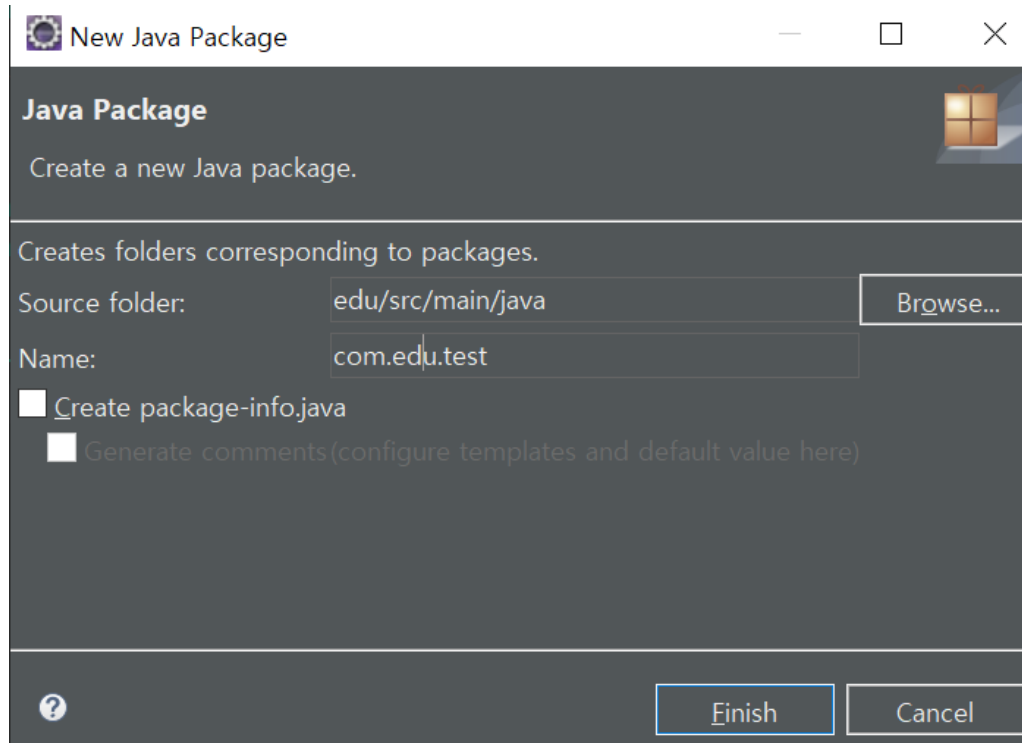


웹 프로그래밍이란

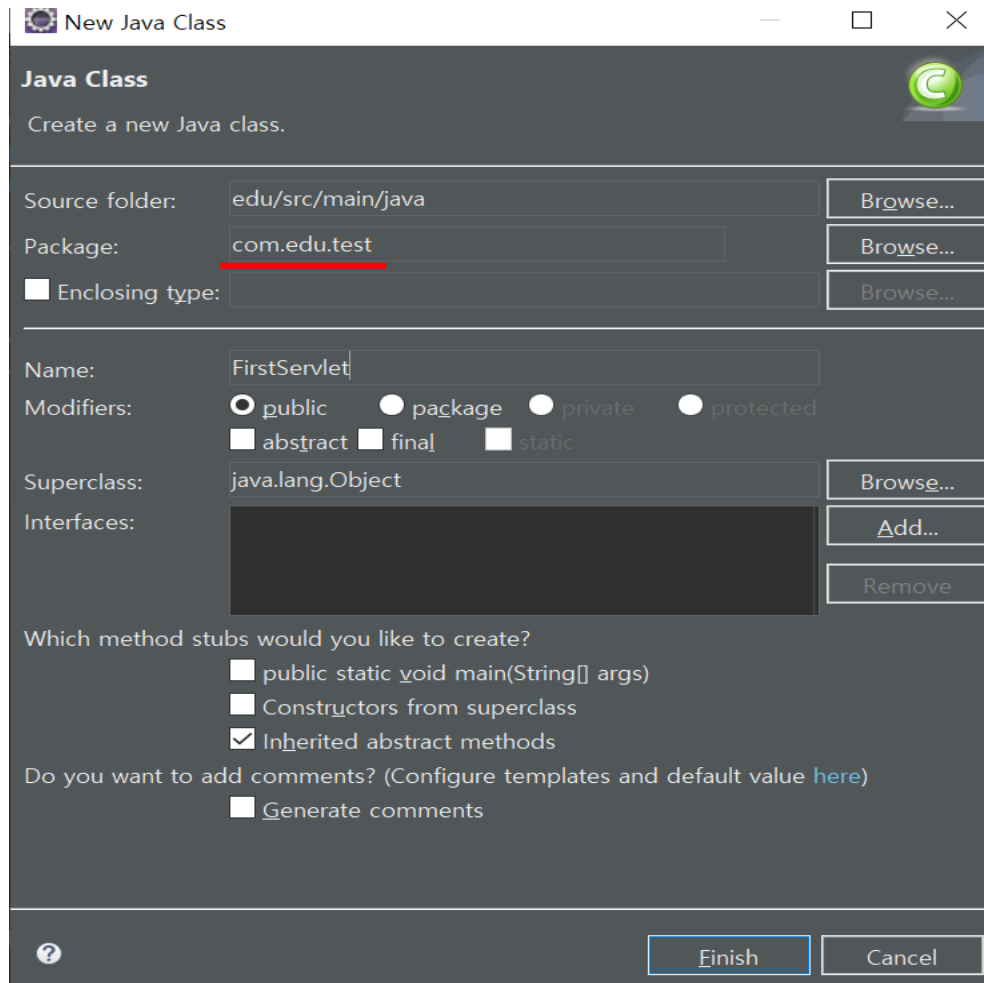
서블릿 작성

프로젝트 Explorer 에서 src폴더에 마우스 오른쪽 버튼을 누르고 패키지를 추가한다.



웹 프로그래밍이란

새로운 자바 클래스 파일(서블릿X)을 만듭니다. com.edu.test에 클래스 파일을 추가합니다.



The image shows a 'New Java Class' dialog box from an IDE. The title bar says 'New Java Class'. The main title is 'Java Class' with a subtitle 'Create a new Java class.' and a green 'C' icon. The 'Source folder' is 'edu/src/main/java' with a 'Browse...' button. The 'Package' is 'com.edu.test' with a 'Browse...' button. There is an unchecked checkbox for 'Enclosing type' with a 'Browse...' button. The 'Name' field contains 'FirstServlet'. The 'Modifiers' section has radio buttons for 'public' (selected), 'package', 'private', and 'protected', and checkboxes for 'abstract', 'final', and 'static'. The 'Superclass' is 'java.lang.Object' with a 'Browse...' button. The 'Interfaces' section is empty with 'Add...' and 'Remove' buttons. Below this, it asks 'Which method stubs would you like to create?' with checkboxes for 'public static void main(String[] args)', 'Constructors from superclass', and 'Inherited abstract methods' (checked). It then asks 'Do you want to add comments? (Configure templates and default value [here](#))' with a checkbox for 'Generate comments'. At the bottom are 'Finish' and 'Cancel' buttons.

New Java Class

Java Class
Create a new Java class.

Source folder: edu/src/main/java Browse...

Package: com.edu.test Browse...

☐ Enclosing type: Browse...

Name: FirstServlet

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

? Finish Cancel

웹 프로그래밍이란

edu\src\main\java\com\edu\test

FirstServlet 클래스 내용

```
1 package com.edu.test;  
2  
3 import javax.servlet.http.HttpServlet;  
4  
5 public class FirstServlet extends HttpServlet{  
6  
7 }  
8 |
```

HttpServlet 는 서블릿이 웹상에서 HTTP 프로토콜을 이용해 서비스를 처리하기 위해 반드시 상속받아야 하는 클래스입니다. 즉, 모든 서블릿의 상위 클래스는 HttpServlet이어야 합니다.

웹 프로그래밍이란

FirstServlet 클래스를 재정의 하여봅시다.

```
package com.edu.test;

import javax.servlet.ServletException;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;

public class FirstServlet extends HttpServlet {

    private static final long serialVersionUID = 1L; // FirstServlet 는 디폴트 UID 값을 갖고 있어야함,

    public void init() {
        System.out.println("init");
    }

    public void service(ServletRequest request, ServletResponse response) {
        System.out.println("service");
    }
}
```



톰캣 연결

웹 프로그래밍이란

`Service()` 메소드가 끝나면 서버에서의 실행은 끝납니다. 서버 프로그램 실행이 완료된 후에는 서블릿 컨테이너가 실행결과를 웹서버에 전달하고, 웹서버는 서비스를 요청한 클라이언트에 응답합니다. 이로써 웹에서 하나의 요청에 대한 처리가 완료됩니다.

웹 프로그래밍이란

웹서버를 구동시키면 확인하면 에러가 납니다.

HTTP 상태 404 – 찾을 수 없음

타입 상태 보고

메시지 요청된 리소스 [/edu/servlet/com.edu.test.FirstServlet]은(는) 가용하지 않습니다.

설명 Origin 서버가 대상 리소스를 위한 현재의 representation을 찾지 못했거나, 그것이 존재하는지를 밝히려 하지 않습니다.

Apache Tomcat/8.5.70

웹 프로그래밍이란

src/main/webapp/WEB-INF/web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_0.xsd">
3   <display-name>edu</display-name>
4   <welcome-file-list>
5     <welcome-file>index.html</welcome-file>
6     <welcome-file>index.htm</welcome-file>
7     <welcome-file>index.jsp</welcome-file>
8     <welcome-file>default.html</welcome-file>
9     <welcome-file>default.htm</welcome-file>
10    <welcome-file>default.jsp</welcome-file>
11  </welcome-file-list>
12  <servlet>
13    <servlet-name>FirstServlet</servlet-name> <!-- 자바 파일을 FirstServlet라는 이름으로 등록 -->
14    <servlet-class>com.edu.test.FirstServlet</servlet-class> <!-- .java생략, 컨트롤키 누르고 밑줄쳐서 -->
15  </servlet>
16  <!-- 2.사용자의 요청 -->
17  <servlet-mapping>
18    <servlet-name>FirstServlet</servlet-name> <!-- 서블릿 중에 FirstServlet 라는 애 있니 -->
19    <url-pattern>/FirstServlet</url-pattern> <!-- FirstServlet 라는 요청이 오면 -->
20  </servlet-mapping>
21 </web-app>
```

경로 체크

웹 프로그래밍이란

src/main/webapp/WEB-INF/web.xml

<!-- 1.서블릿 등록 -->

<servlet>

<servlet-name>FirstServlet</servlet-name> // 해당 클래스 파일을 FirstServlet 명칭으로 등록한다. 클라이언트에서 FirstServlet 요청이 오면 해당 클래스 파일을 전송한다.

<servlet-class>com.edu.test.FirstServlet</servlet-class> //해당 클래스 파일

</servlet>

<!-- 2.사용자의 요청 -->

<servlet-mapping>

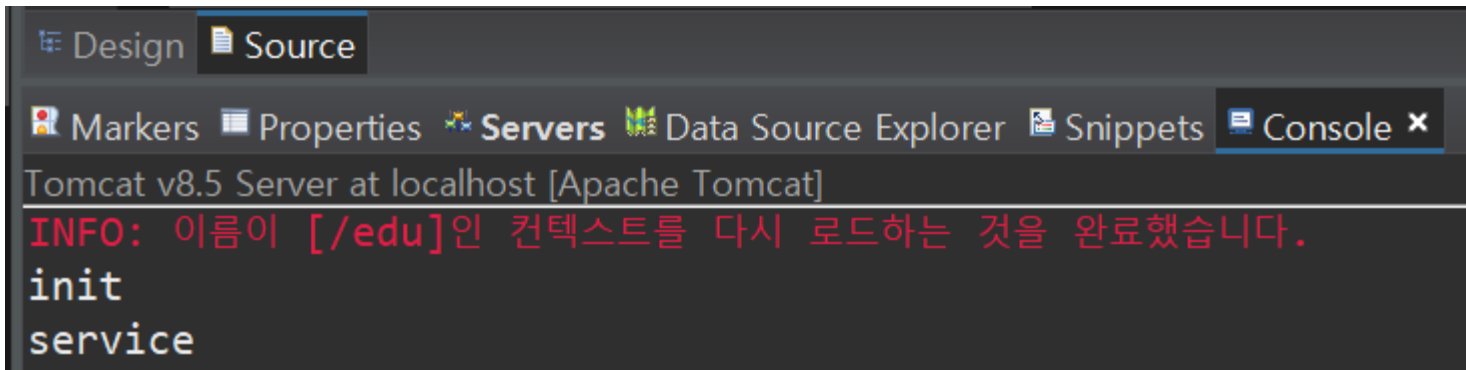
<servlet-name>FirstServlet</servlet-name> // 서버에게 FirstServlet 이라는 애가 있는지 요청한다.

<url-pattern>/FirstServlet</url-pattern> // 클라이언트에서 /FirstServlet 이라는 요청이 오면

</servlet-mapping>

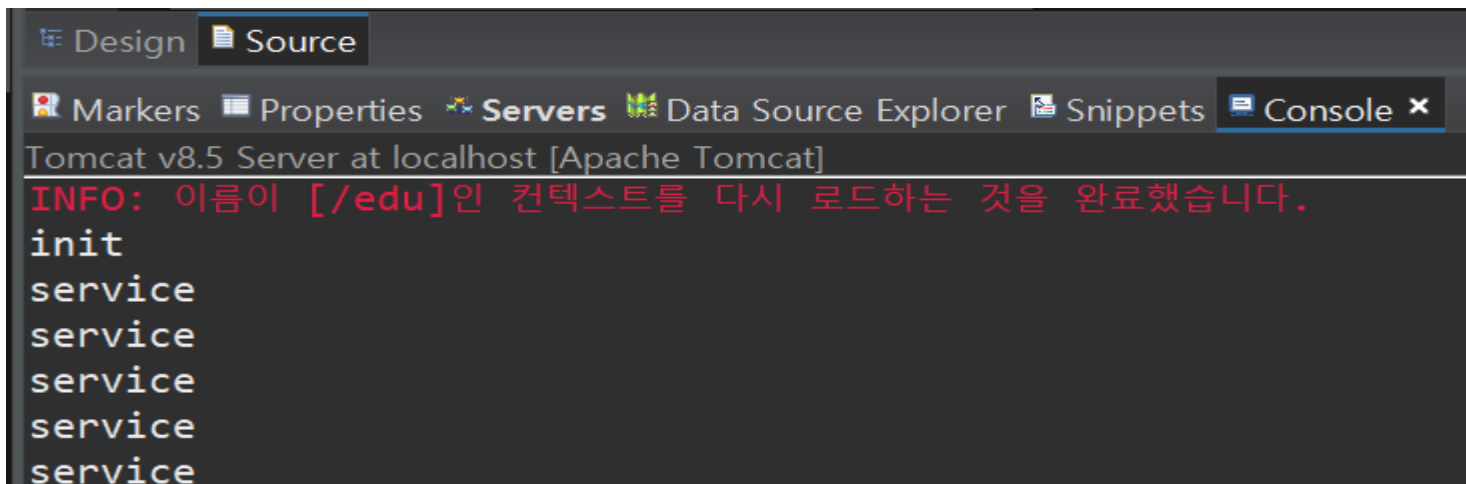
웹 프로그래밍이란

구동하면 콘솔 창에는



