## **JSP**

초기 웹 프로그램 개발에서는 서블릿이 클라이언트로부터 요청을 받아 비즈니스 작업을 처리한 후 그 결과를 클라이언트의 브라우저로 전송하는 방식으로 작업했다.

클라이언트가 서블릿에 요청을 하면 먼저 톰캣 컨테이너가 받는다.

사용자의 요청이나 응답에 대한 HttpServletRequest객체와 HttpServleyResponse객체를 만들고 doXXX()메소드를 호출하면 이 객체들을 전달한다.

톰캣이 사용자 요청에 대한 정보를 모든 HttpServletRequest 객체의 속성으로 담아 메소드로 전달하므로 HttpServletRequest에서 제공하는 메소드들은 매개변수로 넘어온 객체들을 이용해 사용자가 전송한 데이터를 받아오거나 응답할 수 있는 것이다.

| @WebServlet("/test")  **public class** test **extends** HttpServlet {  **private static final long** serialVersionUID = 1L;  public test() { }  protected void **init**(ServletConfig **config**) throws ServletException {  super.init(config);  System.out.println("첫 요청 때 한 번 호출");  }  protected void **doGet**(HttpServletRequest **request**, HttpServletResponse **response**) throws ServletException, IOException {  **response**.getWriter().print("Hello Servlet");  }  protected void **doPost**(HttpServletRequest **request**, HttpServletResponse **response**) throws ServletException, IOException {  **doGet**(**request**, **response**);  }  } |
| --- |

### **- [ 단축키 ]**

| **단축키** | **뜻** | |
| --- | --- | --- |
| Ctrl + Shift + / | 주석 | |
| Ctrl + f11 | 서버 실행 | |
|  |  | |
|  |  | |
|  |  | |
|  |  | |
| Ctil + Shift + O | import 단축키 | |

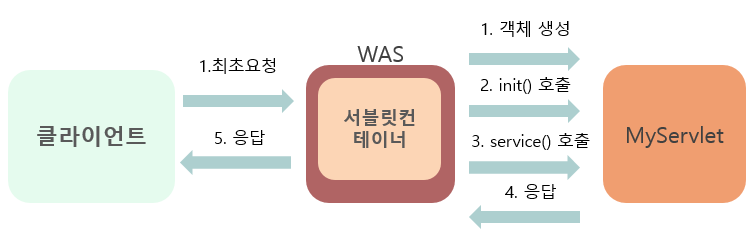
### **- [ 라이프 사이클 ]**

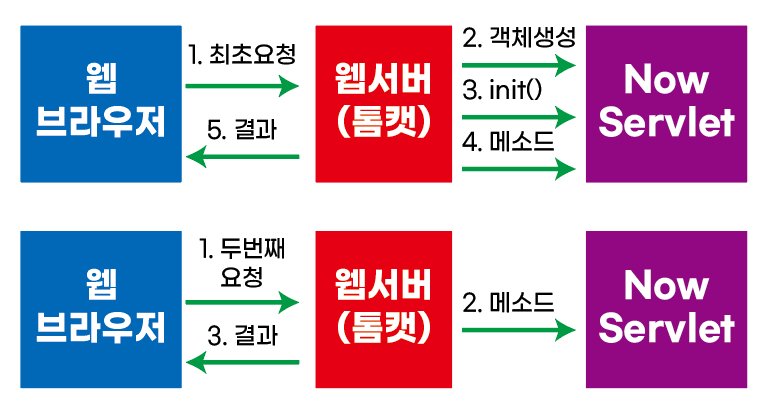
클라이언트(웹 브라우저)가 서블릿에 요청(request) 시, 서블릿이 바로 호출되지 않는다.

웹 컨테이너에 해당 서블릿 클래스가

* **로딩되어 있지 않다면 init() 메소드** 호출 후 service() 메소드 & doGet()/doPost() 호출
* **로딩 되어 있다면** service() 메소드 호출

JSP 페이지도 서블릿 클래스로 변환된 후 똑같은 라이프 사이클 수행





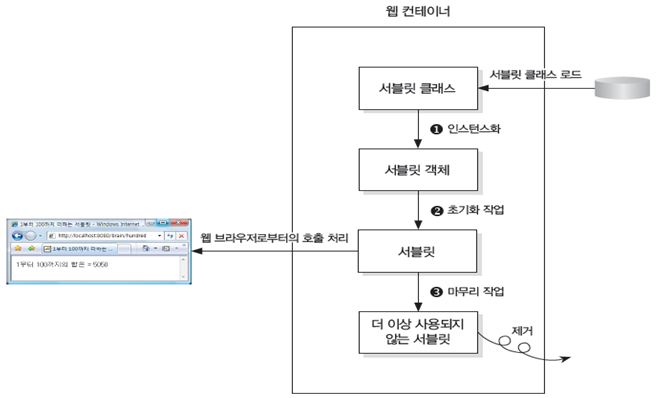
1. 요청이 오면, Servlet 클래스가 로딩되어 요청에 대한 Servlet 객체가 생성됩니다.

