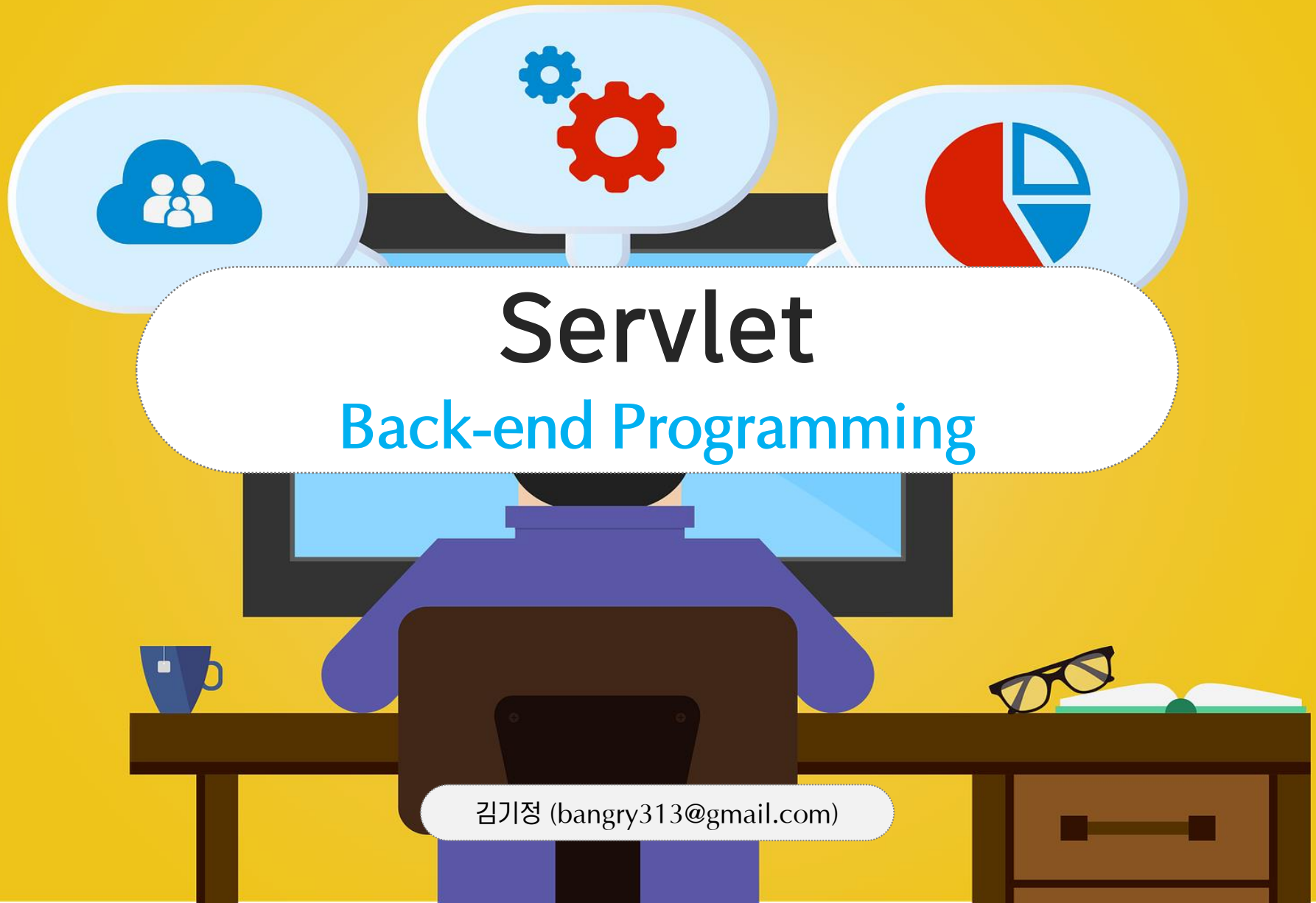


Servlet

Back-end Programming

김기정 (bangry313@gmail.com)

목차 (Table of Contents)

1. Web 기본 개념

1.1 HTTP 프로토콜
1.2 TCP 포트와 서비스
1.3 Web Server
1.4 CGI (Common Gateway Interface)
1.5 JavaEE 기반 표준 기술

1.1 HTTP 프로토콜 (1/5) – 개요

✓ Web Programming

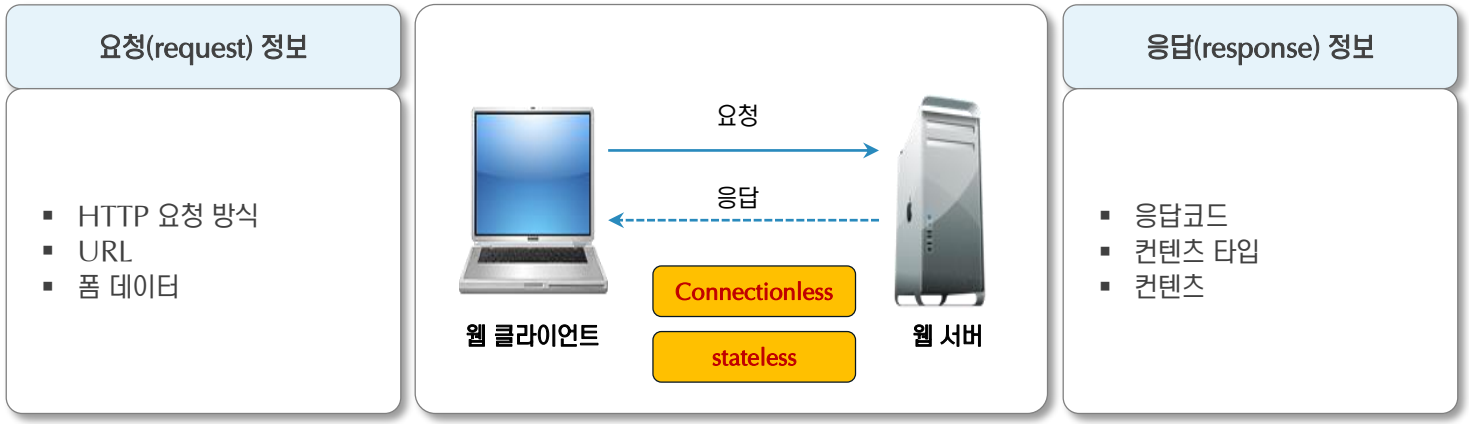
- Web 상에서 HTTP 응용 프로토콜을 사용하여 다양한 데이터를 송수신하는 네트워크 프로그램을 말한다.
- 웹 클라이언트(Front-end) 프로그래밍과 웹 서버(Back-end) 프로그래밍으로 구분할 수 있다.

✓ HTTP(HyperText Transfer Protocol) 표준 응용 프로토콜

- TCP/IP 프로토콜을 기반으로 한 대표적인 응용 프로토콜의 하나로 웹 브라우저의 요청과 웹 서버의 응답으로 데이터를 주고 받기 위한 통신 규약을 말한다.
- 다양한 데이터(HTML, CSS, JavaScript, XML, JSON, 멀티미디어 등)를 송수신할 때 사용한다.

✓ HTTP 프로토콜 특징

- HTTP 프로토콜은 비 연결 지향 응용 프로토콜로 웹 브라우저가 웹 서버에 요청을 보내고, 응답을 수신하면 웹 서버와 연결이 끊어지는(Socket 연결 종료) 특징을 가진다.(Connectionless)
- 이러한 특징으로 인해 웹 서버에서는 웹 브라우저의 상태 정보를 유지할 수 없다(Stateless)



1.1 HTTP 프로토콜 (2/5) – 요청 방식(Method)

- ✓ HTTP 요청 방식은 웹 클라이언트의 요청 종류를 웹 서버에 알려주기 위해 사용하는 것으로 다양한 방식이 존재한다.
- ✓ 대부분의 웹 클라이언트에서 웹 서버에 데이터를 요청할 때는 GET 방식을 사용하며, 폼 입력 데이터를 전송할 때는 POST 방식을 주로 사용한다.
  - 대부분의 웹 브라우저(크롬, 사파리 등)들은 GET과 POST 방식만을 지원하며, API 웹 서비스에서는 HTTP 메소드의 목적에 맞게 다양한 요청 방식을 사용할 수 있다.

GET	URL을 이용하여 웹 서버의 자원(리소스)을 요청 (Retrieve)
POST	요청 메시지 Body에 정보를 포함시켜 웹 서버로 전송 (Create)
PUT	웹 서버의 특정 리소스를 변경하기 위해 요청 메시지 Body에 정보를 포함시켜 웹 서버로 전송 (Update)
DELETE	웹 서버의 특정 리소스를 삭제하기 위해 요청 메시지 Body에 정보를 포함시켜 웹 서버로 전송 (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 URL을 이용하여 웹 서버의 특정 자원(리소스)을 요청할 때 사용한다.

- URL 맨 뒤에 요청 파라미터 (쿼리 스트링)를 추가하는 방식으로 웹 서버에 데이터를 전달할 때도 사용한다.

✓ 요청 파라미터는 URL을 포함하여 최대 2KB 용량만 전송할 수 있다.

1.1 HTTP 프로토콜 (4/5) – POST 방식

✓ POST 방식은 웹 서버에 데이터를 전송할 때 주로 사용한다.

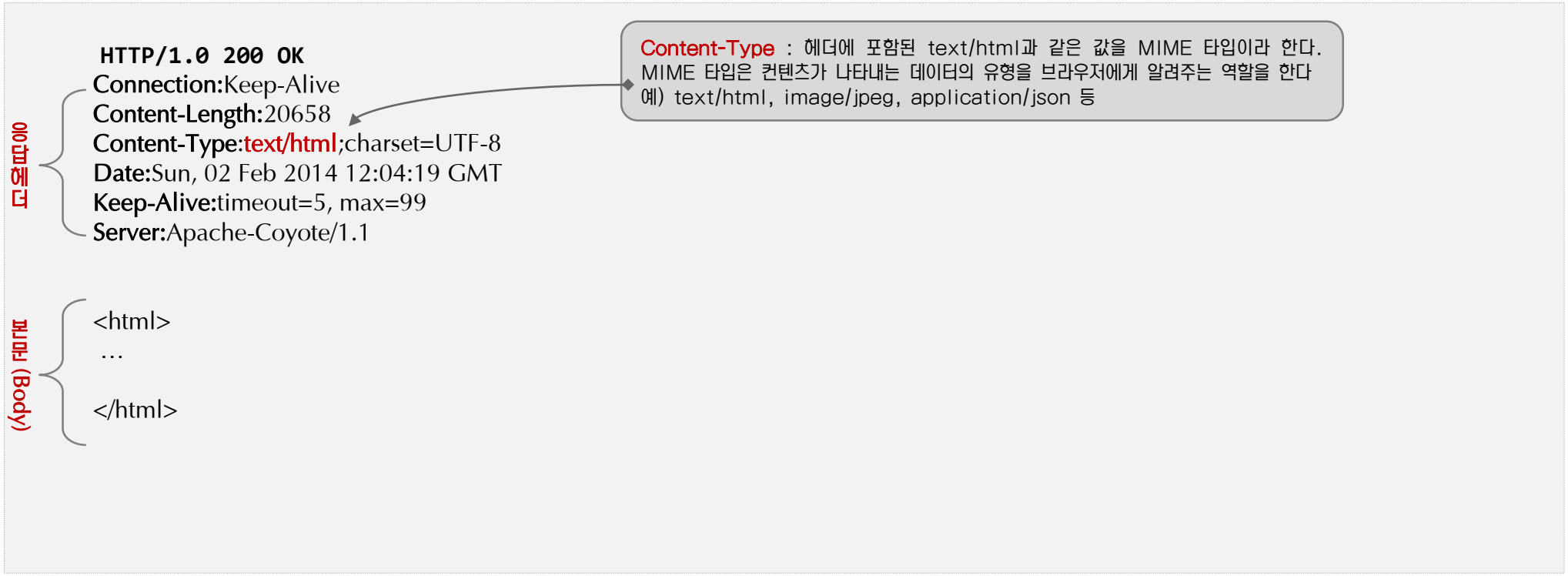
- 요청 메시지 바디에 파라미터를 포함시켜 웹 서버로 데이터를 전송한다.
- URL에 파라미터 값들이 노출되지 않는 장점이 있다.
- GET 방식과 달리 전달하는 요청 파라미터 용량에 대한 제약이 없다.

