자바 서블릿 프로그래밍

(1999 년 10 월 작성)

from Yongwoo's Park

목 차

자바 서블릿 프로그래밍

from Yongwoo's Park

1. 자바 서블릿(Servlet)

1.1일반 서블릿(GenericServlet)

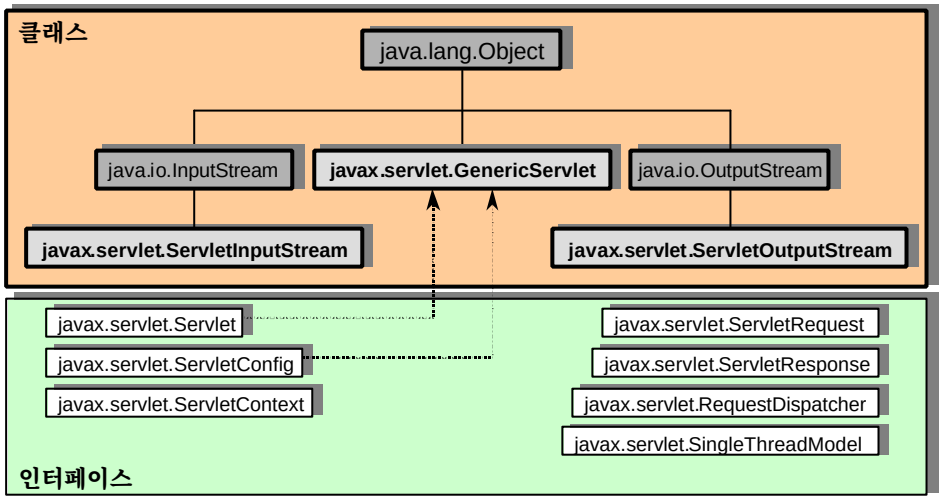


그림 1. javax.servlet 패키지의 클래스 계층도

1.1.1Servlet 인터페이스

자바에서 작성될 수 있는 모든 서블릿이 구현해야 될 메소드들을 정의하고 있습니다. 서블릿은 웹 서버에서 실행되는 자바 프로그램입니다. 서블릿은 일반적으로 HTTP(Hyper-Text Transfer Protocol)을 통해서 웹 클라이언트로부터 요청을 받고, 웹 서버는 해당 요청을 위한 작업을 하고, 그 결과를 웹 클라이언트에게 다시 되돌려 줍니다. Servlet 인터페이스를 구현하여 서블릿을 작성하기 위해서는, javax.servlet.GenericServlet 클래스를 상속하여 확장하는 하위클래스인 일반 서블릿(generic Servlet)을 작성하거나 또는 javax.servlet.http.HttpServlet 을 상속하여 확장하는 HTTP 서블릿을 작성하는 것입니다. Servlet 인터페이스에서 정의하고 있는 메소드는 다음과 같이 구분하고, 이러한 메소드들을 서블릿의 생명주기(life-cycle) 메소드라 하며, 호출 순서는 다음과 같습니다.

- 서블릿의 초기화: 서블릿이 생성될 때, init 메소드를 이용하여 서블릿의 초기화를

수행합니다.

- 서블릿의 동작: 클라이언트로부터 온 요청을 받아 처리하고 그 결과를 되돌려 줍니다.
- 서블릿의 종료: 서블릿의 수행을 마치고, destroy 메소드를 이용하여 마무리하며, 쓰레기 수거를 하고, finalize 합니다.

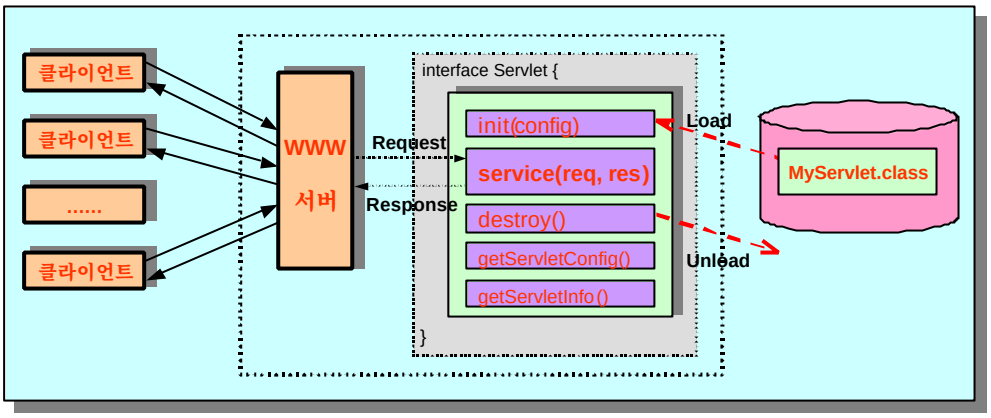


그림 2. 서블릿의 생명 주기

위와 같이, 서블릿의 생명 주기 메소드뿐만 아니라, 서블릿이 시작(startup) 정보를 얻을 수 있도록 해 주기 위한 getServletConfig 메소드와 서블릿에 대한 작성자(author), 버전(version), 저작권(copyright) 등과 같은 정보를 제공해 줄 수 있도록 하기 위해 getServletInfo 메소드를 정의하고 있습니다. 이러한 Servlet 인터페이스가 정의하고 있는 메소드를 살펴보면, 다음과 같습니다.

- void init(ServletConfig config): 서블릿을 시작할 때 호출하여, 필요한 자원을 할당하는 등의 서블릿을 초기화 하고, 서비스를 시작할 수 있도록 합니다.
- void service(ServletRequest req, ServletResponse res): 서블릿이 초기화 된 후 클라이언트로부터 온 요청에 대한 서비스를 수행합니다.
- void destroy(): 서블릿을 종료할 때 호출하여, 필요한 자원을 할당 해제 하는 등의 작업을 할 수 있도록 합니다.
- ServletConfig getServletConfig(): 서블릿에 대한 초기화/시작 매개변수 등을 포함하고 있는 ServletConfig 객체를 얻습니다.
- java.lang.String getServletInfo(): 서블릿의 작성자(author), 버전(version), 저작권(copyright) 등과 같은 서블릿 관련 정보를 제공해 주기 위해 재정의해 주어야 합니다.

1.1.2GenericServlet 추상 클래스

GenericServlet 클래스는 추상클래스로서, 일반적이고 프로토콜에 무관한 서블릿을 작성할 수 있는 기능을 제공해 주고 있습니다. 웹 사이트(Web site)를 가지고 사용하기 위한 HTTP 서블릿을 작성하고자 한다면, HttpServlet 클래스를 확장하는 하위클래스를 작성하면 됩니다. GenericServlet 클래스는 Servlet 인터페이스와 ServletConfig 인터페이스를 구현하고 있습니다. 서블릿을 작성할 때, 서블릿이 다른 상위클래스를 가지고 있지않다면 GenericServlet 클래스 또는 HttpServlet 클래스를 확장하는 하위클래스를 작성하면 됩니다. 만약, 서블릿이 GenericServlet 클래스 또는 HttpServlet 클래스가 아닌 다른 클래스를 상위클래스로서 상속해야 한다면, 서블릿은 Servlet 인터페이스를 구현해 주어야 합니다. GenericServlet 클래스는 init 메소드 및 destroy 메소드 등과 같은 생명주기(lifecycle) 메소드와 ServletConfig 인터페이스에서 정의하고 있는 메소드에 대한 간단한 구현을 제공해 주고 있으므로, 서블릿의 작성을 좀 더 쉽게 해 줍니다. 또한, GenericServlet 클래스는 ServletContext 인터페이스에 선언되어 있는 log 메소드를 구현하고 있습니다. 일반적인 서블릿을 작성하기 위해서는 단지 몸체를 가지지 않는 추상 메소드로 선언되어 있는 서비스 메소드를 재정의 해 주기만 하면 됩니다. 이러한 일반 서블릿의 구조를 살펴보면, 다음과 같습니다.

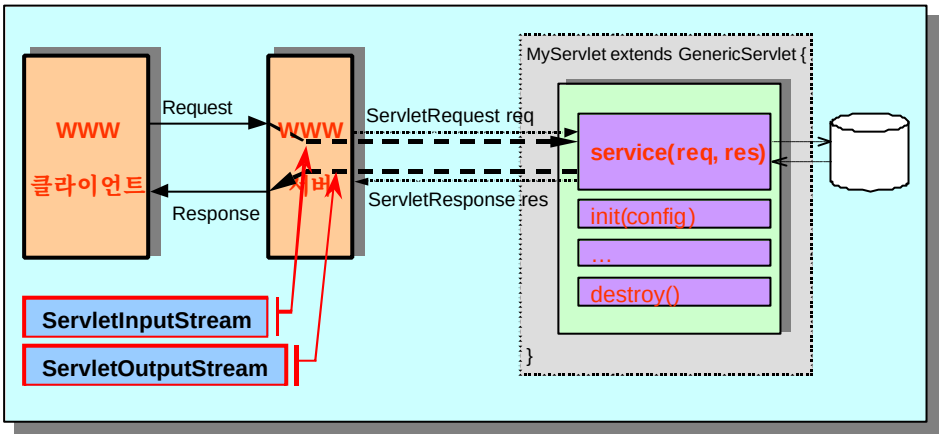


그림 3. 일반 서블릿의 구조

만약 서블릿 엔진을 작성한다면, getServletInfo 메소드를 재정의해야 하고, 만약 서블릿 엔진이 비용이 큰 서블릿-전반(sevlet-wide) 자원을 다루어야 한다면 init 메소드와 destroy 메소드를 구체화시켜 주어야 합니다. 이러한 GenericServlet 클래스가 정의하고 있는 메소드를 살펴보면, 다음과 같습니다.

- void init(): 서블릿의 초기화를 수행합니다.
- void init(ServletConfig config): 서블릿을 시작할 때 호출하여, 필요한 자원을 할당하는 등의 서블릿을 초기화 하고, 서비스를 시작할 수 있도록 합니다.
- abstract void service(ServletRequest req, ServletResponse res): 서블릿이 초기화 된 후 클라이언트로부터 온 요청에 대한 서비스를 수행합니다. 이 메소드는 추상 메소드이므로, 서블릿을 작성하기 위해 GenericServlet 클래스를 상속하는 모든 하위클래스에서 반드시

구현해 주어야 하는 메소드입니다.

- void destroy(): 서블릿을 종료할 때 호출하여, 필요한 자원을 할당 해제 하는 등의 작업을 할 수 있도록 합니다.
- java.lang.String getInitParameter(java.lang.String name): 주어진 이름의 초기화 매개변수의 값을 포함하고 있는 문자열을 얻습니다.
- java.util.Enumeration getInitParameterNames(): 서블릿을 위한 초기화 매개변수의 이름을 문자열 객체의 enumeration 으로 얻습니다.
- ServletConfig getServletConfig(): 서블릿에 대한 초기화/시작 매개변수 등을 포함하고 있는 ServletConfig 객체를 얻습니다.
- ServletContext getServletContext(): 서블릿이 실행되는 서블릿 엔진에 대한 정보를 포함하고 있는 ServletContext 객체를 얻습니다.
- java.lang.String getServletInfo(): 서블릿의 작성자(author), 버전(version), 저작권(copyright) 등과 같은 서블릿 관련 정보를 제공해 주기 위해 재정의해 주어야 합니다.
- void log(java.lang.String msg): 서블릿의 클래스 이름과 서블릿 예외 메시지를 서블릿의 로그 파일에 기록합니다.
- void log(java.lang.String message, java.lang.Throwable t): 시스템 예외 메시지를 서블릿 로그 파일에 기록합니다.

1.1.3ServletRequest 인터페이스

서블릿 엔진이 클라이언트로부터 온 요청에 대한 정보를 서블릿에게 전달하기 위해 사용하는 기능을 정의하고 있습니다. ServletRequest 객체는 다음과 같은 기본적인 데이터를 제공해 주고 있습니다.

- 매개변수의 이름과 값
- 매개변수의 속성
- 입력 스트림

또한, ServletRequest 인터페이스를 확장함으로써 프로토콜 관련 데이터와 같은 추가적인 정보도 제공해 줄 수 있습니다. 예를 들면, HttpServletRequest 에 의해 제공되는 HTTP 와 관련된 정보가 있습니다. 일반적으로 서블릿에 서비스를 요청하기 위해 사용하는 데이터는 MIME 형식입니다. 따라서, 서블릿이 이 MIME 형식의 데이터를 읽어들여야 하는데, 이러한 MIME 형식의 데이터는 텍스트 또는 바이너리 형태일 수 있습니다. 먼저, 문자 인코딩을 포함하는 텍스트일 경우에는 getReader 메소드를 이용하여 입력 스트림을 열고, 바이너리일 경우에는 getInpuetStream 메소드를 사용하여 입력 스트림을 열도록 하고 있습니다. 이 때, Multipart MIME 형식의 데이터는 바이너리

데이터로 취급됩니다. 이러한 ServletRequest 인터페이스가 정의하고 있는 메소드를 살펴보면, 다음과 같습니다.

- java.util.Enumeration getAttributeNames(): 이 요청이 갖는 속성들의 이름에 대한 Enumeration 객체를 얻습니다.
- java.lang.Object getAttribute(java.lang.String name) : 주어진 이름을 갖는 속성값을 얻습니다.
- void setAttribute(java.lang.String key, java.lang.Object o): 주어진 이름의 속성을 설정합니다.
- java.lang.String getCharacterEncoding(): 이 요청에 사용된 문자 인코딩을 얻습니다.
- int getContentLength(): 이 요청에 포함되어 있는 데이터의 길이를 구하며, 만약 길이를 알 수 없는 경우에는 –1 이 리턴됩니다.
- java.lang.String getContentType(): 요청에 포함되어 있는 내용에 대한 MIME 타입 또는 모를 경우에는 null 을 얻습니다.
- java.util.Enumeration getParameterNames(): 매개변수들의 이름에 대한 Enumeration 객체를 얻습니다.
- java.lang.String getParameter(java.lang.String name): 주어진 이름의 매개변수가 갖는 값을 얻습니다.
- java.lang.String[] getParameterValues(java.lang.String name): 주어진 이름으로 전달된 매개변수가 갖는 모든 값을 문자열 배열로 얻습니다. 매개변수가 다중 선택이 가능한 리스트(list) 또는 선택박스(choicebox)의 값이라면, 여러 개의 값이 하나의 이름으로 전달될 수 있습니다.
- java.io.BufferedReader getReader(): 요청 바디로부터 문자 인코딩에 따라 텍스트를 읽어들이기 위한 BufferedReader 객체를 얻습니다.
- ServletInputStream getInputStream(): 이 요청의 바디로부터 바이너리 데이터를 읽어들이기 위해, 한 번에 한 라인씩 읽을 수 있는 ServletInputStream 객체를 얻습니다.
- java.lang.String getProtocol(): “HTTP/1.1” 과 같은 형식으로 프로토콜 및 major/minor 버전을 얻습니다.
- java.lang.String getRemoteAddr(): 요청한 클라이언트의 IP(Internet Protocol) 주소를 얻습니다.
- java.lang.String getRemoteHost(): 요청한 클라이언트의 호스트 이름을 얻습니다.
- java.lang.String getScheme(): http, https, 또는 ftp 등과 같은 요청을 위해 사용된 방법의 이름을 얻습니다.
- java.lang.String getServerName(): 요청을 받은 서버의 이름을 얻습니다.
- int getServerPort(): 요청을 받은 포트 번호를 얻습니다.
- java.lang.String getRealPath(java.lang.String path): 자바 서블릿 API 2.1 에서부터는 ServletContext.getRealPath(java.lang.String) 메소드로 바뀌었습니다.