2. 서버는 init() 메소드를 호출해서 Servlet을 초기화 합니다.

3. service() 메소드를 호출해서 Servlet이 브라우저의 요청을 처리하도록 합니다.

4. 특정 HTTP 요청(GET, POST 등)을 처리하는 메서드 (doGet(), doPost() 등)를 호출합니다.

5. 서버는 destroy() 메소드를 호출하여 Servlet을 제거합니다.



#### **init(ServletConfig)**

초기화 메소드

| public void init(ServletConfig config) throws ServletException; |
| --- |

init() 메서드는 HttpServlet의 최상위 클래스인 Servlet 인터페이스에 정의되어 있다. 때문에 우리가 HttpServlet을 상속받아 직접 서블릿 클래스를 만들때 init() 메서드를 오버라이딩 하지 않는다면 기본으로 GenericServlet의 init()메서드가 호출된다.

| @Override  public void init(ServletConfig config) throws ServletException {  this.config = config;  this.init();  }    public void init() throws ServletException {  } |
| --- |

서블릿의 기능이나 역할에 따라 서블릿 객체가 생성되는 시점에 필요한 행동들이 있으며, 그에 대한 내용을 원하는 init() 메서드를 오버라이딩하여 구현한다.

init() 메서드를 오버라이딩 한 이후에 service() 메서드나 doGet() doPost() 등 클라이언트의 요청 처리시 사용할 때에는 다음과 같이 조상 클래스인 GenericServlet의 init 메서드(super.init(config);)를 호출해야 한다.

| @Override  public void init(ServletConfig config) throws ServletException {  //초기화 이후에도 ServletConfig를 사용하고 싶다면  super.init(config);  } |
| --- |

\* init() 메소드를 재정의할 때는 doGet()과 doPost() 중 하나는 반드시 재정의해야 한다.

#### **service()**

최초 클라이언트의 요청을 받았을 때, 컨테이너는 새로운 스레드를 생성하거나, 아니면 스레드 풀로부터 하나를 가지고 와서 서블릿의 service() 메소드를 호출한다.

클라이언테의 HTTP 메소드(GET, POST 등)를 참조하여 doGet()을 호출할지, doPost()를 호출할지 판단한다.

재정의는 거의 하지 않는다. service() 메소드를 재정의할 필요는 없다. doPost()나 doGet()을 재정의하여 HttpServlet의 service()가 이를 실행하도록 한다.

| @Override  public void **service**(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException{  System.out.println("요청이 들어와서 서비스 실행");  System.out.println("요청 메소드: " + req.getMethod());  getServletContext().setAttribute("servletApplicationData", "우리가 저장한 데이터");  doGet(req, resp);  } |
| --- |

#### **doGet()**

클라이언트가 request한 어떤 정보를 가져오는 방식. 위와 같이 사용할 경우 길이 제한이 있기 때문에 주로 선택이나 짧은 길이의 정보들을 다룰 때 주로 사용된다. 가장 큰 특징: **URL에 변수를 포함시켜 요청**,

보안에 취약하므로 이럴 경우 POST 방식을 이용할 수 있다.

#### **doPost()**

길이 제한이 없어서 게시판이나 긴 글을 업로드하는 형식에 사용될 수 있다. 또한 GET방식과 달리 URL에 **데이터가 노출되지 않는다.** URL에 Mapping 정보만 표시된다. 보안에 더 좋은 장점을 가지고 있어 개인 정보를 다루는 데이터는 POST 방식을 선택하는 것이 바람직하다.

