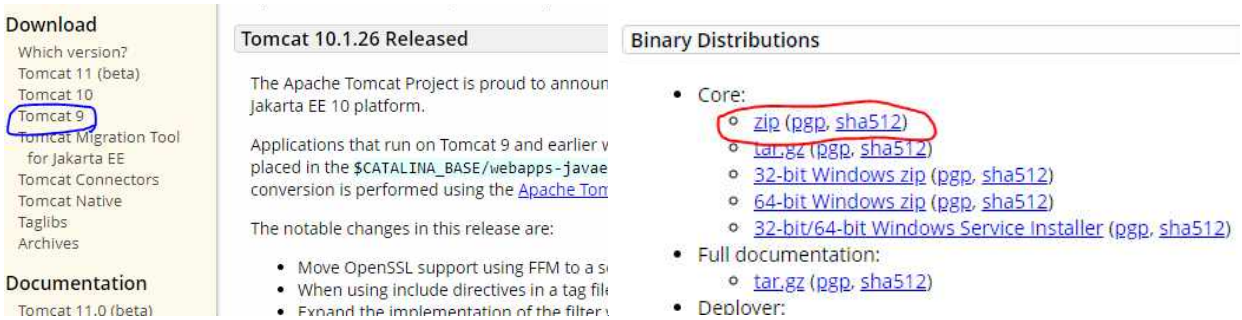
Tomcat 설치

-https://tomcat.apache.org/ (톰캣사이트)



- 다운로드 받은 파일 압출 해제
-

Web & Server Application 기초

1. URL과 웹 페이지

https://www.naver.com이나 https://www.google.com 과 같이 웹브라우저의 주소줄에 표시되는 것을 URL이라고 부른다. 웹 브라우저의 주소줄에 URL을 입력하면 웹 브라우저에 URL에 해당하는 내용이 출력되는데, 웹 브라우저에 출력된 내용을 웹 페이지라고 부른다. 웹 사이트는 이런 웹 페이지의 묶음이다.
웹 페이지의 주소를 표현하는 URL은 일반적으로 다음과 같은 구성 요소로 나눈다.

```
프로토콜://서버이름/경로?쿼리문자열
```

프로토콜 (Protocol): 웹 브라우저와 웹 서버 간 통신 규약을 정의하는 부분으로, 주로 "https://" 또는 "http://"와 같이 사용된다.

서버 주소 (서버이름 또는 도메인): 해당 웹 페이지를 호스팅하는 서버의 주소를 나타낸다. 서버 주소는 "www.naver.com" 또는 "www.google.com"과 같이 표현된다.

따라서 예시로 주어진 URL "https://www.naver.com/search?word=js"에서 프로토콜은 "https://", 서버 주소는 "www.naver.com"이 된다. 이를 설명하는 텍스트에서는 다음과 같이 표현할 수 있다.

"URL은 프로토콜과 서버 주소로 구성되며, 예를 들어 'https://www.naver.com/search?word=js'에서는 'https://'가 프로토콜이고, 'www.naver.com'이 서버 주소이다."

경로 : URL의 경로(path)는 호스트와 쿼리 문자열 사이의 부분으로, 리소스의 위치 또는 경로를 나타낸다. https://www.naver.com는 호스트이다.
/products/electronics/laptops는 경로이다.

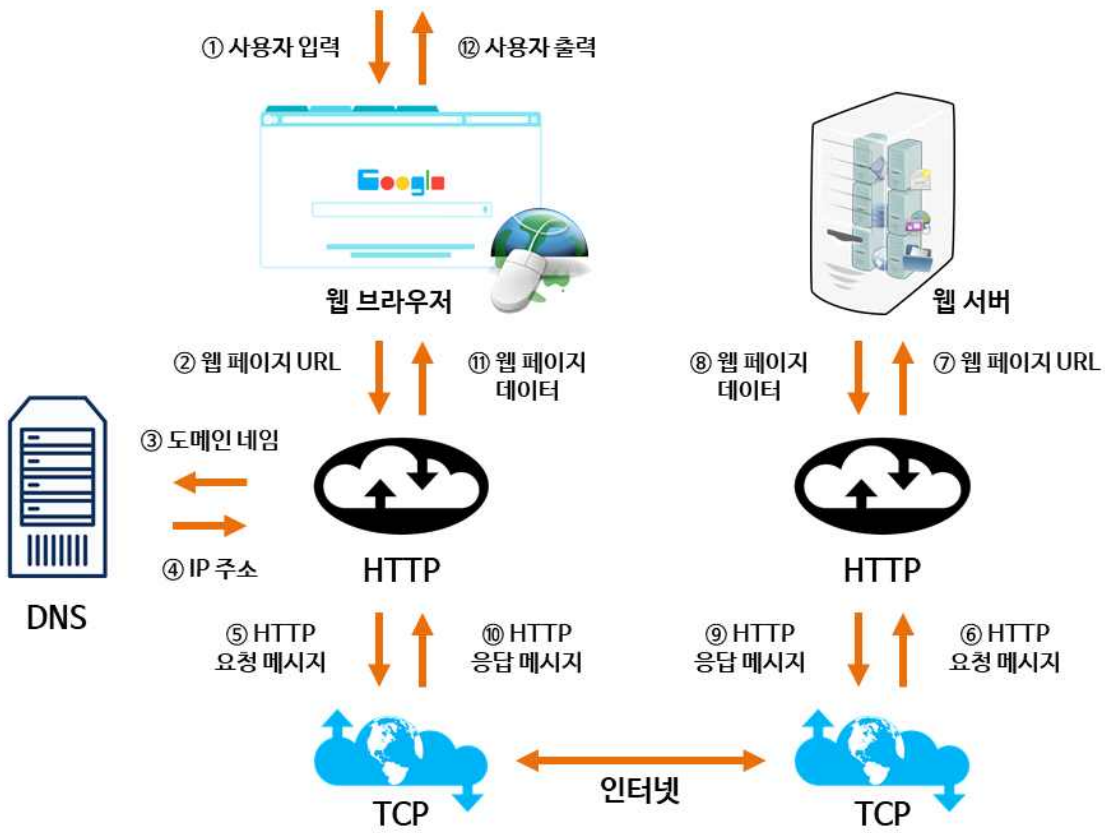
쿼리문자열 : ? 이후의 부분이 쿼리 문자열이다.
q=query, lang=en, page=1은 각각 키-값 쌍이다.
각 키와 값은 등호(=)로 구분되고, 키-값 쌍들은 앰퍼샌드(&)로 구분된다.