1.1.4ServletResponse 인터페이스

ServletResponse 인터페이스는 서블릿에서 처리된 결과를 클라이언트에게 되돌려 주어야 할 경우 사용할 수 있는 기능들을 정의하고 있습니다. 이 때, 서블릿에서 클라이언트로 보내지는 모든 데이터는 MIME 형식으로 인코딩되어 전송되어야 하며, ServletResponse 인터페이스는 이를 지원해 주기 위한 기능 역시 정의하고 있습니다. 서블릿 엔진은 서블릿의 결과를 얻어 클라이어트에게 다시 되돌려 주기 위해 서블릿을 호출할 때, ServletResponse 객체를 생성하여 service 메소드의 매개변수로서 ServletRequest 객체와 함께 전달해 줍니다. 이 때, 서블릿은 MIME 형식으로 바이너리 데이터를 보내고자 할 때는 ServletResponse 객체의 getOutputStream 메소드를 호출하여 ServletOutputStream 객체를 얻어 그 출력 스트림 객체에 출력하면 되고, 이와 달리 MIME 형식의 텍스트 데이터를 보내기 위해서는 getWriter 메소드를 이용하여 PrintWriter 객체를 얻은 후, 이 출력 스트림 객체에 출력하면 됩니다. 이러한 ServletResponse 인터페이스가 정의하고 있는 메소드를 살펴보면, 다음과 같습니다.

- java.lang.String getCharacterEncoding(): 클라이언트에 대한 응답에 해당하는 MIME 데이터를 보낼 때 사용하기 위해 현재 설정된 문자 인코딩을 얻습니다.
- ServletOutputStream getOutputStream(): 클라이언트에 대한 응답으로 바이너리 데이터를 보내기 위해 사용할 ServletOutputStream 객체를 얻습니다.
- java.io.PrintWriter getWriter(): 클라이언트에 대한 응답으로 텍스트 데이터를 보내기 위해 사용할 PrintWriter 객체를 얻습니다.
- void setContentLength(int len): 클라이언트에 대한 응답으로 보내지는 데이터의 길이를 설정합니다.
- void setContentType(java.lang.String type): 클라이언트에 대한 응답으로 보내지는 데이터의 형식을 설정합니다.

1.1.5ServletInputStream 클래스

ServletInputStream 클래스는 클라이언트로부터 온 데이터를 한번에 한 라인씩 읽을 수 있도록 해 주는 readLine 메소드를 제공해 줍니다. 다시 말해서, 클라이언트의 요청으로부터 바이너리 데이터를 읽기 위한 입력 스트림(input stream)을 제공합니다. HTTP POST 및 PUT 등과 같은 프로토콜에서는 클라이언트로부터 온 데이터를 읽기 위해 이 ServletInputStream 객체를 이용하기도 합니다. 일반적으로, ServletRequest.getInputStream 메소드를 사용하여 ServletInputStream 객체를 얻을 수 있습니다. 이러한 ServletInputStream 클래스는 서블릿 엔진이 구현해 주도록 하고 있습니다. 다시 말해서, 서블릿 엔진은 클라이언트로부터 요청을 받고 이 요청을 서블릿에게 전달하기 위해 service 메소드를 사용하게 되는데, 이 때 서블릿 엔진은 ServletInputStream 객체와 ServletOutputStream 객체를 생성하여 이 두 객체를 service 메소드를 호출할 때 매개변수로

넘겨주게 됩니다. 그리고, 서블릿 개발자가 ServletInputStream 클래스의 하위클래스를 직접 작성하려 한다면 java.io.InputStream.read 메소드를 반드시 구현해 주어야 합니다. 이러한 ServletInputStream 클래스에서 제공해 주는 메소드를 살펴보면, 다음과 같습니다.

- int readLine(byte[] b, int off, int len): 입력 스트림에서 한번에 한 라인씩 읽어들입니다. 만약, 결과값이 –1 이면 더 이상의 데이터가 존재하지 않는다는 것을 의미합니다.

1.1.6ServletOutputStream 클래스

ServletOutputStream 클래스는 서블릿이 클라이언트로부터 온 요청을 처리하고 그 결과를 되돌려 줄 때, 클라이언트에게 바이너리 데이터를 보낼 수 있도록 하기 위한 출력 스트림(output stream)을 제공해 줍니다. ServletInputStream 객체를 얻을 때와 비슷한 방식으로 ServletResponse.getOutputStream 메소드를 호출함으로써 ServletOutputStream 객체를 사용할 수 있습니다. ServletInputStream 클래스와 마찬가지로 ServletOutputStream 클래스 역시 추상클래스이므로, 서블릿 엔진이 이 클래스를 확장하고 각 추상 메소드를 구현해 주고 있습니다. 만약, 서블릿 개발자가 직접 ServletOutputStream 클래스의 하위클래스를 생성하고자 한다면, 반드시 java.io.OutputStream 클래스의 write(int) 메소드를 구현해 주어야 합니다. 이러한 ServletOutputStream 클래스에서 제공해 주는 메소드를 살펴보면, 다음과 같습니다.

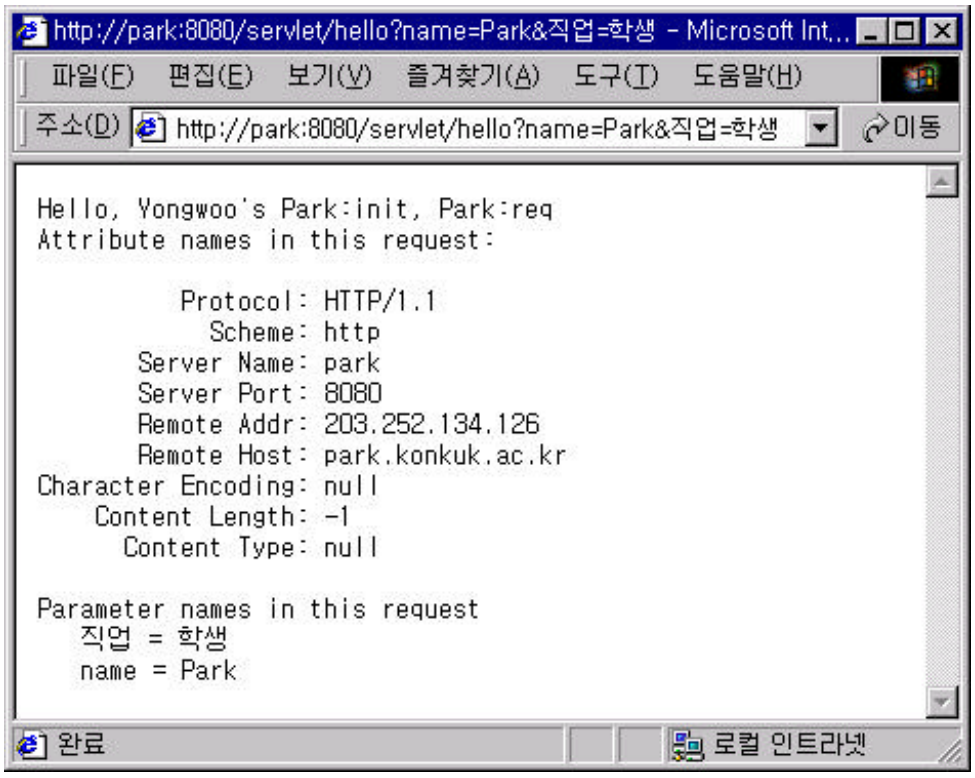
- void print(boolean b): b 를 출력 스트림에 출력합니다.
- void print(char c): c 를 출력 스트림에 출력합니다.
- void print(double d): d 를 출력 스트림에 출력합니다.
- void print(float f): f 를 출력 스트림에 출력합니다.
- void print(int i): i 를 출력 스트림에 출력합니다.
- void print(long l): l 를 출력 스트림에 출력합니다.
- void print(java.lang.String s): s 를 출력 스트림에 출력합니다.
- void println(): CRLF(carriage return-line feed) 문자를 출력 스트림에 출력합니다.
- void println(boolean b): b 와 CRLF 문자를 출력 스트림에 출력합니다.
- void println(char c): c 와 CRLF 문자를 출력 스트림에 출력합니다.
- void println(double d): d 와 CRLF 문자를 출력 스트림에 출력합니다.
- void println(float f): f 와 CRLF 문자를 출력 스트림에 출력합니다.
- void println(int i): i 와 CRLF 문자를 출력 스트림에 출력합니다.
- void println(long l): l 와 CRLF 문자를 출력 스트림에 출력합니다.
- void println(java.lang.String s): s 와 CRLF 문자를 출력 스트림에 출력합니다.

1.1.7일반 서블릿(GenericServlet) 기본 예제

지금까지 살펴본 것과 같이, 자바에서 서블릿을 작성하기 위해서는 Setvlet 인터페이스를 구현하고 있는 GenericServlet 클래스를 상속하는 하위클래스를 작성해 주기만 하면됩니다. 이 때, 서블릿의 생명주기에 해당하는 init, service, destroy 메소드를 각각 호출되는 특징에 따라 구현해 주면 되고, 필요에 따라 getServletInfo 메소드와 getServletConfig 메소드를 사용할 수 있습니다. 이러한 전반적인 과정은 자바 애플릿을 작성하기 위해 java.applet.Applet 클래스를 상속하는 하위클래스를 작성해 주기만 했던 것과 같습니다. 또한, 필요에 따라 애플릿의 생명주기에 해당하는 init, start, stop, destroy 메소드를 적당하게 구현해 주었고, 또한 필요에 따라 paint, getAppletInfo 등과 같은 메소드를 구현해 주었습니다. ***마지막으로, 새롭게 작성한 자바 서블릿을 테스트 하기 위해서는 항상 자바 서버를 새롭게 시작해 주어야 한다는 것을 명심하시기 바랍니다.*** 다음은 일반 서블릿(GenericServlet)을 작성하기 위한 간단한 예를 보여주는 자바 서블릿 예제 프로그램입니다.

```
import java.io.*;
import java.util.Enumeration;
import javax.servlet.*;

public class HelloGenericServlet extends GenericServlet {
   static String initialName;

   public void init(ServletConfig config) throws ServletException {
        super.init(config);
        initialName=config.getInitParameter("name");
    }

   public void service(ServletRequest req, ServletResponse res)
    throws ServletException, IOException {
        PrintWriter out=res.getWriter();
        String name=req.getParameter("name");

        out.println("Hello, "+initialName+":init, "+name+":req");

       out.println("Attribute names in this request:");
       Enumeration e = req.getAttributeNames();
        while(e.hasMoreElements()) {
            String key =(String)e.nextElement();
           Object value = req.getAttribute(key);

          out.println("    " + key + " = " + value);
         }
         out.println();

         out.println("              Protocol: " + req.getProtocol());
         out.println("                Scheme: " + req.getScheme());
        out.println("           Server Name: " + req.getServerName());
        out.println("           Server Port: " + req.getServerPort());
        out.println("           Remote Addr: " + req.getRemoteAddr());
        out.println("           Remote Host: " + req.getRemoteHost());
       out.println("Character Encoding: " + req.getCharacterEncoding());
        out.println("      Content Length: " + req.getContentLength());
         out.println("         Content Type: " + req.getContentType());
```

```
        out.println();

       out.println("Parameter names in this request");
        e = req.getParameterNames();
        while(e.hasMoreElements()) {
            String key =(String)e.nextElement();
            String[] values = req.getParameterValues(key);

           out.print("    " + key + " = ");
            for(int i=0;i<values.length;i++) {
               out.print(values[i] + " ");
             }
             out.println();
         }
    }
}

/*
 * File: d:\java\jsdk2.1\webpages\Web-inf\servlets.properties
 ...

 hello.code=HelloGenericServlet
 hello.initparams=name=Yongwoo's Park
```



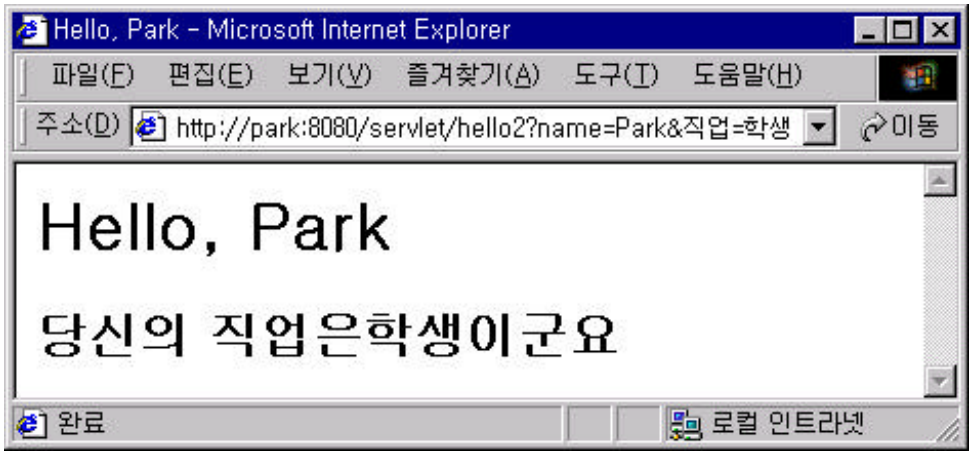

```
*/
```

<프로그램 1. HelloGenericServlet.java>

위의 서블릿은 단순한 텍스만을 서블릿의 실행결과로서 되돌려 주고 있습니다. 그러나, 단순한 텍스트만을 결과로 되돌려준다고 하면, 클라이언트가 보면 결과 페이지는 너무나 초라해 보일 것입니다. 다시 말해서, 서블릿의 실행 결과를 앞에서와 같이 단순한 텍스트가 아닌 기존의 HTML

페이지와 같은 HTML 코드를 이용하여 되돌려 주도록 하면 훨씬 더 보기좋은 결과 페이지를 클라이언트는 볼 수 있다는 것입니다. 다음은 일반 서블릿(GenericServlet)에서 텍스트 페이지가 아닌 HTML 페이지를 클라이언트에서 결과 페이지로서 되돌려 주기 위한 간단한 예를 보여주는 자바 서블릿 예제 프로그램입니다.

```
import java.io.*;
import java.util.Enumeration;
import javax.servlet.*;

public class HelloGenericServlet2 extends GenericServlet {
   public void service(ServletRequest req, ServletResponse res)
   throws ServletException, IOException {
        res.setContentType("text/html; charset=euc-kr");

        PrintWriter out=res.getWriter();
        String name=req.getParameter("name");

         out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello, "+name+"</TITLE></HEAD>");
         out.println("<BODY>");
        out.println("<H1>Hello, "+name+"</H1>");
         out.println("<H2>당신의 직업은"+req.getParameter("직업")+"이군요</H2>");
         out.println("</BODY>");
         out.println("</HTML>");
    }
}

/*
 * File: d:\java\jsdk2.1\webpages\Web-inf\servlets.properties
 ...

 hello2.code=HelloGenericServlet2
 hello2.initparams=
```




```
 */
```

<프로그램 2. HelloGenericServlet2.java>

위의 자바 서블릿 예제에서 첫 번째 서블릿과 한 가지 다른점은 ServletResponse 클래스에서 제공해 주는 setContentType 메소드를 사용한다는 것입니다. 이 메소드는 서블릿의 결과 데이터에 대한 형식을 지정하기 위한 것입니다. 따라서, 예제에서와 같이 함으로써 결과 데이터(페이지)가 MIME 형식 중 text/html 형식이라고 지정할 수 있습니다. 다시 말해서, 서블리스이 결과 페이지가 HTML 페이지라는 것을 지정할 수 있다는 것입니다. 이렇게 하면 다음과 같이 설정됩니다.

```
Content Type: text/html; charset=euc-kr
```

이 때, MIME 형식의 출력 데이터에 대한 문자 인코딩 방식을 지정할 수 있고, 위와 같이 함으로써 일반적인 한글 인코딩 방식을 지정할 수 있습니다. 디폴트 한글 인코딩 방식은 ISO-8859-1 입니다.