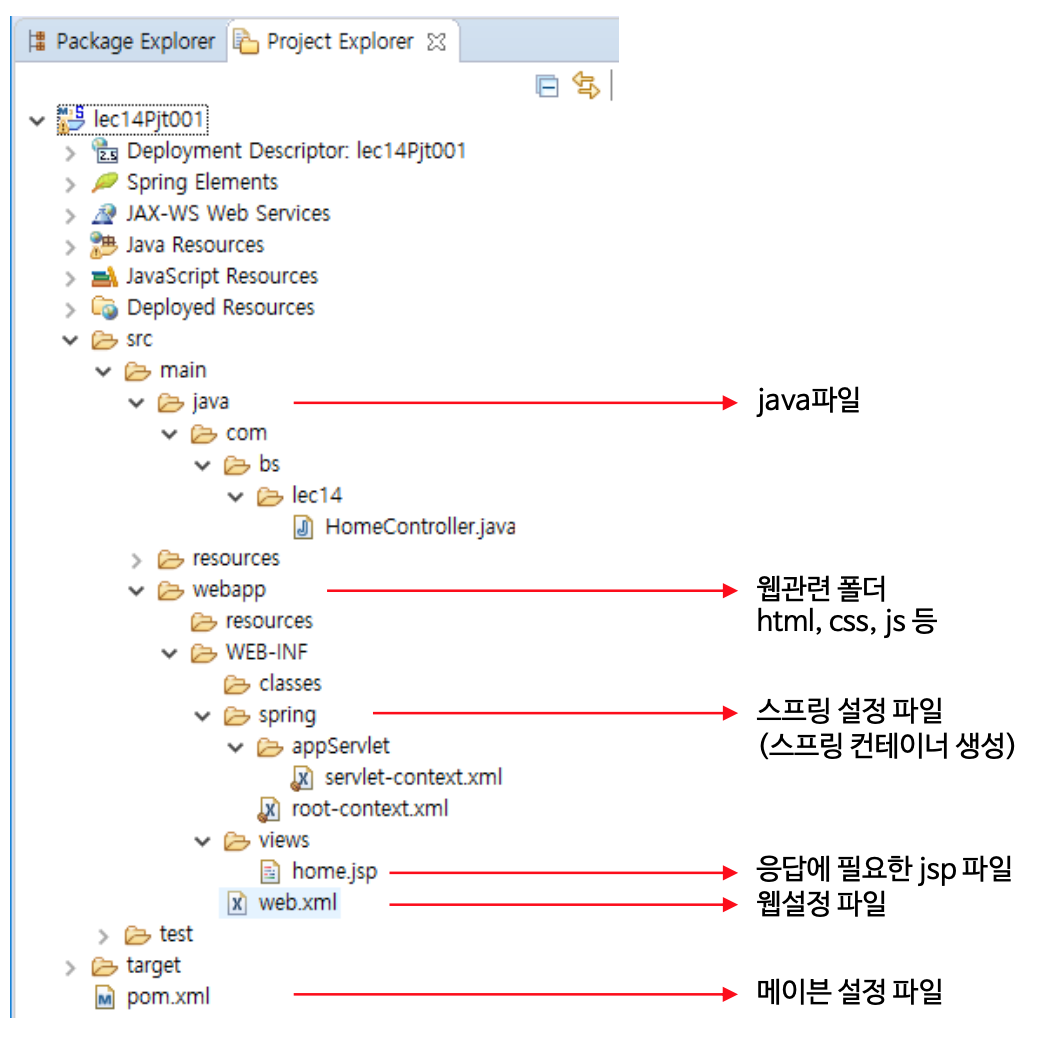
#### **destroy()**

한 번만 수행된다. Web Application이 갱신되거나 WAS가 종료될 때 호출됨

역할 : Servlet 객체를 메모리에서 제거한다.

### **- [ 구성 ]**

| **종류** | **위치** | |
| --- | --- | --- |
| Java 파일 | Java 파일들이 위치한다. 주로 패키지로 묶어서 관리  웹 어플리케이션에 사용되는 Controller, Service, DAO 객체들이 위치 | |
| webapp | 웹과 관련된 파일들이 위치한다  : 스프링 설정 파일, JSP파일, HTML파일 등 | |
| resources (static) | JSP파일을 제외한 html, css, js 파일 등이 위치한다 | |
| spring 폴더 | 스프링 컨테이너를 생성하기 위한 스프링 설정파일이 위치한다 | |
| views 폴더 | View로 사용될 JSP파일이 위치한다 | |
| pom.xml 파일 | 메인 레파지토리에서 필요한 라이브러리를 내려받기 위한 메이븐 설정 파일 | |



#### **WEB-INF 폴더**

WEB-INF 폴더의 경우에는 브라우저에서 직접적으로 접근이 불가한 경로이다.

Controller를 통해야만 접근이 가능하며, 사용자가 직접 접근이 불가하여 보안성이 높다.

이건 jsp뿐만이 아니라, html, js, css와 같은 정적파일들도 포함된다.

그래서 WEB-INF안에 있는 js나 css에는 접근을 할 수 없다. 만약 WEB-INF 하위에 .js나 .css를 생성하고 호출을 하면 404에러가 뜰 것이다.

WEB-INF 아래의 .js를 호출할 경우 아래와 같이 사용할 수 있다.

| <script>  <%@include file="/WEB-INF/경로" %>  </script> |
| --- |
| <script>  <%@include file="/WEB-INF/jsp/testJSP/testJs1.js" %>  </script> |

#### **필터**

Filter는 서블릿을 향하는 클라이언트의 요청(request)과 응답(response)를 중간에 가로채서 필터링하는 것을 목적으로 하는 객체를 정의하고 메서드를 제공하는 인터페이스다.