```
Design Source
Markers Properties Servers Data Source Explorer Snippets Console x
Tomcat v8.5 Server at localhost [Apache Tomcat]
INFO: 이름이 [/edu]인 컨텍스트를 다시 로드하는 것을 완료했습니다.
init
service
```

새로고침을 하게 되면



```
Design Source
Markers Properties Servers Data Source Explorer Snippets Console x
Tomcat v8.5 Server at localhost [Apache Tomcat]
INFO: 이름이 [/edu]인 컨텍스트를 다시 로드하는 것을 완료했습니다.
init
service
service
service
service
service
service
```

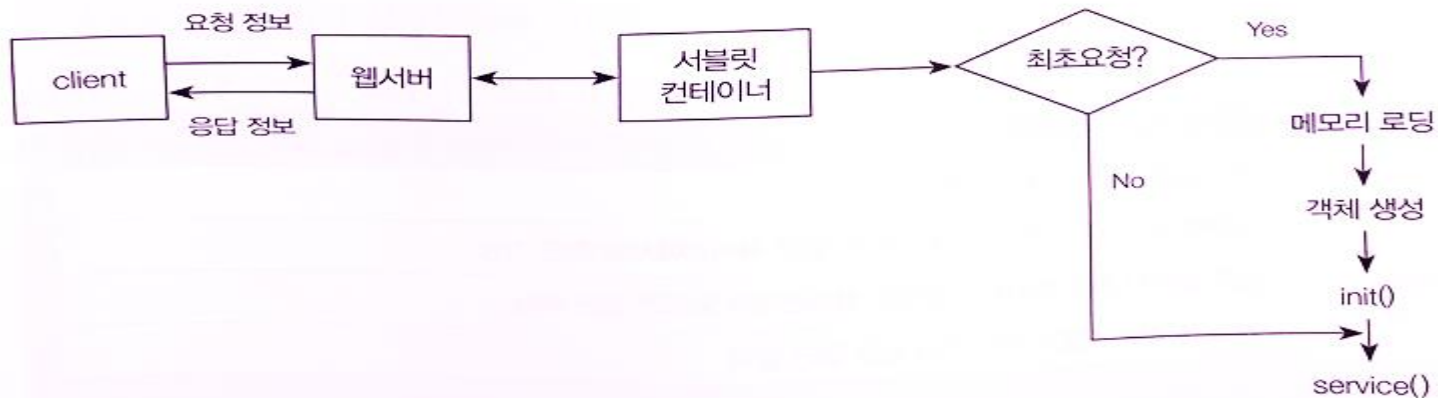
서블릿 프로그램을 만들어 보자

- 서블릿 클래스 요청을 위한 URL 매핑을 보다 쉽게 처리하기 위한 어노테이션을 작성해야 한다.
- 서블릿 버전 3.0이전에는 web.xml을 직접 작성해야 한다.
 - 우리는 우선 web.xml을 편집하여 구동할 것이다.
 - 매핑 주소는 기본값을 클래스 이름이지만 변경이 가능하다.

웹 프로그래밍이란

서블릿의 실행 순서

1. 클라이언트로부터 처리 요청을 받는다.
2. 최초의 요청 여부 판단
3. 서블릿 객체 생성
4. `init()` 메소드 실행
5. `service()` 메소드 실행



웹 프로그래밍이란

출력응답

HttpServletResponse 객체를 활용하여 클라이언트 쪽으로 문자열을 전송한 다음, 웹 브라우저에 문자열을 출력하는 예제를 작성하겠습니다.

SecondServlet.java 서블릿을 작성하여주세요.

```
package com.edu.test;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SecondServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("SecondServlet!!");
        PrintWriter out = response.getWriter();
        out.print("<html><head><title>Test</title></head>");
        out.print("<body><h1>have a nice day!!</h1></body>");
        out.print("</html>");
        out.close();
    }
}
```

웹 프로그래밍이란

response의 타입은 HttpServletResponse입니다. HttpServletResponse는 웹서버에서 클라이언트로 보내는 응답에 관한 일을 처리해주는 객체이므로, 소스에서 response.~는 응답 관련 처리라고 생각하시면 됩니다.

메소드이름이 get으로 시작하는데 어떠한 값을 받아오는 getter 메소드입니다. 따라서 response.get~ 까지 분석하면 응답 관련 작업을 수행하는데 무엇인가를 추출하는구나 라고 생각할 수 있습니다.

그 다음 Writer 라는 이름은 자바에서 외부로 데이터를 출력하기 위한 출력 스트림을 나타낼 때 사용하는 단어입니다.

이어서 out 의 변수타입이 PrintWriter인데 역시 Writer라는 단어가 있지요 이는 출력스트림을 의미합니다.

(서버 ->(out):스트림 클라이언트)

웹 프로그래밍이란

edu\src\main\java\com\edu\test

ThirdServlet 클래스 한글 출력해 보기

```
public class ThirdServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    • public void init(ServletConfig config) throws ServletException {
        System.out.println("init method");
    }
    • public void destroy() {
        System.out.println("destroy method");
    }
    // protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    //     System.out.println("service method");
    // } //브라우저 요청과 상관없는 통신
    • protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("doGet method"); |
        PrintWriter out = response.getWriter(); //String객체
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World </title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello Servlet</h1>");
        out.println("<div>한글을 적어봅시다.</div>");
        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

웹 프로그래밍이란

한글이 깨지는 것을 방지하기 위해서는 서버가 클라이언트로 보내는 데이터의 문서타입과 한글을 지원하는 문자 셋을 응답정보 헤더에 설정해서 보내야 합니다. 문서 타입과 문자셋을 설정하려면 `HttpServletResponse`의 `setContentType()` 메소드를 사용합니다.

소스를 수정해봅시다.

`ContentType` 헤더에는 두 가지 정보를 설정해야 하는데요, 세미콜론(;)을 기준으로 앞에는 문서의 타입을 지정하고, 뒤에는 `charset=` 다음에 문자셋을 지정합니다.

웹 프로그래밍이란

ThirdServlet 클래스 한글 출력해 보기

```
private static final long serialVersionUID = 1L;
public void init(ServletConfig config) throws ServletException {
    System.out.println("init method");
}
public void destroy() {
    System.out.println("destroy method");
}
// protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
//     System.out.println("service method");
// }//브라우저 요청과 상관없는 통신
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    System.out.println("doGet method");
    //한글 깨짐 방지
    // response.setContentType("text/html");
    // response.setCharacterEncoding("UTF-8"); //문자만 설정
    response.setContentType("text/html; charset=UTF-8");
    PrintWriter out = response.getWriter(); //String객체
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Hello World </title>");
    out.println("</head>");
    out.println("<body>");
        out.println("<h1>Hello Servlet</h1>");
        out.println("<div>한글을 적어봅시다.</div>");
    out.println("</body>");
    out.println("</html>");
    out.close();
}
```


서블릿 프로그램을 만들어 보자

- 응답객체 – response

setContentType메소드는 응답할 페이지의 환경을 설정해 준다.

가장 기본적인 응답방식 text/html

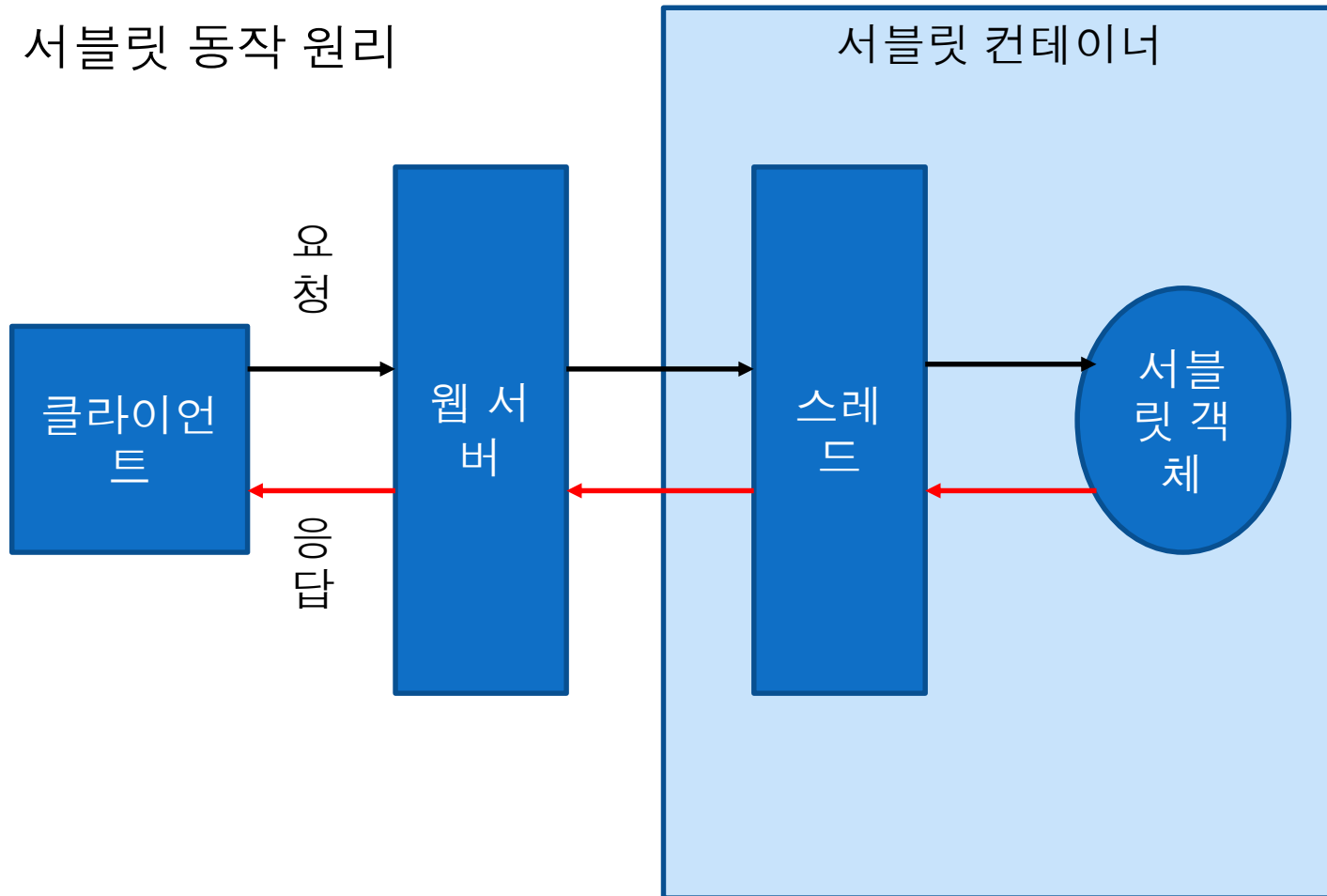
문자 인코딩 방식 charset=UTF-8

getWriter() 결과 출력을 위한 출력 스트림을 만들어 준다.

출력 스트림으로 만들어진 문자열을 응답해 준다.