1.1.8ServletConfig 인터페이스

서블릿을 초기화하기 위해 서블릿 엔진이 서블릿에게 정보를 전달해야 하는데, 이 때 사용하기 위한 서블릿 구성 객체를 정의하고 있습니다. 이러한 구성 정보는 name/value 쌍으로 구성된 초기화 매개변수, 서버와 관련된 서블릿 정보를 제공해 주는 ServletContext 객체 등을 포함하고 있습니다. 이러한 ServletConfig 인터페이스가 정의하고 있는 메소드를 살펴보면, 다음과 같습니다.

- java.lang.String getInitParameter(java.lang.String name): name/value 쌍으로 주어진 초기화 매개변수 중, 주어진 이름으로 전달된 값을 얻습니다.
- java.util.Enumeration getInitParameterNames(): 서블릿의 초기화 매개변수들의 이름에 대한 Enumeration 객체를 얻습니다.
- ServletContext getServletContext(): 자바 서버가 서블릿에게 전달해 준 ServletContext 객체를 얻습니다.

1.1.9ServletContext 인터페이스

서블릿이 서블릿 엔진과 통신하기 위해 사용할 수 있는 메소드를 정의하고 있습니다. 이러한 메소드를 통하여 할 수 있는 작업을 예로 들면 다음과 같습니다.

- 파일의 MIME 타입을 얻습니다.
- 다른 서블릿을 실행시킵니다.
- 서블릿에 대한 로그 파일을 작성합니다.

또한, 서블릿 엔진은 서블릿의 실행 환경에 대한 정보를 제공해 주는 ServletContext 객체를 리턴해줌으로써 서블릿과 통신합니다. 이 때, 서블릿은 ServletConfig.getServletContext() 메소드를 이용하여 ServletContext 객체를 얻을 수 있습니다. 만약, 서버가 여러 개의(multiple) 또는 가상(virtual) 호스트를 제공한다면, ServletContext 객체는 유일해야 합니다. ServletContext 객체는 서블릿이 초기화 될 때 웹 서버가 제공하는 ServletConfig 객체에 포함되어 있습니다. 따라서, Servlet.getServletConfig 메소드를 이용하여 이 ServletConfig 객체를 참조할 수 있습니다. 이러한 ServletContext 인터페이스가 정의하고 있는 메소드를 살펴보면, 다음과 같습니다.

- java.lang.Object getAttribute(java.lang.String name): 주어진 이름의 서블릿 엔진 속성을 얻습니다.
- java.util.Enumeration getAttributeNames(): 서블릿 엔지의 속성에 대해 이름들을 Enumeration 객체로 얻습니다.
- void removeAttribute(java.lang.String name): 서블릿 컨텍스트(contect)로부터 주어진 이름의 속성을 제거합니다.
- void setAttribute(java.lang.String name, java.lang.Object object): 서블릿 컨텍스트(contect)에 주어진 이름의 속성을 추가합니다.
- ServletContext getContext(java.lang.String uripath): 주어진 서버상의 URL 에 해당하는 ServletContext 객체를 얻습니다.
- int getMajorVersion(): 웹 서버가 지원하는 자바 서블릿 API 의 major version 을 얻습니다.
- int getMinorVersion():웹 서버가 지원하는 자바 서블릿 API 의 minor version 을 얻습니다.
- java.lang.String getMimeType(java.lang.String file): 문자열로 주어진 파일에 대한 MIME 타입을 얻습니다.
- java.lang.String getRealPath(java.lang.String path): 주어진 가상 디렉토리의 실제 경로를 얻습니다.
- RequestDispatcher getRequestDispatcher(java.lang.String urlpath): 주어진 경로에 위치에 있는 자원을 위한 wrapper 로 작동하는 RequestDispatcher 객체를 얻습니다.
- java.net.URL getResource(java.lang.String path): 주어진 경로에 매핑되는 자원을 얻습니다.
- java.io.InputStream getResourceAsStream(java.lang.String path): 주어진 경로에 매핑되는 자원으로부터 입력받을 수 있는 입력 스트림을 얻습니다.
- java.util.Enumeration getServlets(): 자바 서블릿 API 2.0 에서부터는 제거되었습니다.
- void log(java.lang.Exception exception, java.lang.String msg) : 자바 서블릿 API 2.1 에서부터는 log(String message, Throwable throwable) 메소드로 바뀌었습니다.
- void log(java.lang.String msg): 서블릿 로그 파일에 주어진 메시지를 기록합니다. 일반적으로 이벤트 로그입니다.
- void log(java.lang.String message, java.lang.Throwable throwable): 스택의 내용과 주어진

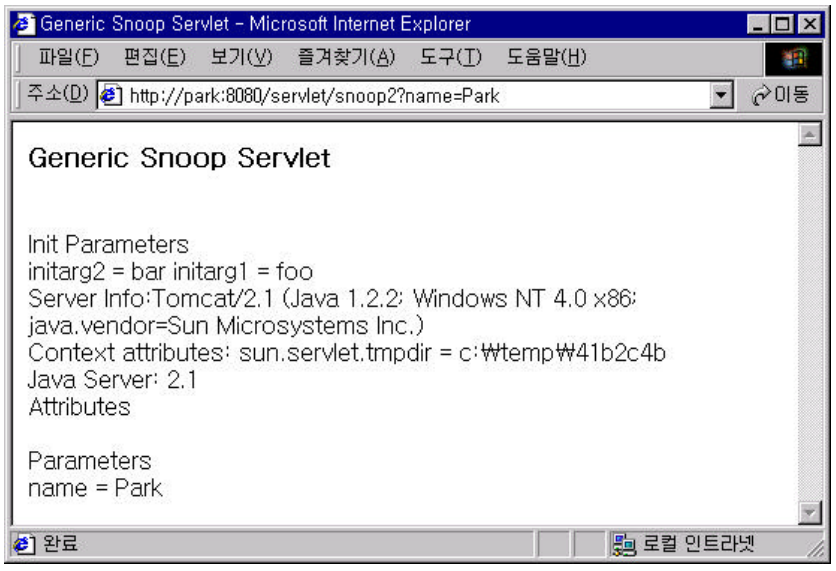
예외가 가지는 확장 메시지를 서블릿 로그 파일에 기록합니다.

- java.lang.String getServerInfo(): 서블릿이 실행되고 있는 서블릿 엔진의 이름과 버전을 얻습니다.
- Servlet getServlet(java.lang.String name): 자바 서블릿 API 2.1 에서부터는 제거되었습니다.
- java.util.Enumeration getServletNames(): 자바 서블릿 API 2.1 에서부터는 제거되었습니다.

다음은 위의 두 개의 클래스를 서블릿(GenericServlet)에서 사용하는 간단한 예를 보여주는 자바 서블릿 예제 프로그램입니다.

```
/* $Id: SnoopServlet.java,v 1.1 1999/03/17 01:17:20 duncan Exp $
 *
 */

import java.io.IOException;
import java.util.Enumeration;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * @author James Duncan Davidson <duncan@eng.sun.com>
 */

public class SnoopGenericServlet extends GenericServlet {
   public void service(ServletRequest req, ServletResponse res)
   throws ServletException, IOException {
               ServletConfig config = getServletConfig();
       ServletOutputStream out = res.getOutputStream();

        res.setContentType("text/html; charset=8859-1");

        out.println("<html>");
       out.println("<head><title>Generic Snoop Servlet</title></head>");

        out.println("<body>");

       out.println("<H3>Generic Snoop Servlet</H3><br>");

                out.println("Init Parameters<br>");
               Enumeration e = config.getInitParameterNames();
                while(e.hasMoreElements()) {
                    String key =(String)e.nextElement();
                   String value = config.getInitParameter(key);
                     out.println("\t" + key + " = " + value + "\t");
                }
                out.println("<br>");
       out.println("Server Info:"+getServletConfig().getServletContext().getServerInfo());
        out.println("<br>");

                out.println("Context attributes:");
               ServletContext context = config.getServletContext();
               e = context.getAttributeNames();
               while(e.hasMoreElements()) {
                    String key =(String)e.nextElement();
          Object value = context.getAttribute(key);
```

```
            out.println("\t" + key + " = " + value + "<br>");
                }
              out.println("Java Server: "+context.getMajorVersion()+"."+context.getMinorVersion());
               out.println("<br>");

        out.println("Attributes<br>");
       e = req.getAttributeNames();
        while(e.hasMoreElements()) {
           String key =(String)e.nextElement();
          Object value = req.getAttribute(key);

            out.println("\t" + key + " = " + value + "<br>");
        }
                out.println("<br>");

        out.println("Parameters<br>");
       e = req.getParameterNames();
        while(e.hasMoreElements()) {
           String key =(String)e.nextElement();
          String[] values = req.getParameterValues(key);

            out.print("\t" + key + " = ");
           for(int i=0;i<values.length;i++) {
                out.print(values[i] + " ");
            }
                out.println("<br>");
        }
        out.println("</body>");
        out.println("</html>");
    }
}

/*
 *

 # $Id: servlets.properties,v 1.2 1999/04/02 02:04:22 duncan Exp $
 ...
 snoop2.code=SnoopGenericServlet
 snoop2.initparams=initarg1=foo,initarg2=bar
```




```
 */
```

<프로그램 3. SnoopGenericServlet.java>

1.2HTTP 서블릿(HttpServlet)

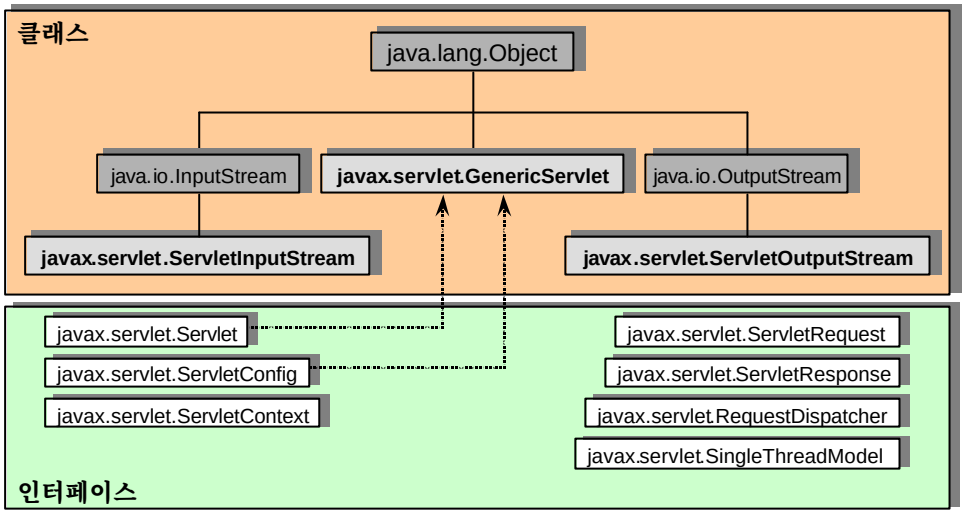


그림 4. javax.servlet.http 패키지의 클래스 계층도

1.2.1HttpServlet 클래스

HttpServlet 클래스는 HTTP 서블릿을 생성할 수 있도록 기능을 제공해 주는 추상클래스입니다. HTTP 서블릿은 일반적으로 웹 클라이언트로부터 온 요청을 받아서 처리하고, 그 결과를 다시 웹 클라이언트에게 되돌려 주는 작업을 수행합니다. 따라서, 이와 같은 작업을 하기 위해 HttpServlet 클래스를 상속하는 하위클래스를 작성할 때는 다음의 메소드 중 하나를 반드시 재정해 주어야 합니다.

- doGet 메소드: 서블릿이 HTTP GET 요청을 처리하기 위해서 재정의합니다.
- doPost 메소드: 서블릿이 HTTP POST 요청을 처리하기 위해서 재정의합니다.
- doPut 메소드: 서블릿이 HTTP PUT 요청을 처리하기 위해서 재정의합니다.
- doDelete 메소드: 서블릿이 HTTP DELETE 요청을 처리하기 위해서 재정의합니다.
- init 와 destroy 메소드: 서블릿이 자원을 할당하여 사용할 경우, 자원의 할당을 위해 init 메소드를 정의하고, 이 때 반드시 destroy 메소드를 재정의 함으로써 할당한 자원을 반납하도록 해야 합니다.
- getServletInfo: 서블릿 자신에 대한 정보를 제공하기 위해 재정의합니다.

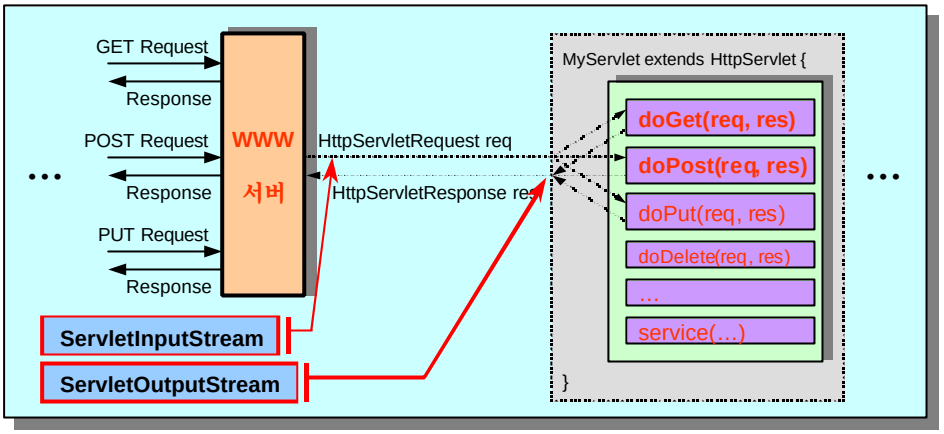


그림 5. HTTP 서블릿의 구조

1.2.2 HttpServletRequest 인터페이스

ServletRequest 인터페이스를 확장하는 HttpServletRequest 인터페이스는 HTTP 방식을 사용하는 HTTP 서블릿에 정보를 전달하기 위한 추가적인 기능을 제공해 주고 있습니다. 서블릿 엔진은 클라이언트로부터 HTTP 프로토콜을 사용하여 전달된 정보를 HttpServlet 의 service 메소드에 전달하기 위해 HttpServletRequest 객체를 생성합니다. HttpServletRequest 인터페이스는 이렇게 HTTP 서블릿에게 클라이언트로부터 온 정보를 전달하기 위해 필요한 기능들을 제공해 주고 있고, 이 기능들을 살펴보면 다음과 같습니다.

- java.lang.String getAuthType(): 서버에서 사용하는 인증 방법(authentication scheme)의 이름을 얻습니다. 예를 들면, "BASIC" 또는 "SSL" 등이 있습니다.
- Cookie[] getCookies(): 브라우저에서 이 요청을 통해 보낸 모든 쿠키 객체에 대한 배열을 얻습니다.
- long getDateHeader(java.lang.String name): 주어진 이름의 요청 헤더의 값을 Date 객체를 표현하는 long 형 값으로 얻습니다.
- java.lang.String getHeader(java.lang.String name): 주어진 이름의 요청 헤더의 값을 문자열로 얻습니다.
- java.util.Enumeration getHeaderNames(): 요청 헤더의 모든 이름에 대한 Enumeration 객체를 얻습니다.
- int getIntHeader(java.lang.String name): 주어진 이름의 요청 헤더의 값을 정수값으로 얻습니다.
- java.lang.String getMethod(): GET, POST, PUT, DELETE 등과 같은 요청 메소드(request method)를 얻습니다.