서블릿 컨테이너에는 수 많은 서블릿이 존재한다. 그런데 이 모든 서블릿의 요청이나 응답할 때 공통적으로 수행해야 하는 작업이 있다면 굉장히 피곤해진다. 매번 모든 서블릿에 해당 작업 코드를 집어 넣어야 하기 때문이다. 그럴 때 필터를 쓰면 편하다.

필터의 구조는 다음과 같다.



클라이언트가 서블릿에 대해 요청하거나 서블릿이 클라이언트에 응답할 때 중간에서 필터링하는 역할을 한다. 가장 대표적인 예가 인코딩 작업이다.

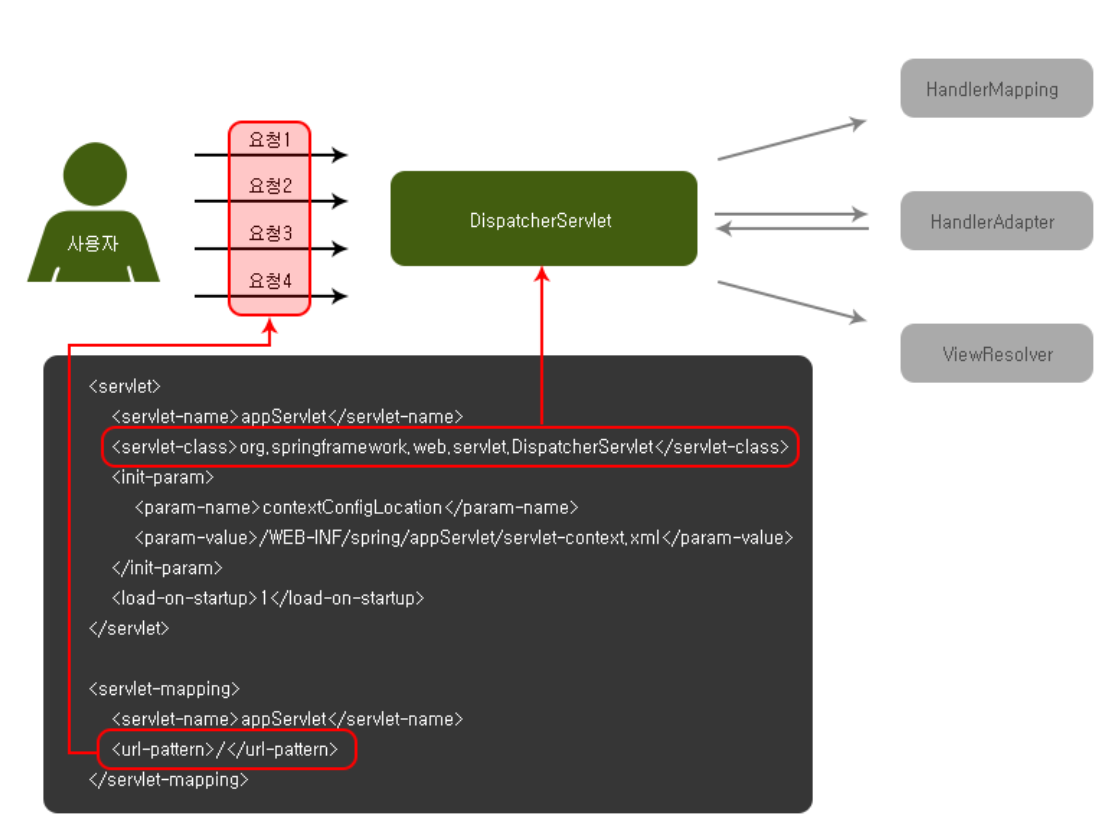
| **public void** **doFilter**(ServletRequest **request**, ServletResponse **response**, FilterChain chain) {  **request**.**setCharacterEncoding**("utf-8");  // 위는 필터링 전에 실행되는 부분  **chain**.**doFilter**(**request**, **response**);  // 아래는 필터링 후 서블릿 거친 후에 브라우저로 response 전달 전 실행부분 **response**.**setCharacterEncoding**("utf-8");  } |
| --- |

#### **web.xml**

[참고링크](https://velog.io/@909snare/%EC%8A%A4%ED%94%84%EB%A7%81-MVC-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8%EC%9D%98-%EA%B5%AC%EC%A1%B0)

웹 어플리케이션에서 최초 사용자의 요청이 발생하면 DispatcherServlet이 사용자의 요청자를 받는다고 하였다. 따라서 개발자는 DispatcherServlet을 서블릿으로 등록해주는 과정을 설정해 주어야 한다.

사용자의 모든 요청을 받기 위해서 서블릿 맵핑 경로는 '/'로 설정한다



#### **view**

클라이언트 요청 정보 (URL 매핑 값)에 해당하는 JSP파일 실행합니다.



