println("파라메터 이름 : " + item.getFieldName());
         System.out.println("파라메터 값 : " + item.getString("utf-8"));
      }else {
         System.out.println("파라메터 이름 : " + item.getFieldName());
         System.out.println("파일명 : " + item.getName());
         System.out.println("파일사이즈 : " + item.getSize());
         if(item.getSize() > 0){
            String separator = File.separator;
            int index =  item.getName().lastIndexOf(separator);
            String fileName = item.getName().substring(index  + 1);
            File uploadFile = new File(fileStorage +  File.separator + fileName);
            item.write(uploadFile);
         }
      }
   }
```

2.5 파일 업로드 (6/6)

```
            out.println("<html>");
            out.println("<body>");
            out.println("<h2>파일 업로드 완료!</h2>");
            out.println("</body>");
            out.println("</html>");
            //response.sendRedirect("/파일목록처리 서블릿");
        } catch (Exception e) {
            e.printStackTrace();
            new ServletException(e.getMessage());
        }
    }
}
```

```
<context-param>
    <param-name>fileStorage</param-name>
    <param-value>d:/xxx/yyy/zzz/fileStorage</param-value>
</context-param>
<context-param >
    <param-name>limitFileSize</param-name>
    <param-value>5242880</param-value>
</context-param>
<servlet>
    <servlet-name>FileUploadServlet</servlet-name>
    <servlet-class>namoo.servlet.FileUploadServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>FileUploadServlet</servlet-name>
    <url-pattern>/upload.do</url-pattern>
</servlet-mapping>
```

2.6 파일 다운로드 (1/4)

✓ **HTML 링크 태그 사용**

✓ **<a href="다운로드 파일명" >**

- 웹 브라우저가 처리할 수 있는 Content-type의 경우 직접 렌더링(HTML, XML, GIF, JPG, PNG 등)
- 웹 서버 디렉토리 구조가 노출되는 보안상의 문제 발생

✓ **Servlet에서 응답헤더 설정을 통한 다운로드 구현**

- 웹 서버 디렉터리 구조 노출되지 않음

2.6 파일 다운로드 (2/4)

✓ FileDownloadServlet.java

```
public class FileDownloadServlet extends HttpServlet {

    private String fileStorage;

    @Override
    public void init() throws ServletException {
        fileStorage = getServletContext().getInitParameter("fileStorage");
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException{
        process(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException{
        process(request, response);
    }

public void process(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
        String fileName = request.getParameter("file");
        if (fileName == null || fileName.equals(""))
            return;
        String filePath = fileStorage + File.separator + fileName;
        File file = new File(filePath);
```

2.6 파일 다운로드 (3/4)

```
        // HTTP 버전별 브라우저 캐시 사용 않도록 응답헤더 설정
        String httpVersion = request.getProtocol();
        if (httpVersion.equals("HTTP/1.0")) {
            response.setDateHeader("Expires", 0);
            response.setHeader("Pragma", "no-cache");
        } else if (httpVersion.equals("HTTP/1.1")) {
            response.setHeader("Cache-Control", "no-cache");
        }
        // 파일 다운로드 처리를 위한 응답헤더에 마임타입 설정
        response.setContentType("application/octet-stream");
        fileName = URLEncoder.encode(fileName, "utf-8");
        response.setHeader("Content-Disposition", "attachment;filename=" + fileName + ";");
        response.setHeader("Content-Length", file.length());
        FileInputStream in = new FileInputStream(file);
        OutputStream out = response.getOutputStream();
        try{
            byte[] buffer = new byte[1024];
            int count = 0;
            while ((count = in.read(buffer)) != -1) {
                out.write(buffer, 0, count);
            }
        }finally{
            if(out != null) out.close();
            if(in != null)  in.close();
        }
    }
}
```

2.6 파일 다운로드 (4/4)

```
<servlet>
    <servlet-name>FileDownloadServlet</servlet-name>
    <servlet-class>namoo.servlet.FileDownloadServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>FileDownloadServlet</servlet-name>
    <url-pattern>/download.do</url-pattern>
</servlet-mapping>
```

End of Document

✓ **Q&A**

감사합니다...