- java.lang.String getPathInfo(): 클라이언트가 보낸 URL 과 연관된 추가 경로 정보를 얻습니다.
- java.lang.String getPathTranslated(): “servlet” 이후부터 쿼리 문자열(query string) 이전 까지의 추가 경로 정보를 얻어, 실제 경로로 변환한 경로명을 얻습니다.
- java.lang.String getQueryString(): 이 요청에 포함되어 있는 쿼리 문자열을 얻습니다.
- java.lang.String getRemoteUser(): 만약, 사용자가 HTTP 인증(authentication)을 통해 로그인 했을 경우, 이 요청을 만든 사용자의 이름을 얻습니다.
- java.lang.String getRequestedSessionId(): 클라이언트에 의해 주어진 세션(session) ID 를 얻습니다.
- java.lang.String getRequestURI(): HTTP 요청의 첫줄에 있는 URL 중 프로토콜부터 쿼리 문자열까지의 부분을 얻습니다.
- java.lang.String getServletPath(): URL 중 서블릿 호출을 위한 부분을 얻습니다.
- HttpSession getSession(): 이 요청과 관련된 현재 세션을 얻습니다. 만약 세션이 없을 경우, 새롭게 생성합니다.
- HttpSession getSession(boolean create): 이 요청과 관련된 현재 세션을 얻습니다. 만약 세션이 없을 경우, create 값에 따라 생성 여부를 결정합니다.
- boolean isRequestedSessionIdFromCookie(): 주어진 세션 ID 가 쿠키에 대한 것인지를 얻습니다.
- boolean isRequestedSessionIdFromUrl(): 자바 서블릿 API 2.1 에서부터는 isRequestedSessionIdFromURL() 메소드로 바뀌었습니다.
- boolean isRequestedSessionIdFromURL(): 주어진 세션 ID 가 URL 의 부분에 대한 것인지를 얻습니다.
- boolean isRequestedSessionIdValid(): 요청이 HttpSessionContext 의 객체인 현재 세션 컨텍스트 내에 유효한 세션을 가지고 있는지를 얻습니다.

1.2.3 HttpServletResponse 인터페이스

웹 서버에서 실행되고 있는 HTTP 서블릿이 클라이언트의 요청을 받아 처리하고 그 결과를 HTTP 를 이용하여 클라이언트에게 되돌려 주기 위해 HttpServletResponse 객체를 이용합니다. HttpServletResponse 인터페이스는 서블릿의 service 메소드가 HTTP 헤더를 적당하게 설정할 수 있도록 해 주고, 클라이언트에게 데이터를 되돌려 줄 수 있도록 해 줍니다. 따라서, HTTP 서블릿과 클라이언트 사이를 중재하는 서블릿 엔진은 HttpServletResponse 인터페이스를 구현해 주어야 하며, HttpServletResponse 인터페이스가 정의하고 있는 기능을 살펴보면, 다음과 같습니다.

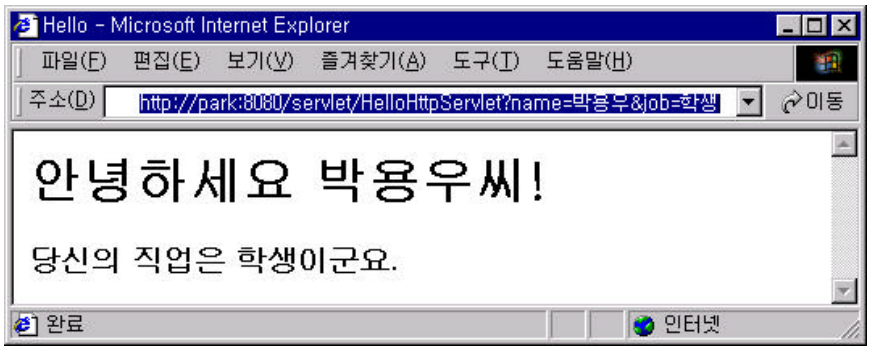
- void addCookie(Cookie cookie): 주어진 쿠키를 응답에 추가합니다.

- boolean containsHeader(java.lang.String name): 응답 메시지 헤더에 주어진 이름의 항목이 있는지를 얻습니다.
- java.lang.String encodeRedirectUrl(java.lang.String url): encodeRedirectURL(String url) 메소드로 바뀌었습니다.
- java.lang.String encodeRedirectURL(java.lang.String url): 주어진 URL 을 sendRedirect 메소드 내에서 사용하기 위해 인코딩합니다.
- java.lang.String encodeUrl(java.lang.String url) : encodeURL(String url) 메소드로 바뀌었습니다.
- java.lang.String encodeURL(java.lang.String url): 주어진 URL 에 세션 ID 를 포함하여 인코딩합니다.
- void sendError(int sc): 주어진 상태코드와 그 코드에 해당하는 디폴트 메시지를 사용하여 클라이언트에게 에러를 응답합니다.
- void sendError(int sc, java.lang.String msg): 주어진 상태코드와 메시지를 사용하여 클라이언트에게 에러를 응답합니다.
- void sendRedirect(java.lang.String location): 응답을 주어진 URL 로 리다이렉트 하도록 합니다.
- void setDateHeader(java.lang.String name, long date): 주어진 이름과 날자를 갖도록 항목을 추가합니다.
- void setHeader(java.lang.String name, java.lang.String value) : 주어진 이름과 값을 갖도록 항목을 추가합니다.
- void setIntHeader(java.lang.String name, int value): 주어진 이름과 정수값을 갖도록 항목을 추가합니다.
- void setStatus(int sc): 이 응답에 대한 상태코드를 설정합니다.
- void setStatus(int sc, java.lang.String sm): 이 응답에 대한 주어진 상태코드와 메시지를 설정합니다.

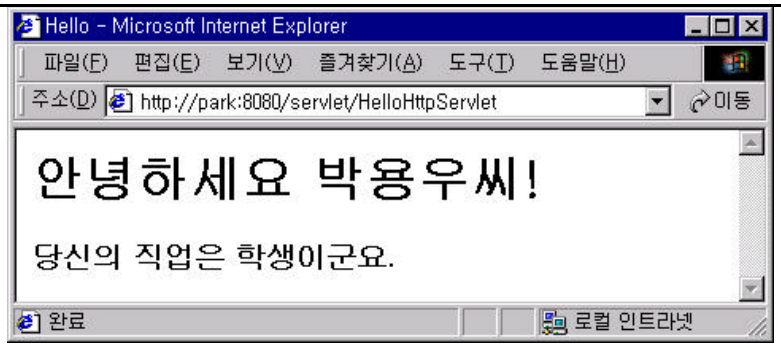
다음은 HTTP 서블릿(GenericServlet)을 사용하여 클라이언트로부터 GET 또는 POST 방식으로 온 요청에 대해, 간단한 결과 페이지를 클라이언트에게 되돌려주기 위한 예를 보여주는 자바 서블릿 프로그램입니다.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloHttpServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws IOException, ServletException {
       res.setContentType("text/html; charset=euc-kr");

       PrintWriter out = res.getWriter();
```

```
        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<HEAD>");
        out.println("<TITLE>Hello</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<H1>안녕하세요 "+req.getParameter("name")+"씨!</H1>");
        out.println("<H3>당신의 직업은 "+req.getParameter("job")+"이군요.</H3>");
        out.println("</BODY>");
        out.println("</HTML>");
    }

   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws IOException, ServletException {
       res.setContentType("text/html; charset=euc-kr");

       PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<HEAD>");
        out.println("<TITLE>Hello</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<H1>안녕하세요 "+toKSC5601(req.getParameter("name"))+"씨!</H1>");
        out.println("<H3>당신의 직업은 "+toKSC5601(req.getParameter("job"))+"이군요.</H3>");
        out.println("</BODY>");
        out.println("</HTML>");
    }

   public static String toKSC5601(String str)
       throws UnsupportedEncodingException {
       if(str == null) {
           return(null);
        }
       return(new String(str.getBytes("8859_1"), "KSC5601"));
    }
}

/*
 *
 1) GET 방식으로 전달
```



1) POST 방식으로 전달

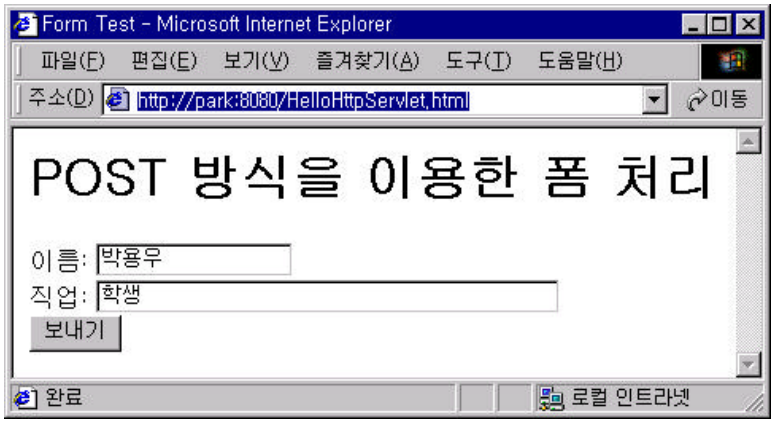


*/

<프로그램 4. HelloHttpServlet.java>

다음은 간단한 폼을 보여주고, 사용자가 입력을 마친 후 위의 HTTP 서블릿(GenericServlet)을 호출하여 POST 방식으로 데이터를 전달해 주는 간단한 HTML 페이지 입니다.

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=euc-kr">
<TITLE>Form Test</TITLE>
</HEAD>

<BODY>
<H1>POST 방식을 이용한 폼 처리</H1>
<P>
<FORM action="http://park:8080/servlet/HelloHttpServlet" method="POST">
이름: <input type="TEXT" name="name" size=16><BR>
직업: <input type="TEXT" name="job" size=40><BR>
<input type="SUBMIT" value="보내기">
</FORM>
</BODY>
</HTML>

/*
 *
```



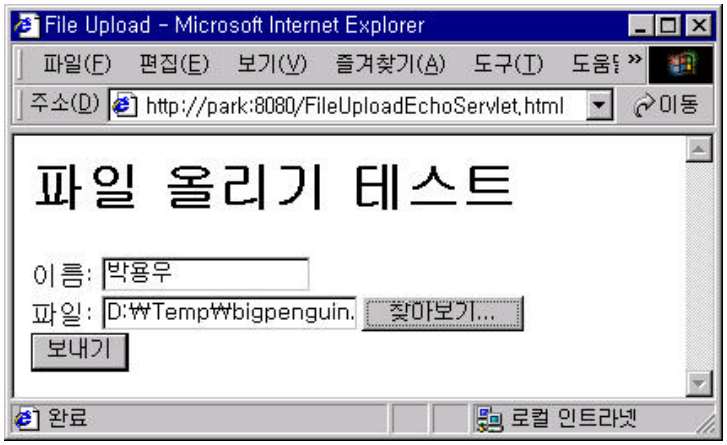
*/

<프로그램 5. HelloHttpServlet.html>

1.2.4HTTP 서블릿을 활용한 파일 업로드 서블릿 작성하기

파일 업로드를 위한 HTTP 서블릿을 작성하기 전에, 파일 업로드를 할 때 내부적으로 어떻게 데이터가 전달 되는지를 살펴보도록 하겠습니다. 먼저, 파일 업로드를 위한 HTML 페이지부터 설계를 해야 되겠지요. 그리고 나서, 이 HTML 페이지를 이용하여 호출된 HTTP 서블릿에 대해서 살펴보도록 하겠습니다. 다음은 파일 업로드를 위한 간단한 HTML 페이지에 대한 예제 입니다.

```
<HTML>
<HEAD>
<TITLE>File Upload</TITLE>
</HEAD>

<BODY>
<H1>파일 올리기 테스트</H1>
<P>
<FORM method="POST"
      action="http://park.konkuk.ac.kr:8080/servlet/FileUploadEchoServlet"
      enctype=multipart/form-data>
이름: <input type="TEXT" name="name" size=16><BR>
파일: <input type="file" name="binary"><BR>
      <input type="submit" value="보내기">
</FORM>
</BODY>
</HTML>

/*
 *
```




```
 */
```

<프로그램 6. FileUploadEchoServlet.html>

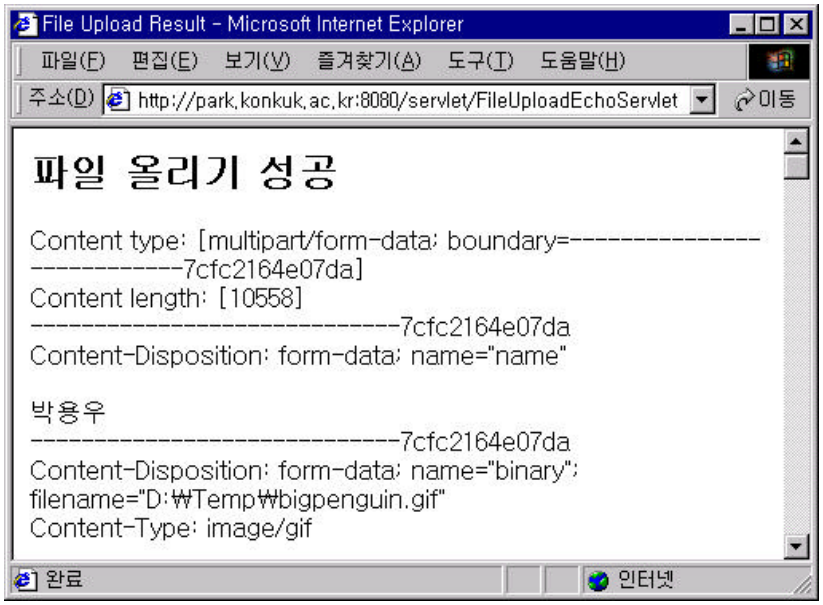
먼저, 주의 깊게 살펴볼 부분이 아무래도 <FORM> 태그내에 설정되어 있는 속성들이지요. 파일을 업로드하기 위해서는 <FORM> 태그내의 속성을 다음과 같이 지정해 주어야 합니다.

- method="POST" – 메소드를 POST 방식으로 설정합니다. 왜냐하면, GET 방식으로 전달할 경우 환경변수에 내용이 저장되므로 그 크기에 한계가 있습니다. 또한, 파일의 경우 대부분 바이너리 형식으므로 POST 방식으로 전달하는 것이 안전합니다.
- action="http://park.konkuk.ac.kr:8080/servlet/FileUploadEchoServlet" – 물론 파일 업로드를 위한 서블릿을 지정해 주어야 겠지요.
- enctype=multipart/form-data – 파일 업로드를 위해서 중요한 속성입니다. 다시 말해서, 업로드 할 파일의 형식을 지정한는 것으로서, 항상 "multipart/form-data"와 같이 해 주어야 합니다.
- 파일: <input type="file" name="binary"> - 파일을 업로드하기 위해 input 의 형식을 "file"로 지정해 주어야 합니다. 이는 웹 브라우저가 파일을 선택할 수 있는 버튼을 자동으로 제공하도록 함으로써, 사용자가 보내고자 하는 파일을 선택할 수 있도록 하는 것입니다.