### **- [ 태그 ]**

| **특수코드값** | **실제표현** | **비고** |
| --- | --- | --- |
| 선언문 | <%! … %> | 자바 변수나 메소드를 정의  전역변수&전역 메소드로 사용된다 |
| 스크립틀릿 | <% … %> | 자바 로직 코드를 작성하는데 사용  JSP 페이지가 서블릿 프로그램에서 서블릿 클래스로 변환할 때 JSP 컨테이너가 자바 코드가 삽입되어 있는 스크립트 태그를 처리하고 나머지는 HTML 코드나 일반 텍스트로 간주 |
| 표현문 | <%= … %> | 변수, 계산식, 메소드 호출 결과를 문자열 형태로 출력 |
| 주석 | <%-- --%> | Ctrl + Shift + / |
| 지시자 | <%@ %> | 페이지의 속성  <%@ page import="java.util.Arragys"%>  <%@ include file="include01.jsp" %> |

#### 

## **HTTP 프로토콜**

상호 간에 정의한 규칙을 의미하며 특정 기기 간에 데이터를 주고받기 위해 정의다. 웹에서는 브라우저와 서버 간에 데이터를 주고받기 위한 방식으로 HTTP 프로토콜을 사용한다.

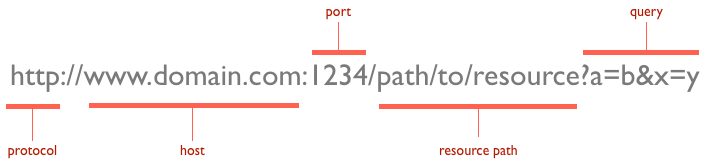
HTTP 프로토콜은 상태가 없는(stateless) 프로토콜이다. 데이터를 주고 받기 위한 각각의 데이터 요청이 서로 독립적으로 관리가 된다는 뜻으로 이전 데이터 요청과 다음 데이터 요청이 서로 관련이 없다는 말이다.

이러한 특징 덕택에 서버는 세션과 같은 별도의 추가 정보를 관리하지 않아도 되고, 다수의 요청 처리 및 서버의 부하를 줄일 수 있는 성능 상의 이점이 생긴다.

HTTP 프로토콜은 일반적으로 TCP/IP 통신 위에서 동작하며 기본 포트는 80번이다.

### **- [ URL ]**

서버에 자원을 요청하기 위해 입력하는 영문 주소이다. 구조는 아래와 같다.



### **- [ 요청 메소드 ]**

HTTP Verbs라고도 불리우며 아래와 같이 주요 메서드를 갖고 있다. 조회, 생성, 변경, 삭제 동작을 HTTP 요청 메서드로 정의할 수 있다. 참고로 때에 따라서는 POST 메서드로 PUT, DELETE의 동작도 수행할 수 있다.

| **메소드** | **비고** |
| --- | --- |
| GET | 존재하는 자원에 대한 요청 |
| POST | 새로운 자원을 생성 |
| PUT | 존재하는 자원에 대한 변경 |
| DELETE | 존재하는 자원에 대한 삭제 |

기타 요청 메서드는 다음과 같다.

| **메소드** | **비고** |
| --- | --- |
| HEAD | 서버 헤더 정보를 획득. GET과 비슷하나 Response Body를 반환하지 않음 |
| OPTIONS | 서버 옵션들을 확인하기 위한 요청. CORS에서 사용 |

### **- [ 응답코드 ]**

서버에서 설정해주는 응답(Response) 정보이다.

상태 코드는 3자리 숫자로 만들어져 있으며, 첫번째 자리는 1에서 5까지 제공됩니다. 첫번째 자리가 4와 5인 경우는 정상적인 상황이 아니므로 사이트 관리자가 유의해야 한다.

| **코드** | **비고** |
| --- | --- |
| 1xx(정보) | 요청을 받았으며 프로세스를 계속 진행합니다. |
| 2xx(성공) | 요청을 성공적으로 받았으며 인식했고 수용하였습니다. |
| 3xx(리다이렉션) | 요청 완료를 위해 추가 작업 조치가 필요합니다. |
| 4xx(클라이언트 오류) | 요청의 문법이 잘못되었거나 요청을 처리할 수 없습니다. |
| 5xx(서버 오류) | 서버가 명백히 유효한 요청에 대한 충족을 실패했습니다. |



## **내장 객체**

개발자가 객체를 생성하지 않고 바로 사용할 수 있는 객체가 내부객체이다. JSP 에서 제공되는 내부객체는 JSP 컨테이너에 의해 Servlet으로 변화될 때 자동으로 객체가 생성된다.