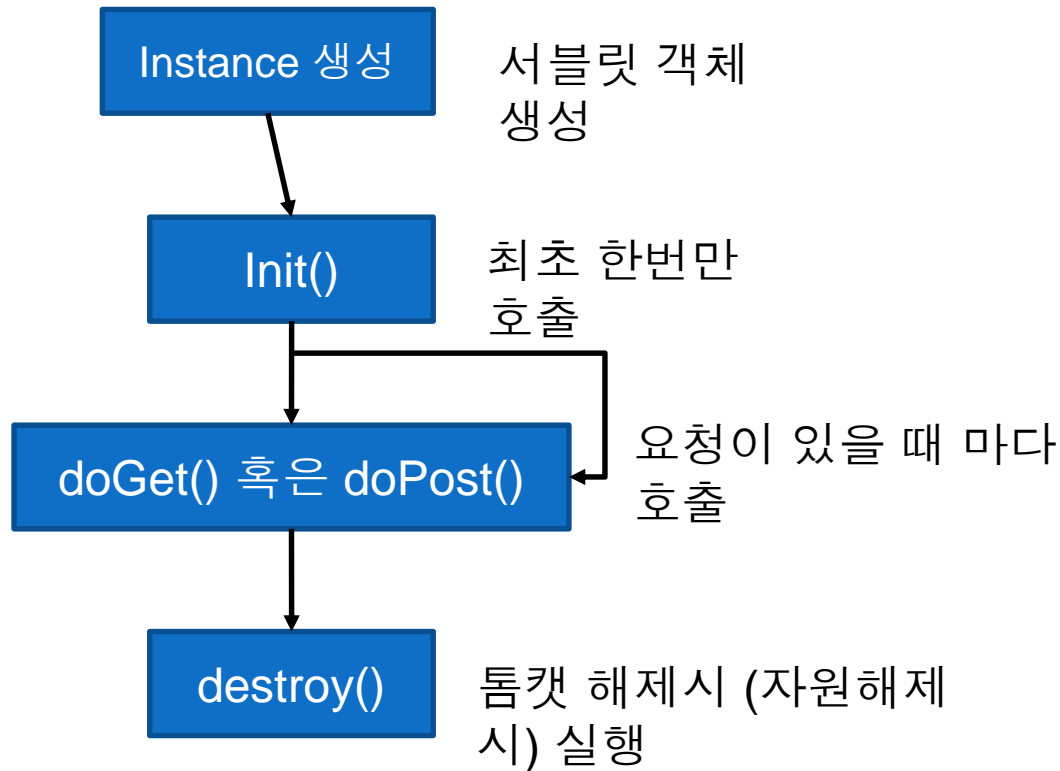
서블릿 프로그램을 만들어 보자

- 서블릿 동작 원리



서블릿 프로그램을 만들어 보자

- 서블릿의 라이프 사이클



요청 정보 처리

- **HttpServletRequest**

- 서블릿은 주로 웹서버의 애플리케이션 기술로 활용되므로 **HTTP** 프로토콜 기반의 애플리케이션이라고 할 수 있습니다. 그러므로 클라이언트에서 웹서버로 서블릿 수행을요청할 때는 **HTTP** 프로토콜의 요청 규약을 사용하는데, 이때 **HTTP** 요청 규약에 알맞게 다양한 요청 정보들을 전달합니다.
- 브라우저에 적절한 **URL** 문자열을 이용하여 웹서버에 서블릿수행을 요청할 때 일정한 형식의 다양한 정보를 서버로 전달합니다.
 - 클라이언트 IP주소, 포트번호
 - 클라이언트가 전송한 요청 헤더정보 (클라이언트에서 처리 가능한문서 타입의 종류,클라이언트 프로그램 정보, 처리 가능한 문자셋 정보, 쿠키)
 - 요청방식,요청 프로토콜의 종류와 버전, 요청하는 파일의 **URI**, 요청 받은 서버의 정보
 - 서버의 호트스 이름, 포트 번호
 - 사용자가 서블릿 요청 시 추가로 전달한 정보
 - 질의 문자열(name=value 형태)

요청 정보 처리

- **HttpServletRequest**

- 이와 같은 다양한 요청 정보는 HttpServletRequest 인터페이스의 get~() 메소드를 통해 추출할 수 있습니다
- 이 정보들은 service() 나 doGet(), doPost() 메소드의 첫 번째 인자로 전달됩니다.
- HttpServletRequest 인터페이스는 ServletRequest를 상속받습니다.
 - ServletRequest: 일반적인 네트워크 통신에서의 요청 관련 메소드 제공
 - HttpServletRequest: HTTP 통신 기반의 요청 관련 메소드를 확장하여 제공

요청 정보 처리

- `HttpServletRequest`

- `ServletRequest`는 클라이언트로부터 서비스 요청이 들어올 때 호출되는 `HttpServlet`의 `service(ServletRequest, ServletResponse)` 메소드의 첫 번째 인자로 전달되며, 이 메소드는 다시 `service(HttpServletRequest, HttpServletResponse)` 메소드를 호출합니다.

요청 정보 처리

- ServletRequest 의 주요 메소드

반환값	메소드	사용 용도
void	setCharacterEncoding(String env)	요청 페이지에 env의 인코딩 방법을 적용
String	getParameter(String name)	name의 요청 인자 값을 반환, 없으면 null을 반환, 만일 값이 여러 개이면 첫 번째 값만 반환
String[]	getParameterValues(String name)	지정한 name의 요청 인자 값을 문자열 배열로 반환, 없으면 null을 반환
Enumeration	getParameterNames()	모든 인자의 이름을 Enumeration으로 반환
String	getProtocol()	사용중인 프로토콜을 반환
String	getRemoteAddr()	클라이언트의 IP 주소를 반환
String	getRemoteHost()	클라이언트의 호스트 이름을 반환
String	getServerName()	요청된 서버의 호스트 이름을 반환
int	getServerPort()	요청된 서버의 포트 번호를 반환

요청 정보 처리

- HttpServletRequest 의 주요 메소드

반환값	메소드	사용 용도
Cookie[]	getCookies()	클라이언트에 보내진 쿠키 배열을 반환
String	getQueryString()	URL에 추가된 Query 문자열을 반환
String	getRequestURI()	클라이언트가 요청한 URI 반환, URI는 프로토콜, 서버이름, 포트번호를 제외한 서버의 컨텍스트와 파일의 문자열
String	getRequestURL()	클라이언트가 요청한 URL 반환, URL은 프로토콜과 함께 주소 부분에 기술된 모든 문자열
HttpSession	getSession()	현재의 세션을 반환, 세션이 없으면 새로 만들어 반환
String	getMethod()	요청 방식인 get, post 중의 하나를 반환

요청 정보 처리

- 예제

```
package com.edu.test;

import java.io.*;

@WebServlet("/netInfo")
public class NetInfoServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html;charset=EUC-KR");
        PrintWriter out = res.getWriter();
        out.print("<html>");
        out.print("<head><title>Request 정보 출력 Servlet</title></head>");
        out.print("<body>");
        out.print("<h3>Request 정보 출력</h3>");
        out.print("Request Scheme : " + req.getScheme() + "<br/>");
        out.print("Server Name : " + req.getServerName() + "<br/>");
        out.print("Server Address : " + req.getLocalAddr() + "<br/>");
        out.print("Server Port : " + req.getServerPort() + "<br/>");
        out.print("Client Address : " + req.getRemoteAddr() + "<br/>");
        out.print("Client Host : " + req.getRemoteHost() + "<br/>");
        out.print("Client Port : " + req.getRemotePort() + "<br/>");
        out.print("</body></html>");
        out.close();
    }
}
```

Request 정보 출력

Request Scheme : http
Server Name : localhost
Server Address : 0:0:0:0:0:0:0:1
Server Port : 8000
Client Address : 0:0:0:0:0:0:0:1
Client Host : 0:0:0:0:0:0:0:1
Client Port : 63849

요청 정보 처리

```
out.print("Request Scheme : " + req.getScheme() + "<br/>");
```

req.getScheme(): 클라이언트가 웹 서버에 서비스를 요청할 때 사용한 프로토콜 이름을 반환한다.

```
out.print("Server Name : " + req.getServerName() + "<br/>");
```

req.getServerName(): 서버의 호스트 이름을 반환하다. 별도의 호스트 이름이 없으면 IP 반환

```
out.print("Server Address : " + req.getLocalAddr() + "<br/>");
```

req.getLocalAddr(): 클라이언트의 서비스를 요청 받아 처리하는 서버의 포트 번호를 반환한다.

```
out.print("Client Address : " + req.getRemoteAddr() + "<br/>");
```

req.getRemoteAddr(): 서비스를 요청한 클라이언트의 IP주소 반환한다.

```
out.print("Client Host : " + req.getRemoteHost() + "<br/>");
```

req.getRemoteHost(): 서비스를 요청한 클라이언트의 호스트 이름을 반환합니다.

```
out.print("Client Port : " + req.getRemotePort() + "<br/>");
```

req.getRemoteHost(): 서비스를 요청한 클라이언트의 소스 포트번호를 반환

요청 정보 처리

- URL 정보

- 이번에는 `HttpServletRequest` 인터페이스에서 제공하는 메소드 중에서 URL 정보를 추출하는 메소드들을 실행한 후 결과값을 확인하는 예제를 작성해보겠습니다.

요청 정보 처리

- URLInfoServlet

```
@WebServlet("/urlInfo")
public class URLInfoServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html;charset=EUC-KR");
        PrintWriter out = res.getWriter();
        out.print("<html>");
        out.print("<head><title>Request 정보 출력 Servlet</title></head>");
        out.print("<body>");
        out.print("<h3>요청 헤더 정보10</h3>");
        out.print("Request URI : " + req.getRequestURI() + "<br/>");
        out.print("Request URL : " + req.getRequestURL() + "<br/>");
        out.print("Context Path : " + req.getContextPath() + "<br/>");
        out.print("Request Protocol : " + req.getProtocol() + "<br/>");
        out.print("Servlet Path : " + req.getServletPath() + "<br/>");
        out.print("</body></html>");
        out.close();
    }
}
```

요청 헤더 정보10

Request URI : /edu/urlInfo

Request URL : http://localhost:8000/edu/urlInfo

Context Path : /edu

Request Protocol : HTTP/1.1

Servlet Path : /urlInfo

요청 정보 처리

```
out.print("Request URI : " + req.getRequestURI() + "<br/>");
```

req.getRequestURI() 는 클라이언트가 요청한 문서정보를 반환합니다. 클라이언트가 서비스를 요청했을 때 HTTP 프로토콜에 의해 요청정보가 웹서버에 전달됩니다.

```
out.print("Request URL : " + req.getRequestURL() + "<br/>");
```

req.getRequestURL()은 클라이언트가 서비스를 요청한 문서의 전체정보, 즉 URL을 반환합니다.

```
out.print("Context Path : " + req.getContextPath() + "<br/>");
```

req.getContextPath()는 웹 애플리케이션의 경로 정보를 반환합니다. 서블릿 페이지에 접근하기 전에 먼저 서블릿이 소속된 웹 애플리케이션에 접근해야 합니다. 웹 애플리케이션에 접근할 때 URL에서 포트번호와 슬래시 다음, 웹 애플리케이션 이름을 지정합니다. getContextPath()는 바로 이 값을 추출하는 메소드 입니다.

```
out.print("Request Protocol : " + req.getProtocol() + "<br/>");
```

req.getProtocol()은 서비스를 처리하면서 사용되는 프로토콜의 구체적인 정보를 반환합니다.

```
out.print("Servlet Path : " + req.getServletPath() + "<br/>");
```

req.getServletPath()는 웹 애플리케이션을 루트 디렉터리를 기준으로 서블릿의 경로를 반환합니다.

모든 헤더 정보

- HTTP 프로토콜의 요청정보는 헤더와 몸체(body)로 구성되어 있고, 헤더는 두 번째 줄 이후부터는 “name:valaue” 형태로 헤더 정보들이 들어있습니다.
- HttpServletRequest 인터페이스에서 제공하는 메소드를 이용하여 요청정보의 헤더 정보들을 추출한 후 결과값을 확인하는 예제를 작성해보겠습니다.

요청 정보 처리

- HeaderInfoServlet.java

```
@WebServlet("/headerInfo")
public class HeaderInfoServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html;charset=EUC-KR");
        PrintWriter out = res.getWriter();
        out.print("<html>");
        out.print("<head><title>Request 정보 출력 Servlet</title></head>");
        out.print("<body>");
        out.print("<h3>요청 헤더 방식</h3>");
        Enumeration<String> em = req.getHeaderNames();
        while (em.hasMoreElements()) {
            String s = em.nextElement();
            out.println(s + " : " + req.getHeader(s) + "<br/>");
        }
        out.print("</body></html>");
        out.close();
    }
}
```

요청 정보 처리

- HeaderInfoServlet.java

```
@WebServlet("/headerInfo")
public class HeaderInfoServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html;charset=EUC-KR");
        PrintWriter out = res.getWriter();
        out.print("<html>");
        out.print("<head><title>Request 정보 출력 Servlet</title></head>");
        out.print("<body>");
        out.print("<h3>요청 헤더 방식</h3>");
        Enumeration<String> em = req.getHeaderNames();
        while (em.hasMoreElements()) {
            String s = em.nextElement();
            out.println(s + " : " + req.getHeader(s) + "<br/>");
        }
        out.print("</body></html>");
        out.close();
    }
}
```


요청 정보 처리

- HeaderInfoServlet.java

요청 헤더 방식

```
host : localhost:8000
connection : keep-alive
sec-ch-ua : "Chromium";v="94", "Google Chrome";v="94", ";Not A Brand";v="99"
sec-ch-ua-mobile : ?0
sec-ch-ua-platform : "Windows"
upgrade-insecure-requests : 1
user-agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.81 Safari/537.36
accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
sec-fetch-site : none
sec-fetch-mode : navigate
sec-fetch-user : ?1
sec-fetch-dest : document
accept-encoding : gzip, deflate, br
accept-language : ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7,zh-CN;q=0.6,zh;q=0.5
```

요청 정보 처리

```
Enumeration<String> em = req.getHeaderNames();
```

req.getHeaderNames() 는 요청정보의 헤더 안에 있는 정보 중 헤더 이름들만 모아 Enumeration 객체에 담아서 반환하고, 반환된 값의 시작 주소를 em 변수에 저장합니다. Enumeration 객체도 java.util 패키지에 있으며, 배열처럼 데이터 그룹으로 구성된 collection 객체입니다. 제네릭을 이용하여 Enumeration <String> 객체 안에 저장되는 데이터의 타입을 String 으로 선언하고 있습니다.