위와 같이, 업로드하는 사람의 이름과 업로드 할 파일을 선택할 수 있도록 해 주는 HTML 페이지를 작성하였습니다. 이 때, 호출되는 HTTP 서블릿은 파일을 업로드 할 때, 내부적으로 어떻게 데이터가 전달되는 지를 알아보기 위해 웹 클라이언트로부터 전달되어 온 데이터를 그대로 출력하도록 하는 서블릿을 작성하고 테스트 결과를 확인해 보도록 하겠습니다. 다음에 나오는 자바 프로그램은 웹 클라이언트로부터 전달되어 온 데이터를 그대로 출력하는 HTTP 서블릿과 실행 결과입니다.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FileUploadEchoServlet extends HttpServlet {
   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
        res.setContentType("text/html; charset=euc-kr");
       PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<HEAD>");
       out.println("<TITLE>File Upload Result</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("<H2>파일 올리기 성공</H2>");

       out.println("  Content type: ["+req.getContentType()+"]<BR>");
       out.println("Content length: ["+req.getContentLength()+"]<BR>");
```

```
        InputStreamReader isr = new InputStreamReader(req.getInputStream(), "EUC-KR");
          BufferedReader br = new BufferedReader(isr);

         for(String line;(line=br.readLine())!=null;) {
              out.println(line+"<BR>");
          }

          out.close();

          out.println("</BODY>");
          out.println("</HTML>");
      }
}

/*
 *
```




```
 */
```

<프로그램 7. FileUploadEchoServlet.java>

위와 같이, 실제로 파일을 업로드 했을 경우, 내부적으로 전달되는 데이터를 그대로 출력한 결과 HTML 페이지는 이상한 문자들이 찍혀 있습니다. 이는 바이너리 파일을 업로드 했고, 그 내용이 HTTP 서블릿에게 전달되고, HTTP 서블릿은 이를 그대로 여과없이 결과 HTML 페이지에 출력했기 때문입니다. 위의 결과로 생성된 HTML 페이지를 살펴보면 다음과 같습니다.

```
<HTML>
<BODY>
<HEAD>
<TITLE>File Upload Result</TITLE>
</HEAD>
<BODY>
```

<H2>파일 올리기 성공</H2>
Content type: [multipart/form-data; boundary=----------7cfc2164e07da]
Content length: [10558]
----------7cfc2164e07da
Content-Disposition: form-data; name="name"

박용우
----------7cfc2164e07da
Content-Disposition: form-data; name="binary"; filename="D:\Temp\bigpenguin.gif"
Content-Type: image/gif

파일의 이진 데이터..
----------7cf1124e07da--

그림 6. 파일 업로드 서블릿이 생성한 결과 페이지

위의 그림을 살펴보면, 실제로 파일 업로드 서블릿이 생성한 HTML 태그를 제외한 부분들이 웹 클라이언트로부터 내부적으로 전달된 데이터들입니다. 이 데이터를 분석하여 파일 업로드 서블릿을 실제로 작성할 수 있습니다. 먼저, 위의 결과를 살펴보면, 다음과 같습니다.

- Content Type – req.getContentType 메소드를 이용하여 내용에 해당하는 데이터의 MIME 형식을 얻을 수 있는데, 이 때 MIME 형식이 "multipart/form-data" 일 경우에는 각 항목을 구분하기 위한 구분자를 얻을 수 있습니다. 위의 예에서는 다음과 같은 경계 정보를 얻을 수 있습니다.
  - multipart/form-data; boundary=----------7cfc2164e07da
- ----------7cf1124e07da – "enctype=multipart/form-data" 형식으로 웹 클라이언트로부터 넘겨져 온 폼의 항목을 구분하기 위한 구분줄입니다. 위의 예에서는 이름(name) 항목과 파일(binary) 항목을 구분하고 있고, 마지막으로 웹 클라이언트로부터 전달되어 온 입력 스트림의 끝을 나타내기 위해 사용된 것을 알 수 있습니다.
- Content-Disposition: – 폼의 각 항목에 대한 설명줄입니다. 이 라인을 분석하여 항목의 이름(name="…")을 얻을 수 있습니다. 만약, 항목이 파일일 경우에는 파일의 이름(filename="…")도 얻을 수 있습니다. 이 때, 항목이 파일일 경우에는 바로 다음 줄에 파일의 내용에 대한 MIME 형식이 따르게 됩니다. 그리고 나서, 한 줄을 건너뛰고, 항목에 해당하는 실제 내용이 나옵니다.
  - Content-Disposition: form-data; name="name" – 폼의 이름(name) 항목에 대한 지시줄로서, 폼 데이터이고, name 이 "name"임을 알 수 있습니다.
  - Content-Disposition: form-data; name="binary"; filename="D:\Temp\bigpenguin.gif" – 폼의 파일(ninary) 항목이 대한 지시줄로서 폼 데이터이고, name 이 "binary"이며,

filename 이 "D:\Temp\bigpenguin.gif" 임을 알 수 있습니다.

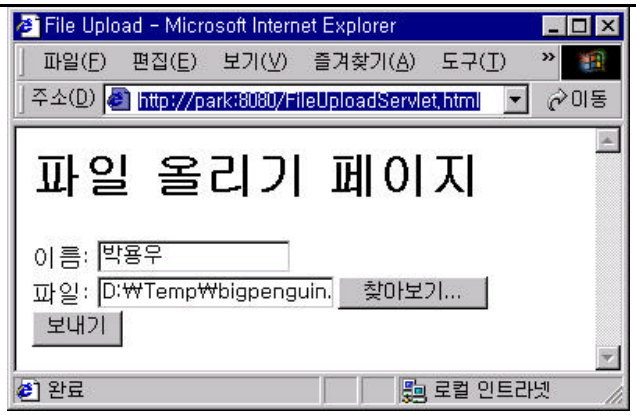
?? Content-Type: image/gif – 항목에 해당하는 내용에 대한 MIME 형식을 필요할 경우 나타냅니다. 위의 예에서는 파일에 대해 그 내용의 MIME 형식이 image/gif 임을 나타내고 있습니다.

- 빈줄 – 각 항목에 대한 설명이 끝나고 항목이 같는 내용값이 나오기 전에 한 줄을 건너뜁니다.
- 항목에 해당하는 내용값 – 구분줄과 설명줄들이 나온후, 한 줄을 띄고, 항목에 해당하는 내용이 나옵니다. 위의 예에서는 "박용우"가 하나의 항목에 대한 실제 내용값이고, 파일에 대한 내용값은 바이너리 값이기 때문에 생략하였습니다.
- ----------7cf1124e07da-- - 입력 스트림의 끝을 나타내는 마침줄입니다. 각 항목을 나타내는 구분줄과 같은 것 같지만, 한 가지 차이점은 구분줄의 ID 다음에 "--"가 더 있다는 것입니다. 따라서, 이 라인이 나오면 웹 클라이언트로부터 전달되어 온 데이터의 끝이라는 것을 알 수 있습니다.

위와 같이, 파일 업로드를 위한 전반적인 내용을 살펴보았습니다. 위의 분석 내용을 바탕으로 실제 파일 업로드를 수행할 HTTP 서블릿을 작성할 수 있습니다. 다음은 실제 파일 업로드를 수행하기 위한 HTML 페이지와 HTTP 서블릿에 대한 간단한 예를 보여주고 있습니다.

서블릿에 대해서 살펴보도록 하겠습니다. 다음은 파일 업로드를 위한 간단한 HTML 페이지에 대한 예제 입니다.

```
<HTML>
<HEAD>
<TITLE>File Upload</TITLE>
</HEAD>

<BODY>
<H1>파일 올리기 페이지</H1>
<P>
<FORM method="POST"
      action="http://park.konkuk.ac.kr:8080/servlet/ FileUploadServlet "
      enctype=multipart/form-data>
이름: <input type="TEXT" name="name" size=16><BR>
파일: <input type="file" name="binary"><BR>
      <input type="submit" value="보내기">
</FORM>
</BODY>
</HTML>

/*
 *
```

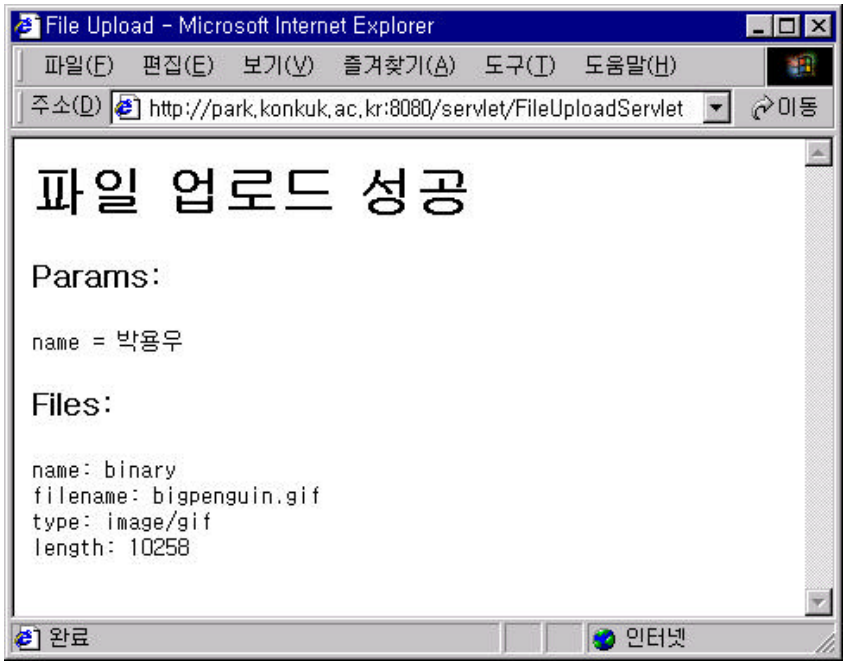


*/

<프로그램 8. FileUploadServlet.html>

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FileUploadServlet extends HttpServlet {
   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
       res.setContentType("text/html; charset=euc-kr");
      PrintWriter out = res.getWriter();
/*
      OutputStreamWriter osw = new OutputStreamWriter(res.getOutputStream();
     PrintWriter out = new PrintWriter(osw, "KSC5601") );
*/
       try {
         MultipartRequest multi = new MultipartRequest(req, ".", 5*1024*1024);

           out.println("<HTML>");
          out.println("<HEAD><TITLE>File Upload</TITLE></HEAD>");
           out.println("<BODY>");
           out.println("<H1>파일 업로드 성공</H1>");

           out.println("<H3>Params:</H3>");
           out.println("<PRE>");

          Enumeration params = multi.getParameterNames();
           while(params.hasMoreElements()) {
              String name = (String)params.nextElement();
              String value = multi.getParameter(name);

            out.println(name + " = " + value);
           }
           out.println("</PRE>");

           out.println("<H3>Files:</H3>");
           out.println("<PRE>");
          Enumeration files = multi.getFileNames();
           while(files.hasMoreElements()) {
```

```
                String name = (String)files.nextElement();
                String filename = multi.getFilesystemName(name);
                String type = multi.getContentType(name);
                File f = multi.getFile(name);

                out.println("name: " + name);
                out.println("filename: " + filename);
                out.println("type: " + type);
                if(f != null) {
                     out.println("length: " + f.length());
                     out.println();
                  }
                  out.println("</PRE>");
               }
          } catch(Exception e) {
               out.println("<PRE>");
               e.printStackTrace(out);
               out.println("</PRE>");
           }
           out.println("</BODY></HTML>");
      }
}

/*
 *
```




```
 */
```