2. 웹 브라우저와 웹 서버

웹 브라우저와 서버 간의 통신은 주로 HTTP(HyperText Transfer Protocol)를 기반으로 이루어집니다. 아래는 웹 브라우저와 서버 간의 기본적인 통신 과정을 설명한 것입니다.

1. **사용자의 요청** : 사용자가 웹 브라우저를 통해 특정 웹페이지에 접속하거나, 어떤 동작(클릭, 양식 제출 등)을 수행합니다.
2. **URL 해석** : 브라우저는 사용자가 입력한 URL을 해석하여 호스트명, 프로토콜, 경로 등의 정보를 추출합니다.
3. **DNS 조회** : 브라우저는 호스트명을 IP 주소로 변환하기 위해 DNS(Domain Name System) 서버에 조회를 요청합니다.
4. **TCP/IP 연결** : 브라우저는 서버의 IP 주소로 TCP/IP 연결을 수립합니다. 일반적으로 3-way handshake를 통해 연결을 설정합니다.
5. **HTTP 요청 생성** : 브라우저는 HTTP 요청 메시지를 생성합니다. 이는 웹페이지 요청의 성격에 따라 GET, POST 등의 메서드를 사용할 수 있습니다. 요청 헤더에는 브라우저 정보, 쿠키, 요청하는 페이지 등의 정보가 포함될 수 있습니다.
6. **서버의 응답** : 서버는 요청을 받고, 해당 요청에 대한 응답을 생성합니다. 응답은 HTTP 상태 코드, 응답 헤더, 응답 본문 등으로 구성됩니다.
7. **HTTP 응답 전송** : 서버는 생성된 HTTP 응답을 브라우저로 전송합니다.
8. **브라우저의 렌더링** : 브라우저는 받은 응답을 해석하고, 웹페이지를 렌더링합니다. 이 과정에는 HTML, CSS, JavaScript 등의 리소스를 해석하고 화면에 표시하는 과정이 포함됩니다.
9. **TCP/IP 연결 종료** : 웹페이지가 렌더링되면 브라우저는 서버와의 TCP/IP 연결을 종료합니다.

이러한 과정을 통해 사용자는 브라우저를 통해 서버에서 제공하는 웹페이지를 요청하고, 서버는 해당 요청에 대한 응답을 보내어 웹페이지가 표시되게 됩니다.