| **내장객체** | **생성시점** | **소멸시점** |
| --- | --- | --- |
| request | 해당 페이지 요청 시점 | 해당 페이지 로딩 완료 시점  : 필터에서 전처리용.  정보 저장해서 서블릿 연결 가능하다 |
| session | 해당 컨텍스트 내 특정 파일 요청 시점 | 웹 브라우저 종료 or 일정시간 경과 시점  : 쿠키. 기본 30분 정도 |
| application | 웹 애플리케이션 시작 시점 | 웹 애플리케이션 종료 시점  : 서버가 꺼질 때까지 영구 저장 |

| **내장객체** | **용도** | **활용** |
| --- | --- | --- |
| Test | 위에 3종류 확인 |  |
| RepositoryFilter | 리퀘스트 전처리 |  |

### **- [ 입출력 객체 ]**

#### **request**

| **입출력 객체:** 사용자 요청과 관련된 기능을 제공하는 내장객체  **jsp를 servlet 서버로 띄우는 방법**  **request**.**getRequestDispatcher**("/WEB-INF/경로/파일명")**.forward(request, response);** |
| --- |

주로 클라이언트에서 서버로 전달되는 정보를 처리하려고 사용한다. 가장 대표적인 유형은 HTML 폼을 통해 입력된 값을 JSP에서 가져올 때 사용하는 것이다. request의 원래 클래스는 javax.servlet.http.HttpServletRequest이다.

| **실제표현** | **비고** |
| --- | --- |
| getParameterNames() | 현재 요청에 포함된 매개변수의 이름을 열거 형태로 리턴 |
| getParameter(name) | 문자열 name에 매칭된 value를 리턴 |
| getParameterValues(name) | 문자열 name에 매칭된 value를 배열 형태로 가져옴 |
| getCookies() | 모든 쿠키 값을 javax.servlet.http.Cookie의 배열 형태로 리턴 |
| getRemoteAddr() | 클라이언트의 IP 주소를 리턴한다. |
| setCharacterEncoding() | 현재 JSP로 전달되는 내용을 지정한 캐릭터셋으로 변환해준다.  HTML 폼에서 한글 입력을 정상적으로 처리하기 위해 필수 |

| @WebServlet("/content/type")  **public class** ContentTypeServlet extends HttpServlet {  protected void **doGet**(HttpServletRequest **request**, HttpServletResponse **response**) throws ServletException, IOException {  System.out.println( "요청 URL: " + **request**.**getRequestURI**() );  System.out.println( "요청 Method: " + **request**.**getMethod**() );  System.out.println( "요청 param(id): " + **request**.**getParameter**("id") );  System.out.println( "요청 param(password): " + **request**.**getParameter**("password") );  System.out.println( "요청 param(email): " + **request**.**getParameter**("email") );  **request**.**getRequestDispatcher**("**/WEB-INF**/dispatcher-test.html").forward(**request**, **response**);  }  } |
| --- |
| 요청 URL: **/content/type**  요청 Method: **GET**  요청 param(id): **null**  요청 param(password): **null**  요청 param(email): **null** |

#### **response**

| **입출력 객체:** 사용자 응답과 관련된 기능을 제공하는 내장객체 |
| --- |

사용자 요청을 처리하고 응답을 다른 페이지로 전달하는 기능을 제공한다. javax.servlet.http.HttpServletResponse 객체에 대한 참조 변수로, request에 비해 사용되는 메서드는 적지만 setContentType, sendRedirect와 같은 메서드는 잘 알아두어야 한다.

| **실제표현** | **비고** |
| --- | --- |
| setContentType(type) | 문자열 형태의 type에 지정된 MIME Type으로 contentType을 설정 |
| sendRedirect | 클라이언트 요청을 다른 페이지로 보낸다. |

#### **out**

| **입출력 객체:** 출력 스트림으로써, 사용자 웹 브라우저로 출력하기 위한 채널이다. |
| --- |

스트립트릿에서 브라우저로 텍스트 출력하는 데 사용하며, out은 javax.servlet.jsp.JspWriter 객체의 참조 변수로, 버퍼 관련 메서드와 출력 관련 메서드로 구성되며 out을 이용해서 출력된 내용은 서버의 콘솔이 아닌 사용자에게 전달된다.

| **실제표현** | **비고** |
| --- | --- |
| getBufferSize() | output buffer의 크기를 바이트로 알려준다. |
| getRemaining() | 남아 있는 버퍼의 크기 중 사용 가능한 비율을 알려준다. |
| clearBuffer() | 버퍼에 있는 콘텐츠를 모두 지운다. |
| flush() | 버퍼를 지우고 output stream도 비운다. |
| close() | output stream을 닫고 버퍼를 비운다. |
| println(content) | content의 내용을 newline과 함께 출력한다. |
| print(content) | content의 내용을 출력한다.  **out**.print(gson .toJson(**user**));  servlet에서는 PrintWriter out = response.getWriter(); 로 써야?  ㄴjsp에는 이게 내장되어 있어서 그렇다 |