<프로그램 9. FileUploadServlet.java>

```
import java.io.*;
import java.util.*;
import javax.servlet.*;

public class MultipartRequest {
```

```
private static final int DEFAULT_MAX_POST_SIZE=1024*1024;  // 1 Mega
private ServletRequest req;
private File dir;
private int maxLen;

private Hashtable parameters = new Hashtable();  // name - value
private Hashtable files = new Hashtable();         // name - UploadedFile

public MultipartRequest(ServletRequest req, String uploadDir)
throws IOException {
    this(req, uploadDir, DEFAULT_MAX_POST_SIZE);
}

public MultipartRequest(ServletRequest req, String uploadDir, int maxLen)
throws IOException {
    // Sanity check values
    if(req == null) {
       throw new IllegalArgumentException("request cannot be null");
     }
    if(uploadDir == null) {
       throw new IllegalArgumentException("saveDirectory cannot be null");
     }
    if(maxLen <= 0) {
        throw new IllegalArgumentException("maxPostSize must be positive");
     }

   // Save the request, dir, and max size
    this.req = req;
    dir = new File(uploadDir);
    this.maxLen = maxLen;

   // Check saveDirectory is truly a directory
    if(!dir.isDirectory()) {
      throw new IllegalArgumentException("Not a directory: " + uploadDir);
     }
   // Check saveDirectory is writable
    if(!dir.canWrite()) {
      throw new IllegalArgumentException("Not writable: " + uploadDir);
     }
  // Now parse the request saving data to "parameters" and "files";
  // write the file contents to the saveDirectory
     readRequest();
}

public Enumeration getParameterNames() {
    return parameters.keys();
}

public Enumeration getFileNames() {
    return files.keys();
}

public String getParameter(String name) {
    try {
        String param =(String)parameters.get(name);
        if("".equals(param)) {
            return(null);
         }
        return(param);
   } catch(Exception e) {
        return(null);
    }
```

```
  }

public String getFilesystemName(String name) {
    try {
        UploadedFile file =(UploadedFile)files.get(name);

        return(file.getFilesystemName());   // may be null
    } catch(Exception e) {
        return null;
     }
 }

public String getContentType(String name) {
    try {
        UploadedFile file =(UploadedFile)files.get(name);

        return(file.getContentType());   // may be null
    } catch(Exception e) {
        return(null);
     }
 }

public File getFile(String name) {
    try {
        UploadedFile file =(UploadedFile)files.get(name);

        return(file.getFile());   // may be null
    } catch(Exception e) {
        return(null);
     }
 }

protected void readRequest() throws IOException {
    String type = req.getContentType();

    if((type == null)||(!type.toLowerCase().startsWith("multipart/form-data"))) {
        throw new IOException("Posted content type isn't multipart/form-data");
     }

  // Check the content length to prevent denial of service attacks
   int len = req.getContentLength();
   if(len > maxLen) {
      throw new IOException("Content length: "+len+", Limit: "+maxLen);
     }

  // Get the boundary string; it's included in the content type.
   // Should look something like "-----------------------12012133613061"
   String boundary = extractBoundary(type);
   if(boundary == null) {
        throw new IOException("Separation boundary was not specified");
     }

  // Construct the special input stream we'll read from
    MultipartInputStreamHandler in =
        new MultipartInputStreamHandler(req.getInputStream(), boundary, len);

    // Read the first line, should be the first boundary
   String line = in.readLine();
   if(line == null) {
      throw new IOException("Corrupt form data: premature ending");
     }
```

```
    // Verify that the line is the boundary
      if(!line.startsWith(boundary)) {
       throw new IOException("Corrupt form data: no leading boundary");
      }

   // Now that we're just beyond the first boundary, loop over each part
    boolean done = false;
     while(!done) {
        done = readNextPart(in, boundary);
     }
 }

protected boolean readNextPart(MultipartInputStreamHandler in, String boundary)
 throws IOException {
   // Read the first line, should look like this:
     // content-disposition: form-data; name="field1"; filename="file1.txt"
    String line = in.readLine();
    if(line == null) {
       // No parts left, we're done
          return(true);
     }

    // Parse the content-disposition line
    String[] dispInfo = extractDispositionInfo(line);
     String disposition = dispInfo[0];
    String name = dispInfo[1];
    String filename = dispInfo[2];

   // Now onto the next line.  This will either be empty
    // or contain a Content-Type and then an empty line.
     line = in.readLine();
     if(line == null) {
       // No parts left, we're done
          return(true);
      }

   // Get the content type, or null if none specified
    String contentType = extractContentType(line);
    if(contentType != null) {
       // Eat the empty line
          line = in.readLine();
      if((line == null)||(line.length() > 0)) {   // line should be empty
          throw new IOException("Malformed line after content type: " + line);
          }
    } else {
       // Assume a default content type
          contentType = "application/octet-stream";
     }

   // Now, finally, we read the content(end after reading the boundary)
    if(filename == null) {
       // This is a parameter
       String value = readParameter(in, boundary);

          parameters.put(name, value);
    } else {
       // This is a file
        readAndSaveFile(in, boundary, filename);
         if("unknown".equals(filename)) {
            files.put(name, new UploadedFile(null, null, null));
        } else {
              files.put(name, new UploadedFile(dir.toString(), filename, contentType));
```

```
            }
        }
        return(false);  // there's more to read
    }

    protected String readParameter(MultipartInputStreamHandler in, String boundary)
    throws IOException {
        StringBuffer sbuf = new StringBuffer();
        String line;

        while((line = in.readLine()) != null) {
            if(line.startsWith(boundary)) {
                break;
            }
            sbuf.append(line + "\r\n");  // add the \r\n in case there are many lines
        }
        if(sbuf.length() == 0) {
            return(null);  // nothing read
        }
        sbuf.setLength(sbuf.length() - 2);  // cut off the last line's \r\n

        return(sbuf.toString());  // no URL decoding needed
    }

    protected void readAndSaveFile(MultipartInputStreamHandler in,
                                   String boundary,
                                   String filename)
    throws IOException {
        File f = new File(dir + File.separator + filename);
        FileOutputStream fos = new FileOutputStream(f);
        BufferedOutputStream out = new BufferedOutputStream(fos, 8 * 1024); // 8K

        byte[] bbuf = new byte[100*1024];  // 100K
        int result;
        String line;

        // ServletInputStream.readLine() has the annoying habit of
        // adding a \r\n to the end of the last line.
        // Since we want a byte-for-byte transfer, we have to cut those chars.
        boolean rnflag = false;
        while((result = in.readLine(bbuf, 0, bbuf.length)) != -1) {
            // Check for boundary
            if(result > 2 && bbuf[0] == '-' && bbuf[1] == '-') { // quick pre-check
                line = new String(bbuf, 0, result, "ISO-8859-1");
                if(line.startsWith(boundary)) {
                    break;
                }
            }
            // Are we supposed to write \r\n for the last iteration?
            if(rnflag) {
                out.write('\r'); out.write('\n');
                rnflag = false;
            }
            // Write the buffer, postpone any ending \r\n
            if((result>=2)&&(bbuf[result-2]=='\r')&&(bbuf[result-1]=='\n')) {
                out.write(bbuf, 0, result - 2);  // skip the last 2 chars
                rnflag = true;  // make a note to write them on the next iteration
            } else {
                out.write(bbuf, 0, result);
            }
        }
        out.flush();
```

```
        out.close();
        fos.close();
    }

  // Extracts and returns the boundary token from a line.
    //
  private String extractBoundary(String line) {
      int index = line.indexOf("boundary=");

      if(index == -1) {
           return null;
        }
     String boundary = line.substring(index + 9);  // 9 for "boundary="

    // The real boundary is always preceeded by an extra "--"
      boundary = "--" + boundary;

       return boundary;
    }

  // Extracts and returns disposition info from a line, as a String array
  // with elements: disposition, name, filename.  Throws an IOException
  // if the line is malformatted.
    //
  private String[] extractDispositionInfo(String line) throws IOException {
      // Return the line's data as an array: disposition, name, filename
      String[] retval = new String[3];

    // Convert the line to a lowercase string without the ending \r\n
    // Keep the original line for error messages and for variable names.
       String origline = line;
       line = origline.toLowerCase();

    // Get the content disposition, should be "form-data"
      int start = line.indexOf("content-disposition: ");
      int end = line.indexOf(";");
      if(start == -1 || end == -1) {
          throw new IOException("Content disposition corrupt: " + origline);
        }
     String disposition = line.substring(start + 21, end);
       if(!disposition.equals("form-data")) {
         throw new IOException("Invalid content disposition: " + disposition);
        }

     // Get the field name
      start = line.indexOf("name=\"", end);  // start at last semicolon
      end = line.indexOf("\"", start + 7);   // skip name=\"
      if(start == -1 || end == -1) {
         throw new IOException("Content disposition corrupt: " + origline);
        }
     String name = origline.substring(start + 6, end);

     // Get the filename, if given
      String filename = null;
      start = line.indexOf("filename=\"", end + 2);  // start after name
      end = line.indexOf("\"", start + 10);             // skip filename=\"
      if(start != -1 && end != -1) {                    // note the !=
         filename = origline.substring(start + 10, end);
        // The filename may contain a full path.  Cut to just the filename.
          int slash = Math.max(filename.lastIndexOf('/'), filename.lastIndexOf('\\'));
          if(slash > -1) {
            filename = filename.substring(slash + 1);  // past last slash
```

```
            }
            if("".equals(filename)) {
              filename = "unknown"; // sanity check
            }
          }

        // Return a String array: disposition, name, filename
          retval[0] = disposition;
          retval[1] = name;
          retval[2] = filename;

          return(retval);
      }

    // Extracts and returns the content type from a line, or null if the
    // line was empty.  Throws an IOException if the line is malformatted.
      //
    private String extractContentType(String line) throws IOException {
        String contentType = null;

        // Convert the line to a lowercase string
          String origline = line;
          line = origline.toLowerCase();

        // Get the content type, if any
          if(line.startsWith("content-type")) {
            int start = line.indexOf(" ");
            if(start == -1) {
                throw new IOException("Content type corrupt: " + origline);
            }
            contentType = line.substring(start + 1);
        } else if(line.length() != 0) {  // no content type, so should be empty
            throw new IOException("Malformed line after disposition: " + origline);
          }

          return contentType;
      }
}

// A class to hold information about an uploaded file.
//
class UploadedFile {
    private String dir;
    private String filename;
    private String type;

    UploadedFile(String dir, String filename, String type) {
        this.dir = dir;
        this.filename = filename;
        this.type = type;
      }

    public String getContentType() {
        return(type);
      }

    public String getFilesystemName() {
        return(filename);
      }

    public File getFile() {
        if(dir == null || filename == null) {
```

```
            return(null);
        } else {
            return(new File(dir + File.separator + filename));
        }
    }
}

// A class to aid in reading multipart/form-data from a ServletInputStream.
// It keeps track of how many bytes have been read and detects when the
// Content-Length limit has been reached.  This is necessary since some
// servlet engines are slow to notice the end of stream.
//
class MultipartInputStreamHandler {
    ServletInputStream in;
    String boundary;
    int totalExpected;
   int totalRead = 0;
   byte[] buf = new byte[8 * 1024];

    public MultipartInputStreamHandler(ServletInputStream in,
                                       String boundary,
                                      int totalExpected) {
        this.in = in;
       this.boundary = boundary;
       this.totalExpected = totalExpected;
    }

   // Reads the next line of input.  Returns null to indicate the end
    // of stream.
     //
   public String readLine() throws IOException {
       StringBuffer sbuf = new StringBuffer();
       int result;
       String line;

       do {
          result = this.readLine(buf, 0, buf.length);  // this.readLine() does +=
           if(result != -1) {
              sbuf.append(new String(buf, 0, result)); //, "ISO-8859-1"));
           }
      } while(result == buf.length);  // loop only if the buffer was filled

       if(sbuf.length() == 0) {
          return(null);  // nothing read, must be at the end of stream
       }

       sbuf.setLength(sbuf.length() - 2);  // cut off the trailing \r\n
       return(sbuf.toString());
    }

   // A pass-through to ServletInputStream.readLine() that keeps track
  // of how many bytes have been read and stops reading when the
   // Content-Length limit has been reached.
    //
   public int readLine(byte b[], int off, int len) throws IOException {
       if(totalRead >= totalExpected) {
            return(-1);
       } else {
          int result = in.readLine(b, off, len);

           if(result > 0) {
               totalRead += result;
```

```
            }
            return(result);
        }
    }

    public static String toKSC5601(String str)
        throws UnsupportedEncodingException {
        if(str == null) {
            return(null);
        }
        return(new String(str.getBytes("8859_1"), "KSC5601"));
    }
}
```

<프로그램 10. MultipartRequest.java>

1.3쿠키와 세션

1.3.1Cookie 클래스

Cookie 란 서블릿이 어떤 적은양의 정보를 웹 브라우저에게 보내고, 웹 브라우저는 그 정보를 저장하고 있다가, 나중에 다시 이 웹페이지를 방문할 때 웹 서버에게 제출하도록 하는 것입니다. 쿠키 값은 클라이언트에 대해 유일한 값이 될 수도 있는데, 이러한 쿠키는 일반적으로 세션 관리를 위해 사용됩니다. 쿠키는 이름과 그 이름에 해당하는 하나의 값을 갖고 있으며, 설명(commnent), 경로와 도메인 구분자(path and domain qualifiers), 유효 기간(maximum age), 그리고 버전(version number) 등과 같은 추가적인 속성을 가질 수 있습니다. 쿠키는 처음엔 웹브라우저가 웹서버에 접속했을 때 받아가는 HTTP-Response Header 에 정보가 담겨와서 세팅된다. 이 헤더에 있는 Set-Cookie 라는 필드를 웹브라우저가 읽어서 메모리상에 저장하면 Cookie 가 세팅되는 것인데, 다음과 같은 형식으로 헤더에 적혀서 온다.

```
Set-Cookie: NAME=VALUE; expires=DATE;
path=PATH; domain=DOMAIN_NAME; secure
```

그림 7. 쿠키의 간단한 구조

위의 그림에서 보면 간단하게 두 줄이 있는데, 반드시 기술해 주어야 하는 것은 NAME=VALUE 이고 나머지는 없어도 된다. 쿠키의 각 항목을 살펴보면, 다음과 같습니다.

- Set-Cookie: – 쿠키에 대한 설정임을 나타냅니다.
- NAME=VALUE; – 쿠키의 이름과 값을 설정합니다. 여러 개가 동시에 나타날 수 있습니다. 하나 이상 반드시 나타나야 합니다.
- expires=DATE; – 쿠키의 유효기간을 나타냅니다. 다시 말해서, 주어진 시간이 경과하면 쿠키는 더 이상 쓸모가 없게 됩니다. 이 값을 지정하지 않으면, 웹 브라우저를 종료하자만자 쿠키는 쓸모없게 되며, 지정한 시간이 지나지 않았다면, 웹 브라우저는 종료하면서 쿠키를 웹 브라우저가 있는 컴퓨터의 하드 디스크에 저장하게 됩니다. 일반적으로, "WINDOW_ROOT\\Profiles\\사용자 ID\\Cookie" 디렉토리입니다.
- domain=DOMAIN_NAME; – 웹 브라우저가 어떤 도메인에 접속 했을 때, 이 쿠키가 유효한지를 나타냅니다. 이 값이 세팅되어 있으면 이 도메인으로 접속할 때만 웹 브라우저가 쿠키를 웹서버에게 보내게 됩니다.
- path=PATH; – 쿠키가 유효한 도메인 내의 경로명을 나타냅니다.
- Secure – 이 설정은 보안관련 설정으로서, secure 가 설정되어 있으면 SSL 같은 보안채널을 통해서만 억세스될 수 있고, 일반 HTTP 를 통해서는 쿠키를 억세스할 수 없습니다.

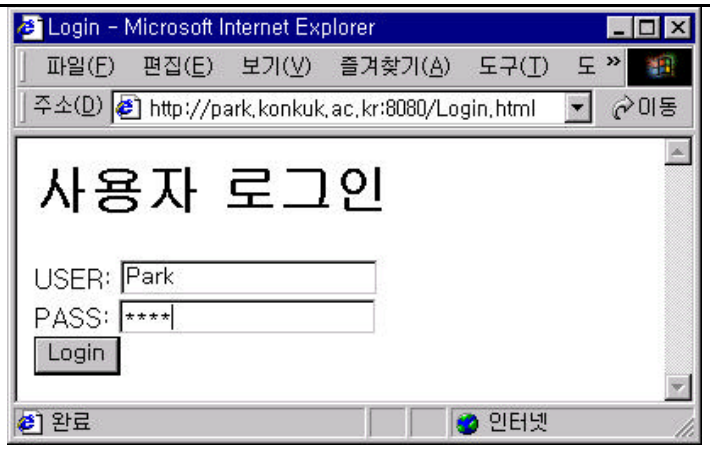
서블릿은 HttpServletResponse 객체의 addCookie 메소드를 이용하여 웹 브라우저에게 쿠키를 보낼 수 있습니다. 이 때, HttpServletResponse 객체의 addCookie 메소드는 웹 브라우저에게 쿠키를 보내기 위해 한 번에 한 개씩 HTTP 응답 헤더(HTTP response headers)에 필드를 추가하게 됩니다. 브라우저는 각 웹 서버당 20 개의 쿠키를 지원하는 것으로 되어 있고, 이는 적어도 4KB 정도의 크기입니다. 브라우저는 HTTP 요청 헤더(HTTP request headers)에 쿠키 항목을 추가함으로써 서블릿에게 쿠키를 보내게 됩니다. 이 때, 서블릿은 이 쿠키들을 가져오기 위해서는 HttpServletRequest 객체의 getCookies 메소드를 사용할 수 있습니다. 어떤 쿠키는 이름은 같지만 경로 속성이 서로 다를 수 있습니다. Cookie 클래스는 이러한 쿠키의 기능을 제공해 주며, 현재 Netscape 에 의해 정의된 버전 0 과 RFC 2109 에 정의된 버전 1 을 지원하고 있습니다. 쿠키가 생성되는 디폴트 버전은 최상의 호환성을 위해 버전 0 으로 생성됩니다. 이렇게 쿠키의 기능을 지원하기 위해 Cookie 클래스에서 제공해 주고 있는 메소드를 살펴보면, 다음과 같습니다.

- Cookie(java.lang.String name, java.lang.String value): 주어진 이름과 값을 갖는 쿠키를 생성합니다.
- java.lang.String getComment(): 쿠키의 목적을 설명하고 있는 설명 속성을 얻습니다.
- java.lang.String getDomain(): 쿠키를 위한 도메인 이름 속성을 얻습니다.
- int getMaxAge(): 쿠키에 주어진 유효기간을 초단위로 얻습니다.
- java.lang.String getName(): 쿠키그이 이름을 얻습니다.
- java.lang.String getPath(): 브라우저가 쿠키를 되돌려 준 서버상의 경로를 얻습니다.
- boolean getSecure(): 브라우저가 보안 프로토콜을 통해서 쿠키를 보내도 있는 지를 얻습니다.
- java.lang.String getValue(): 쿠키의 값을 얻습니다.

- int getVersion(): 쿠키가 따르는 프로토콜의 버전을 얻습니다.
- void setComment(java.lang.String purpose): 쿠키의 목적을 설명하는 설명 속성을 설정합니다. 이 메소드는 version 값이 0일 때는 사용할 수 없습니다.
- void setDomain(java.lang.String pattern): 쿠키의 도메인 속성을 설정합니다.
- void setMaxAge(int expiry): 쿠키의 유효기간을 초단위로 설정합니다.
- void setPath(java.lang.String uri): 쿠키의 경로 속성을 설정합니다.
- void setSecure(boolean flag): HTTPS 또는 SSL 등과 같은 보안 프로토콜을 이용하여 쿠키가 보내져야 하는지를 설정합니다.
- void setValue(java.lang.String newValue): 쿠키의 값을 설정합니다.
- void setVersion(int v): 쿠키의 버전을 설정합니다.