✓ 웹 페이지에서 웹 서버로 폼 입력 데이터를 전송하거나, 파일을 업로드 할 때 주로 사용한다.

1.1 HTTP 프로토콜 (5/5) – Response (응답 메시지)

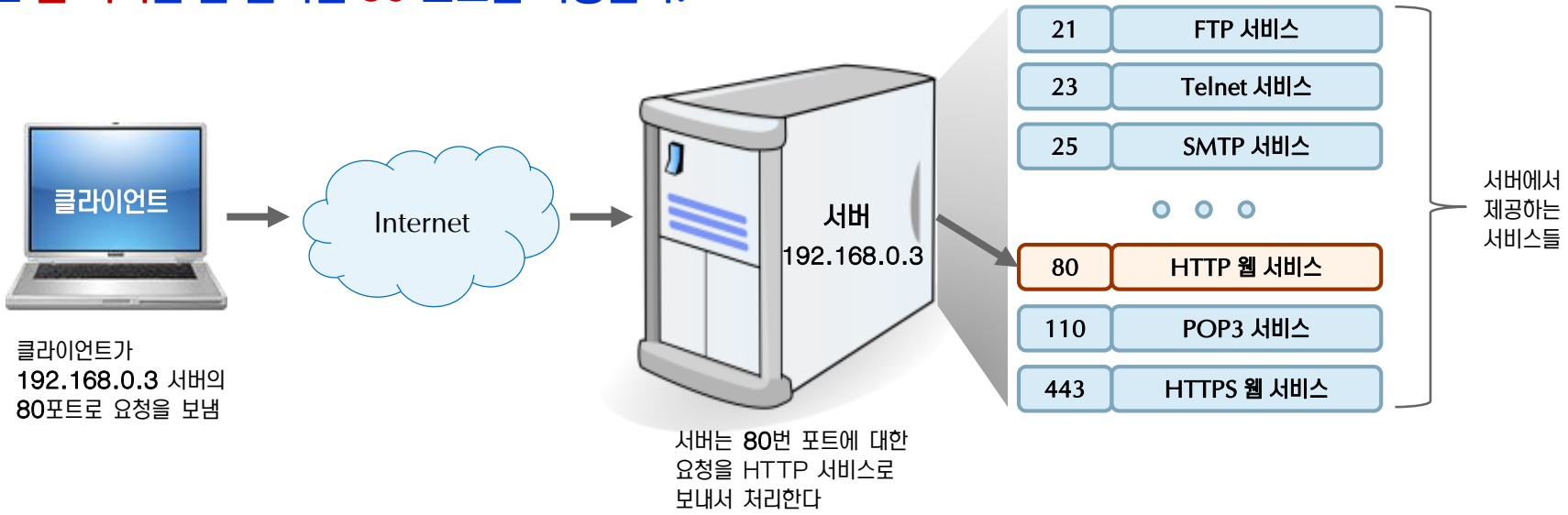
✓ **응답 메시지(Response)란 웹 서버가 웹 클라이언트로 보내는 메시지를 말한다.**

- 응답 메시지는 응답라인, 응답헤더, 본문(Body)으로 구성된다.
- 응답 라인에는 사용 프로토콜, 요청 성공 여부 코드가 포함되며, 응답 헤더에는 본문에 포함된 컨텐츠의 종류 등이 포함되어 있다.
- 응답 메시지 바디에는 HTML, 이미지와 같은 웹 브라우저에서 사용할 컨텐츠가 포함된다.



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 서버에 접속하기를 원하는지 알 수 있다
- ✓ 일반적으로 웹 서버는 잘 알려진 80 포트를 사용한다.



1.3 Web Server (1/4)

✓ 웹 서버는 HTTP 프로토콜을 기반으로 웹 클라이언트(브라우저)가 요청한 자원(리소스)을 찾아 서비스하는 서버를 말한다.

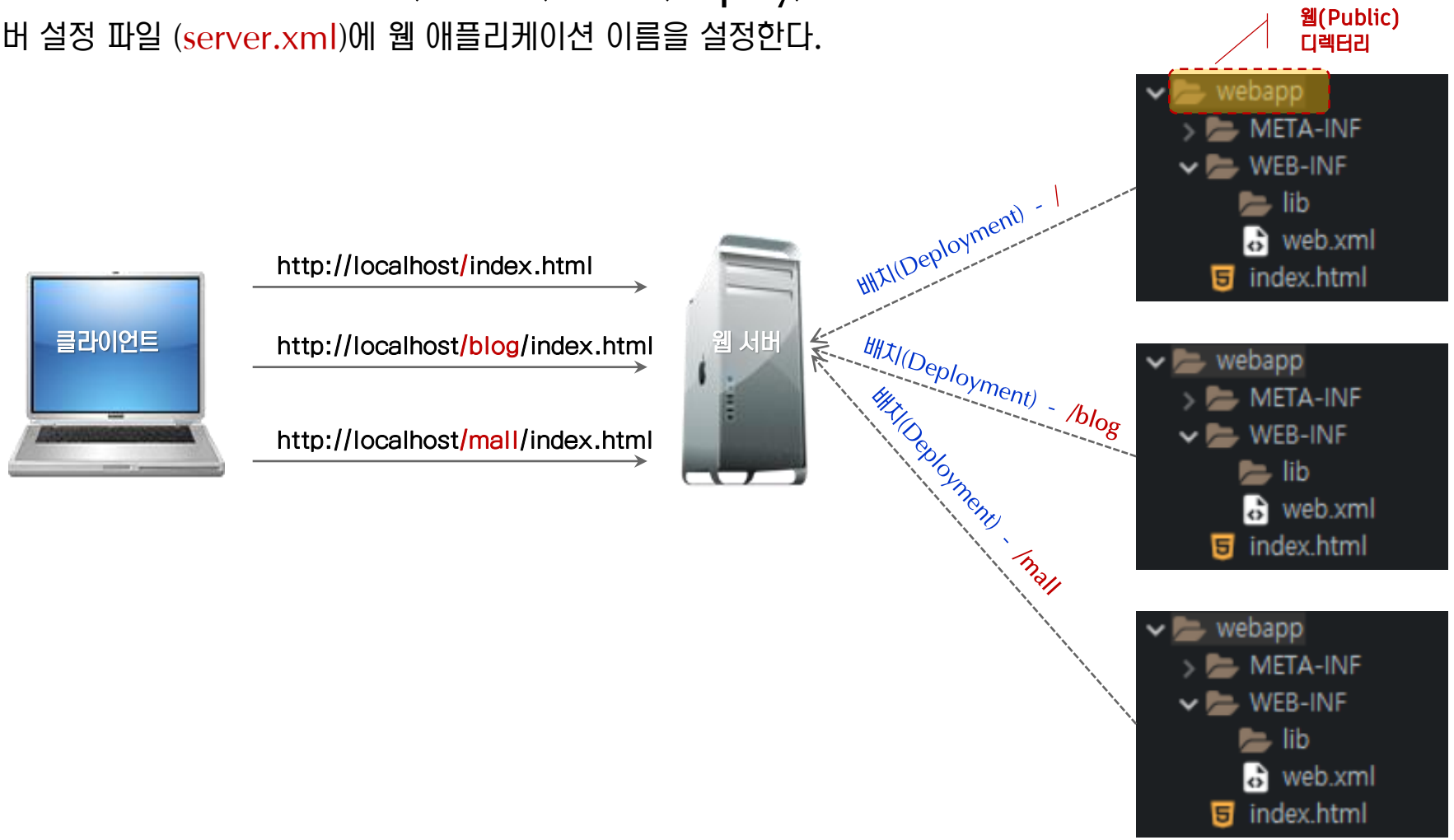
- 웹 서버는 단지 웹 브라우저가 요청한 정적 리소스(HTML 문서, 이미지 등)를 찾아서 그대로 웹 브라우저에게 전송하는 역할이다.
- 웹 클라이언트가 요청한 리소스가 웹 서버에 존재하지 않을 경우 웹 서버는 응답라인에 404 (Not Found) 응답 코드를 전송한다.

✓ 웹 서버는 동적으로 컨텐츠를 생성할 수 없기 때문에 동적 컨텐츠 서비스를 위해 CGI와 같은 기술이 등장하였다.

1.3 Web Server (2/4) – Web Application 배치

✓ 웹 서버에 1개 이상의 웹 애플케이션(디렉토리)을 배치(Deploy)할 수 있다.

- 웹 서버 설정 파일 (server.xml)에 웹 애플리케이션 이름을 설정한다.



1.3 Web Server (3/4) – Apache Tomcat 설치 및 환경 설정

✓ Apache Tomcat 다운로드 및 설치
- https://tomcat.apache.org/
- Portable(압축) 버전(apache-tomcat-9.0.xx-windows-x64.zip) 다운로드 및 압축 해제

✓ OS 환경 변수 설정
- JAVA_HOME = JDK_홈 디렉토리

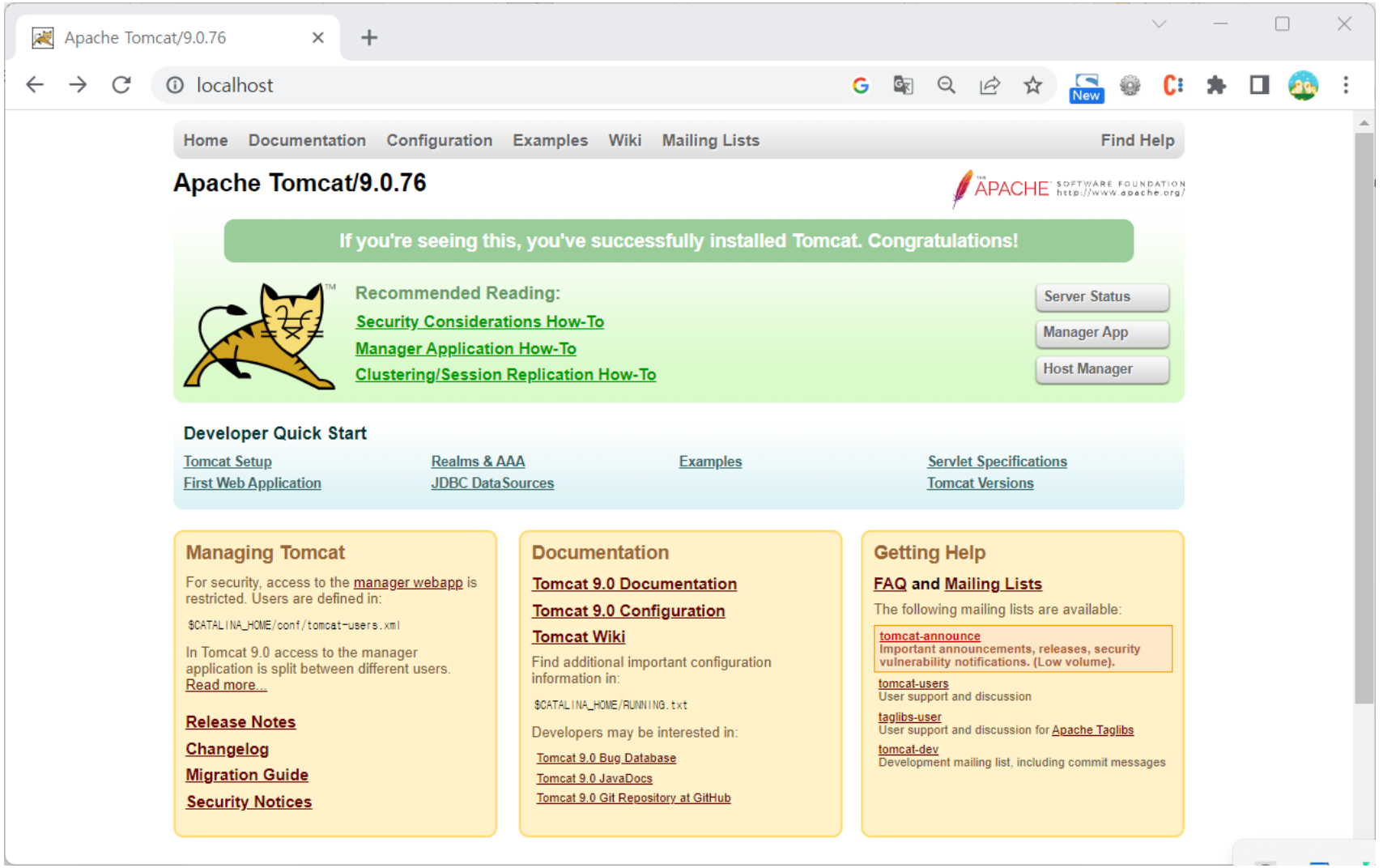
✓ Apache Tomcat 설정
- /config/server.xml
- 포트번호 변경 : 8080 -> 80
- 문자 인코딩 설정 : utf-8

```
<Connector port="80" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"
           maxParameterCount="1000"
           URIEncoding="utf-8"
           />
```