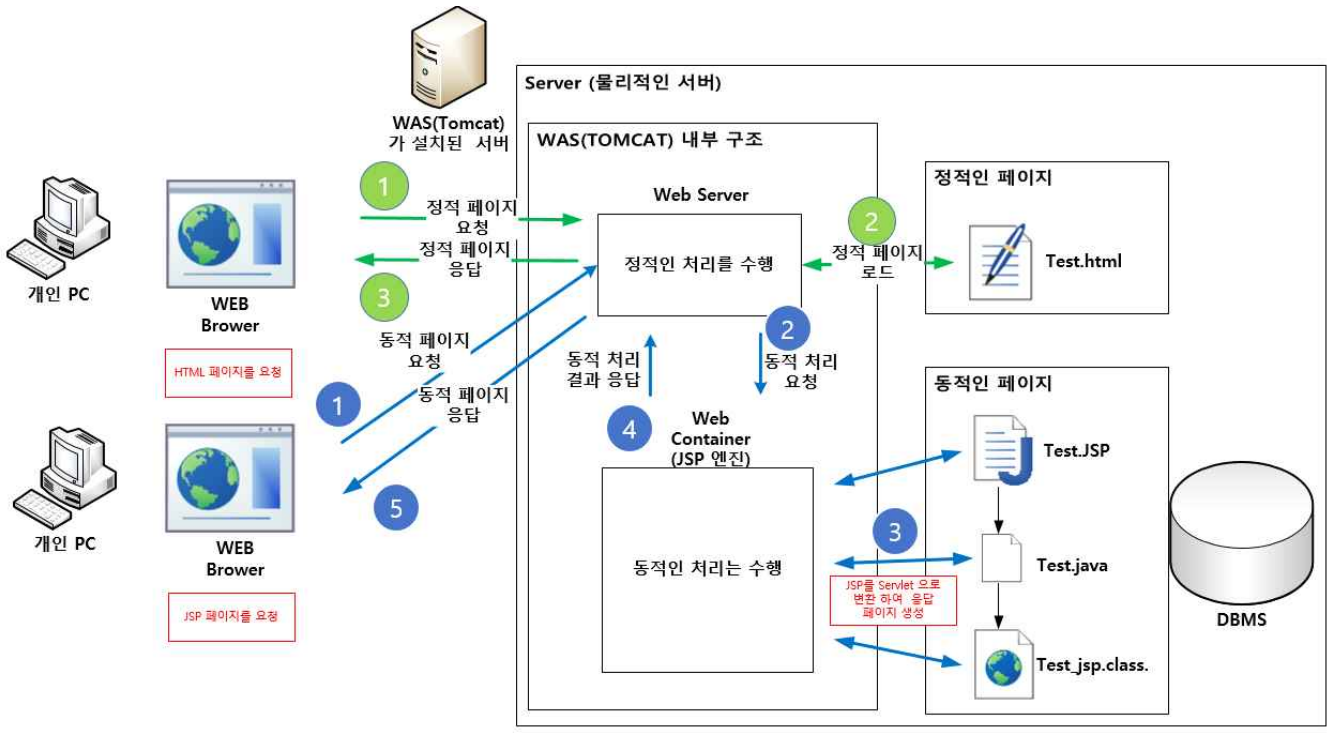


3. 아파치 톰캣

Apache Tomcat은 Java Servlet, JavaServer Pages (JSP), 그리고 Java 언어로 작성된 웹 어플리케이션을 실행하는 오픈 소스 웹 서버 및 서블릿 컨테이너입니다. 톰캣의 동작 원리는 다음과 같습니다:

1. **톰캣 서버 시작** : 사용자가 톰캣 서버를 시작하면, Java의 main 메서드를 실행하여 톰캣 서버를 초기화합니다.
2. **서버 소켓 바인딩** : 톰캣은 네트워크에서 클라이언트의 요청을 받을 수 있도록 서버 소켓을 생성하고 특정 포트에 바인딩합니다. 기본적으로 HTTP 요청은 8080 포트에서 수신됩니다.
3. **웹 애플리케이션 로딩** : 톰캣은 각각의 웹 애플리케이션(웹 어플리케이션은 WAR 파일로 패키지됨)을 로드하고 컨텍스트에 배치합니다.
4. **웹 애플리케이션 초기화** : 각 웹 애플리케이션은 자신만의 서블릿 컨테이너, 컨텍스트, 클래스로더 등을 초기화합니다.
5. **HTTP 요청 수신** : 클라이언트가 HTTP 요청을 보내면, 톰캣 서버는 해당 요청을 받아들입니다.
6. **요청 분석** : 톰캣은 요청을 분석하여 어떤 웹 애플리케이션으로 보낼지를 결정합니다.
7. **서블릿 호출** : 요청이 특정 웹 애플리케이션에 속한다면, 해당 웹 애플리케이션의 서블릿 컨테이너에서 적절한 서블릿이나 JSP로 요청을 전달합니다.
8. **서블릿 처리** : 서블릿은 요청을 처리하고 필요한 로직을 실행합니다. 이 때, 필요한 자원은 웹 애플리케이션의 클래스로더를 통해 로드됩니다.
9. **HTTP 응답 생성** : 서블릿이나 JSP는 HTTP 응답을 생성하고, 이를 클라이언트로 전송합니다.
10. **클라이언트에게 응답 전송** : 톰캣은 생성된 응답을 클라이언트에게 전송하고, 연결을 닫습니다.
11. **웹 애플리케이션 언로드** : 톰캣은 요청 처리가 끝난 후에 해당 웹 애플리케이션에서 사용된 자원을 해제하고, 필요한 경우 웹 애플리케이션을 언로드합니다.

톰캣은 이러한 방식으로 여러 웹 애플리케이션을 동시에 실행하고, 각각을 격리된 환경에서 실행할 수 있는 웹 컨테이너 역할을 합니다.



3.1 서블릿 컨테이너

서블릿 컨테이너는 웹 애플리케이션을 실행하고 관리하는 소프트웨어입니다. 여기서 웹 애플리케이션은 자바 서블릿과 JSP(JavaServer Pages)를 포함한 웹 어플리케이션을 의미합니다. 톰캣은 이러한 웹 애플리케이션을 처리하는 역할을 수행합니다.

간단하게 설명하면:

서블릿 컨테이너 역할:

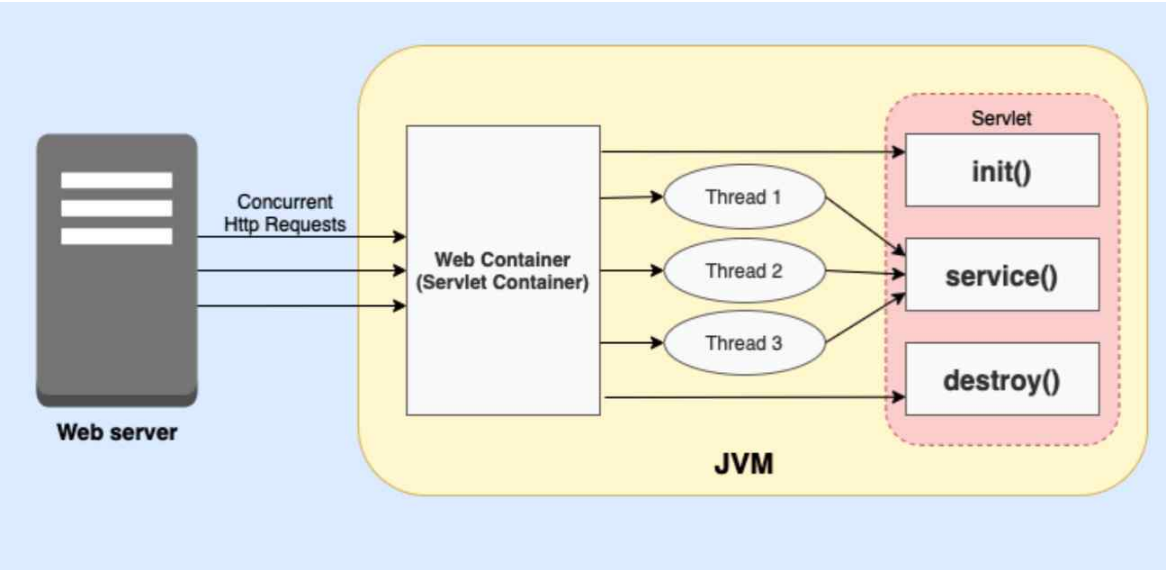
톰캣은 자바 서블릿을 실행하기 위한 서블릿 컨테이너를 제공합니다. 서블릿은 동적인 웹 페이지를 생성하고 관리하는 자바 클래스입니다. 톰캣은 이러한 서블릿을 실행하고 요청에 따라 적절한 서블릿을 호출합니다.
JSP 처리:

JSP는 HTML 코드 안에 자바 코드를 넣어 동적인 웹 페이지를 만들 수 있는 기술입니다. 톰캣은 JSP를 처리하여 동적인 컨텐츠를 생성하고 웹 애플리케이션에 표시합니다.
웹 애플리케이션 관리:

톰캣은 여러 개의 웹 애플리케이션을 동시에 실행하고, 각각의 애플리케이션을 격리된 환경에서 실행할 수 있도록 관리합니다. 웹 애플리케이션의 배포, 시작, 중지, 제거 등을 담당합니다.
HTTP 요청 처리:

클라이언트로부터 오는 HTTP 요청을 받아들이고, 해당 요청을 처리하는데 필요한 서블릿이나 JSP 등을 호출하여 동적인 응답을 생성합니다.
정적 리소스 제공:

톰캣은 HTML, CSS, 이미지와 같은 정적인 리소스를 제공하는데 사용됩니다. 이러한 리소스들은 웹 애플리케이션의 일부 또는 별도의 디렉터리에서 관리됩니다.
톰캣 컨테이너는 웹 애플리케이션을 실행하는 데 필요한 환경을 제공하며, 다양한 웹 기술을 지원하여 동적이고 유연한 웹 애플리케이션을 개발하고 배포할 수 있게 해줍니다.



3.2 서블릿

서블릿(Servlet)은 Java 언어로 작성된 웹 애플리케이션의 구성 요소 중 하나로, 동적인 웹 페이지를 생성하고 관리하는 자바 클래스입니다. 서블릿은 서버 측에서 동작하며, 클라이언트의 요청에 따라 동적으로 응답을 생성합니다.

서블릿의 특징과 동작 방식을 자세히 설명해보겠습니다:

1. 서블릿의 특징
   가. Java 기반 : 서블릿은 Java 언어로 작성되어 있습니다. Java의 강력한 객체 지향 프로그래밍 기능을 활용할 수 있습니다.
   나. 플랫폼 독립성 : Java의 특성에 따라 서블릿은 플랫폼 독립적이며, 어떤 웹 서버에서든 실행될 수 있습니다.
   다. 웹 애플리케이션의 일부 : 서블릿은 주로 웹 애플리케이션 내에서 사용되며, 동적인 웹 페이지 생성, 데이터 처리, 비즈니스 로직 구현 등에 활용됩니다.
   라. 라이프사이클 관리 : 서블릿은 라이프사이클을 가지며, 초기화, 서비스 처리, 소멸 등의 단계를 거칩니다. 이를 통해 서블릿은 효율적으로 관리됩니다.
   마. HTTP 프로토콜 지원 : 주로 HTTP 프로토콜을 통해 클라이언트와 통신하며, 웹 브라우저로부터의 요청에 대한 응답을 생성합니다.

2. 서블릿의 동작 방식
   가. 클라이언트 요청 (Request) :
   클라이언트가 웹 브라우저를 통해 URL을 입력하거나 폼을 제출하여 HTTP 요청을 보냅니다. 이 요청은 웹 서버(Apache Tomcat)로 전달됩니다.
   나. 서블릿 컨테이너와 서블릿 매핑
   웹 서버는 서블릿 컨테이너에 요청을 전달합니다.
   서블릿 컨테이너는 요청된 URL과 매핑된 서블릿을 찾습니다. 이 매핑 정보는 web.xml 파일이나 애노테이션(@WebServlet)을 통해 정의됩니다.
   다. 서블릿 인스턴스 생성 및 초기화 (Initialization)
   서블릿 컨테이너는 서블릿의 인스턴스가 존재하지 않으면 이를 생성하고, init() 메서드를 호출하여 초기화 작업을 수행합니다.
   init() 메서드는 서블릿이 처음 생성될 때 한 번만 호출되며, 서블릿이 필요한 초기화 작업(예: 자원 설정 등)을 수행합니다.
   라. 요청 처리 (Request Handling)
   서블릿 컨테이너는 클라이언트의 요청을 받아 service() 메서드를 호출합니다.
   service() 메서드는 요청의 HTTP 메서드(GET, POST, PUT, DELETE 등)에 따라 적절한 메서드(doGet(), doPost() 등)를 호출하여 요청을 처리합니다.
   예를 들어, HTTP GET 요청이면 doGet() 메서드가 호출되고, HTTP POST 요청이면 doPost() 메서드가 호출됩니다.
   마. 응답 생성 (Response Generation)
   요청을 처리한 후, 서블릿은 HttpServletResponse 객체를 통해 클라이언트에게 응답을 보냅니다.
   응답에는 상태 코드, 헤더, 콘텐츠 타입, 실제 응답 데이터(HTML, JSON 등)가 포함될 수 있습니다.
   바. 클라이언트로 응답 전송 (Response Transmission)
   서블릿 컨테이너는 서블릿이 생성한 응답을 웹 서버로 전달하고, 웹 서버는 이를 클라이언트에게

전송합니다.

클라이언트는 응답을 받아 웹 브라우저에 표시하거나 다른 방식으로 처리합니다.

사. 서블릿 종료 및 소멸 (Termination and Destruction)

서블릿이 더 이상 필요하지 않으면, 서블릿 컨테이너는 서블릿의 destroy() 메서드를 호출하여 자원을 해제하고, 서블릿 인스턴스를 소멸시킵니다.

destroy() 메서드는 서블릿이 소멸될 때 한 번만 호출되며, 서블릿이 사용하던 자원(예: 데이터베이스 연결 등)을 정리하는 작업을 수행합니다.