쿠키는 처음에는 메모리에 저장되기 때문에 사용자가 임의로 조작할 수 없으므로, 사용자 인증 또는 세션 관리를 위해 주로 사용됩니다. 이 때, 사용자 인증을 끝내면 인증이라는 이름으로 쿠키를 만듦니다. 그런 다음 인증이 필요한 모든 웹페이지에서 이 쿠키의 유무를 체크하고 서비스를 제공하게 됩니다. 쿠키는 디폴트로 웹브라우저가 종료될 때까지만 살아있기 때문에 브라우저를 종료하기 전까지는 인증에 필요한 쿠키가 유효하므로, 위와 같은 사용자 인증이 가능하게 됩니다. 다음에 나오는 예제는 이와 같은 사용자 인증을 위해 서블릿에서 쿠키를 사용하는 예를 보여주는 자바 프로그램입니다.

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=euc-kr">
<TITLE>Login</TITLE>
</HEAD>

<BODY>
<H1>사용자 로그인</H1>
<P>
<FORM action="http://park.konkuk.ac.kr:8080/servlet/LoginServlet"
        method="POST">
USER: <input type=text name=user><BR>
PASS: <input type=password name=pass><BR>
        <input type=submit value="Login">
</FORM>

</BODY>
</HTML>

/*
 *
```

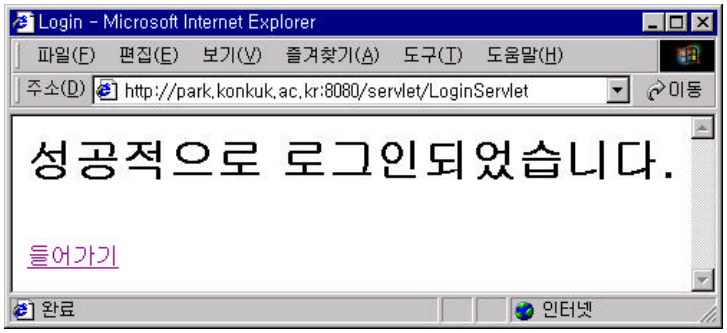


*/

<프로그램 11. Login.html>

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoginServlet extends HttpServlet {
   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
         res.setContentType("text/html;charset=euc-kr");
        PrintWriter out = res.getWriter();

         out.println("<HTML>");
         out.println("<BODY>");
         out.println("<HEAD>");
         out.println("<TITLE>Login</TITLE>");
         out.println("</HEAD>");

         out.println("<BODY>");
        String user = req.getParameter("user");
        user = new String(user.getBytes("ISO-8859-1"), "EUC-KR");
        String pass = req.getParameter("pass");

        if(("Park".equals(user)) && ("Park".equals(pass))) {
            Cookie loginCookie = new Cookie("BBS-Park", "Login");
            Cookie userCookie = new Cookie("Username", user);

              res.addCookie(loginCookie);
              res.addCookie(userCookie);
              out.println("<H1>성공적으로 로그인되었습니다.</H1><BR>");
               out.println("<A   HREF=\"http://park.konkuk.ac.kr:8080/servlet/StartServiceServlet\">들어
가기");
        } else {
              out.println("<H1>사용자 이름 또는 암호가 틀립니다.</H1><BR>");
             out.println("<A  HREF=\"/Login.html\">되돌아가기");
         }
         out.println("</BODY>");
         out.println("</HTML>");
```

```
        out.close();
    }
}

/*
 *
```



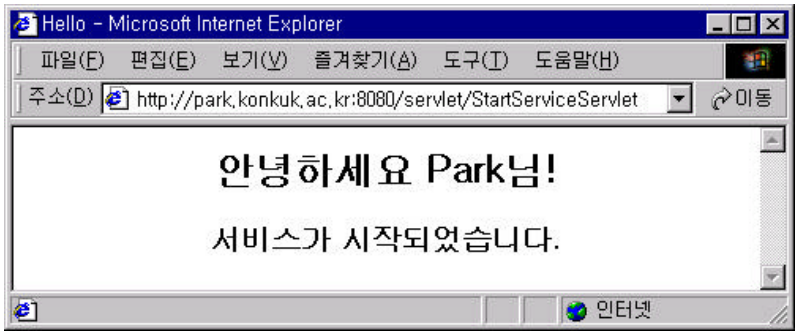

```
 */
```

<프로그램 12. LoginServlet.java>

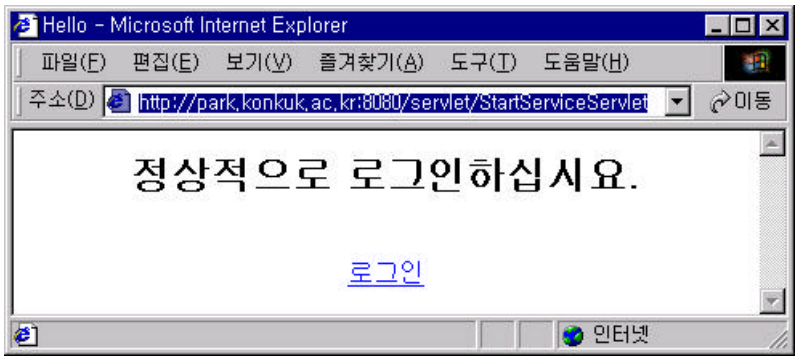
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class StartServiceServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws IOException, ServletException {
       res.setContentType("text/html; charset=euc-kr");

       PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<HEAD>");
        out.println("<TITLE>Hello</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");

        int match=0;
       String user = null;
        boolean login=false;
       Cookie[] cookies = req.getCookies();
       if(cookies != null) {
           for(int i=0;i<cookies.length;i++) {
               if(("BBS-Park".equals(cookies[i].getName())) &&
                   ("Login".equals(cookies[i].getValue()))) {
                   login = true;
                    match++;
               } else if("Username".equals(cookies[i].getName())) {
                    user = cookies[i].getValue();
                     match++;
                }
               if(match >= 2) {
                     break;
                }
```

```
            }
        }

        out.println("<CENTER>");
        if(login) {
            out.println("<H2>안녕하세요 "+user+"님!</H2>");
            out.println("<H3>서비스가 시작되었습니다.</H3>");
        } else {
            out.println("<H2>정상적으로 로그인하십시요.</H2><BR>");
            out.println("<A HREF=\"/Login.html\">로그인");
        }

        out.println("</BODY>");
        out.println("</HTML>");
    }

   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws IOException, ServletException {
       doGet(req, res);
    }
}

/*
 *
```

1) 정상적으로 로그인 한 경우



2) 정상적으로 로그인 하지 않고 바로 실행한 경우
(브라우저에서 http://park.konkuk.ac.kr:8080/servlet/StartServiceServlet 실행한 경우)




```
*/
```

<프로그램 13. LoginServlet.java>

위와 같은 구조로 사용자 인증을 할 수 있습니다. 위의 각 파일들을 살펴보면, 다음과 같습니다.

- Login.html: 사용자 로그인을 위한 HTML 페이지 입니다.
- LoginServlet.java: 사용자 로그인 HTML 페이지로부터 폼 데이터를 입력받아 사용자 인증을 수행하고, 정상적으로 로그인했다면, 쿠키를 설정합니다. 이 쿠키는 정상적으로 로그인 했다는 정보와 사용자 이름을 위한 두 개의 쿠키가 있습니다.
- StartServiceServlet.java: 실제 서비스를 수행하는 서블릿입니다. 이 페이지는 사용자가 정상적으로 로그인 했을 때 설정된 쿠키 정보를 기반으로 정상적으로 로그인 한 경우에만 사용할 수 있고, 쿠키가 설정되어 있지 않다면 정상적으로 로그인 한 것이 아니므로, 로그인 HTML 페이지로 되돌아 갈 수 있도록 해 줍니다.

이러한 쿠키는 잘 사용하면 아주 유용하지만, 보안상 무척 취약합니다. 그래서, 보안이 중요시 되는 곳에서는 아주 유의하여 사용해야 합니다.

1.3.2HttpSession 인터페이스

HttpSession 인터페이스는 같은 웹 사이트 내에서 여러 페이지를 둘러볼 경우, 그 사용자를 구별할 수 있도록 하는 기능을 제공해 줍니다. 카운터를 예로들면, 한 사용자가 어떤 웹 사이트의 시작 페이지에 처음 접속할 때 자동으로 증가합니다. 이 사용자가 같은 웹 사이트 내의 다른 페이지를 보다가 다시 시작 페이지로 되돌아 왔을 경우, 이 카운터가 자동으로 증가하면 안됩니다. 이를 위해, 이 사용자가 이미 연결 설정되어 있는 사용자인지를 구별해 주어야 합니다. 이렇게 웹 클라이언트(사용자)와 웹 서버 간에 연결을 유지해야 하는데, 이러한 것을 가능하게 해 주는 것이 세션이라는 개념입니다. 일반적으로, 세션은 한 사용자와 웹 서버간에 존재하는 가상 연결이라 할 수 있고, 사용자(웹 브라우저)와 웹 서버간에 하나의 세션이 성립되도록 되어 있습니다. 이를 위해, HttpSession 인터페이스는 세션 구분자(session identifier), 생성 시간(creation time), 세션 문맥(session context) 등과 같은 정보를 유지할 수 있는 기능을 제공해 줍니다. 이렇게 HttpSession 인터페이스에서 정의하고 있는 기능들을 살펴보면, 다음과 같습니다.

- java.lang.String getId(): 세션에 유일하게 할당된 ID를 얻습니다.
- long getCreationTime(): 세션이 생성된 날자를 얻습니다. 시간은 January 1, 1970 GMT 자정부터 경과한 밀리초 단위입니다..
- long getLastAccessedTime(): 클라이언트가 이 세션에 마지막으로 요청을 보낸 시간을 얻습니다.
- int getMaxInactiveInterval(): 서블릿 엔진이 이 세션이 열려도록 유지하기 위한 최대 시간 간경을 얻습니다. 서블릿 엔진은 이 시간이 경과하도록 클라이언트로부터 요청이 없으면

세션을 닫습니다.

- java.lang.Object getValue(java.lang.String name): 이 세션에 주어진 이름으로 연결된 객체를 얻습니다.
- java.lang.String[] getValueNames(): 이 세션에 연결된 모든 객체들의 이름을 배열로 얻습니다.
- void invalidate(): 이 세션을 무효화하여, 세션에 연결된 객체들에 대해 연결 해제 합니다.
- boolean isNew(): 웹 서버가 새롭게 이 세션을 생성하고 아직 클라이언트가 연결하지 않은 새로운 세션인지를 얻습니다.
- void putValue(java.lang.String name, java.lang.Object value): 주어진 이름을 이용하여 객체를 이 세션에 연결합니다.
- void removeValue(java.lang.String name): 주어진 이름의 객체를 이 세션으로부터 연결 해제 합니다.
- void setMaxInactiveInterval(int interval): 서블릿 엔진이 이 세션이 열려도록 유지하기 위한 최대 시간 간경을 설정합니다. 서블릿 엔진은 이 시간이 경과하도록 클라이언트로부터 요청이 없으면 세션을 닫습니다.

1.3.3HttpSessionBindingListener 인터페이스

HttpSessionBindingListener 인터페이스는 객체가 세션에 연결되거나 또는 연결 해제될 때 작동할 수 있는 리스너를 정의하고 있는 인터페이스입니다. HttpSessionBindingListener 객체는 HttpSessionBindingEvent 객체를 받아 처리하도록 되어 있습니다. 이러한 HttpSessionBindingListener 인터페이스에서 정의하고 있는 메소드를 살펴보면 다음과 같습니다.

- void valueBound(HttpSessionBindingEvent event): 세션에 연결될 때 발생하는 이벤트를 처리하기 위한 메소드입니다.
- void valueUnbound(HttpSessionBindingEvent event): 세션으로부터 연결 해제될 때 발생하는 이벤트를 처리하기 위한 메소드입니다.

1.3.4HttpSessionBindingEvent 클래스

HttpSessionBindingEvent 클래스는 객체가 세션에 연결되거나 또는 연결 해제될 때, HttpSessionBindingListener 구현하는 객체에게 주어지는 이벤트에 대한 기능을 제공해 주고 있습니다. 세션은 HttpSession 클래스가 제공해 주는 putValue 메소드를 호출함으로써 객체를 세션에 연결하고, HttpSession 클래스가 제공해 주는 removeValue 메소드를 호출함으로써 객체를 세션에서 연결 해제 합니다. 이러한 HttpSessionBindingEvent 클래스가 제공해 주는 메소드를

살펴보면, 다음과 같습니다.

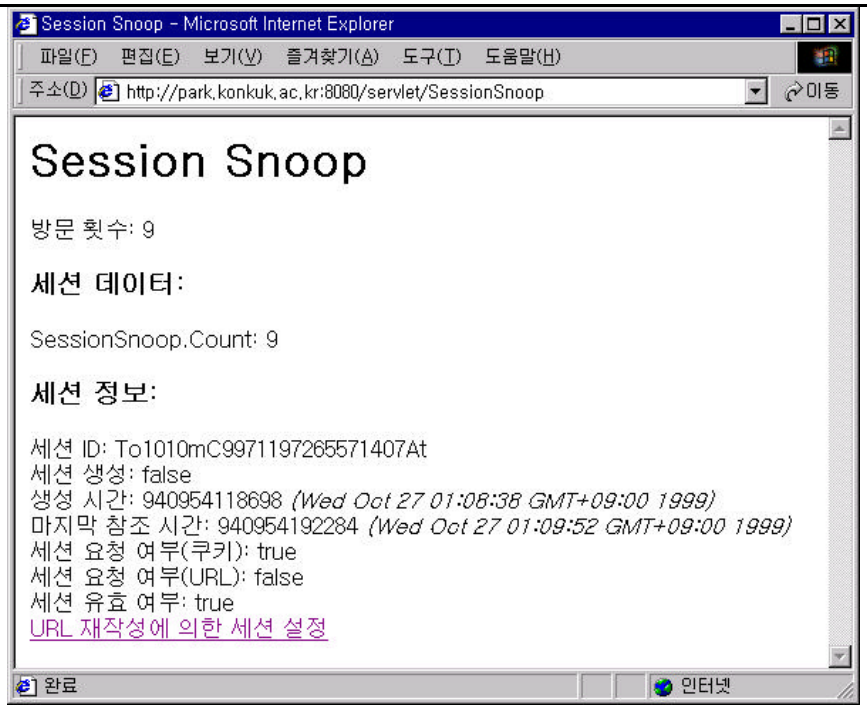
- HttpSessionBindingEvent(HttpSession session, java.lang.String name): 객체가 세션에 연결되었는지 세션으로부터 연결 해제되었는지를 알리기 위한 이벤트를 생성합니다.
- java.lang.String getName(): 객체가 세션에 연결되었거나 또는 세션으로부터 연결 해제될 때 사용한 이름을 얻습니다.
- HttpSession getSession(): 객체가 연결되었거나 또는 연결 해제된 세션을 얻습니다.

1.3.5세션 유지 서블릿 예제

HTTP 의 특성상 연속된 연결은 존재할 수 없습니다. HTTP 는 웹서버에 접속했다가 데이터의 전송이 끝나면 곧바로 연결을 끊어버립니다. 즉, 연결에 대한 영속성(persistence)이 없다는 것입니다. 이러한 방법은 부족한 네트워크 자원을 효율적으로 사용할 수는 있지만, 웹 서비스를 위한 사용자 인증 및 유지와 같은 사용자 관리상의 비효율적인 면들이 존재합니다. 물론, 앞서 배운 쿠키를 사용하면 사용자 인증을 그럴듯하게 만들 수 있지만 쿠키를 이용한 사용자 인증은 관리의 책임이 모두 프로그래머에게 있고, 쿠키를 브라우저 설정에 따라 사용하지 못할 수도 있기 때문에 완벽한 사용자 관리를 할 수 없습니다. 그래서, 서블릿에서는 세션 관리를 위한 전반적인 기능을 제공해 주는 HttpSession 인터페이스를 제공해 주고 있습니다. 즉, 하나의 사용자는 언제나 같은 컴퓨터에서 접속을 한다는 사실을 이용한 것입니다. 사용자는 여러 컴퓨터를 돌아가면서 같은 싸이트에 접속하지는 않기 때문에 서비스를 제공하는 서버측에서는 클라이언트의 정보를 체크해서 동일한 클라이언트인지를 체크할 수 있습니다. 이 때, 클라이언트에 대한 정보로는 쿠키를 사용합니다. 이렇게 함으로써, 쿠키를 직접 이용하는 것 보다 더 효과적인 클라이언트 관리가 이루어질 수 있습니다. 다음에 나오는 예제는 HTTP 서블릿에서 사용하는 예를 보여주는 간단한 자바 프로그램입니다.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SessionSnoop extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
        res.setContentType("text/html;charset=euc-kr");
       PrintWriter out = res.getWriter();
       HttpSession session = req.getSession(true);

        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<HEAD>");
       out.println("<TITLE>Session Snoop</TITLE>");
        out.println("</HEAD>");
```

```
        out.println("<BODY>");
        out.println("<H1>Session Snoop</H1>");

       Integer count = (Integer)session.getValue("SessionSnoop.Count");
       if(count == null) {
          count = new Integer(1);
        } else {
          count = new Integer(count.intValue() + 1);
         }
         session.putValue("SessionSnoop.Count", count);

         out.println("방문 횟수: "+count+"<BR>");

         out.println("<P>");

         out.println("<H3>세션 데이터:</H3>");
        String[] names = session.getValueNames();
        for(int i=0;i<names.length;i++) {
            out.println(names[i]+": "+session.getValue(names[i])+"<BR>");
         }

         out.println("<H3>세션 정보:</H3>");
         out.println("세션 ID: " + session.getId() + "<BR>");
         out.println("세션 생성: " + session.isNew() + "<BR>");
         out.println("생성 시간: " + session.getCreationTime());
         out.println("<I>(" + new Date(session.getCreationTime()) + ")</I><BR>");
         out.println("마지막 참조 시간: " + session.getLastAccessedTime());
       out.println("<I>(" + new Date(session.getLastAccessedTime()) + ")</I><BR>");

         out.println("세션 요청 여부(쿠키): " +
                     req.isRequestedSessionIdFromCookie() + "<BR>");
         out.println("세션 요청 여부(URL): " +
                     req.isRequestedSessionIdFromURL() + "<BR>");
         out.println("세션 유효 여부: " +
                     req.isRequestedSessionIdValid() + "<BR>");

         out.println("<A HREF=\""+res.encodeURL(req.getRequestURI())+"\">");
        out.println("URL 재작성에 의한 세션 설정</A>");

         out.println("</BODY>");
         out.println("</HTML>");
    }
}

/*
 *

 1) 정상적으로 로그인 한 경우
```