제네릭은 Collection 객체에 담기는 데이터의 타입으로 Collection 객체 생성 시 미리 선언하는 기능입니다.

ArrayList<String> list = new ArrayList<String>(); 이렇게 선언하면 list 안에는 String 타입으로 데이터만 저장하겠다는 의미여서 String이 아닌 데이터를 저장하면 오류가 발생합니다. 또한 추출할 때는 자동으로 String 타입으로 변환됩니다.

Enumeration객체가 Set, List, Map 계열의 Collection 객체와 다른 점은 그룹의 데이터에 접근할 때 인덱스나 키가 아닌 커서라는 개념으로 접근한다는 사실입니다.

req.getHeaderNames() 메소드가 헤더 정보의 이름들을 String 타입으로 Enumeration 객체에 담아 반환하면 Enumeration 의 첫 번째 요소 앞에 커서가 위치합니다.

요청 정보 처리

```
while (em.hasMoreElements()) {
```

em.hasMoreElements()는 em 이 가리키는 Enumeration 객체의 커서 다음에 데이터가 있는지 없는지 판단하여 있으면 true, 없으면 false 를 반환합니다. 커서가 마지막 요소에 있을 때 비로소 false를 반환하고 while 반복문을 빠져나옵니다.

```
String s = em.nextElement();
```

em.nextElement()는 em의 커서 다음에 있는 요소를 반환하고 커서를 다음 요소로 이동시킵니다. em변수를 선언할 때 Enumeration<String> 으로 선언했으므로 반환하는 값은 String 타입이며, 반환 값을 String s 변수에 저장합니다.

```
out.println(s + " : " + req.getHeader(s) + "<br/>");
```

S 변수에는 요청정보의 헤더 이름이 들어있습니다. 헤더 이름을 getHeader() 메소드의 인자로 지정하면 해당 이름을 찾아 값을 반환합니다. 즉, getHeader() 는 헤더의 값을 추출할 때 사용하는 메소드입니다.

잠깐 Collection 이란?

프로그램에서 데이터를 그룹으로 처리할 수 있는 방법은 배열을 이용하는 것입니다. 그런데 배열은 크기를 변경할 수 없고, 하나의 배열에 다른 타입의 데이터를 저장할 수 없는 제약 사항이 있습니다.

그래서 `collection`이라는 객체를 이용하면 이러한 제한사항을 벗어나 데이터를 그룹으로 처리할 수 있습니다.

`Collection`은 `java.util` 패키지 안에 있으며, 객체 이름에 `Set`, `List`, `Map` 의 단어가 포함되어 있습니다. `Java.util` 패키지에는 `set`, `List`, `Map` 인터페이스를 상속받는 하위 객체가 많습니다.

`Enumeration` 과 `Iterator` : 이 두 객체는 그룹 안에 있는 요소에 접근할 때 인덱스나 키로 접근하는 것이 아니고, 커서 개념으로 접근합니다. 주로 `collection` 안에 있는 모든 요소를 차례로 접근할 때 사용하면 편리하며, 속도 면에서도 빠릅니다.

추가 정보

- `HttpServletRequest` 인터페이스에서 제공하는 메소드 중에서 질의 문자열이나 추가 경로 정보를 추출하는 메소드를 실행한 후 결과값을 확인 예제를 작성하겠습니다.

추가 정보

@WebServlet("/addInfo/*")

```
public class AdditionalInfoServlet extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/html;charset=EUC-KR");  
        PrintWriter out = res.getWriter();  
        out.print("<html>");  
        out.print("<head><title>Request 정보 출력 Servlet</title></head>");  
        out.print("<body>");  
        out.print("<h3>추가적인 요청 정보</h3>");  
        out.print("Request Method : " + req.getMethod() + "<br/>");  
        out.print("Path Info : " + req.getPathInfo() + "<br/>");  
        out.print("Path Translated : " + req.getPathTranslated() + "<br/>");  
        out.print("Query String : " + req.getQueryString() + "<br/>");  
        out.print("Content Length : " + req.getContentLength() + "<br/>");  
        out.print("Content Type : " + req.getContentType() + "<br/>");  
        out.print("</body></html>");  
        out.close();  
    }  
}
```

추가 정보

```
out.print("Request Method : " + req.getMethod() + "<br/>");
```

- 클라이언트는 여러 가지의 요청방식을 지정하여 서비스를 요청할 수 있습니다. 이때 지정한 요청방식은 요청정보의 헤더에서 첫 번째 단어로 표시되어 전달됩니다.
- req.getMethod() 는 바로 이 부분의 정보를 추출하는 메소드입니다. 즉, 클라이언트의 요청방식을 알고자 할 때 사용하는 메소드입니다. (기본 값은 GET)
- 앞에서 서블릿을 요청할 수 있는 방법은 두가지가 있다고 했습니다.
 - 1. web.xml 수정
 - 2. @WebServlet 어노테이션 이용
- 이번에는 web.xml을 수정하여 봅시다.

```
//@WebServlet("/addInfo")
public class AdditionalInfoServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html; charset=EUC-KR");
```

추가 정보 (web.xml)

```
<servlet>
  <servlet-name>addInfo</servlet-name> ③ - 자바 파일을 addInfo라는 이름으로 등록 -->
  <servlet-class>com.edu.test.AdditionalInfoServlet</servlet-class> ④ - .java생략, 컨트롤러 누르고 밑줄쳐져야 정상 -->
</servlet>
<!-- 2.사용자의 요청 -->
<servlet-mapping>
  <servlet-name>addInfo</servlet-name> ② -- 서블릿 중에 addInfo 라는 애 있네 -->
  <url-pattern>/addInfo</url-pattern> ① -- addInfo 라는 요청이 오면 -->
</servlet-mapping>
```

- ① 에서 **<url-pattern>/addInfo</url-pattern>** 으로 지정했으므로 클라이언트로부터 <http://localhost:8000/edu/addInfo> 요청이 들어오면 ② 에 지정된 이름과 같은 이름을 찾습니다. ③에서 찾았습니다. 그 다음 **<servlet-class>**에 지정된 서블릿을 실행합니다. 즉, **com.edu.test.AdditionalInfoServlet**의 **service()** 메소드가 실행됩니다.
- 그런데 클라이언트 요청 URL 과 매핑되는 **<url-pattern>**을 지정할 때 다음 예처럼 * 기호를 사용할 수 있습니다.

추가 정보 (web.xml)

```
<!-- 1.서블릿 등록 -->
<servlet>
  <servlet-name>addInfo</servlet-name> <
  <servlet-class>com.edu.test.Additional
</servlet>
<!-- 2.사용자의 요청 -->
<servlet-mapping>
  <servlet-name>addInfo</servlet-name> ②
  <url-pattern>/addInfo/*</url-pattern> ①
</servlet-mapping>
```

- ① 에서 /addInfo/*으로 수정한게 보이시죠.
- <url-pattern>의 값은 클라이언트가 서버에 서비스를 요청할 때 보내는 URL에서 웹 애플리케이션 이름 다음에 오는 값을 의미합니다. 현재 웹 애플리케이션을 찾아오기까지의 URL 정보는 이미 (<http://localhost:8000/edu>)로 고정되어 있어서 생략할 수 있습니다.
- * 라는 문자는 그 자리에 어떤 경로가 와도 상관 없다는 의미입니다.

Request 정보 출력 Servlet

localhost:8080/JspWeb/addInfo/a

추가적인 요청 정보

Request Method : GET
Path Info : /a
Path Translated : C:\Dev\MyWorkspace\metadata\plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\JspWeb\addInfo/a
Query String : null
Content Length : -1
Content Type : null

Request 정보 출력 Servlet

localhost:8080/JspWeb/addInfo/qwerty

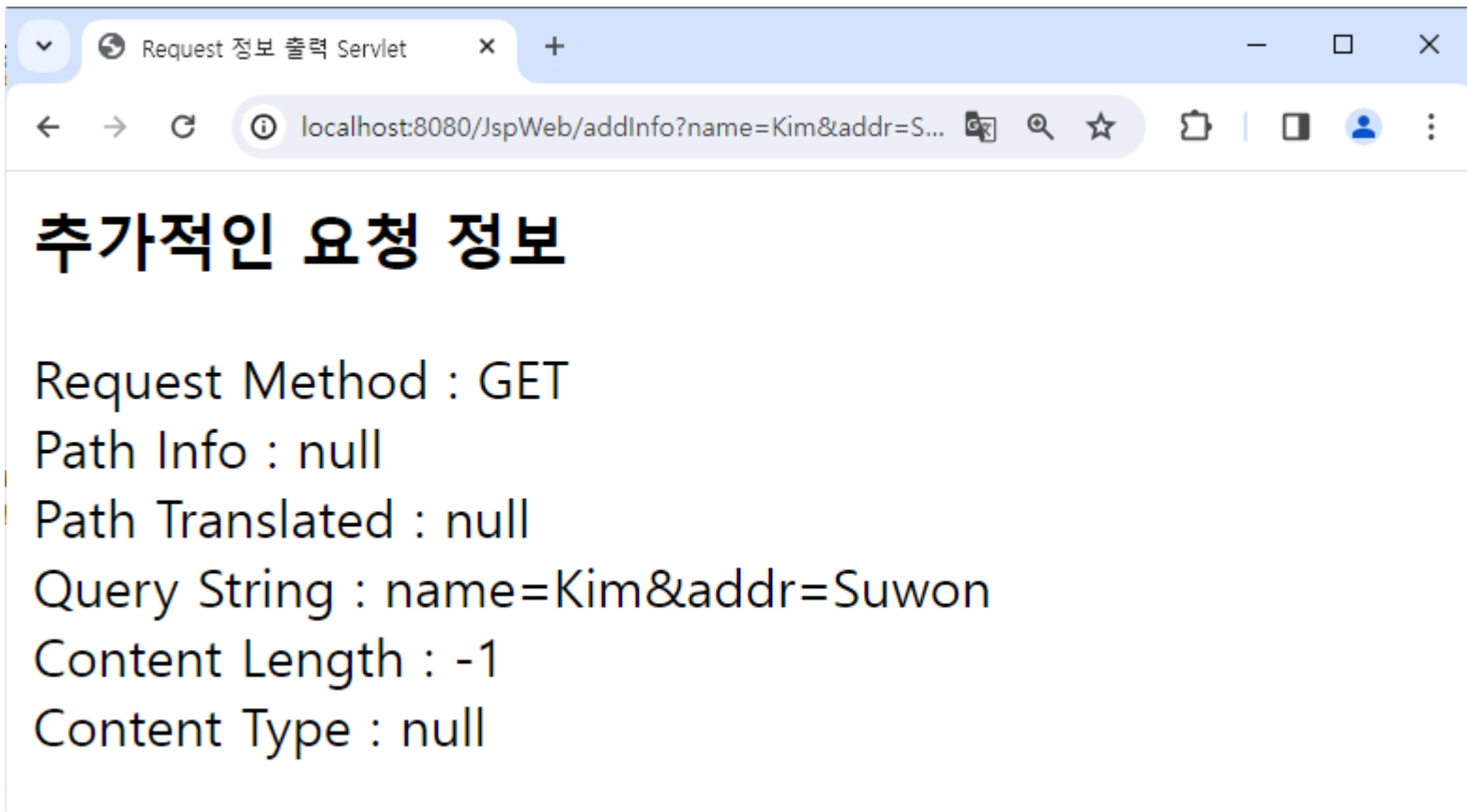
추가적인 요청 정보

Request Method : GET
Path Info : /qwerty
Path Translated : C:\Dev\MyWorkspace\metadata\plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\JspWeb\addInfo/qwerty
Query String : null
Content Length : -1
Content Type : null

추가 정보

```
out.print("Query String : " + req.getQueryString() + "<br/>");
```

- req.getQueryString()는 요청 URL에서 ?를 찾아서 다음에 있는 문자열 전체를 추출하여 반환합니다.
 - http://localhost:8000/edu/addInfo?name=Kim&addr=Suwon



질의 문자열

- 질의 문자열이란? 웹 클라이언트에서 웹서버에 정보를 요청할 때 정해진 방식으로 데이터를 전달할 수 있으며, 이때 사용하는 문자열을 질의 문자열이라고 합니다.
- 웹에서 질의 문자열을 사용하게 되면서 웹을 서비스 제공자와 이용자의 상호작용 매체로 활용할 수 있게 되었습니다. 질의 문자열 처리는 지금처럼 웹이 다양한 목적으로 사용되는 데 중요한 역할을 담당하는 기술입니다.
- 질의 문자열이 무엇인지와 인코딩 규칙, 전달 방식의 규칙 등을 알아보겠습니다.