✓ Apache Tomcat 실행 및 종료
- /bin/startup.bat
- /bin/shutdown.bat

1.3 Web Server (4/4) – 웹 브라우저 요청 테스트

✓ http://localhost:80



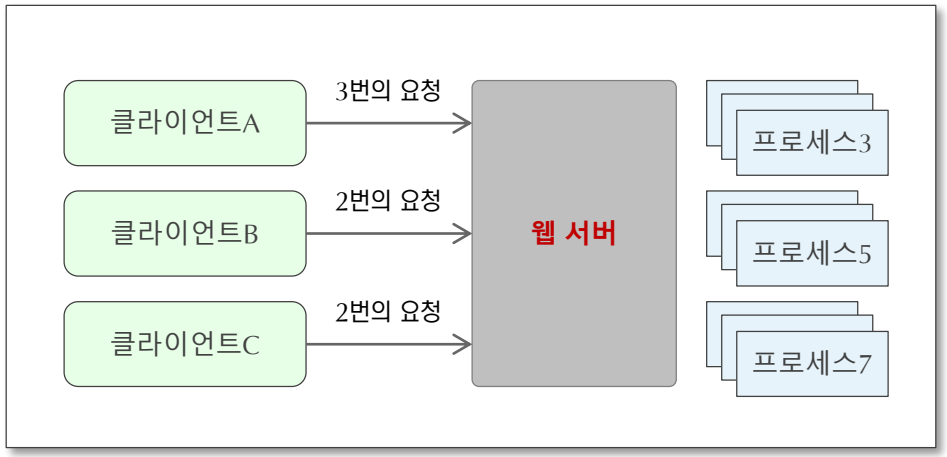
1.4 CGI (Common Gateway Interfaces) 소개

✓ CGI는 웹 브라우저 요청 시 웹 서버에서 동적인 컨텐츠 생성을 위한 컴포넌트에 대한 인터페이스(명세)이다.

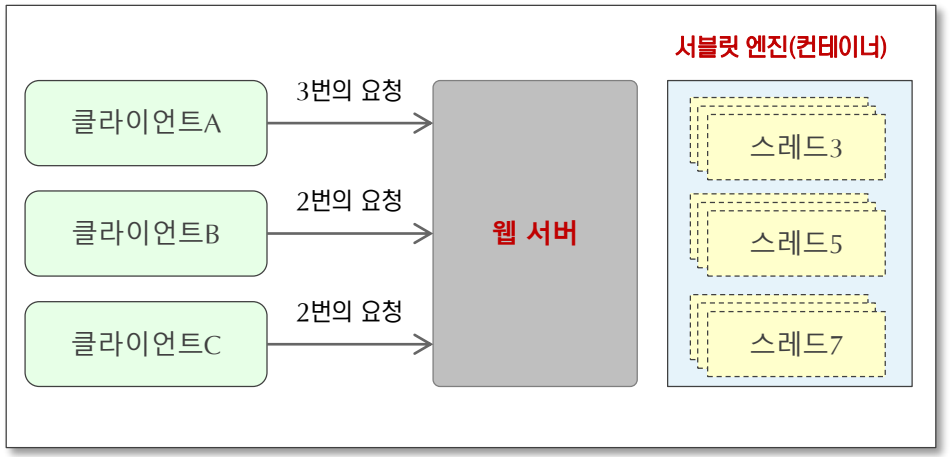
- CGI는 전통적인 개발 방식으로 웹 클라이언트 요청마다 개별 프로세스가 생성되어 시스템 부하가 많이 생기는 단점이 있었다.
- 또한 플랫폼에 종속적이어서 윈도우에서 C언어 등으로 만들어진 CGI 애플리케이션은 리눅스에서 사용할 수 없었다.

✓ 이러한 단점을 해결하기 위하여 웹 클라이언트 요청 시 개별 스레드가 요청을 처리하는 방식으로 발전하였다.

- Servlet, JSP, ASP, ASP.NET, PHP 등



전통적인 프로세스 CGI 방식



요청을 스레드로 처리하는 방식

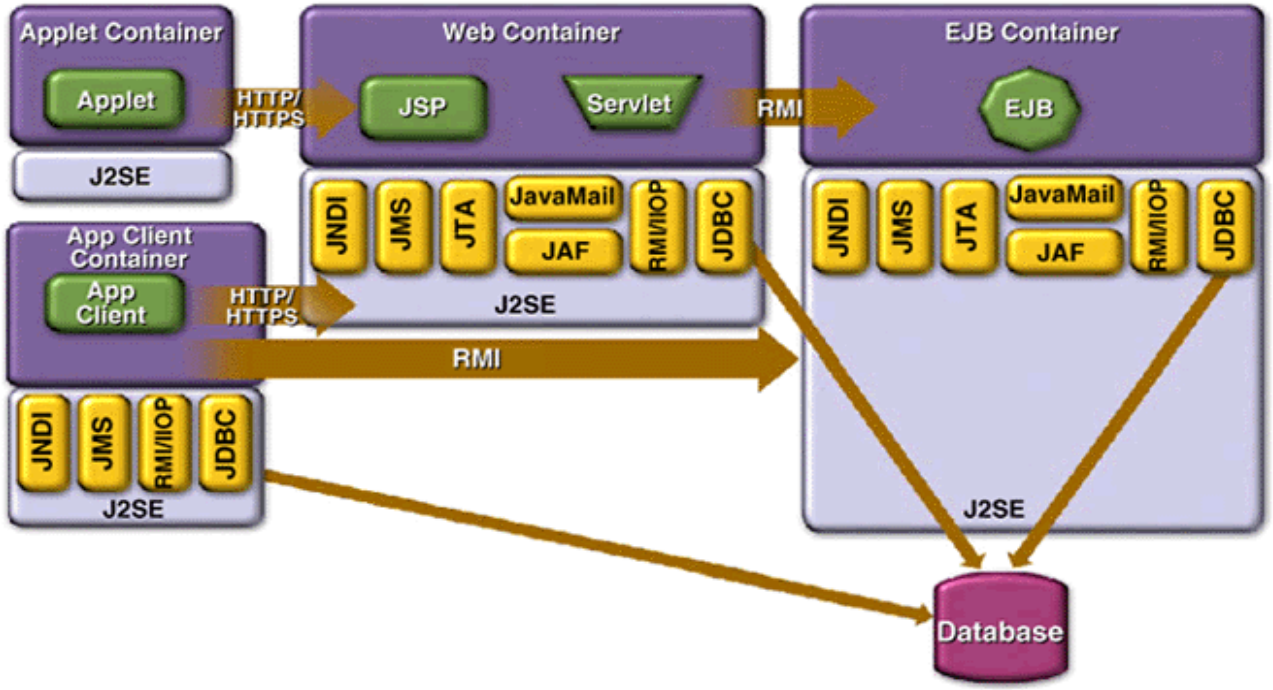
1.5 Java EE 플랫폼 소개 및 표준 기술 명세

✓ Java SE (Standard Edition)

- Stand-alone Application 개발 및 실행을 위한 기본 자바 플랫폼

✓ Java EE (Enterprise Edition)

- Java EE 플랫폼은 Java SE 플랫폼을 기반으로 그 위에 탑재된다.
- 웹 프로그램 개발 및 실행에 필요한 기능을 다수 포함하고 있다.
- Servlet, JSP, JDBC, JNDI, EJB, JTA 등
- 대규모, 다계층, 확장성, 신뢰성, 보안 네트워킹 API, 환경 등을 제공한다.



2. Servlet Programming

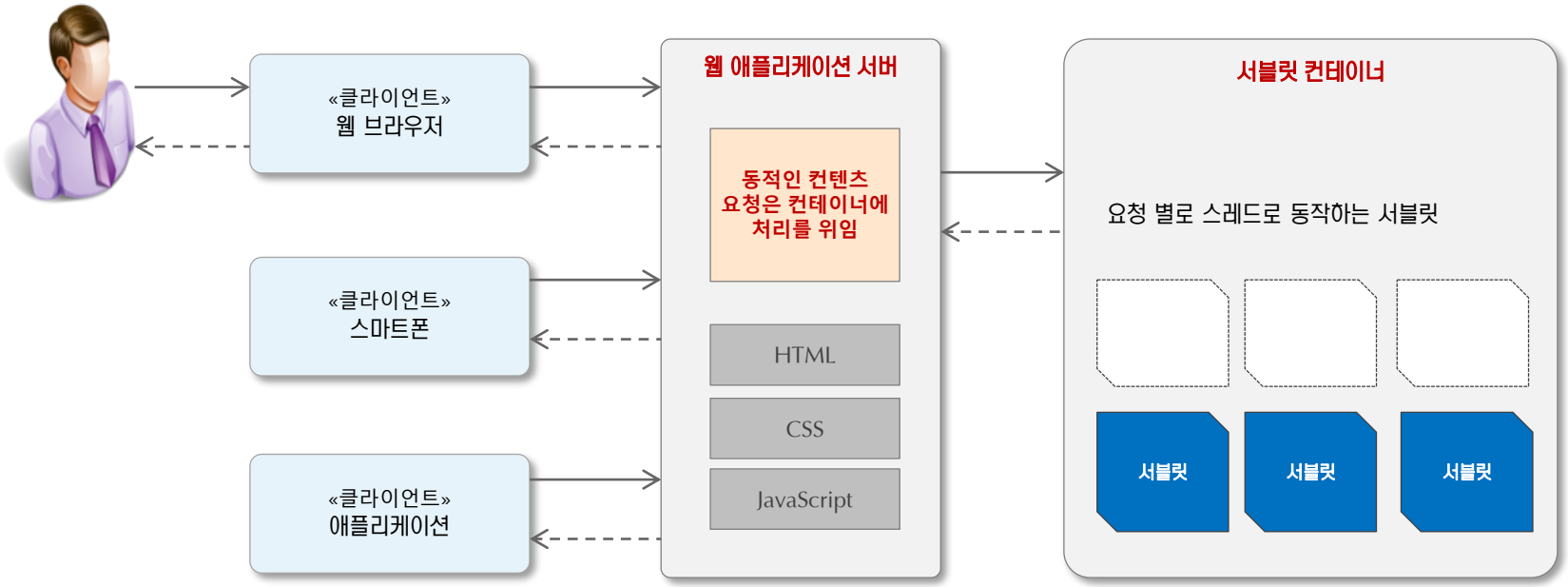
2.1 Servlet 기본 (1/7) – 소개

✓ 서블릿(Servlet)은 브라우저 요청 시 동적 웹페이지 생성을 위한 자바 컴포넌트에 대한 JavaEE 표준명세(인터페이스)이다.

- 서블릿 인터페이스를 구현하면 동적 컨텐츠를 생성하는 서블릿 컴포넌트를 개발하여 서버에 배치할 수 있다.
- Server + ~let 합성어

✓ 서블릿은 WAS(Web Application Server)에 배치되고 관리된다.