#### **예문**

| **String** user\_id = **request**.**getParameter**("user\_id");  **String** user\_password = **request**.**getParameter**("user\_password");  **String** user\_name = **request**.**getParameter**("user\_name");  **String** user\_email = **request**.**getParameter**("user\_email");  **Map**<String, String> **user** = new **HashMap**<String, String>();  user.put("user\_id", user\_id );  user.put("user\_password", user\_password );  user.put("user\_name", user\_name );  user.put("user\_email", user\_email );  **response**.**setContentType**("application/json; charset=utf-8");  **Gson** gson = new **GsonBuilder**().**setPrettyPrinting**().**create**();  **out**.print(gson .toJson(**user**)); |
| --- |
|  |

### **- [ ServletContext ]**

서블릿 컨테이너와 통신하기 위해서 사용되는 메소드를 지원하는 인터페이스이다. 서블릿 컨테이너가 시작될 때 웹서버에 등록된 웹 애플리케이션 단위로 하나의 ServletContext 객체가 자동으로 생성된다. 그리고 웹 애플리케이션 서비스가 중지될 때 소멸한다.

웹 애플리케이션 내에 있는 모든 서블릿 그리고 **JSP 간에 정보를 공유**할 수 있고, 서블릿 컨테이너에 대한 **정보를 추출**할 수 있게 하는 기술이다.

ServletContext 객체는 메소드를 이용해 추출된다. 웹 애플리케이션 단위로 사용하기 때문에 **동일한 웹 애플리케이션에 존재하는 서블릿들은 동일한 ServletContext 객체를 사용**한다. ServletContext를 추출하는 메소드는 **ServletConfig**의 **getServletContext**()이다.

#### **application**

application은 웹 에플리케이션 전체를 관리하는 객체이다. application 객체를 통해 각 서블릿이나 JSP에서 공유하려고 하는 각종 정보를 설정하고 참조할 수 있다.

application은 javax.servlet.ServletContext 객체에 대한 참조 변수로서, config 객체를 통해 생성한다. ServletContext 객체는 컨테이너와 관련된 여러 정보를 제공하며, application 참조 변수를 통해서 서블릿이 실행되는 환경, 서버 자원과 관련된 정보를 얻거나 로그 파일을 기록하는 작업을 수행한다.

| **실제표현** | **비고** |
| --- | --- |
| log(message) | 문자열 message의 내용을 로그 파일에 기록한다. |
| log(message, exception) | 예외 상황에 대한 정보를 포함하여 로그 파일에 기록한다. |
| getAttribute(String name) | 문자열 name에 해당하는 속성 값이 있다면 Object 형태로 반환, 형변환이 필요하다. |
| getAttributeNames() | application 객체에 저장된 속성들의 이름을 열거 형태로 가져온다. |
| setAttribute(String name, Object value) | 문자열 name의 이름으로 Object형 데이터를 저장한다. |
| removeAttribute(String name) | 문자열 name에 해당하는 속성을 삭제한다. |

### **- [ 기타 객체 ]**

#### **session**

| **세션 객체:** HTTP 프로토콜이 비연결형 프로토콜이라는 문제를 해결 |
| --- |

HTTP 프로토콜은 한 페이지가 출력된 다음에는 클라이언트와 서버의 연결이 끊어진다. 따라서 사용자가 특정 행동을 할 때 까지 보관해야 할 정보가 있다면 처리가 매우 곤란해진다. 이러한 문제점을 해결하기 위해 등장한 것이 세션과 쿠키이다. 쿠키는 사용자와 관련된 정보를 PC에 저장하고, 세션은 서버에 보관하는 방식인데, 세션이 쿠키보다 보안 문제에 유리하고 제약사항이 적은 편이라서 최근에는 대부분 세션을 사용한다.

| **실제표현** | **비고** |
| --- | --- |
| getId() | 세션 고유의 ID를 문자열 형태로 리턴 |
| getCreatingTime() | 세션 생성 시간을 January 1. 1970 GMT 부터 long 형 밀리세컨드 값으로 반환 |
| getLastAccessedTime() | 현재 세션으로 마지막 작업한 시간을 ong형 밀리세컨드 값으로 반환 |
| getMaxInactiveInterval() | 세션의 유지시간을 초로 반환 |
| setMaxInactiveInterval(t) | 세션의 유효 시간을 t에 설정된 초 값으로 설정 |
| invalidate() | 세션 종료 |
| getAttribute(attr) | 문자열 attr로 설정된 세션 값을 Object 형태로 가져옴 |
| setAttribute(name, attr) | 문자열 name으로 Object attr을 저장 |

#### **config**

서블릿이 최초로 메모리에 적재될 때 컨테이너는 서블릿 초기화와 관련된 정보를 읽고 javax.servlet.ServletConfig 객체에 저장한다. 웹 에플리케이션 하나에는 서블릿, JSP, 빈즈 클래스 등이 여러 개로 구성되는데 config 내장객체는 서로 독립적인 파일로 구성된 JSP나 서블릿들이 서로 다른 시점에 임의의 정보를 서로 교환할 수 있는 기법 중 하나로 활용할 수 있다.