질의 문자열

- 검색 사이트에 접속하여 검색어를 입력하거나, 회원가입시 입력한 데이터들 모두가 질의 문자열에 속합니다.
- 이처럼 웹 클라이언트에서 웹서버에 정보를 요청할 때 정해진 방식으로 데이터를 전달할 수 있으며, 이때 사용하는 문자열을 질의 문자열이라고 합니다.
- 대표적인 웹 클라이언트인 웹 브라우저에서 다양한 정보를 전달할 수 있도록 입력 양식을 제시할 수 있는데 이때 사용되는 HTML 태그가 바로 `<form>` 태그입니다. 웹 브라우저 화면에는 버튼, 텍스트 필드, 체크박스, 라디오버튼과 같은 GUI환경의 입출력 도구들을 출력하는 기능과 서버 프로그램의 URI를 설정하는 기능 등이 있습니다.

질의 문자열 규칙

- 질의 문자열 작업을 하려면 먼저 사용자가 데이터를 입력할 수 있는 화면을 만들어야 합니다.
- 질의 문자열들이 클라이언트에서 서버 쪽으로 전달될 때 입력한 데이터 그대로 전달되는 것이 아니라, 정해진 규칙으로 인코딩되어 전달됩니다. 이 규칙은 요청방식과 관계없이 동일하게 적용됩니다.
- 다음 규칙이 적용됩니다.
 - ① name=value 형식으로 전달되며, 여러 개의 name=value 쌍이 있을 때는 & 를 구분자로 사용합니다.
 - ② 영문자, 숫자, 일부 특수문자는 그대로 전달되고, 이를 제외한 나머지 문자는 % 기호와 함께 16진수로 바뀌어 전달됩니다.
예) id-geust&name=%C9%AB%B1%E6
 - ③ 공백은 + 기호로 변경되어 전달됩니다.

데이터 통신

- `<form action="서버프로그램 경로" method="요청방식">`
- **action:** 클라이언트가 태그 사이에 입력한 질의 문자열을 전달 받아 처리할 서버 프로그램을 지정합니다. (JSP , servlet 등)
- **action** 속성으로 프로그램의 위치를 지정할 때는 상대 경로나 절대 경로를 사용할 수 있는데요 자주 사용하는 것은 상대 경로입니다. 상대 경로는 현 위치에 따라 지정하는 위치가 달라집니다.
- 현재 위치를 점(.)으로 표시, 상위 디렉토리를 점 두 개(..)로 표시, 하위 디렉터리는 / 기호 다음에 표시합니다.
- **method:** GET or POST 등 여러 가지가 있지만 일반적으로 이 두 개를 가장 많이 사용합니다.
- **form**태그에 적용된 내용을 전달하기 위해서는 반드시 전송 버튼이 있어야 한다.
 - `<input type="submit" value="전송">`

데이터 통신

- Get 전송 예제

```
<form method="get" action="MethodServlet">  
    <input type="submit" value="get 방식으로 호출하기">  
</form>
```

- POST 전송 예제

```
<form method="post" action="MethodServlet">  
    <input type="submit" value="post 방식으로 호출하기">  
</form>
```

- 서블릿

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) {  
    response.setContentType("text/html; charset=UTF-8");  
    PrintWriter out = response.getWriter();  
    out.print("<h1>get 방식으로 처리됨</h1>");  
    out.close();  
}
```

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) {  
    response.setContentType("text/html; charset=UTF-8");  
    PrintWriter out = response.getWriter();  
    out.print("<h1>post 방식으로 처리됨</h1>");  
    out.close();  
}
```


데이터 통신 – 질의 문자열

- 앞서 본 예는 클라이언트가 정보를 보내는 방식에 대해 알아보았다면 반대로 서버에서 정보를 읽어오는 방법에 대해 알아보자
- 서버에서는 클라이언트가 보낸 정보를 읽어 들이기 위해서 질의 문자열이라는 기술을 사용한다.
- 질의 문자열을 사용하는 이유는 웹 페이지의 특성 때문인데 웹 페이지는 특성상 현재 페이지의 데이터를 다음 페이지가 인지 하지 못하고 전부 잃어버리기 때문이다.
- 그래서 페이지간의 정보를 교환할 필요를 질의 문자열을 통해서 처리한다.

데이터 통신 – 질의 문자열

- 클라이언트에서 서버로 데이터를 전송하기 위해서는 기본적으로 데이터를 입력할 수 있는 텍스트 박스가 필요하다.

```
<input type="text" name="id">  
<input type="text" name="age">
```

- 위 데이터는 서버로 데이터를 전송하는 데 name부분의 데이터를 ‘이름’ 입력된 값을 ‘값’ 형태의 질의 문자열형태로 전송된다.
- 서버에서는 질의 문자열로 전송되는 데이터를 어떻게 읽어 드릴까?

데이터 통신 – 질의 문자열

- 서버에서는 클라이언트로부터 오는 모든 데이터를 요청 객체 (request)에 담아서 활용한다.

- 필요한 데이터는 request객체로 부터 얻을 수 있다.

```
request.getParameter('이름');
```

- 위 메소드로 얻어온 데이터는 문자열 형태로 사용가능 하다.
- 숫자가 필요하다면 문자열을 숫자로 변환해서 사용한다.

요청방식에 따른 처리

지금까지 클라이언트가 서버로 전달하기 위한 값들을 입력 받는 화면을 만들기 위한 내용에 대해 알아보았습니다. 이제 화면에서 입력 받은 값들을 서버로 전달하기 위한 방식을 알아보겠습니다. 클라이언트가 입력 또는 선택한 질의 문자열들을 서버로 전달하기 위한 방식은 여러 가지가 있습니다. 그 중에서 대표적인 요청방식인 **GET**과 **POST** 방식에 대해 살펴보겠습니다.

요청방식에 따른 처리

GET 방식으로 처리

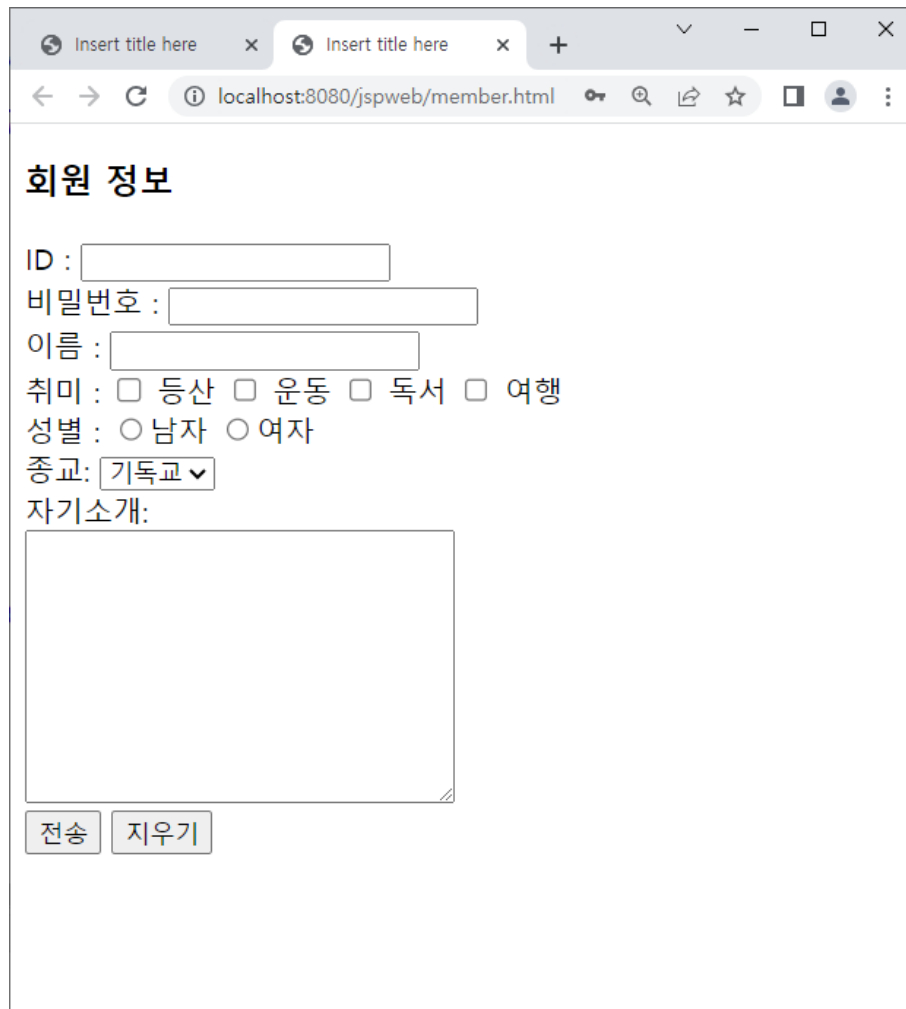
Src/webapp/member.html

```
<body>
  <h3>회원 정보</h3>
  <form action="queryTest" method="GET">
    ID : <input type="text" name="id" /><br/>
    비밀번호 : <input type="password" name="pwd" /> <br/>
```

action과 method 값을 입력한 후 run on server를 실행해 봅시다.

요청방식에 따른 처리

값을 입력한 후 전송을 클릭해봅시다.



The screenshot shows a web browser window with two tabs, both titled 'Insert title here'. The address bar displays 'localhost:8080/jspweb/member.html'. The page content is titled '회원 정보' (Member Information). The form includes the following fields and options:

- ID :
- 비밀번호 :
- 이름 :
- 취미 : ☐ 등산 ☐ 운동 ☐ 독서 ☐ 여행
- 성별 : ☐ 남자 ☐ 여자
- 종교: (dropdown menu)
- 자기소개:

At the bottom of the form, there are two buttons: '전송' (Submit) and '지우기' (Clear).

요청방식에 따른 처리

우리는 아직 서버 프로그램을 만들지 않았으므로 “404” 에러 페이지를 확인할 수 있습니다.

HTTP 상태 404 – 찾을 수 없음

타입 상태 보고

메시지 요청된 리소스 [/edu/queryTest]은(는) 가용하지 않습니다.

설명 Origin 서버가 대상 리소스를 위한 현재의 representation을 찾지 못했거나, 그것이 존재하는지를 밝히려 하지 않습니다.

Apache Tomcat/8.5.70

URI 를 보면 폼에서 입력한 데이터가 name = value 형태로 표시된 것을 확인할 수 있습니다.

요청방식에 따른 처리

만일 클라이언트가 `id=guest&name=lee`이라는 두개의 값으로 질의 문자열을 **GET** 방식으로 서버 프로그램 (`/edu/queryTest`)에 요청했다면, 요청 정보 헤더는 다음과 같이 구성됩니다.

```
GET /edu/queryTest?id=guest&name=lee HTTP/1.1  
HOST: localhost:8000
```

요청 정보의 첫 줄에는 요청 방식과 **URI**, **HTTP**버전 등 3개의 정보가 나옵니다. 이처럼 **GET** 방식은 질의 문자열을 요청 정보 헤더의 **URI**에 포함하므로 서버로 전달되는 값이 브라우저 주소 줄에 모두 노출됩니다.

GET 방식은 데이터의 길이가 255바이트 미만이고, 외부에 노출되어도 상관없는 데이터를 전달 할 때 적합한 요청방식입니다.