- WAS는 웹 서버 + 서블릿 엔진(컨테이너)으로 구성된다.
- 웹 서버 : 웹 브라우저의 정적 리소스(html, css, javascript, image 등) 요청 처리
- 서블릿 컨테이너 : 웹 브라우저의 서블릿 요청 처리



2.1 Servlet 기본 (2/7) – 특징

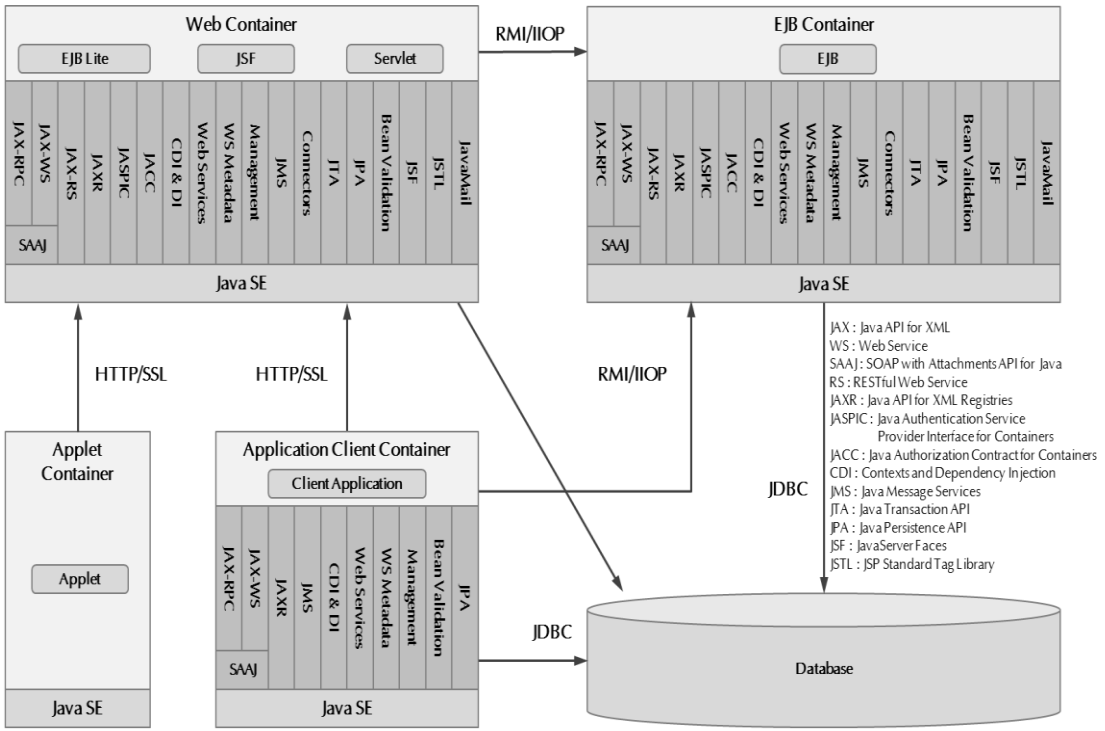
✓ 서블릿 특징

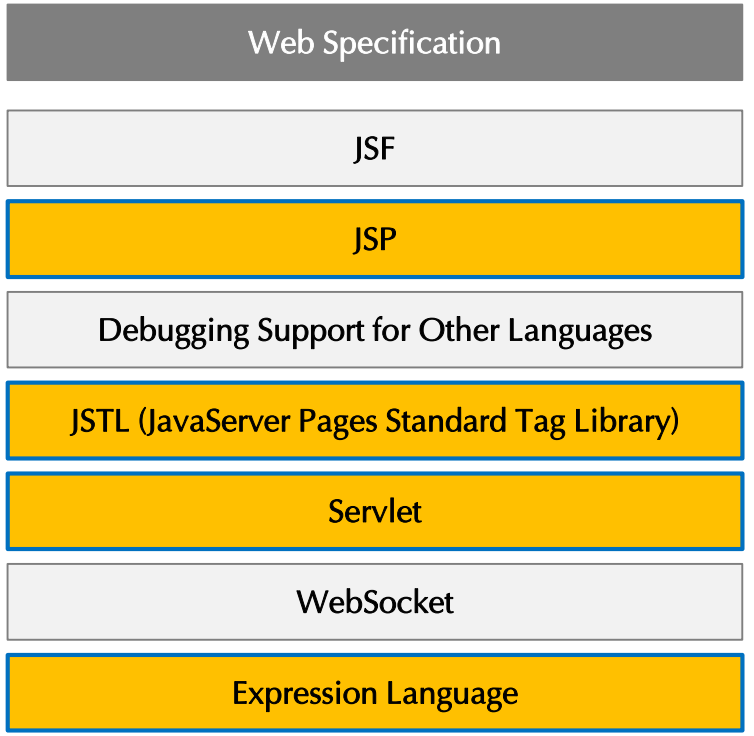
- 웹 브라우저의 요청에 동적으로 응답하는 웹 어플리케이션 컴포넌트이다.
- 자바 언어를 사용하므로 플랫폼에 독립적이고, 스레드 기반으로 좀 더 효율적인 멀티 태스킹을 지원한다.
- HTML을 생성하여 응답한다.
- 서블릿 컨테이너의 쓰레드에 의해 실행된다.
- HTTP 프로토콜 서비스를 지원하는 Servlet API는 개발에 필요한 다양한 인터페이스와 클래스를 제공한다.
- 서블릿 컨테이너는 서블릿이 동작하는 실행환경으로 서블릿의 생명주기(생성, 실행, 삭제)를 관리한다.
- 웹 애플리케이션 MVC 디자인 패턴의 Controller 역할을 담당한다.
- HTML 변경 시 서블릿을 재 컴파일해야 한다는 단점이 존재한다.

✓ 서블릿은 WAS내의 서블릿 컨테이너에 의해 관리되며, 웹 브라우저 요청(Request)을 받으면 요청에 맞는 로직을 실행하고 웹 브라우저에게 HTTP 응답 메시지 형식으로 응답(Response)하게 된다.

2.1 Servlet 기본 (3/7) – JavaEE 표준 기술 명세

- ✓ JavaEE 명세는 자바 엔터프라이즈 애플리케이션 개발에 필요한 기술들에 대한 표준이다.
- ✓ JavaEE 명세 중 웹 애플리케이션 개발과 관련된 기술들을 모아둔 것을 Web Specification 이라고 한다.
- ✓ JavaEE 모든 명세를 구현한 서버를 WAS(Web Application Server)라고 하며, 이 중 웹과 관련된 기술만을 처리하는 엔진을 웹 컨테이너라 한다 (예: WebLogic, Websphere, JEUS, Apache Tomcat 등)
- ✓ 웹과 관련된 기술은 Servlet, JSP, JSTL, EL 등 이다.





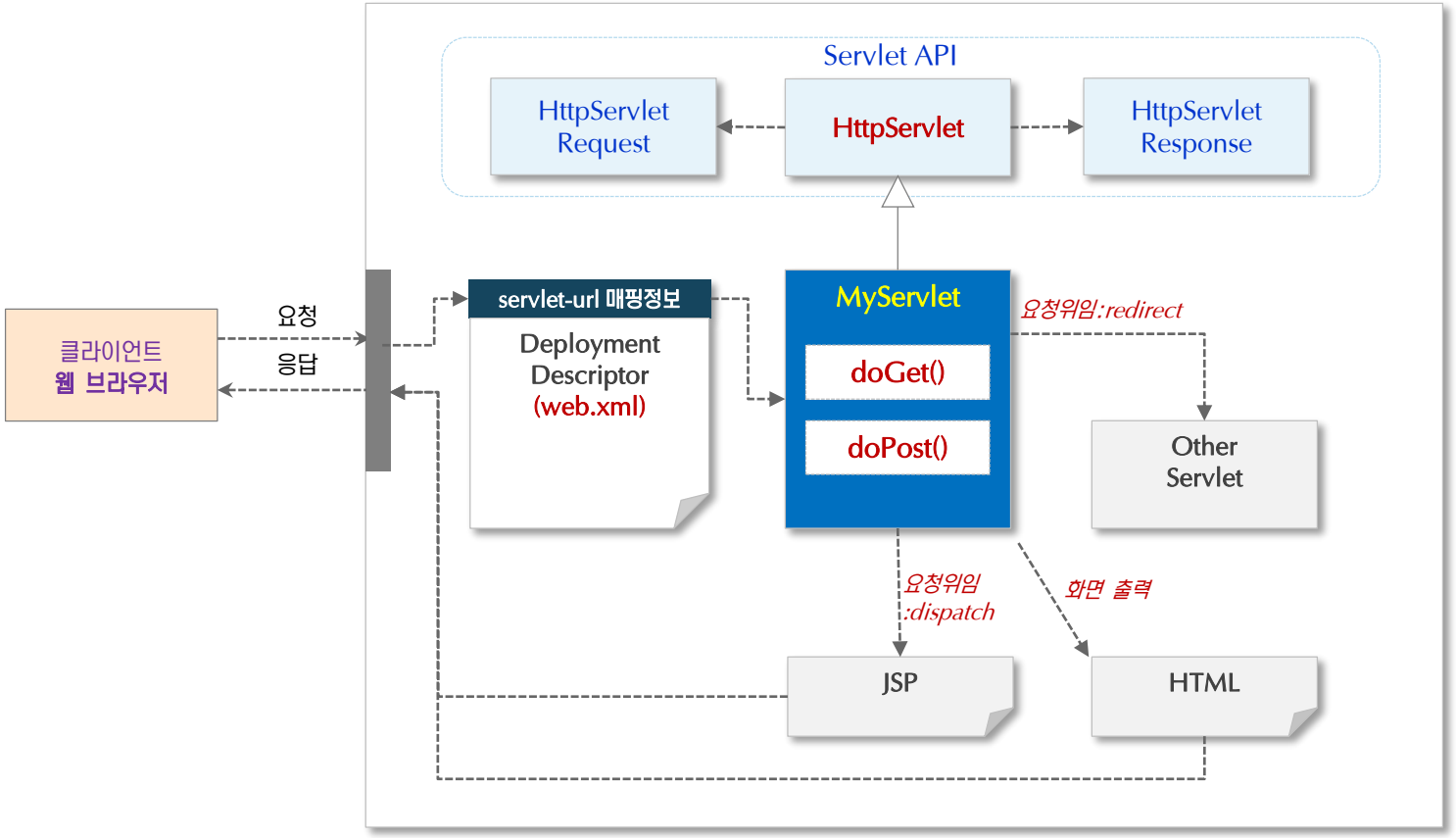
2.1 Servlet 기본 (4/7) – 서블릿 웹 프로그래밍 핵심 구성 요소

✓ 웹 클라이언트의 요청은 web.xml(Deployment Descriptor) 파일에 설정된 URL 매핑 서블릿으로 위임된다.

- Servlet 명세 3.0 부터는 @WebServlet 어노테이션으로 매핑 가능하다.

✓ 서블릿은 Servlet API의 HttpServlet 클래스를 상속받아 개발하며, 웹 브라우저 요청방식에 따른 처리를 구현할 수 있다.

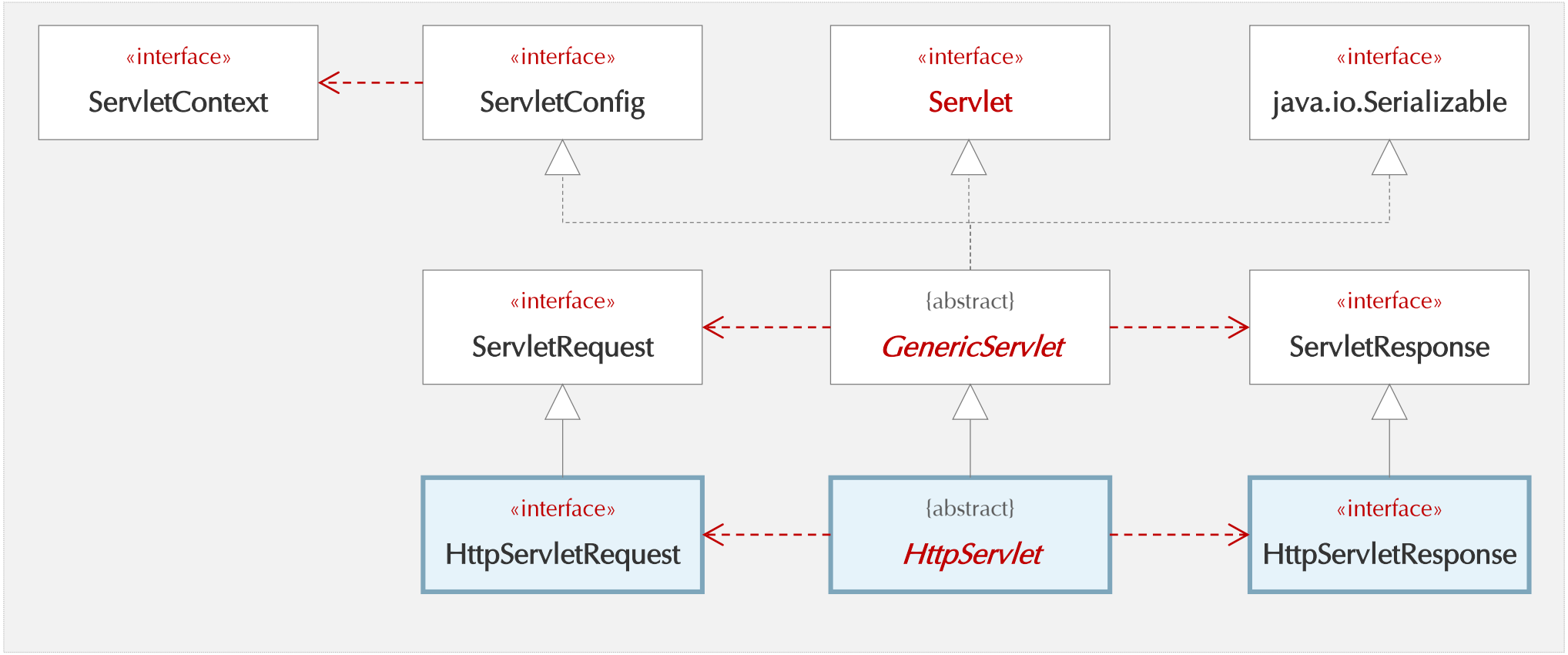
✓ 브라우저에 의해 요청 된 서블릿이 직접 처리하여 출력하거나, 또는 다른 서블릿, JSP, HTML로 요청을 위임할 수 있다.