JSP 기초

1. JSP 기본 코드 구조

JSP 페이지는 HTML 문서를 생성하는 데 사용되며, 다음은 JSP의 기본 코드 구조입니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>JSP HTML 생성 예제</title>
</head>
<body>

    <%-- Java 코드를 여기에 삽입 --%>

    <h1>Hello, JSP!</h1>

    <%-- 또 다른 Java 코드를 여기에 삽입 --%>

</body>
</html>
```

2. JSP 페이지의 구성요소

2.1 디렉티브(Directive)

디렉티브는 JSP 페이지의 속성을 지정하는데 사용되며 <%@ %>태그로 시작합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
```

2.2 스크립트 요소

JSP 페이지 내에 Java 코드를 삽입하기 위한 스크립트 요소는 다음과 같습니다.

2.2.1 스크립트릿(Scriptlet)

Java 코드를 실행하는 데 사용됩니다.

```
<%
    // Java 코드 작성
    String message ="Hello, JSP!";
    out.println(message);
%>
```

2.2.2 표현식(Expression)

값을 출력하기 위해 사용됩니다.

```
<%=message%>
```

2.2.3 선언부(Declaration)

전역 변수와 메소드를 선언하는 데 사용됩니다.

```
<%! int count =0; %>
```

2.3 내장 객체

JSP 페이지에서 사용할 수 있는 기본 객체는 다음과 같습니다.

2.3.1 request 객체

request 객체는 클라이언트의 요청 정보를 포함합니다. 주로 HTTP 요청 파라미터, 헤더 정보, 세션 정보 등에 접근하는 데 사용됩니다.

메서드

String getParameter(String name): 요청 파라미터의 값을 반환합니다.
Enumeration<String> getParameterNames(): 모든 요청 파라미터 이름을 반환합니다.
String[] getParameterValues(String name): 요청 파라미터의 모든 값을 배열로 반환합니다.
Map<String, String[]> getParameterMap(): 모든 요청 파라미터와 그 값을 맵으로 반환합니다.
String getHeader(String name): 지정된 요청 헤더의 값을 반환합니다.
Enumeration<String> getHeaderNames(): 모든 요청 헤더 이름을 반환합니다.
Cookie[] getCookies(): 클라이언트가 전송한 쿠키 배열을 반환합니다.
HttpSession getSession(): 현재 세션을 반환합니다. 세션이 없으면 새로 생성합니다.

HttpSession getSession(boolean create): 현재 세션을 반환합니다. create가 true인 경우, 세션이 없으면 새로 생성합니다.
ServletInputStream getInputStream(): 요청 본문을 읽기 위한 입력 스트림을 반환합니다.
BufferedReader getReader(): 요청 본문을 읽기 위한 버퍼드 리더를 반환합니다.
String getMethod(): HTTP 메서드(GET, POST 등)를 반환합니다.
String getRequestURI(): 요청된 URI를 반환합니다.
StringBuffer getRequestURL(): 요청된 URL을 반환합니다.
String getContextPath(): 컨텍스트 경로를 반환합니다.
String getServletPath(): 서블릿 경로를 반환합니다.
String getPathInfo(): 추가 경로 정보를 반환합니다.
String getQueryString(): 요청의 쿼리 문자열을 반환합니다.
String getRemoteAddr(): 클라이언트의 IP 주소를 반환합니다.
String getRemoteHost(): 클라이언트의 호스트 이름을 반환합니다.
int getRemotePort(): 클라이언트가 사용한 포트를 반환합니다.
String getServerName(): 서버의 이름을 반환합니다.
int getServerPort(): 서버의 포트를 반환합니다.
String getScheme(): 요청의 스킴(프로토콜)을 반환합니다.
String getProtocol(): 요청의 프로토콜을 반환합니다.
RequestDispatcher getRequestDispatcher(String path): 요청을 다른 리소스로 포함하거나 포워드하는 디스패처를 반환합니다.
String getCharacterEncoding(): 요청의 문자 인코딩을 반환합니다.
void setCharacterEncoding(String env): 요청의 문자 인코딩을 설정합니다.
int getContentLength(): 요청 본문의 길이를 반환합니다.
String getContentType(): 요청 본문의 MIME 타입을 반환합니다.
void setAttribute(String name, Object o): 요청 속성을 설정합니다.
Object getAttribute(String name): 요청 속성을 반환합니다.
Enumeration<String> getAttributeNames(): 요청 속성 이름의 열거형을 반환합니다.
void removeAttribute(String name): 요청 속성을 제거합니다.

[redacted]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Request 객체 예제</title>
</head>
<body>
    <h1>Request 객체 예제</h1>
    <%
        String userName = request.getParameter("userName");
        String userAgent = request.getHeader("User-Agent");
        String clientIP = request.getRemoteAddr();
    %>
    <p>사용자 이름: <%= userName %></p>
    <p>사용자 에이전트: <%= userAgent %></p>
    <p>클라이언트 IP: <%= clientIP %></p>
```