요청방식에 따른 처리

GET 방식으로 요청되는 상황

1. <a>태그를 클릭하여 요청하는 경우
2. 브라우저 주소 줄에 URL을 입력하여 요청하는 경우
3. <form>태그에서 method 속성을 생략하여 요청하는 경우

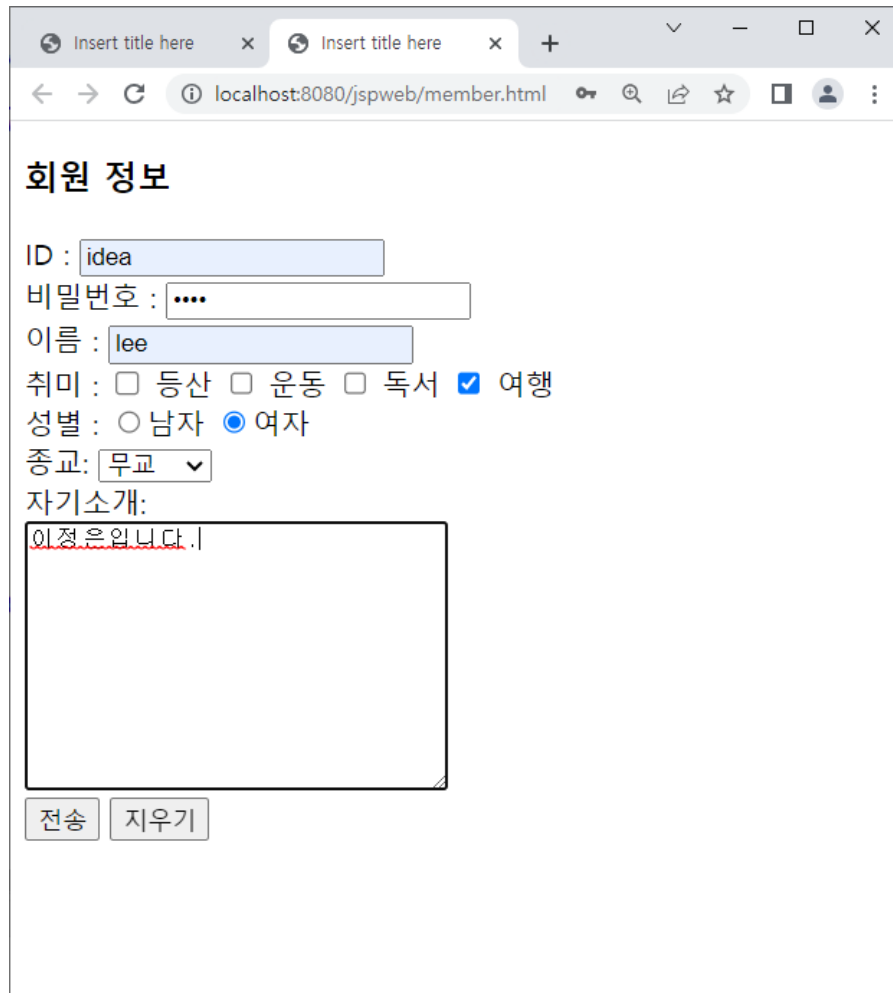
요청방식에 따른 처리

POST 방식으로 처리

```
<body>
  <h3>회원 정보</h3>
  <form action="queryTest" method="POST">
    ID : <input type="text" name="id" /><br/>
    비밀번호 : <input type="password" name="pwd" /> <br/>
```

요청방식에 따른 처리

값을 입력한 후 전송을 클릭해봅시다.



The screenshot shows a web browser window with two tabs, both titled "Insert title here". The address bar displays "localhost:8080/jspweb/member.html". The page content is a registration form titled "회원 정보" (Member Information). The form includes the following fields and options:

- ID :
- 비밀번호 :
- 이름 :
- 취미 : ☐ 등산 ☐ 운동 ☐ 독서 ☒ 여행
- 성별 : ☐ 남자 ☒ 여자
- 종교: (dropdown menu)
- 자기소개:

이정은입니다.

At the bottom of the form, there are two buttons: "전송" (Submit) and "지우기" (Clear).

요청방식에 따른 처리

HTTP 상태 404 – 찾을 수 없음

타입 상태 보고

메시지 요청된 리소스 [/edu/queryTest]은(는) 가용하지 않습니다.

설명 Origin 서버가 대상 리소스를 위한 현재의 representation을 찾지 못했거나, 그것이 존재하는지를 밝히려 하지 않습니다.

Apache Tomcat/8.5.70

이번에는 질의 문자열은 보이지 않고 클라이언트가 서버에 요청한 서버 프로그램의 경로만 보입니다.

이것이 GET과 POST의 가장 눈에 띄게 큰 차이입니다.

요청방식에 따른 처리

POST 방식은 질의 문자열이 요청정보의 헤더가 아닌 몸체에 포함합니다. 따라서 외부에 노출되지 않고 서버에 전달되며 문자열 길이에 제한도 없습니다.

그리고 POST 방식은 클라이언트 측에서 요청정보의 몸체에 전달하는 질의 문자열들을 인코딩해서 보내고, 이를 전달받은 서버 측에서는 다시 디코딩하는 추가 작업이 필요합니다.

POST 방식의 특징

1. 전달되는 질의 문자열이 요청정보의 몸체에 포함되어 전달
2. 전달되는 질의 문자열이 외부에 노출되지 않는다.
3. 전달되는 질의 문자열의 길이에 제한이 없다.
4. <form>태그를 사용해야만 요청할 수 있다.

서블릿 작성

웹에서는 클라이언트가 전달하는 모든 정보들은 HTTP의 요청정보에 포함되어 서버로 전달되어 처리됩니다. 서버에서 요청정보를 처리할 때 사용하는 객체는 `HttpServletRequest`입니다.

`QueryTestServlet.java` 서블릿 파일을 만들어 봅시다.

문자열 추출

클라이언트로 부터 전달된 질의 문자열들을 서블릿에서 추출하는 방법에 대해서 알아보시다.

1. String getParameter(String name)

- 이 메소드는 질의 문자열로 넘어온 값을 하나씩 추출할 때 사용한다. 질의 문자열에서 name 이 중복되지 않고 유일한 값이 넘어올 때 사용한다.

2. String[] getParameterValues(String name)

- 이 메소드는 같은 이름으로 여러 개의 변수가 전달되었을 때 한번에 모든 값을 추출하여 String 타입의 배열로 받고 싶을 때 사용합니다.

3. String getQueryString()

- 이 메소드는 클라이언트가 전달한 질의 문자열 전체를 하나의 String으로 추출해줍니다. 그런데 이 메소드는 GET 방식에서만 사용할 수 있습니다. 왜냐, URL의 ?뒤 내용을 전부 추출하기 때문입니다.

4. ServletInputStream getInputStream() throws IOException

- 이 메소드는 HTTP의 요청정보 몸체와 연결된 입력스트림을 생성하여 반환합니다. 이는 POST 방식의 질의 문자열 전체를 한 번에 추출할 때 사용할 수 있습니다. 반환 받은 후엔 readLine() 메소드를 사용합니다.

문자열 추출

member.html 소스의 method값을 get으로 수정합니다.

```
<body>
  <h3>회원 정보</h3>
  <form action="queryTest" method="GET">
    ID : <input type="text" name="id" /><br/>
    비밀번호 : <input type="password" name="pwd" /> <br/>
```


문자열 추출

QueryTestServlet.java 소스의 doGet() 메소드에 추가해 보세요.

```
String id = req.getParameter("id");
String password = req.getParameter("pwd");
String name = req.getParameter("name");
String[] hobbies = req.getParameterValues("hobby");
String gender = req.getParameter("gender");
String religion = req.getParameter("religion");
String intro = req.getParameter("introduction");

out.print("ID : " + id + "<br/>");
out.print("비밀번호 : " + password + "<br/>");
out.print("이름 : " + name + "<br/>");
out.print("취미 : ");
for (int i = 0; i < hobbies.length; i++) {
    out.print(hobbies[i] + " ");
}
out.print("<br/>");
out.print("성별 : " + gender + "<br/>");
out.print("종교 : " + religion + "<br/>");
out.print("소개 : " + intro + "<br/>");
out.print("전체 Query 문자열 : " + req.getQueryString());

out.println("</body></html>");
out.close();
```

문자열 추출

member.html 소스의 method값을 post으로 수정합니다.

```
<body>
  <h3>회원 정보</h3>
  <form action="queryTest" method="POST">
    ID : <input type="text" name="id" /><br/>
    비밀번호 : <input type="password" name="pwd" /> <br/>
```

문자열 추출

QueryTestServlet.java 소스의 doPost() 메소드에 추가해 보세요.

실행한 후 영어를 입력하세요.
한글을 입력하면 깨집니다.

```
out.print("<body>");
out.print("<h1>POST 방식으로 요청되었습니다</h1>");

String id = req.getParameter("id");
String password = req.getParameter("pwd");
String name = req.getParameter("name");
String[] hobbies = req.getParameterValues("hobby");
String gender = req.getParameter("gender");
String religion = req.getParameter("religion");
String intro = req.getParameter("introduction");

out.print("ID : " + id + "<br/>");
out.print("비밀번호 : " + password + "<br/>");
out.print("이름 : " + name + "<br/>");
out.print("취미 : ");
for (int i = 0; i < hobbies.length; i++) {
    out.print(hobbies[i] + " ");
}
out.print("<br/>");
out.print("성별 : " + gender + "<br/>");
out.print("종교 : " + religion + "<br/>");
out.print("소개 : " + intro + "<br/>");

out.println("</body></html>");
out.close();
```

문자열 추출

POST 전체 문자열 추출

```
@WebServlet("/queryTest2")
public class QueryTest2Servlet extends HttpServlet {
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();
        out.print("<html><head><title>Query 문자열 테스트</title></head>");
        out.print("<body>");
        out.print("<h1>Post방식 Query 문자열 추출</h1>");
        ServletInputStream input = req.getInputStream();
        int len = req.getContentLength();
        byte[] buf = new byte[len];
        input.readLine(buf, 0, len);
        String s = new String(buf);
        out.print("전체 문자열 : " + s);
        out.println("</body></html>");
        out.close();
    }
}
```

문자열 추출

```
ServletInputStream input = req.getInputStream();
```

req.getInputStream()은 요청정보의 몸체와 서블릿 프로그램 간에 연결된 입력 스트림을 생성하여 반환해주는 메소드입니다. getInputStream() 메소드에서 반환된 input 은 요청정보 몸체와 현재 서블릿 간에 연결된 ServletInputStream 객체입니다.

```
int len = req.getContentLength();
```

req.getContentLength()은 요청정보의 몸체에 담겨있는 문자열의 길이를 반환합니다. 요청정보 몸체의 문자열들을 0릭어 들일 때 사용하기 위해 몸체의 길이를 구해서 변수 len에 저장합니다.

```
input.readLine(buf, 0, len);
```

Input 입력스트림으로부터 줄 단위로 데이터를 읽어 들입니다. 메소드의 첫 번째 인자는 한 줄 읽어 들인 문자열을 저장할 바이트 배열을 지정합니다. 두 번째 인자는 저장소에 저장할 시작 위치를 의미합니다. 세 번째 인자는 저장할 문자열의 길이입니다.

문자열 추출

```
String s = new String(buf);
```

buf 배열은 바이트 배열로서 input에서 읽어 들인 한 줄의 데이터가 바이트 형태로 저장되어 있습니다. New String(buf)는 buf배열의 데이터를 String 타입의 데이터로 새로 생성합니다. buf 배열을 이용해 새로 생성한 String의 시작 주소를 String 타입의 변수 s에 저장합니다.

```
out.print("전체 문자열 : " + s);
```

요청정보의 몸체에서 바이트 형태로 읽어 들인 데이터를 buf 배열에 저장했다가 String으로 변환하였습니다. 위 코드는 String으로 변환한 요청 정보 몸체로부터 읽어 들인 문자열(s 변수)을 웹 브라우저로 출력합니다.

한글 처리

POST 방식일 때 클라이언트가 전송한 문자열에 한글이 있을 때 깨졌던 것을 확인했었죠?

한글이 깨지지 않고 올바르게 표현하는 방법에 대해서 알아보겠습니다.
먼저 name.html 을 작성해봅시다.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" >
<title>이름 입력</title>
</head>
<body>
  <form action="queryTest3" method="post">
    이름 : <input type="text" name="name">
    <input type="submit" value="전송">
  </form>
</body>
</html>
```

한글 처리

그 다음 QueryTest3Servlet.java 파일을 작성해봅시다.

```
@WebServlet("/queryTest3")
public class QueryTest3Servlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();
        out.print("<html><head><title>Query 문자열 테스트</title></head>");
        out.print("<body>");
        out.print("<h1>GET 방식으로 요청되었습니다</h1>");

        String name = req.getParameter("name");
        System.out.println(name);
        out.print("이름 : " + name + "<br/>");

        out.println("</body></html>");
        out.close();
    }
}
```


한글 처리

그 다음 QueryTest3Servlet.java 파일을 작성해봅시다.

```
@Override
public void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    resp.setContentType("text/html;charset=UTF-8");
    PrintWriter out = resp.getWriter();
    out.print("<html><head><title>Query 문자열 테스트</title></head>");
    out.print("<body>");
    out.print("<h1>POST 방식으로 요청되었습니다</h1>");

    req.setCharacterEncoding("UTF-8");
    String name = req.getParameter("name");
    out.print("이름 : " + name + "<br/>");

    out.println("</body></html>");
    out.close();
}
```

한글 처리 **주의! Charset은 반드시 통일! Html에서 메타정보도 통일!**

```
req.setCharacterEncoding("UTF-8");
```

해당 메소드는 인자값으로 지정된 문자코드를 사용하여 클라이언트가 전달한 요청정보 몸체에 있는 문자열들을 인코딩해주는 메소드입니다. 클라이언트로부터 질의 문자열이 전달된 후 서블릿이 실행될 때 `setCharacterEncoding()` 메소드로 인코딩 작업을 처리한 후 `getParameter()` 메소드로 추출하여 사용합니다.