2.1 Servlet 기본 (5/7) – 서블릿 API 구조

✓ 서블릿 API는 서블릿 구현체(컴포넌트) 작성을 위한 인터페이스 및 클래스로 javax.servlet 패키지에 존재한다.

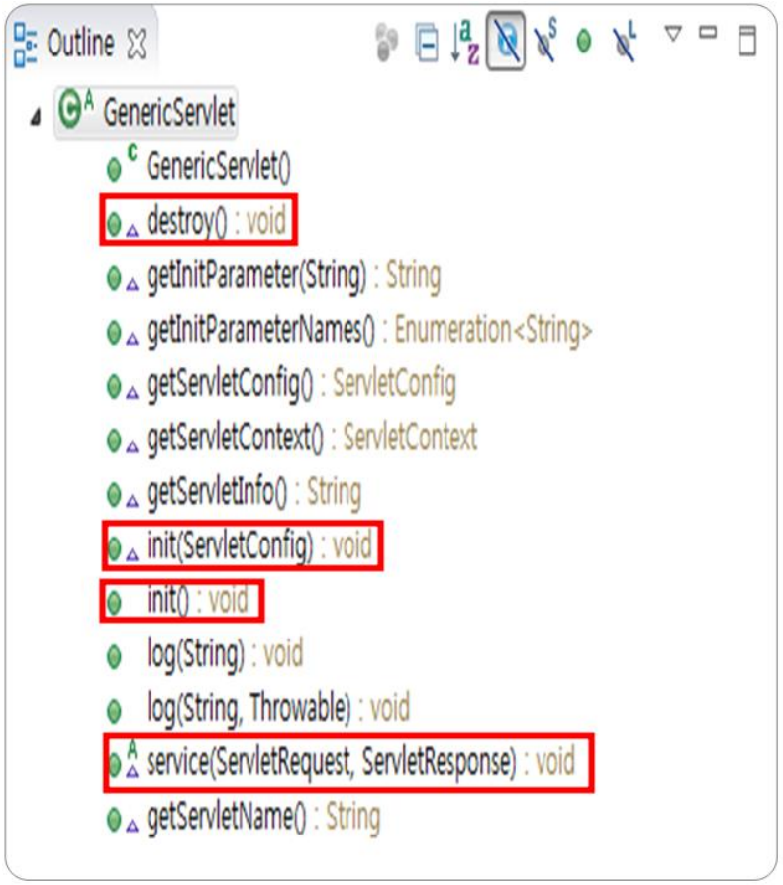
✓ 서블릿을 사용한 웹 프로그래밍을 할 때는 일반적으로 HttpServlet을 상속 받아 구현한다.

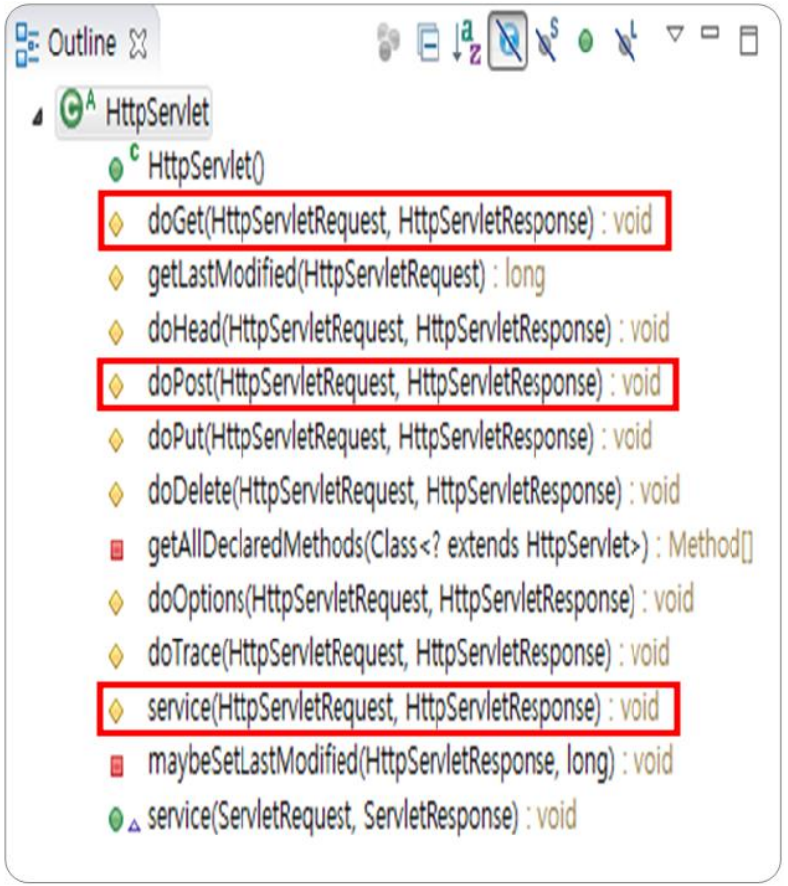


서블릿 인터페이스 및 클래스 관계

2.1 Servlet 기본 (6/7) – HttpServlet 추상클래스

- ✓ GenericServlet은 서블릿 개발에 필요한 기본 기능들이 미리 구현되어 있는 추상 클래스이다.
- ✓ HttpServlet는 GenericServlet을 상속하며 웹 브라우저의 HTTP 요청 방식별로 메소드가 정의되어 있다.
- ✓ 일반적으로 웹 애플리케이션에서는 HttpServlet 추상클래스를 상속받아 요청 방식에 해당하는 메소드를 재정의한다.
  - HttpServlet의 service() 메소드에서는 요청방식에 따라 해당 메소드를 호출한다(예, GET방식은 doGet() 호출)





2.1 Servlet 기본 (7/7) – 서블릿 등록 (배치)

✓ **웹 애플리케이션에 서블릿을 등록하는 방법**에는 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.

서블릿 3.0 이전

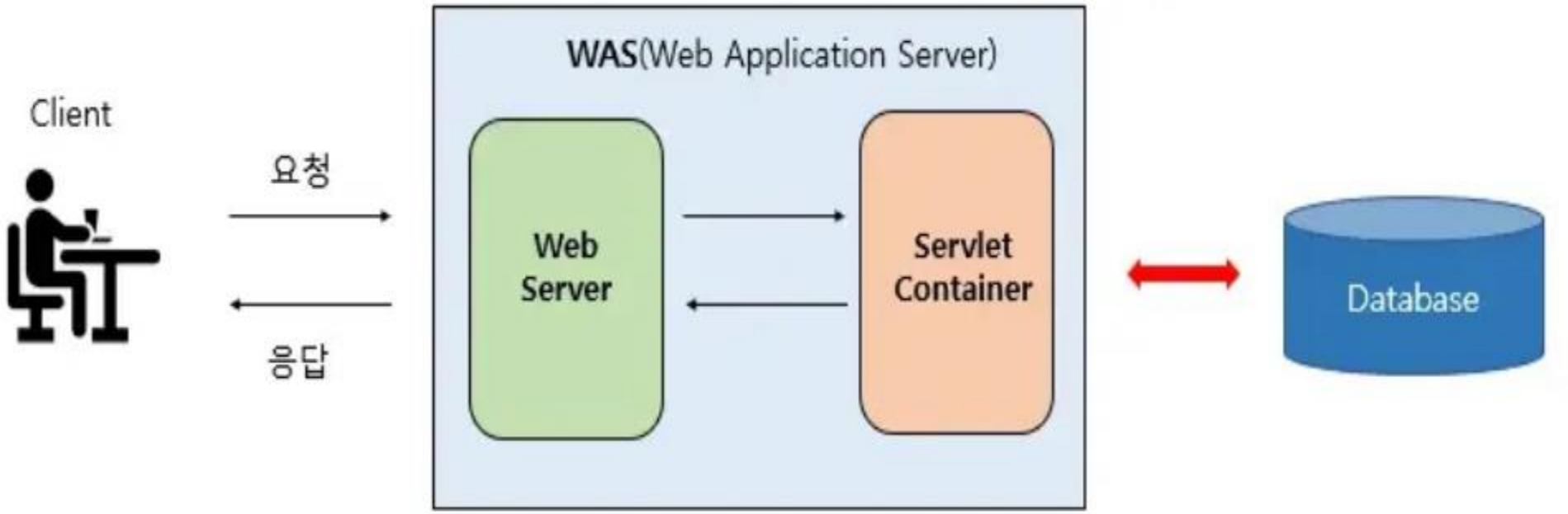
```
<servlet>
  <description>처음 작성하는 서블릿</description>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>ezen.servlet.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/hello.do</url-pattern>
</servlet-mapping>
```

서블릿 3.0 이후

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

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

2.2 Servlet Container (1/5) – 소개 (1/2)

✓ 서블릿 컨테이너는 서블릿을 담고 관리해주는 컨테이너이다. 서블릿 컨테이너는 구현되어 있는 서블릿 클래스의 규칙에 맞게 서블릿을 관리하며, 웹 클라이언트의 요청을 받으면 HttpServletRequest와 HttpServletResponse 객체를 생성하여 GET, POST 요청 방식에 따라 동적인 HTML 페이지를 생성하여 응답한다.



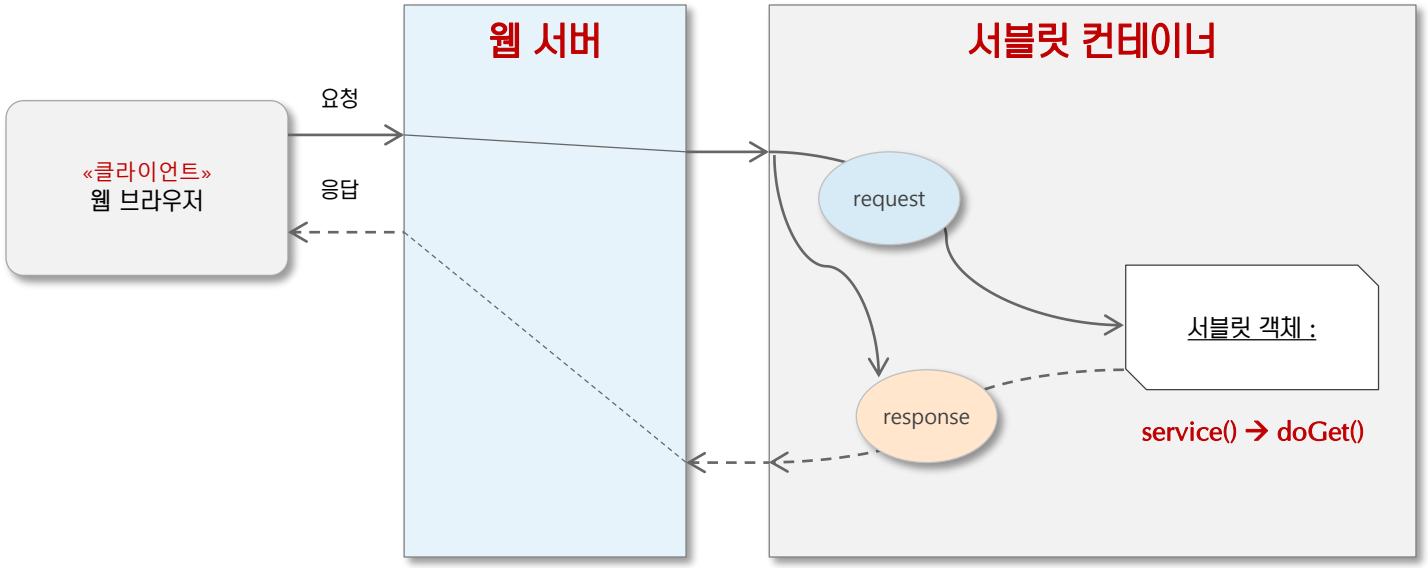
2.2 Servlet Container (2/5) – 소개 (2/2)

✓ 서블릿 컨테이너는 다음과 같은 기능을 수행한다.