```
</body>
</html>
```

2.3.2 response 객체

response 객체는 서버의 응답 정보를 포함합니다. 주로 HTTP 응답 헤더 설정, 쿠키 전송, 리다이렉트 등에 사용됩니다.

메서드

void addCookie(Cookie cookie): 응답에 쿠키를 추가합니다.
boolean containsHeader(String name): 응답에 특정 헤더가 포함되어 있는지 여부를 반환합니다.
String encodeURL(String url): URL을 인코딩하여 세션 ID를 포함시킵니다.
String encodeRedirectURL(String url): 리다이렉트 URL을 인코딩하여 세션 ID를 포함시킵니다.
void sendError(int sc, String msg): 지정된 상태 코드와 메시지를 사용하여 에러 응답을 전송합니다.
void sendError(int sc): 지정된 상태 코드로 에러 응답을 전송합니다.
void sendRedirect(String location): 클라이언트를 다른 URL로 리다이렉트합니다.
void setDateHeader(String name, long date): 응답 헤더에 날짜 값을 설정합니다.
void addDateHeader(String name, long date): 응답 헤더에 날짜 값을 추가합니다.
void setHeader(String name, String value): 응답 헤더에 문자열 값을 설정합니다.
void addHeader(String name, String value): 응답 헤더에 문자열 값을 추가합니다.
void setIntHeader(String name, int value): 응답 헤더에 정수 값을 설정합니다.
void addIntHeader(String name, int value): 응답 헤더에 정수 값을 추가합니다.
void setStatus(int sc): 응답의 상태 코드를 설정합니다.
int getStatus(): 응답의 상태 코드를 반환합니다.
String getHeader(String name): 응답 헤더의 값을 반환합니다.
Collection<String> getHeaders(String name): 응답 헤더의 모든 값을 반환합니다.
Collection<String> getHeaderNames(): 응답 헤더의 모든 이름을 반환합니다.
void setContentLength(int len): 응답의 콘텐츠 길이를 설정합니다.
void setContentLengthLong(long len): 응답의 콘텐츠 길이를 설정합니다 (long 타입).
void setContentType(String type): 응답의 콘텐츠 타입을 설정합니다.
void setCharacterEncoding(String charset): 응답의 문자 인코딩을 설정합니다.
PrintWriter getWriter(): 응답 본문을 작성하기 위한 출력 스트림을 반환합니다.
ServletOutputStream getOutputStream(): 응답 본문을 작성하기 위한 출력 스트림을 반환합니다.
void setBufferSize(int size): 응답 버퍼의 크기를 설정합니다.
int getBufferSize(): 응답 버퍼의 크기를 반환합니다.
void flushBuffer(): 응답 버퍼를 플러시합니다.
void resetBuffer(): 응답 버퍼를 리셋합니다.
boolean isCommitted(): 응답이 커밋되었는지 여부를 반환합니다.
void reset(): 응답을 리셋합니다.
void setLocale(Locale loc): 응답의 로케일을 설정합니다.
Locale getLocale(): 응답의 로케일을 반환합니다.

[redacted]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="UTF-8">
    <title>Response 객체 예제</title>
</head>
<body>
    <h1>Response 객체 예제</h1>
    <%
        response.setContentType("text/html; charset=UTF-8");
        response.addHeader("Custom-Header", "CustomHeaderValue");
        response.setStatus(HttpServletResponse.SC_OK);

        response.sendRedirect("경로");
    %>
    <p>응답 헤더와 상태 코드가 설정되었습니다.</p>
</body>
</html>
```

2.3.3. out 객체

클라이언트에게 출력할 데이터를 쓰는 데 사용되는 객체입니다. JSP 페이지의 HTML 콘텐츠를 생성하는 데 사용됩니다.

메서드
print(String s): 문자열을 출력함
println(String s): 문자열을 출력하고 줄바꿈 함
flush(): 버퍼를 비움

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Out 객체 예제</title>
</head>
<body>
    <h1>Out 객체 예제</h1>
    <%
        out.println("Hello, World!");
        out.println("JSP에서 데이터를 출력합니다.");
    %>
</body>
</html>
```

2.3.4 page 객체

page 객체는 JSP 페이지 자체를 나타내며, 일반적으로 this 키워드와 동일합니다. JSP 페이지의 모든 멤버와 메서드를 참조할 수 있습니다. 이는 특별한 메서드를 제공하지 않지만 JSP 내에서 현재 페이지 객체에 접근할 수 있습니다.

2.3.5 pageContext 객체

PageContext 객체는 JSP 페이지 내에서 다양한 범위의 속성에 접근하고 조작할 수 있는 메서드를 제공합니다. 모든 내장 객체에 접근할 수 있는 메서드를 포함합니다.

메서드

void initialize(...): 페이지 컨텍스트를 초기화합니다. 일반적으로 컨테이너에서 호출합니다.
void release(): 페이지 컨텍스트를 해제합니다. 일반적으로 컨테이너에서 호출합니다.
void setAttribute(String name, Object attribute): 페이지 범위에 속성을 설정합니다.
void setAttribute(String name, Object attribute, int scope): 특정 범위에 속성을 설정합니다.
Object getAttribute(String name): 페이지 범위에서 속성을 가져옵니다.
Object getAttribute(String name, int scope): 특정 범위에서 속성을 가져옵니다.
Object findAttribute(String name): 모든 범위에서 속성을 검색하여 가져옵니다.
void removeAttribute(String name): 페이지 범위에서 속성을 제거합니다.
void removeAttribute(String name, int scope): 특정 범위에서 속성을 제거합니다.
int getAttributesScope(String name): 속성이 존재하는 범위를 반환합니다.
Enumeration<String> getAttributeNamesInScope(int scope): 특정 범위에서 속성 이름을 반환합니다.
JspWriter getOut(): 현재 JSP 출력 스트림을 반환합니다.
HttpSession getSession(): 현재 세션을 반환합니다.
Object getPage(): 현재 페이지 객체를 반환합니다.
ServletRequest getRequest(): 현재 요청 객체를 반환합니다.
ServletResponse getResponse(): 현재 응답 객체를 반환합니다.
Exception getException(): 현재 예외 객체를 반환합니다.
ServletConfig getServletConfig(): 서블릿 구성 객체를 반환합니다.
ServletContext getServletContext(): 서블릿 컨텍스트를 반환합니다.
void forward(String relativeUrlPath): 지정된 URL로 요청을 포워드합니다.
void include(String relativeUrlPath): 지정된 URL의 출력을 포함합니다.
void handlePageException(Exception e): 페이지 예외를 처리합니다.
void handlePageException(Throwable t): 페이지 예외를 처리합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>PageContext 객체 예제</title>
</head>
<body>
    <h1>PageContext 객체 예제</h1>
    <%
        pageContext.setAttribute("myAttribute", "Hello, PageContext!");
```

```
        String myAttribute = (String) pageContext.getAttribute("myAttribute");
    %>
    <p>PageContext 속성 값: <%= myAttribute %></p>

     if( pageContext.getSession().getAttribute("part")!=null ){
        pageContext.include("경로");
     }else{
        pageContext.forward("경로");
     }

</body>
</html>
```