```
resp.setContentType("text/html;charset=UTF-8");
```

이 객체는 서버에서 클라이언트로 보내는 응답에 관한 기능을 처리해줍니다. `setContentType()` 메소드는 서버에서 보내는 문서의 타입과 보내는 문자를 처리할 문자 코드를 지정해줍니다. 응답정보의 헤더에 문서타입은 `text/html`로 문자코드는 `UTF-8`로 지정되어 전달합니다. 클라이언트는 이 정보를 보고 응답 받은 데이터를 `html`로 처리하고 문자를 처리할 때는 `UTF-8` 코드를 사용합니다.

서블릿 설정과 변수

웹 애플리케이션을 개발하면서 여러 개의 서블릿 페이지를 작성하게 되는데요. 서블릿은 웹에서 클라이언트로부터 요청 받아서 실행되는 자바 프로그램으로서, 주로 서비스 처리를 위한 데이터 준비 작업과 메소드 호출 역할을 합니다. 이러한 작업을 하려면 서블릿 페이지 내에서가 아니라 서버에서 설정해야 하는 부분이 있는데요. 이번 절에서는 서블릿이 작업하는데 필요한 내용을 설정하는 방법과 서블릿에서 설정한 내용을 추출하여 사용하는 방법에 대해 살펴보겠습니다.

서블릿 설정과 변수

Web.xml 서버에서 서블릿 실행에 관한 정보를 설정할 때는 web.xml <servlet>태그로 설정합니다. Web.xml 파일은 서버가 시작할 때 웹서버가 사용하는 파일인데, 웹 애플리케이션 서비스 실행에 관한 전반적인 내용을 정의하는 환경설정 파일입니다. 서블릿 또한 웹 애플리케이션 서비스를 실행하기 위해 존재하는 파일이므로 web.xml에 정의합니다.

```
<servlet>
  <servlet-name>initParam</servlet-name>
  <servlet-class>com.edu.test.InitParamServlet</servlet-class>
  <init-param>
    <param-name>id</param-name>
    <param-value>guest</param-value>
  </init-param>
  <init-param>
    <param-name>password</param-name>
    <param-value>1004</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>initParam</servlet-name>
  <url-pattern>/initParamTest</url-pattern>
</servlet-mapping>
```

서블릿 설정과 변수

1)<setvlet>,<servlet-name>,<servlet-class> 태그

- 설정하려는 서블릿을 등록합니다.

2)<init-param>태그

- 서블릿에 변수를 전달할 때 사용합니다.
- 서블릿을 실행하면서 필요한 값을 외부에서 전달받아 실행할 수 있는데, 서블릿 소스에서 직접 값을 지정해서 사용해도 되지만, 실행환경에 맞게 동적으로 값을 할당하고자 할 때 외부에서 값을 전달할 수 있다.

<init-param>태그는 반드시 <param-name>과<param-value>태그로 구성해야 한다.

```
<param-name>id</param-name>
```

<param-name>은 <init-param> 태그 사용 시 반드시 설정해야 하는 태그로서 변수의 이름을 지정합니다. 서블릿 측에서는 <param-name>에 지정한 값을 이용하여 변수의 값을 추출합니다. 이때 변수의 이름은 대소문자, 철자를 구분하여 <param-name>에서 정확히 일치하는 변수의 이름을 찾아서 값을 추출합니다.

```
<param-value>guest</param-value>
```

<param-value>는 매핑되는 <param-name>에 저장되는 변수의 값을 지정하는 태그입니다.

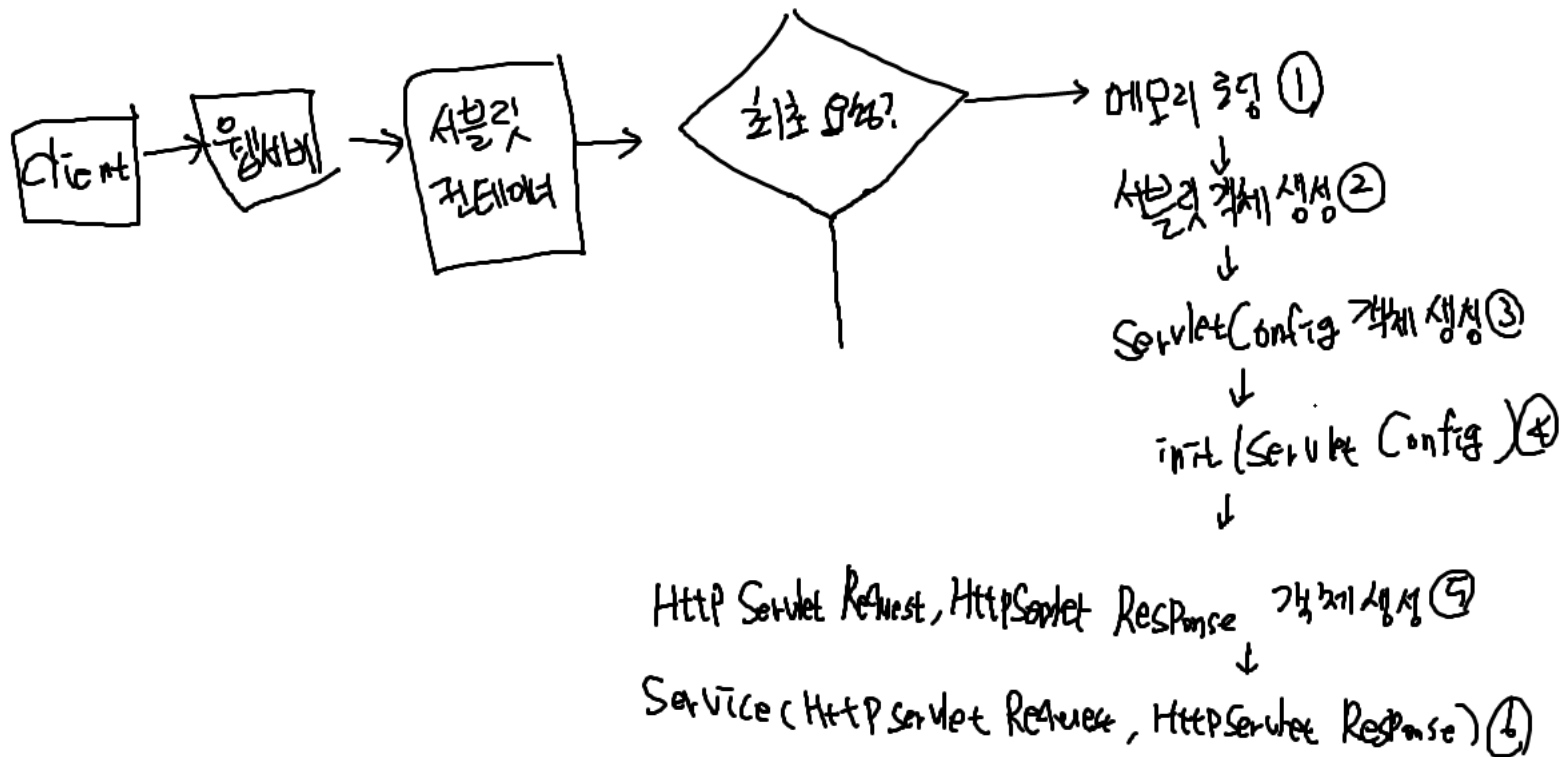
서블릿 설정과 변수

3)<load-on-startup>태그

- 웹 서비스가 시작될 때 서블릿 객체를 생성할 수 있습니다.
 - 서블릿 객체가 메모리에 생성되는 시점은 클라이언트로부터 최초의 요청이 있을 때입니다. 서버에 서블릿 클래스 파일이 존재하더라도 클라이언트로부터 실행 요청이 없으면 객체가 메모리에 생성되지 않습니다.
- <load-on-startup> 태그의 값으로 숫자를 지정하는데, 이 숫자는 객체가 생성되는 우선순위를 의미합니다. 서버가 시작될 때 생성해야 하는 서블릿 객체가 여러 개일 때 <load-on-startup> 태그의 값으로 우선순위를 지정합니다. 숫자 값이 낮을수록 우선순위가 높습니다.

서블릿 설정과 변수

Web.xml의 <servlet> 태그에 설정한 정보를 서블릿 페이지 내에서 추출할 때는 ServletConfig 객체에서 제공하는 메소드를 사용합니다. ServletConfig 객체는 서블릿이 실행될 때 자동으로 생성됩니다.



서블릿 설정과 변수

클라이언트로부터 서블릿 실행 요청이 들어오면 웹서버가 먼저 요청 받고 서블릿 컨테이너에 실행을 요청합니다. 서블릿 컨테이너는 현재 실행할 서블릿이 최초인지를 판단하고, 결과에 따라 서로 다르게 실행합니다.

최초로 실행되는 서블릿이면 ① 메모리에 로딩하고, ② 서블릿 객체를 생성하고, ③ **ServletConfig** 객체를 생성합니다. 그리고 나서 ④init() 메소드를 실행합니다. 이때 방금 생성한 **ServletConfig** 객체를 인자값으로 넘겨줍니다. Web.xml에 설정한 서블릿에 대한 정보를 추출하고자 할 때는 init() 메소드의 인자로 넘어오는 **ServletConfig**를 사용합니다.

⑤, ⑥번은 서블릿의 요청이 최초 요청이든지 두 번째 이후의 요청이든지 상관없이 매번 실행되는 부분입니다. **ServletConfig** 객체는 서블릿이 최초로 요청이 들어왔을 때 한번만 생성됩니다. 따라서 서블릿 페이지당 하나씩 존재하며 서블릿에 대한 정보를 처리하는 기능을 합니다.

서블릿 설정과 변수

다음은 init() 메소드를 재정의하여 ServletConfig 객체를 init() 메소드의 인자로 받아서 web.xml파일에 <servlet>태그로 설정한 서블릿 정보를 추출하여 확인하는 예제입니다.

```
public class InitParamServlet extends HttpServlet{  
    String id, password;  
  
    @Override  
    public void init(ServletConfig config)  
        throws ServletException {  
        id = config.getInitParameter("id");  
        password = config.getInitParameter("password");  
    }  
  
    @Override  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = res.getWriter();  
        out.print("<h2>서블릿 초기 추출 변수 </h2>" );  
        out.print("<h3>ID : "+id+"</h3>");  
        out.print("<h3>PASSWORD : "+password+"</h3>");  
        out.close();  
    }  
}
```

서블릿 초기 추출 변수

ID : guest

PASSWORD : 1004

서블릿 설정과 변수

```
String id, password;
```

서블릿 소스의 멤버변수 id와 password를 String 타입으로 선언하였습니다.

```
public void init(ServletConfig config) throws ServletException {
```

GenericServlet에서 상속받은 init() 메소드를 재정의합니다. init()메소드는 서블릿이 최초로 실행할 때 한번만 호출되는 메소드로서 인자 값으로 ServletConfig 객체가 전달됩니다.

```
id = config.getInitParameter("id");  
password = config.getInitParameter("password");
```

Config는 ServletConfig 객체의 변수이고, getInitParameter()는 ServletConfig가 가지고 있는 메소드로서 web.xml 파일의 <servlet> 속성 중에서 <init-param>으로 지정한 변수의 값을 추출할 때 사용합니다. getInitParameter() 메소드의 인자값으로 지정된 값과 같은 값이 있는 <param-name>을 찾아서 짝이 되는 <para-value>의 값을 반환합니다. 반환 데이터 타입은 String입니다.

서블릿 설정과 변수

```
out.print("<h3>ID : "+id+"</h3>");  
out.print("<h3>PASSWORD : "+password+"</h3>");
```

위 코드는 doGet() 메소드에 있는 실행문으로서 id와 password 변수의 값을 브라우저로 출력합니다. 서블릿이 처음 요청되었을 때 서블릿 객체가 생성되면서 멤버변수 id와 password가 만들어지고, init() 메소드가 호출되면서 각 멤버변수에 web.xml 파일의 <servlet>에 지정된 값을 저장합니다.

그리고 doGet() 메소드가 호출되면서 id와 password 변수의 값을 출력합니다.

서블릿 설정과 변수

GenericServlet 객체가 상속하고 있는 인터페이스를 보면 앞서 다루었던 ServletConfig가 있는 것을 확인할 수 있습니다. GenericServlet 객체는 ServletConfig 객체가 가지고 있는 메소드들을 모두 재정의하고 있습니다. 따라서 서블릿 페이지에서는 init() 메소드를 재정의하지 않고도 ServletConfig 객체의 메소드를 바로 사용할 수 있습니다.

Web.xml 파일을 수정하여 <servlet>의 <init-param> 태그를 활용하는 예제를 작성해보겠습니다.

새로운 서블릿 클래스를 만들고, 파일을 작성하겠습니다.

서블릿 설정과 변수

ServletConfigTestServlet.java

```
public class ServletConfigTestServlet extends HttpServlet {  
    public void doPost(HttpServletRequest req, HttpServletResponse resp)  
        throws ServletException, IOException {  
        resp.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = resp.getWriter();  
        String env =this.getInitParameter("charset");  
        req.setCharacterEncoding(env);  
        out.print("<h3> 이름 :"+req.getParameter("name"));  
        out.close();  
    }  
}
```

서블릿 설정과 변수

```
public void doPost(HttpServletRequest req, HttpServletResponse resp)  
    throws ServletException, IOException {
```

doPost() 메소드를 재정의하였습니다. 이 메소드는 클라이언트로부터 POST 방식으로 요청이 있을 때 HttpServlet의 service() 메소드에서 호출합니다.

```
resp.setContentType("text/html;charset=UTF-8");  
PrintWriter out = resp.getWriter();
```

서비스를 요청한 클라이언트로 데이터를 전송하기 위한 출력 스트림 out을 생성하였습니다. out을 통해 전달되는 데이터 타입은 html로, 문자코드는 UTF-8로 처리됩니다.