1. **서블릿 생명주기 관리**
   - 서블릿 컨테이너는 서블릿의 생성과 죽음을 관리한다. 서블릿 클래스를 로딩하여 인스턴스화하고, 초기화 메서드를 호출하고, 웹 클라이언트 요청이 들어오면 적절한 서블릿 메서드를 찾아서 동작한다.
     또한 서블릿의 생명이 다하면 가비지 컬렉션(Garbage Collection)을 통해 메모리에서 제거한다.
2. **통신 지원**
   - 서블릿 컨테이너는 웹 서버와 소켓을 만들어서 웹 클라이언트의 요청을 받고 응답할 수 있는 HTTP 통신을 지원해준다.
   - HTTP 통신을 하기 위한 Listen, Accept 등의 과정을 API로 제공하여 복잡한 과정을 생략해주기 때문에 개발자가 비즈니스 로직 개발에만 집중할 수 있게 도와준다.
3. **멀티스레드 지원 및 관리**
   - 서블릿 컨테이너는 클라이언트의 요청을 받을 때마다 새로운 자바 스레드를 생성한다.
     따라서 동시에 여러 요청이 들어와도 멀티스레딩 환경에서 동시 다발적인 작업을 관리할 수 있다.
4. **선언적인 보안 관리**
   - 서블릿 컨테이너는 보안 관련 기능을 제공하기 때문에 개발자는 서블릿에 보안 관련 메서드를 구현하지 않아도 된다.

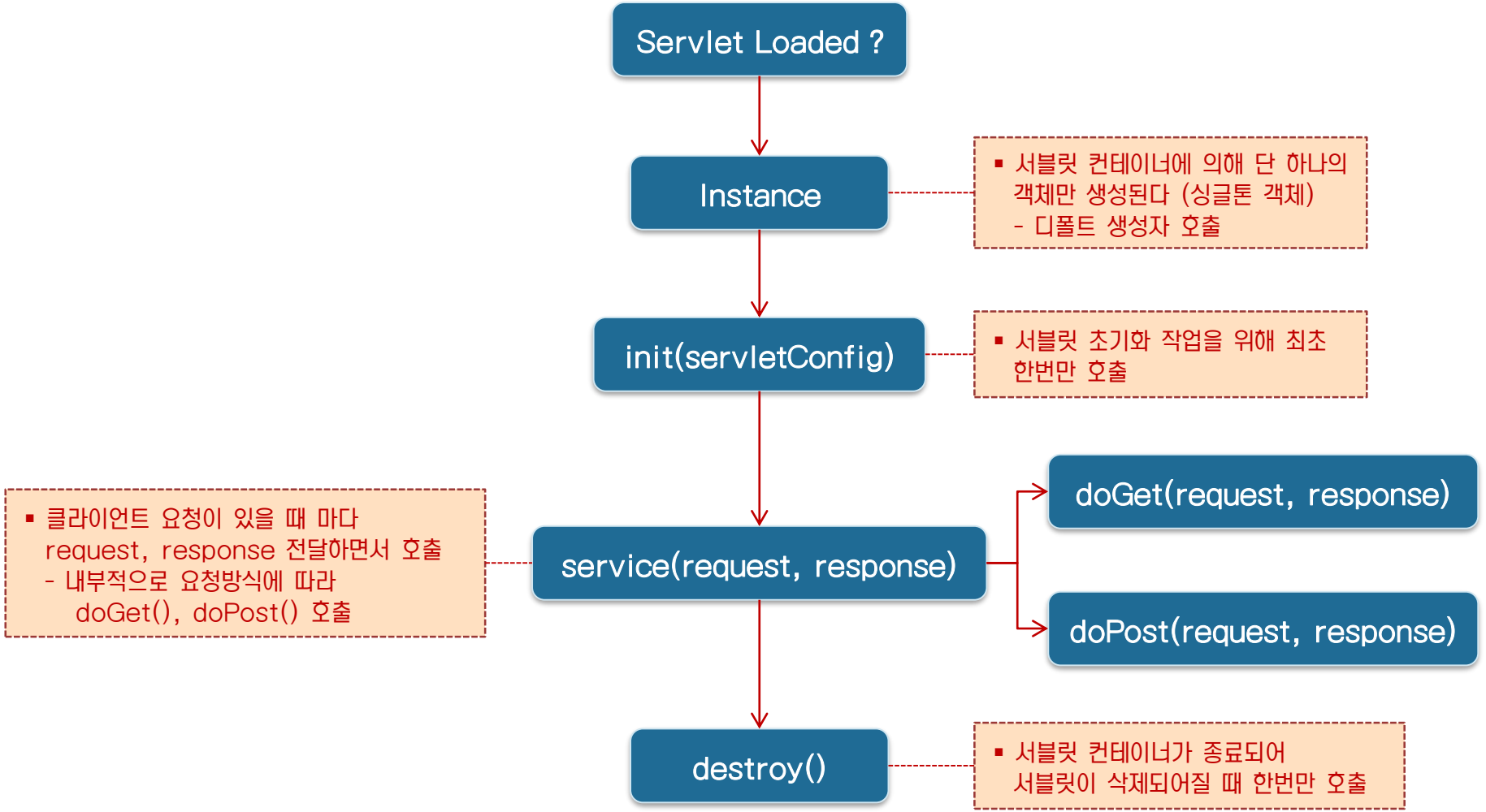
2.2 Servlet Container (3/5) – Servlet 처리 과정

- ✓ 웹 클라이언트가 웹 서버로 HTTP 요청을 보낸다.
  - 웹 클라이언트 요청 메시지 전송
- ✓ 웹 서버는 웹 클라이언트의 요청을 수신하여 정적 리소스(html, css, javascript 등) 요청일 경우 리소스를 서비스하고, 정적 리소스 아닌 경우 서블릿 컨테이너에게 요청을 위임한다.
  - 요청 메시지 분석 및 요청 위임
- ✓ 서블릿 컨테이너는 HTTP Request, HTTP Response 객체를 생성하여 서블릿에 전달한다.
  - HttpServletRequest, HttpServletResponse 생성 및 전달
- ✓ 서블릿은 서비스 하고자 하는 컨텐츠를 응답 객체에 출력하여 웹 클라이언트에게 응답한다.
  - 웹 서버와의 출력스트림 생성 및 컨텐츠 출력



2.2 Servlet Container (4/5) – Servlet 라이프사이클 (1/2)

✓ 서블릿 컨테이너는 서블릿의 생성 및 소멸을 관리한다.



2.2 Servlet Container (5/5) – Servlet 라이프사이클 (2/2)

✓ 서블릿 라이프사이클 메소드

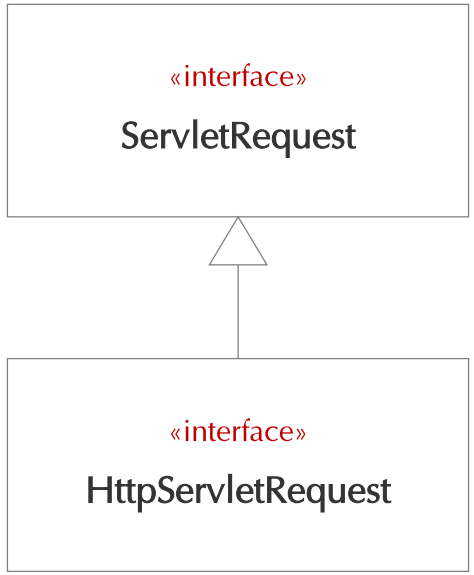
- init(ServletConfig config)
  - 클라이언트의 요청을 처리하기 전에 서블릿을 초기화하는 메소드
  - 컨테이너가 서블릿 인스턴스를 생성한 후 자동 호출한다.
  - 필요한 경우 재정의 가능 (예 : 데이터 베이스 드라이버 로드 및 커넥션 생성 등)
- service(HttpServletRequest request, HttpServletResponse response)
  - 클라이언트 요청을 받아 처리하는 메소드
  - 요청의 HTTP 메소드(GET/POST 등)를 참조하여 doXXX() 메소드를 호출한다.
  - 재정의하지 않음 (상속받은 상위 클래스의 service() 메소드를 그대로 사용)
- doGet(HttpServletRequest request, HttpServletResponse response)
  doPost(HttpServletRequest request, HttpServletResponse response)
  - 동적 컨텐츠 생성 및 웹 브라우저에게 전송
  - 요청 방식에 따라 메소드를 재정의하여 구현한다.
- destory()
  - 요청처리가 끝나면 컨테이너가 호출하는 메소드이다.
  - 특별한 경우 아니면 재정의하지 않는다.

2.3 요청과 응답 (1/4) – HttpServletRequest

✓ HttpServletRequest 객체는 웹 브라우저의 요청 메시지 정보를 담아 제공된다.

✓ HttpServletRequest 주요 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

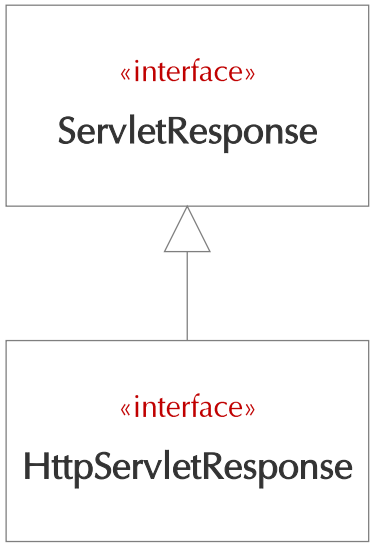


2.3 요청과 응답 (2/4) – HttpServletResponse

✓ HttpServletResponse 객체는 웹 브라우저에게 제공하는 응답 메시지 정보를 담는다.

✓ HttpServletResponse 주요 메소드

- setContentType() : 웹 브라우저에게 전송하는 데이터의 종류 설정(응답 메시지 헤더 설정)
- getWriter() : PrintWriter : 웹 브라우저에게 전송할 문자 스트림 생성(응답 메시지 바디 설정)
- getOutputStream() : ServletOutputStream : 웹 브라우저에게 전송할 바이트 스트림 생성
- setStatus() : 응답 메시지의 응답(상태) 코드 설정



2.3 요청과 응답 (3/4) – ContentType (1/2)

✓ **ContentType은 웹 브라우저에게 전송하는 컨텐츠의 종류를 알려주기 위해 응답 헤더에 추가하는 정보이다.**

- 웹 브라우저는 응답 헤더의 ContentType을 읽고, 응답된 컨텐츠를 적절한 형태로 화면에 렌더링한다.
- 출력 스트림을 이용하여 데이터를 출력하기 전에 response.setContentType("마임타입") 메소드를 호출하여 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 메일 표준 규약을 확장한 규약이다.
- 현재는 MIME TYP은 메일 뿐만 아니라 HTTP 통신에서 파일 시스템 내에 존재하는 많은 파일을 구분하기 위해 사용되고 있다.
- 웹 서버가 웹 브라우저에 전송하는 콘텐츠 종류를 알려주기 위해 응답 헤더의 Content-type에 MIME 타입과 문자 인코딩 셋을 설정하여야 한다.
  - 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 (4/4) – 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 참조

2.4 Servlet API (1/16) – 개요

✓ **서블릿 컨테이너에 관리되는 서블릿의 개발과 실행을 가능하게 하는 인터페이스와 클래스들의 집합을 말한다.**

- JavaEE 명세를 구현한 모든 WAS에서 Servlet API 기본 제공한다.
- **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