2.3.6 config 객체

config 객체는 서블릿 초기화 파라미터와 서블릿 컨텍스트에 접근할 수 있습니다. JSP 페이지의 서블릿 구성 정보를 담고 있습니다.

메서드
String getServletName(): 서블릿의 이름을 반환합니다.
ServletContext getServletContext(): 서블릿 컨텍스트를 반환합니다.
String getInitParameter(String name): 초기화 파라미터 값을 반환합니다.
Enumeration<String> getInitParameterNames(): 초기화 파라미터 이름의 열거형을 반환합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Config 객체 예제</title>
</head>
<body>
    <h1>Config 객체 예제</h1>
    <%
        String servletName = config.getServletName();
        ServletContext context = config.getServletContext();
    %>
    <p>서블릿 이름: <%= servletName %></p>
    <p>서블릿 컨텍스트: <%= context %></p>
</body>
</html>
```

2.3.7 session 객체

session 객체는 세션 범위의 데이터를 저장하고 접근할 수 있습니다. 각 클라이언트에 대해 세션이 생성됩니다.

메서드

long getCreationTime(): 세션이 생성된 시간을 반환합니다.
String getId(): 세션 ID를 반환합니다.
long getLastAccessedTime(): 세션에 마지막으로 접근한 시간을 반환합니다.
ServletContext getServletContext(): 서블릿 컨텍스트를 반환합니다.
void setMaxInactiveInterval(int interval): 세션의 유효 시간을 설정합니다.
int getMaxInactiveInterval(): 세션의 유효 시간을 반환합니다.
HttpSessionContext getSessionContext(): 세션 컨텍스트를 반환합니다 (이 메서드는 더 이상 사용되지 않습니다).
Object getAttribute(String name): 세션 속성을 반환합니다.
Enumeration<String> getAttributeNames(): 세션 속성 이름의 열거형을 반환합니다.
void setAttribute(String name, Object value): 세션 속성을 설정합니다.
void removeAttribute(String name): 세션 속성을 제거합니다.
void invalidate(): 세션을 무효화합니다.
boolean isNew(): 세션이 새로운지 여부를 반환합니다.

[redacted]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Session 객체 예제</title>
</head>
<body>
    <h1>Session 객체 예제</h1>
    <%
        session.setAttribute("sessionAttribute", "Hello, Session!");
        String sessionAttribute = (String) session.getAttribute("sessionAttribute");
    %>
    <p>세션 속성 값: <%= sessionAttribute %></p>
    <p>세션 ID: <%= session.getId() %></p>
</body>
</html>
```

2.3.8 application 객체

application 객체는 애플리케이션 범위의 데이터를 저장하고 접근할 수 있습니다. 모든 사용자와 JSP 페이지에서 공유됩니다.

메서드

String getContextPath(): 컨텍스트 경로를 반환합니다.

ServletContext getContext(String uripath): 지정된 경로의 서블릿 컨텍스트를 반환합니다.

int getMajorVersion(): 서블릿 API의 주요 버전을 반환합니다.

int getMinorVersion(): 서블릿 API의 부 버전을 반환합니다.

int getEffectiveMajorVersion(): 서블릿 API의 실제 주요 버전을 반환합니다.

int getEffectiveMinorVersion(): 서블릿 API의 실제 부 버전을 반환합니다.

String getMimeType(String file): 파일의 MIME 타입을 반환합니다.

Set<String> getResourcePaths(String path): 지정된 경로의 리소스 경로 집합을 반환합니다.

URL getResource(String path): 지정된 경로의 리소스를 반환합니다.

InputStream getResourceAsStream(String path): 지정된 경로의 리소스를 입력 스트림으로 반환합니다.

RequestDispatcher getRequestDispatcher(String path): 지정된 경로의 요청 디스패처를 반환합니다.

RequestDispatcher getNamedDispatcher(String name): 지정된 이름의 요청 디스패처를 반환합니다.

Servlet getServlet(String name): 지정된 이름의 서블릿을 반환합니다 (이 메서드는 더 이상 사용되지 않습니다).

Enumeration<Servlet> getServlets(): 서블릿의 열거형을 반환합니다 (이 메서드는 더 이상 사용되지 않습니다).

Enumeration<String> getServletNames(): 서블릿 이름의 열거형을 반환합니다 (이 메서드는 더 이상 사용되지 않습니다).

void log(String msg): 로그 메시지를 기록합니다.

void log(Exception exception, String msg): 예외와 로그 메시지를 기록합니다 (이 메서드는 더 이상 사용되지 않습니다).

void log(String message, Throwable throwable): 예외와 로그 메시지를 기록합니다.

String getRealPath(String path): 가상 경로의 실제 경로를 반환합니다.

String getServerInfo(): 서버 정보를 반환합니다.

String getInitParameter(String name): 초기화 파라미터 값을 반환합니다.

Enumeration<String> getInitParameterNames(): 초기화 파라미터 이름의 열거형을 반환합니다.

boolean setInitParameter(String name, String value): 초기화 파라미터 값을 설정합니다.