기본적으로 web.xml 파일에 초기화 값을 등록해놓고 JSP에서 필요한 경우 config 내장객체를 이용해 참조하는 방법이 있다.

| **실제표현** | **비고** |
| --- | --- |
| getInitParameterNames() | 초기 매개변수 값들의 설정 이름을 열거 객체로 반환 |
| getInitParameter(name) | 문자열 name에 해당하는 초기화 매개변수 값 반환 |

#### **exception**

exception은 page지시어에서 오류 페이지로 지정된 JSP 페이지에서 예외가 발생할 때 전달되는 java.lang.Throwable의 인스턴스에 대한 참조 변수다. 현재 페이지를 처리하다 발생하는 예외상황에 대한 정보를 가져올 수 있다.

| **실제표현** | **비고** |
| --- | --- |
| getMessage() | 문자열로 된 오류 메시지를 반환한다. |
| printStackTrace() | 스택 추적 정보를 출력한다. |
| toString() | 예외 클래스 이름과 함께 오류 메시지를 반환한다. |

#### 

### **- [ $.ajax ]**

비동기 통신. 클라이언트 - 서버 간에 XML 또는 JSON 데이터를 주고 받는 기술이다. 페이지 이동 없이 데이터 처리가 가능하다. 새로 고침 없이 화면이 변한다.

| < script type="text/javascript" src="<http://code.jquery.com/jquery-latest.min.js>" ></script> |
| --- |

제이쿼리를 사용하면 더 쉽게 활용할 수 있다.

| **실제표현** | **비고** |
| --- | --- |
| type | 통신 타입을 설정 |
| url | 요청할 url |
| async | 비동기로 할 거냐 말 거냐 |
| data | 서버에 요청할 때 보낼 매개변수 설정 |
| dataType | 응답 받을 데이터 타입(XML,JSON,TEXT,HTML 등) |
| success | 요청/응답에 성공했을 때 처리할 구문 |
| error | 요청/응답에 실패했을 때 처리할 구문 |
| complete | 작업을 마친 후에 처리할 구문 |

| **signupButton**.**onclick** = () => {  **const** user = {  **userId**: document.querySelector(".user-id")**.value**,  **userPassword**: document.querySelector(".user-password")**.value**,  **userName**: document.querySelector(".user-name")**.value**,  **userEmail**: document.querySelector(".user-email")**.value**  } // 위 값들은 api로 이동된 후 사용됨  **send**(user);  } |
| --- |
| function **send**(user){  **$.ajax(**{  **async:** false, // async 동기(false), 비동기(true\_default) 선택  **type:** "post", // RequestMethod (get요청 post요청 선택)  **url:** "/api/v1/auth/addition", // 요청 URL (보통 api)  // post로 할 경우 위 링크가 client에게 안 열림  **data:** user, // 전송데이터  **dataType:** "json", // 응답데이터 형식(Response ContentType)  **success:** (response) => { // response 응답 성공시 실행되는 function  console.log(response);  document.querySelector("body").innerHTML = `  <h1> ${response.userId} </h1>  <h1> ${response.userPassword} </h1>  <h1> ${response.userName} </h1>  <h1> ${response.userEmail} </h1>  `  },  **error:** (error) => { // 요청 또는 응답이 실패하였을 때 오류처리  console.log(error);  }  });  } |

----------------------------------------------

[workspace] > [.metadata] > [org.eclipse.wst.server.core] > [tmp0] > [wtpwebapps] > [ServletStudy-20220823-youri] > [WEB-INF] > [classes]

IP: 집주소 느낌. www 인터넷

= IPv4 255.255.255.255

= IPv6

포트번호: 서버 종류 구분하는것? 8000번 8080번 등

web서버 포트는 기본적으로 8080, 80, 8000 등 사용

데이터베이스 : 3306

포트포어딩? KT나 SKT에서 주는 아이디를 외부 IP라고 부른다.

> 공유기가 나옴(여러대 컴퓨터가 쓰기 위함) 공유기는 본인들끼리만 받는다

> 외부에서 8000 8080 서버를 배포할때…. 들어가게 해줌.

MVC > M: / V:JSP /C:Servlet

#### 