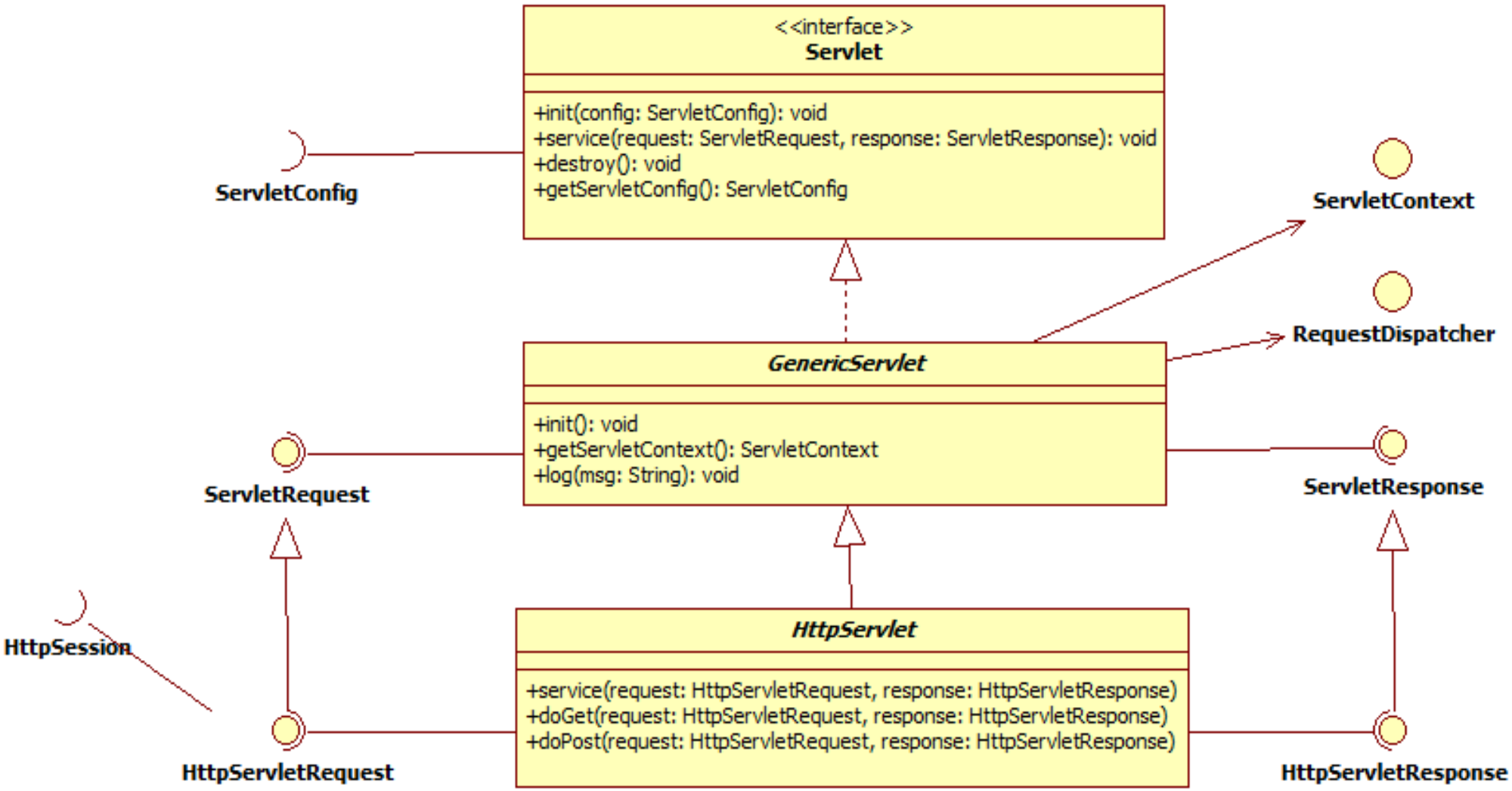
✓ **서블릿 컨테이너에 의해 관리되는 서블릿 구현체는반드시 javax.servlet.Servlet 인터페이스를 구현하여야 한다.**
일반적으로 Servlet 인터페이스를 구현한 javax.servlet.http.HttpServlet 추상클래스를 상속한다.

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

✓ **기타 웹 관련 다양한 서비스 제공을 위한 인터페이스와 클래스를 제공한다.**

- ServletConfig, HttpSession, ServletContext, RequestDispatcher, Cookie 등

2.4 Servlet API (2/16) – 구조



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

✓ ServletRequest :: HttpServletRequest

- 서블릿 컨테이너는 클라이언트 요청 메시지 정보를 담은 HttpServletRequest 객체를 생성하여 서블릿에 전달한다.
- 서블릿 개발자는 HttpServletRequest 객체의 getter 메소드를 이용하여 요청 정보를 추출할 수 있다.

✓ HttpServletRequest 주요 메소드

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

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

✓ 웹 클라이언트에서 전송한 FORM 데이터 처리

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

✓ HTML FORM 태그의 3가지 속성

- METHOD
  - GET : 디폴트 요청방식으로 200바이트 이하 데이터를 URL 쿼리스트링을 통해 정보 전달(예: /someServlet?id=bangry)
  - POST : 요청메시지의 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
- FORM 하위 태그들
  - INPUT, SELECT, TEXTAREA 등

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

✓ HttpServletRequest의 메소드를 이용한 FORM 데이터 수신 방법

- 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 서블릿에 전달한다.
- 개발자는 setter 메소드를 이용하여 클라이언트에 전송할 응답 메시지를 전달하면 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);//302
     // 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

✓ 웹 애플리케이션 설정 파일(/WEB-INF/web.xml)에 등록한 초기 설정 정보를 편리하게 읽어 올 수 있는 기능을 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

✓ ServletContext

- 서블릿 컨테이너 실행 환경 정보를 제공한다.
- 서블릿 컨테이너에 의해 관리되는 서블릿들 간의 데이터 공유를 위해 사용한다.

✓ ServletContext 주요 메소드

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

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

2.4 Servlet API (9/16) – 웹 클라이언트 요청을 위임하는 2가지 방식

✓ 클라이언트 요청 위임이란 요청을 받은 서블릿이 다른 URL(서블릿, JSP, html 등)에 요청을 위임하는 것을 말한다.

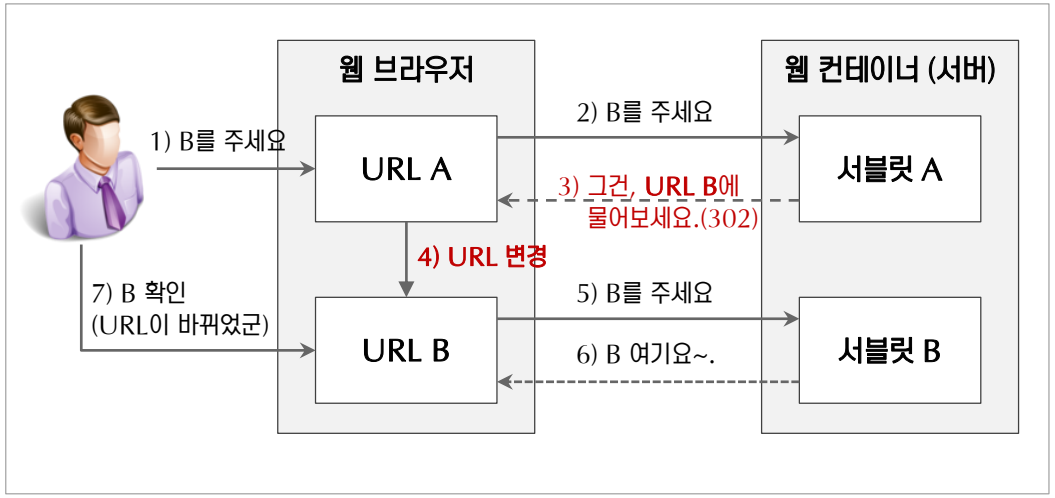
✓ 요청 위임이 클라이언트에서 일어나는지 또는 서버에서 일어나는지에 따라 2가지 방식이 있다.

✓ Redirect 방식

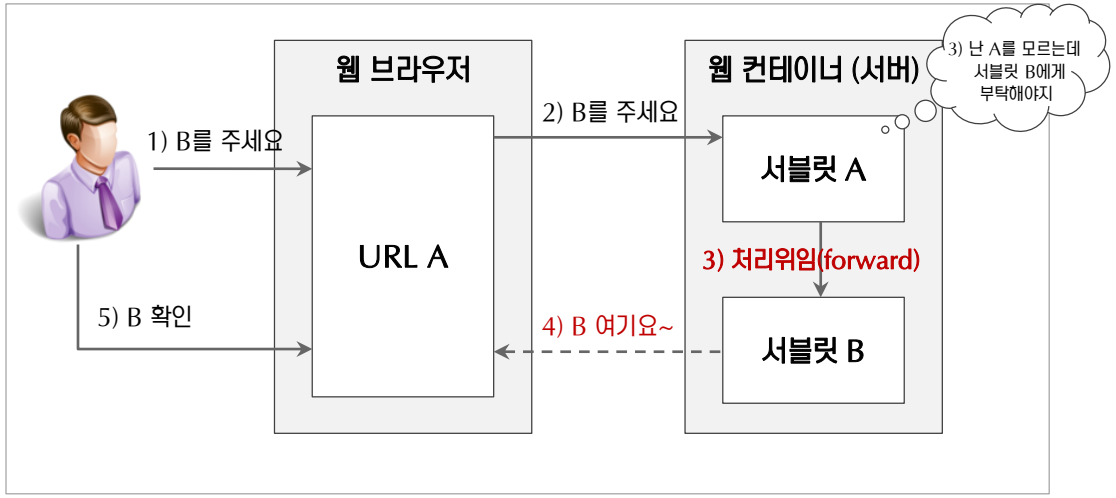
- 웹 클라이언트 즉, 브라우저에 위임할 URL로 다시 요청하게 하는 방식이다.

✓ Request Dispatch

- 웹 서버 내부에서 다른 URL에 요청을 위임하는 방식이다(Forward).



Redirect 방식



Request Dispatch 방식

2.4 Servlet API (10/16) – Redirect 방식

✓ 요청 받은 서블릿은 다른 URL에서 처리해야 할 내용인 경우 HttpServletResponse 객체의 sendRedirect()를 호출한다.

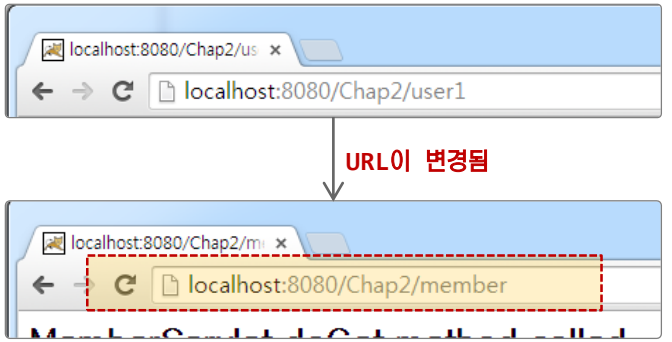
- sendRedirect() 메소드는 HTTP 응답 메시지에 응답코드 302와 Location 헤더에 URL을 설정한다.

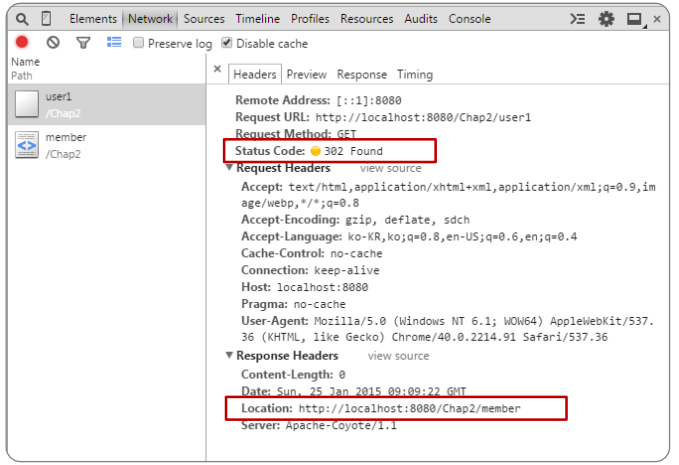
✓ 웹 브라우저는 응답을 받은 후 응답코드가 302 임을 확인하고 Location 헤더에 설정된 URL로 다시 요청한다.

- 사용자는 웹 브라우저의 URL 입력필드에 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

✓ RequestDispatch 방식은 요청을 다른 URL로 보내는 작업이 서버 내부에서 일어난다.