Object getAttribute(String name): 속성을 반환합니다.

Enumeration<String> getAttributeNames(): 속성 이름의 열거형을 반환합니다.

void setAttribute(String name, Object object): 속성을 설정합니다.

void removeAttribute(String name): 속성을 제거합니다.

String getServletContextName(): 서블릿 컨텍스트의 이름을 반환합니다.

[redacted]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Application 객체 예제</title>
</head>
<body>
    <h1>Application 객체 예제</h1>
    <%
        String serverInfo = application.getServerInfo();
```

```
        application.setAttribute("appAttribute", "Hello, Application!");
        String appAttribute = (String) application.getAttribute("appAttribute");
    %>
    <p>서버 정보: <%= serverInfo %></p>
    <p>애플리케이션 속성 값: <%= appAttribute %></p>
</body>
</html>
```

2.3.9 exception 객체

exception 객체는 JSP 페이지에서 발생한 예외를 처리하기 위한 객체입니다. 이 객체는 페이지에서 오류가 발생했을 때, 오류를 처리할 수 있는 특별한 JSP 페이지로 포워드될 때만 사용할 수 있습니다. 이러한 페이지는 일반적으로 isErrorPage="true" 페이지 지시어 속성을 가지고 있습니다.

메서드
String getMessage(): 예외의 설명 메시지를 반환합니다.
String getLocalizedMessage(): 예외의 지역화된 설명 메시지를 반환합니다.
Throwable getCause(): 예외의 원인(중첩된 예외)을 반환합니다.
Throwable initCause(Throwable cause): 예외의 원인을 설정합니다.
String toString(): 예외의 문자열 표현을 반환합니다.
void printStackTrace(): 예외의 스택 추적을 표준 오류 스트림에 출력합니다.
void printStackTrace(PrintStream s): 예외의 스택 추적을 지정된 출력 스트림에 출력합니다.
void printStackTrace(PrintWriter s): 예외의 스택 추적을 지정된 출력 라이터에 출력합니다.
StackTraceElement[] getStackTrace(): 예외의 스택 추적 요소 배열을 반환합니다.
void setStackTrace(StackTraceElement[] stackTrace): 예외의 스택 추적 요소 배열을 설정합니다.
Throwable fillInStackTrace(): 현재 스레드의 스택 추적 요소를 반환합니다.
String getMessage(): 예외의 설명 메시지를 반환합니다.
String getLocalizedMessage(): 예외의 지역화된 설명 메시지를 반환합니다.
Throwable getCause(): 예외의 원인(중첩된 예외)을 반환합니다.
Throwable initCause(Throwable cause): 예외의 원인을 설정합니다.
String toString(): 예외의 문자열 표현을 반환합니다.
void printStackTrace(): 예외의 스택 추적을 표준 오류 스트림에 출력합니다.
void printStackTrace(PrintStream s): 예외의 스택 추적을 지정된 출력 스트림에 출력합니다.
void printStackTrace(PrintWriter s): 예외의 스택 추적을 지정된 출력 라이터에 출력합니다.
StackTraceElement[] getStackTrace(): 예외의 스택 추적 요소 배열을 반환합니다.
void setStackTrace(StackTraceElement[] stackTrace): 예외의 스택 추적 요소 배열을 설정합니다.
Throwable fillInStackTrace(): 현재 스레드의 스택 추적 요소를 반환합니다.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ page isErrorPage="true" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
```

```
    <title>Error Page</title>
</head>
<body>
    <h1>Error Page</h1>
    <p>오류가 발생했습니다.</p>
    <p>오류 메시지: <%= exception.getMessage() %></p>
    <p>오류 클래스: <%= exception.getClass().getName() %></p>
    <p>스택 트레이스:</p>
    <pre>
        <% exception.printStackTrace(new java.io.PrintWriter(out)); %>
    </pre>
</body>
</html>
```

3. 액션태그(Action Tag)

3.1 <jsp:include />

<jsp:include> 태그는 다른 JSP 페이지나 서블릿의 출력을 포함합니다. 이 태그는 정적인 포함이 아닌 동적인 포함을 수행하므로, 포함된 페이지는 호출될 때마다 실행됩니다.

main.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Main Page</title>
</head>
<body>
    <h1>Main Page</h1>
    <jsp:include page="header.jsp" />
    <p>Welcome to the main content of the page.</p>
    <jsp:include page="footer.jsp" />
</body>
</html>
```

header.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<h2>Header Section</h2>
```

footer.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<h2>Footer Section</h2>
```

3.2 <jsp:forward />

<jsp:forward> 태그는 현재 JSP 페이지의 처리를 지정된 다른 JSP 페이지나 서블릿으로 포워딩합니다. 포워딩 후에는 원래 페이지의 실행이 중단됩니다.

page.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Target Page</title>
</head>
<body>
    <h1>이 페이지는 forward 페이지입니다.</h1>
</body>
</html>
```

for.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<jsp:forward page="page.jsp" />
```

3.3 <jsp:param />

<jsp:param> 태그는 jsp:include 또는 jsp:forward 태그와 함께 사용되어 매개 변수를 전달합니다.

main.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Main Page</title>
</head>
<body>
    <h1>Main Page</h1>
    <jsp:include page="include.jsp">
```

```
        <jsp:param name="username" value="이순신" />
    </jsp:include>
</body>
</html>
```

include.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="java.util.*" %>
<!DOCTYPE html>
<html>
<head>
    <title>Target Page</title>
</head>
<body>
    <h1>Target Page</h1>
    <p> 이름 : <%= request.getParameter("username") %></p>
</body>
</html>
```

2.4 표현 언어(Expression Language, EL)

${}를 사용하여 변수나 속성에 접근하는 데 사용됩니다.

[redacted]

```
<p>${message}</p>
```