```
String env =this.getInitParameter("charset");
```

web.xml파일에서 <init-param>으로 지정한 변수 중 “charset”이란 이름으로 설정된 <param-name>을 찾아 그 <param-value>값을 추출하여 변수env에 저장합니다.

서블릿 설정과 변수

```
req.setCharacterEncoding(env);
```

req 변수는 HttpServletRequset 객체로서 요청정보를 처리해주는 객체입니다. 그리고 setCharacterEncoding()메소드는 메소드는 인자 값으로 UTF-8, EUC-KR과 같은 문자코드를 지정하며, 해당 문자코드를 이용해 요청정보 몸체의 데이터를 인코딩 해줍니다.

값을 직접 넣는 것과 web.xml에 값을 지정하고, 이 값을 추출하는 방법과 차이는 없습니다. 다만 유지보수 차원에서 외부에 값을 지정한 다음 추출해서 사용하는 방법이 훨씬 효율적이라 사용하는 방식입니다.

```
out.print("<h3> 이름 :"+req.getParameter("name"));
```

Rwq.getParameter("name")은 클라이언트가 전달한 질의 문자열 중에서 name을 찾아 값을 반환합니다. 이때 추출된 name 값은 앞 단계에서 req.setCharacterEncoding(charset)메소드에 의해 인코딩 처리가 완료된 값이 반환됩니다.


서블릿 설정과 변수

기존 name.html 파일에서 action을 변경하여
ServletConfigTestServlet.java 파일과 연결해봅시다.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" >
<title>이름 입력</title>
</head>
<body>
  <form action="ServletConfigTestServlet" method="post">
    이름 : <input type="text" name="name">
    <input type="submit" value="전송">
  </form>
</body>
</html>
```

작성 후 web.xml 도 수정해봅시다.

전송을 누르면

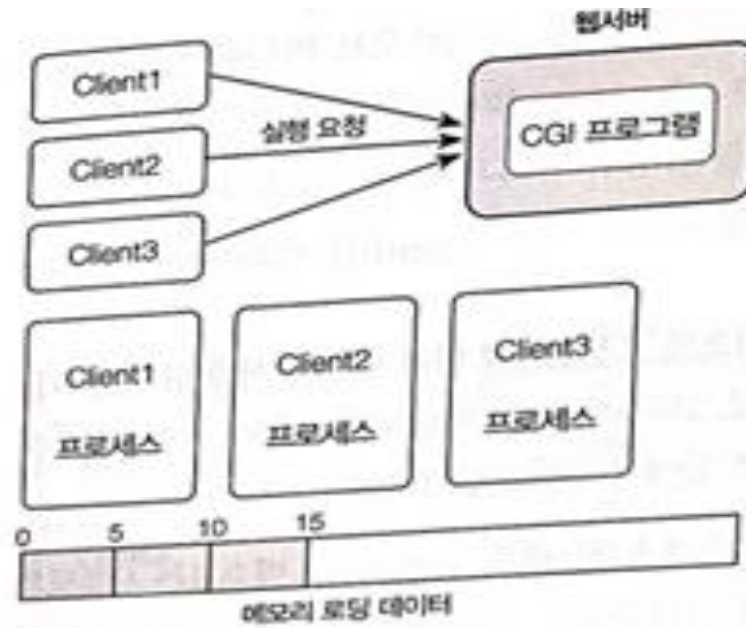


서블릿 변수

서블릿 동시 요청

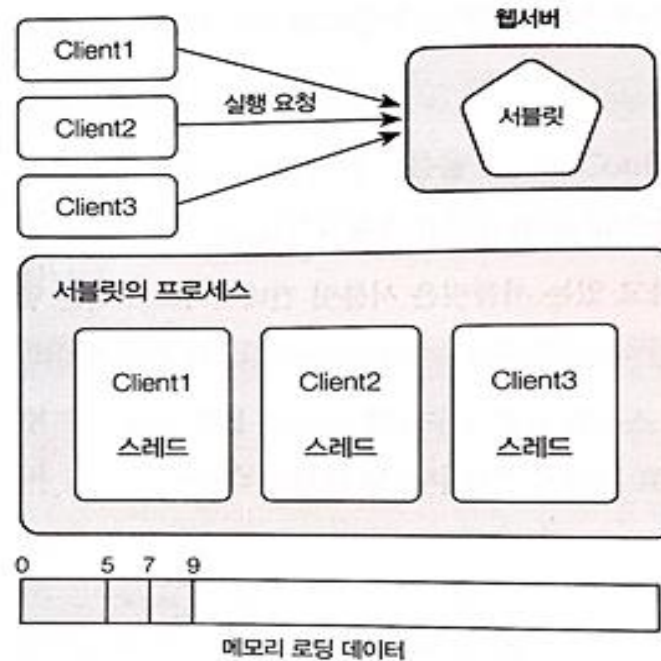
웹 프로그램을 개발하는 방식은 두 가지가 있습니다. 웹서버의 직접적인 호출로 실행하는 **CGI** 방식과 애플리케이션 서버가 실행하는 방식입니다. 지금 학습하고 있는 서블릿은 서블릿 컨테이너라고 하는 웹 애플리케이션 서버가 실행하는 방식입니다. **CGI**와 서블릿 실행 방식은 서로 다른데, 두 가지 방식의 차이점에 대해 알아보겠습니다.

서블릿 변수



CGI 프로그램은 클라이언트로부터 요청이 들어올 때마다 독립적인 프로세스가 만들어지며, 메모리에는 프로세스를 실행하기 위한 데이터가 로딩됩니다. 위의 그림은 하나의 CGI 프로그램에 3명의 클라이언트가 실행 요청할 때를 가정하여 처리 방식을 나타낸 것입니다. 서버에 3개의 요청이 있으므로 각 요청을 처리하기 위한 프로세스 3개가 만들어졌습니다. 그리고 하나의 프로세스를 띄우기 위해 필요한 데이터가 5라고 가정하면 3개의 프로세스이므로 15만큼의 데이터가 메모리에 로딩됩니다.

서블릿 변수



서블릿은 서블릿 컨테이너가 실행하며 해당 서블릿이 최초의 요청인지에 따라 실행 순서가 달라집니다. 최초의 요청이면 서블릿 객체의 메모리 로딩과 객체 생성, 그리고 `init()` 메소드가 호출되는데, 이러한 작업을 진행하기 전에 프로세스가 생성되며 이 프로세스 안에서 위와 같은 작업을 진행합니다. `init()` 메소드 실행이 끝난 다음에는 `service()` 메소드가 호출되는데 `service()` 메소드는 서블릿이 최초 요청 여부와 상관없이 서블릿 요청이 있을 때마다 실행됩니다. `Service()` 메소드는 최초 요청 시 만들어진 프로세스 안에 작은 프로세스를 만들고, 그 작은 프로세스 안에서 실행됩니다. 이와 같은 프로세스를 '스레드'라고 합니다.

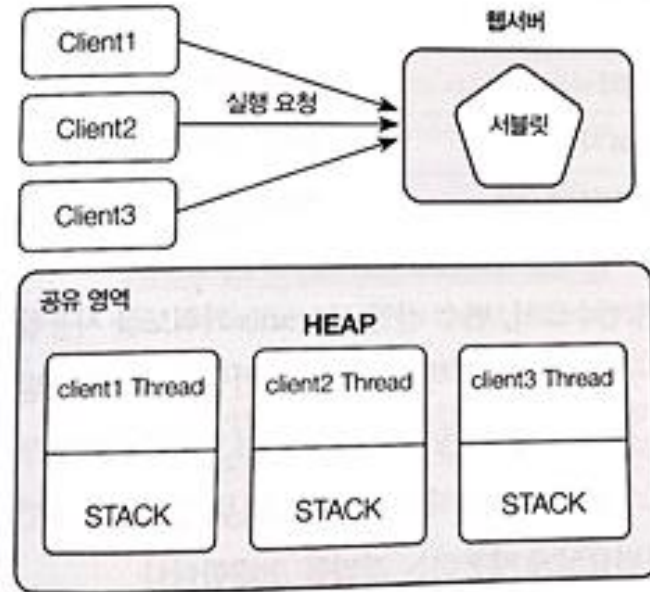
최초 요청일 때 프로세스를 만들고 그 안에 스레드를 만들어서 `service()` 메소드를 실행하며, 두 번째 이후의 요청부터는 이미 만들어진 프로세스 안에 `service()` 메소드를 실행하기 위한 스레드만 새로 생성한 다음, 이 스레드 안에서 `service()` 메소드를 실행합니다.

서블릿 변수특징

서블릿을 개발하면서 변수를 사용할 때 멤버변수인지 지역변수인지 구분하여 사용할 수 있어야 합니다. 서블릿은 웹서버에서 실행하는 프로그램으로서 여러 클라이언트가 하나의 서블릿을 동시에 실행할 수 있어야 하므로 여러 클라이언트가 공통으로 사용하는 데이터가 있을 수 있고, 또는 각각의 클라이언트가 독립적으로 사용해야 하는 데이터가 있을 수 있습니다. 이는 각각 멤버변수, 지역변수로 선언해야 할 것입니다.

서블릿은 하나의 프로세스를 생성한 다음, 동일한 서블릿을 요청하는 클라이언트에 대하여 공통적인 프로세스를 사용하며 `service()` 메소드를 실행하기 위한 스레드만 클라이언트 별로 독립적으로 생성하여 실행합니다.

서블릿 변수



멤버변수는 객체 생성 시 heap 메모리에 생성되며 서블릿을 실행하는 클라이언트들이 공통으로 사용합니다. 그러나 **service()** 메소드가 사용하는 지역변수는 stack 메모리에 생성되며, 클라이언트마다 독립적으로 사용하는데, 이는 서블릿의 **service()** 메소드를 실행하는 스레드마다 stack 메모리가 독립적이기 때문입니다. 따라서 지역변수는 클라이언트마다 별개로 사용됩니다.

서블릿의 지역변수

서블릿의 지역변수는 여러 클라이언트가 동시에 요청했을 때, 요청마다 개별적으로 할당됩니다. 그러므로 각 클라이언트의 요청 수만큼 메모리 영역을 개별적으로 할당하여 지역적으로 처리할 때만 지역변수를 선언하여 활용합니다.

서블릿의 지역변수

LocalTestServlet.java

```
@WebServlet("/local")
public class LocalTestServlet extends HttpServlet{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException{
        int number = 0;
        String str = req.getParameter("msg");
        res.setContentType("text/html;charset=UTF-8");
        PrintWriter out = res.getWriter();
        out.println("<html><head><title>MultiThread Test</title></head>");
        out.println("<body><h2>처리 결과(지역 변수)</h2>");
        while (number++ < 10) {
            out.print(str + " : " + number + "<br>");
            out.flush();
            System.out.println(str + " : " + number);
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
                System.out.println(e);
            }
        }
        out.println("<h2>Done " + str + " !!</h2>");
        out.println("</body></html>");
        out.close();
    }
}
```

서블릿의 지역변수

```
String str = req.getParameter("msg");
```

클라이언트가 보낸 질의 문자열 중에서 msg 의 값을 추출한 후 str 변수에 저장합니다. Get방식으로 전달받은 msg 값은 브라우저마다 URL을 통해 다른 값을 전달하기 때문에 지역변수입니다.

```
while (number++ < 10) {
```

while문으로 반복 실행합니다. number 변수의 값이 10 보다 작은지 판단하고 number 변수 값을 1만큼 증가시킵니다.

```
out.flush();
```

기본적으로 버퍼의 크기만큼 내용이 채워졌을 때 실제 전송이 이루어지는데, flush() 메소드는 버퍼가 채워져 있는지와 상관없이 전송한다.

```
System.out.println(str + " : " + number);
```

out 출력스트림으로 클라이언트 쪽으로 보낸 데이터를 서버 쪽에도 동일하게 출력합니다. 스레드의 실행 순서를 확인하기 위해 출력합니다.

```
Thread.sleep(1000);
```

클라이언트의 요청으로 스레드가 생성되고, 이 스레드가 service() 메소드를 실행합니다. 실행 중인 스레드가 thred.sleep() 메소드를 만나면 실행을 잠시 멈춥니다.

서블릿의 멤버변수

MemberTestServlet.java

```
@WebServlet("/member")
public class MemberTestServlet extends HttpServlet{
    String str;
    public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws ServletException,IOException{
        int number = 0;
        str = req.getParameter("msg");
        res.setContentType("text/html;charset=UTF-8");
        PrintWriter out = res.getWriter();
        out.println("<html><head><title>MultiThread Test</title></head>");
        out.println("<body><h2>처리 결과(멤버 변수)</h2>");
        while (number++ < 10) {
            out.print(str + " : " + number + "<br>");
            out.flush();
            System.out.println(str + " : " + number);
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
                System.out.println(e);
            }
        }
        out.println("<h2>Done " + str + " !!</h2>");
        out.println("</body></html>");
        out.close();
    }
}
```

서블릿의 멤버변수

```
String str;
```

메소드 밖의 멤버변수

동일한 서블릿을 여러 클라이언트가 동시에 요청했을 때 서블릿 객체는 하나만 생성되어 멀티 스레드로 동작하므로 서블릿의 멤버변수는 여러 클라이언트가 공유하게 됩니다. 그러므로 각 클라이언트들의 동시 요청수와 관계없이 하나의 메모리 공간을 할당하여 저녁적으로 처리할 때만 멤버변수를 선언하여 활용합니다.