- 요청을 받은 서블릿은 해당 요청이 다른 URL에서 처리해야 할 대상이라면 RequestDispatcher 객체의 forward()를 호출한다.
- 웹 브라우저의 URL은 변경되지 않으므로 사용자는 위임된 응답인지를 알 수 없다.
- MVC 웹 디자인 패턴에서 웹 브라우저의 요청을 받은 서블릿(Controller)이 요청에 따른 데이터(Model)를 생성한 후, View를 구성하기 위해 JSP로 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) – RequestDispatcher

- ✓ RequestDispatcher는 클라이언트 요청을 서블릿 컨테이너에 의해 관리되는 다른 자원(Servlet, JSP, HTML 등)으로 포워드(forward) 시키거나, 다른 자원의 실행 결과를 현재 서블릿으로 포함(include)시키고자 할 때 사용할 수 있다.
- ✓ 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 {
     // 포워드 URL에 데이터 전달을 위해 request 객체에 속성을 설정할 수 있다.
     // 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 유지하기 (1/6)

✓ 서블릿은 HTTP 프로토콜을 사용하기 때문에 웹 브라우저와 웹 서버와의 연결이 지속적이지 않다(Connectionless)

- 따라서 동일한 사용자가 동일 URL을 여러 번 요청 하더라도 웹 서버는 요청이 같은 사용자가 보낸 것인지 알 수 없다.
- 때문에 사용자 정보를 유지해야 하는 웹 애플리케이션 개발 시 많은 어려움이 따른다(예: 로그인 상태, 쇼핑몰 장바구니 등)

✓ 사용자(브라우저) 상태 정보를 서버에 저장하는지 또는 클라이언트에 저장하는지에 따라 2가지 방식이 있다.

✓ 세션(HttpSession)

- 웹 서버에 개별 클라이언트 상태 정보를 저장하는 방식이다.

✓ 쿠키(Cookie)

- 웹 클라이언트에 상태 정보를 저장하는 방식이다.

2.4 Servlet API (13/16) – 클라이언트 상태 정보 유지하기 (2/6)

✓ 세션(HttpSession)

- 웹 서버에 개별 클라이언트(사용자) 상태 정보를 저장하는 방식이다.
- HttpSession 객체에 키와 값의 쌍으로 사용자 상태 정보를 저장할 수 있다.

✓ 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 유지하기 (3/6)

✓ 쿠키(Cookie)

- 웹 클라이언트에 사용자 상태 정보를 저장하는 방식이다.
- 쿠키는 사용자가 방문한 웹 서버에서 웹 브라우저에 전송하는 ' 일정한 형식이 있는 작은 텍스트 데이터 ' 이다.
  - 예) 웹 서버 방문 시 사용자 방문 기록 등 사용자 관련 정보들이 저장된 텍스트 데이터
- 웹 서버에서 사용자 상태 정보를 유지하기 위해 응답 메시지의 헤더에 쿠키 설정하여 전송하고
- 웹 브라우저는 웹 서버 방문 시 요청 메시지 헤더에 쿠키를 포함시켜 전송한다.

✓ 쿠키(Cookie) 구성 요소

- 쿠키 이름과 값, 도메인, 패스, 유효시간이 포함된다.

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 생성한 패스
domain=domain_name	쿠키가 유효하게 사용될 수 있는 URL 도메인, 생략 시 쿠키를 생성한 도메인

✓ 쿠키 제약 사항

- 쿠키 별 최대 4KB의 텍스트 데이터 만 저장할 수 있다.
- 도메인당 20까지 저장 가능하며, 클라이언트마다 최대 300개의 쿠키를 생성할 수 있다.

2.4 Servlet API (14/16) – 클라이언트 상태 정보 유지하기 (4/6)

✓ 웹 서버에서 웹 브라우저로 응답 메시지 헤더를 통해 전송하는 쿠키 구조

```
Set-Cookie : name=value; expires=date; path=path; domain=domain
```

✓ 웹 브라우저에서 웹 서버로 요청 메시지 헤더를 통해 전송하는 쿠키 구조

```
Cookie : name1=value1; name2=value2 ; name3=value3
```

2.4 Servlet API (15/16) – 클라이언트 상태 정보 유지하기 (5/6)

✓ 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 유지하기 (6/6)

✓ 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;
          }
     }
}
```

2.4 Servlet 간 데이터 공유 방법 3가지 – 정리

✓ HttpServletRequest, HttpSession, ServletContext

이름	용도 및 데이터 공유 가능 범위	라이프 사이클
HttpServletRequest	웹 브라우저부터의 요청을 서블릿으로 전달하는 객체 해당 request를 전달받는 서블릿에서만 공유 가능	웹 브라우저 요청에 대해 response 되기 전까지 살아 있다.
HttpSession	웹 브라우저 사용자 상태 정보를 유지하는 역할 동일한 웹 브라우저에서 요청한 모든 서블릿 사이에서 데이터 공유가 가능	웹 브라우저가 종료되기 전까지 살아 있다.
ServletContext	서블릿 컨테이너에서 관리되는 서블릿 사이를 연결하는 역할 웹 브라우저와 상관 없이 모든 서블릿 사이에서 데이터 공유가 가능	웹 서버가 종료되기 전까지 살아 있다.

✓ 위 3개의 객체는 데이터를 공유할 수 있도록 3개의 동일한 메소드를 제공한다.

- setAttribute(" 키 " , " 값 ");
- getAttribute(" 키 ");
- removeAttribute(" 키 ");

2.5 파일 업로드 (1/4)

✓ 파일 업로드를 위한 HTML FORM

- METHOD
  - GET : 생략 시 디폴트 요청방식으로 200바이트 이하 데이터를 URL 쿼리스트링을 통해 정보 전달한다.
  - POST : 응답메시지의 바디에 파일 데이터를 포함하여 전달한다 (파일 업로드 시 반드시 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= “upfile” />
   <input type= “text” name= “uploader” />
   <input type= “submit” value= “파일 업로드” >
</form>
```

2.5 파일 업로드 (2/4)

✓ multipart/form-data로 업로드 된 파일 확인하기

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    request.setCharacterEncoding("utf-8");
    String writer = request.getParameter("uploader");
    String file = request.getParameter("upfile");
    System.out.println("업로더 : " + writer);
    System.out.println("파일 : " + 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 (3/4) – @MultipartConfig 에노테이션

✓ Servlet 3.0 부터 MultiPart 형식의 파일 업로드를 위한 @MultipartConfig 에노테이션을 제공한다.

- @MultipartConfig 에노테이션을 서블릿 클래스에 선언하면 Multipart 요청을 처리할 수 있다.
- @MultipartConfig 에노테이션의 속성으로 업로드 관련 설정을 할 수 있다.
- 서블릿의 doPost() 메소드에서 업로드 된 파일 정보를 Part 객체를 통해 조회할 수 있다.

서블릿에 파일 업로드 설정 추가하기

```
@MultipartConfig(
    fileSizeThreshold = 1024 * 1024 * 1,
    maxFileSize       = 1024 * 1024 * 10,
    maxRequestSize    = 1024 * 1024 * 15,
    location          = "c:/upload"
)
public class FileUploadServlet extends HttpServlet {
...
}
```

업로드 된 파일의 크기가 1MB가 넘는 경우 c:/upload 경로에 임시파일로 저장한다. 업로드 파일은 최대 10MB를 초과할 수 없고, 요청 파라메터 포함 요청의 최대 크기는 15MB를 초과할 수 없다.

속성	설명	디폴트 값
fileSizeThreshold	업로드 된 파일의 크기가 이 값보다 큰 경우 임시 폴더에 파일로 생성하고 아닌 경우 메모리 상에서 처리됨	0
maxFileSize	업로드 가능한 최대 파일크기 (바이트 수)	무제한
maxRequestSize	업로드 파일을 포함한 요청 메시지 최대크기	무제한
location	임시로 파일을 저장할 폴더로 절대경로만 가능	""

@MultipartConfig 어노테이션에 있는 모든 속성은 생략 가능하다.
이러한 속성 값들은 성능에 직접적인 영향을 주는 부분이므로 웹 애플리케이션이 구동되는 시스템 환경을 잘 고려하여 설정되어야 한다.

2.5 파일 업로드 (4/4) – Part 객체 사용하기

✓ @MultipartConfig 에노테이션이 선언된 서블릿은 Part 객체를 통해 업로드 된 파일을 처리할 수 있다.

- Part 객체는 request 객체로부터 HTML 폼에서 지정한 file 입력 폼의 name 값으로 조회한다.
- Part 객체의 write() 메소드를 이용하여 업로드 된 파일을 서버의 특정 패스에 저장할 수 있다.
- Part 객체에는 업로드 파일의 정보를 조회할 수 있는 다양한 메소드를 제공한다.

Part 객체를 사용한 업로드 파일 저장

```
@MultipartConfig
@WebServlet("/upload.do")
public class FileUploadServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

        // 1. retrieve part object
        Part part = req.getPart("upfile");

        // 2. save folder
        File file = new File("d:/upload/");
        if (!file.exists()) {
            file.mkdirs();
        }

        // 3. save file
        part.write("d:/upload/uploadfile");
    }
}
```

메소드	설명
getSize() : long	현재 파트의 사이즈 반환
getContentType() : String	웹 브라우저에서 전달된 contentType 반환
write(String) : void	업로드 된 파트를 디스크에 저장함
getInputStream() : InputStream	파일의 컨텐츠를 조회할 수 있는 입력 스트림을 반환
getHeader(String) : String	지정한 파트의 헤더 값을 문자열로 반환
getHeaders(String) : Collection<String>	지정한 파트의 모든 헤더 값을 Collection 으로 반환
getName() : String	현재 파트에 대한 multipart form의 이름 반환
delete()	현재 파트와 관련된 모든 임시 파일을 삭제

2.6 업로드 디렉터리의 파일 목록 조회 (1/2)

✓ File 클래스 활용

```java
/**
 * 업로드 파일 목록 조회
 */
@WebServlet("/list")
public class FileListServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private String uploadPath = "c:/ezen-academy/fileStorage";

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throw

        File uploadDir = new File(uploadPath);
        if(!uploadDir.exists()) {
            uploadDir.mkdirs();
        }
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<meta charset='utf-8'>");
        out.println("<title>파일 목록</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>업로드 파일 목록</h2>");
```

2.6 업로드 디렉터리의 파일 목록 조회 (2/2)

```
        out.println("<table border='1'>");
        out.println("<tr>");
        out.println("<th>번호</th><th>파일이름</th><th>파일크기</th>");
        out.println("</tr>");

        File directory = new File(uploadPath);
        File[] fileList = directory.listFiles();
        for (int i=0; i<fileList.length; i++) {
            File file = fileList[i];
            if(file.isFile()) {
                String fileName = file.getName();
                //long fileSize = file.length();
                int fileSize = (int)Math.ceil(file.length()/1024.0);
                out.println("<tr>");
                out.println("<td>"+(i+1)+"</td><td><a href='download?file="+fileName+"'>
                out.println("</tr>");
            }
        }
        out.println("</table>");
    }
}
```

2.7 파일 다운로드 (1/3)

✓ **방법 1) 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 서버의 웹(PUBLIC) 디렉토리 하위에 업로드 디렉터리를 만들어야 하므로 웹 디렉터리 구조가 노출되는 보안상의 문제가 발생 할 수 있다.

✓ **방법 2) Servlet에서 응답 헤더 설정을 통해 다운로드 직접 구현**

- 웹 서버 디렉터리 구조가 노출되지 않는 장점이 있다.

2.7 파일 다운로드 (2/3)

✓ FileDownloadServlet.java

```java
/**
 * 파일 다운로드
 */
@WebServlet("/download")
public class FileDownloadServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private String uploadPath = "c:/ezen-academy/fileStorage";

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws S
        process(request, response);
    }

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

    public void process(HttpServletRequest request, HttpServletResponse response) throws
        // 파라메터로 넘어오는 파일이름 얻기
        String fileName = request.getParameter("file");
        String filePath = uploadPath + File.separator + fileName;
        File file = new File(filePath);
```

2.7 파일 다운로드 (3/3)

```
    // 파일 존재시
    if(file.exists()) {
        // 브라우저 캐시 사용 않도록 응답헤더 설정
        response.setHeader("Cache-Control", "no-cache");
        // 파일 다운로드 처리를 위한 응답헤더에 마임타입 설정
        response.setContentType("application/octet-stream; charset=utf-8");
        fileName = URLEncoder.encode(fileName, "utf-8");
        // Content-Disposition 헤더 설정
        response.setHeader("Content-Disposition", "attachment;filename=" + fileName + ";");
        // Content-Length 헤더 설정
        response.setHeader("Content-Length", String.valueOf(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);
            }
        }catch (Exception e) {
            e.printStackTrace();
        }finally{
            if(out != null) out.close();
            if(in != null)  in.close();
        }
    }else { // 파일이 없을 경우
        response.setContentType("text/html; charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<h3>" + fileName + "파일이 존재하지 않습니다.</h3>");
    }
```

End of Document

✓ Q&A

감사합니다...