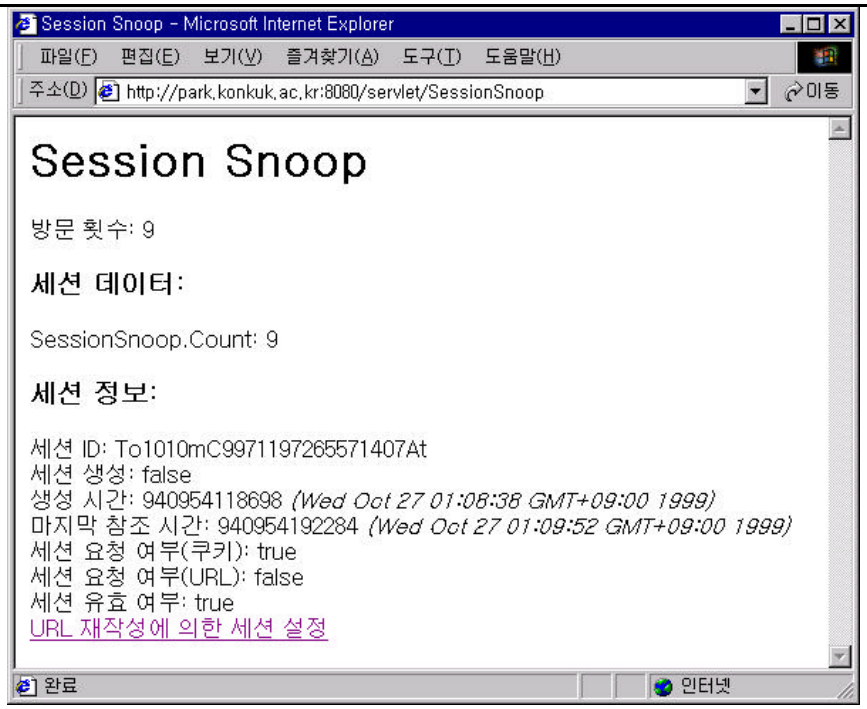
*/

<프로그램 14. LoginServlet.java>

다음에 나오는 예제는 HTTP 서블릿에서 사용하는 예를 보여주는 간단한 자바 프로그램입니다.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SessionSnoop extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
        res.setContentType("text/html;charset=euc-kr");
       PrintWriter out = res.getWriter();
       HttpSession session = req.getSession(true);

        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<HEAD>");
       out.println("<TITLE>Session Snoop</TITLE>");
        out.println("</HEAD>");

        out.println("<BODY>");
       out.println("<H1>Session Snoop</H1>");

       Integer count = (Integer)session.getValue("SessionSnoop.Count");
       if(count == null) {
           count = new Integer(1);
       } else {
```

```
            count = new Integer(count.intValue() + 1);
          }
         session.putValue("SessionSnoop.Count", count);

         out.println("방문 횟수: "+count+"<BR>");

         out.println("<P>");

         out.println("<H3>세션 데이터:</H3>");
        String[] names = session.getValueNames();
        for(int i=0;i<names.length;i++) {
            out.println(names[i]+": "+session.getValue(names[i])+"<BR>");
        }

         out.println("<H3>세션 정보:</H3>");
         out.println("세션 ID: " + session.getId() + "<BR>");
         out.println("세션 생성: " + session.isNew() + "<BR>");
         out.println("생성 시간: " + session.getCreationTime());
       out.println("<I>(" + new Date(session.getCreationTime()) + ")</I><BR>");
         out.println("마지막 참조 시간: " + session.getLastAccessedTime());
       out.println("<I>(" + new Date(session.getLastAccessedTime()) + ")</I><BR>");

         out.println("세션 요청 여부(쿠키): " +
                     req.isRequestedSessionIdFromCookie() + "<BR>");
         out.println("세션 요청 여부(URL): " +
                     req.isRequestedSessionIdFromURL() + "<BR>");
         out.println("세션 유효 여부: " +
                     req.isRequestedSessionIdValid() + "<BR>");

        out.println("<A HREF=\""+res.encodeURL(req.getRequestURI())+"\">");
        out.println("URL 재작성에 의한 세션 설정</A>");

         out.println("</BODY>");
         out.println("</HTML>");
    }
}

/*
 *

 1) 정상적으로 로그인 한 경우
```



*/

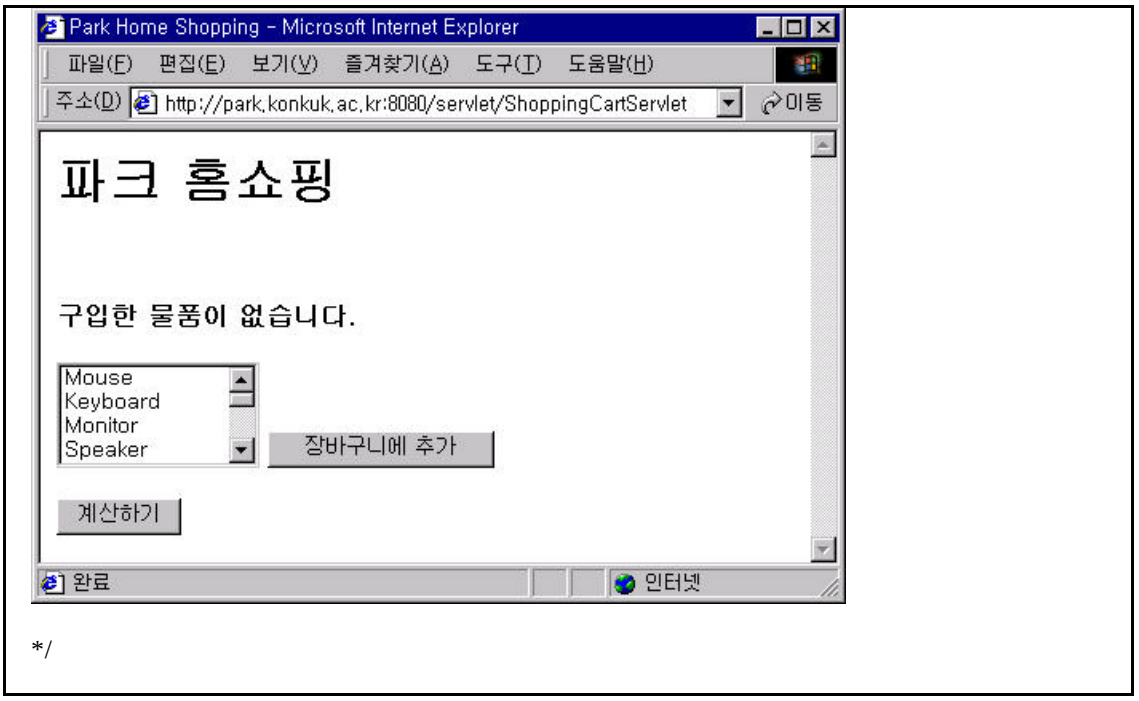
<프로그램 15. SessionSnoop.java>

위의 예제 프로그램을 살펴보면, 우선 HttpSession 객체를 얻기위해 HttpServletRequest 객체의 getSession(boolean) 메소드를 사용하였습니다. 여기서, 인자로 주는 boolean 값은 세션을 새로 만들건지를 결정하는 변수입니다. 이 값은 맨처음 접속했을 때는 HttpSession 이 없는 상태이므로 보통 true 로 줍니다. 이미 HttpSession 이 존재하는 경우에는 true 값을 넣었더라도 기존에 설정되어 있는 HttpSession 객체를 되돌려 줍니다. 이 때, 새로 만든 세션이지 여부를 확인하기 위해 isNew 메소드를 사용할 수 있습니다. 실제로, 세션은 30 분마다 자동으로 갱신되도록 되어 있습니다. 다시 말해서, 이미 세션이 만들어져서 관리되고 있다고 하더라도 30 분 동안 아무런 작업도 하지 않거나 다시 접속이 이루어지지 않으면 연결되어 있는 세션은 끊어지고, 이 때 자시 접속하게 되면 새로운 세션을 생성한다는 것입니다. 그리고, HttpSession 객체를 이용하여 생성 시간, 세션에서 사용될 데이터(위의 예에서는 SessionSnop.count)의 생성 및 관리할 수도 있습니다. 다음에 나오는 프로그램은 세션을 사용하여 쇼핑몰을 구현할 때, 세션을 사용하여 실제로 선택된 상품 정보를 유지 및 관리하는 예를 보여주는 HTTP 서블릿에 대한 자바 프로그램입니다.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ShoppingCartServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
        res.setContentType("text/html;charset=euc-kr");
```

```
PrintWriter out = res.getWriter();
HttpSession session = req.getSession(true);

 out.println("<HTML>");
 out.println("<BODY>");
 out.println("<HEAD>");
out.println("<TITLE>Park Home Shopping</TITLE>");
 out.println("</HEAD>");

 out.println("<BODY>");
 out.println("<H1>파크 홈쇼핑</H1><BR>");

String shoppingItems[] = { "Mouse", "Keyboard", "Monitor",
                           "Speaker", "CDROM", "Floppy Disk",
                           "Hard Disk", "Network Adapter", "Modem",
                           "VGA Adapter", "Printer", "Case" };
int shoppingChecks[] = { 10000, 10000, 200000,
                         30000, 70000, 15000,
                         100000, 50000, 40000,
                         120000, 350000, 50000 };

String shoppingStep = req.getParameter("shoppingStep");
if("checkout".equals(shoppingStep)) {
   String[] items = (String[])session.getValue("ShorppingCart.items");
   if(items == null) {
        out.println("<H2>구입한 물품이 없습니다.</H2>");
       session.putValue("ShorppingCart.check", new Integer(0));
    } else {
        int check=0;
        out.println("구입한 물품:<BR>");
        out.println("<UL>");
        for(int i=0;i<items.length;i++) {
            out.println("<LI>"+items[i]);
           for(int j=0;j<shoppingItems.length;j++) {
               if(shoppingItems[j].equals(items[i])) {
                   check += shoppingChecks[j];
               }
           }
        }
        out.println("</UL>");
       session.putValue("ShorppingCart.check", new Integer(check));
     }
   Integer i = (Integer)session.getValue("ShorppingCart.check");
    out.println("<H2>구입한 물품값은 "+i+"입니다</H2>");
} else {
    if("additem".equals(shoppingStep)) {
       String addedItems[] = req.getParameterValues("AddedItems");
       if(addedItems == null) {
            out.println("<H2>선택한 물품이 없습니다.</H2>");
        } else {
           String oldItems[] = (String[])session.getValue("ShorppingCart.items");
           String newItems[] = null;
           int addedCount = 0;
           if(oldItems != null) {
               newItems = new String[oldItems.length+addedItems.length];
               for(int i=0;i<oldItems.length;i++) {
                   newItems[addedCount++] = oldItems[i];
               }
            } else {
               newItems = new String[addedItems.length];
            }
```

```
                for(int i=0;i<addedItems.length;i++) {
                    newItems[addedCount++] = addedItems[i];
                }
                session.putValue("ShorppingCart.items", newItems);
            }
        }

        String[] items = (String[])session.getValue("ShorppingCart.items");
        if(items == null) {
            out.println("<B>구입한 물품이 없습니다.</B>");
        } else {
            out.println("구입한 물품:<BR>");
            out.println("<UL>");
            for(int i=0;i<items.length;i++) {
                out.println("<LI>"+items[i]);
            }
            out.println("</UL>");
        }

        out.println("<FORM
ACTION=\"http://park.konkuk.ac.kr:8080/servlet/ShoppingCartServlet\" METHOD=POST>");
        out.println("  <input type=hidden name=shoppingStep value=additem multiple>");
        out.println("  <select name=\"AddedItems\" size=4 multiple>");
        for(int i=0;i<shoppingItems.length;i++) {
            out.println("  <option value=\""+shoppingItems[i]+"\">"+shoppingItems[i]);
        }
        out.println("  </select>");
        out.println("  <input type=submit value=\"장바구니에 추가\">");
        out.println("</FORM>");

        out.println("<FORM
ACTION=\"http://park.konkuk.ac.kr:8080/servlet/ShoppingCartServlet\" METHOD=POST>");
        out.println("  <input type=hidden name=shoppingStep value=checkout multiple>");
        out.println("  <input type=submit value=\"계산하기\">");
        out.println("</FORM>");
    }
    out.println("</BODY>");
    out.println("</HTML>");

    out.close();
  }

  public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws IOException, ServletException {
    doGet(req, res);
  }
}

/*
 *
```




```
*/
```

<프로그램 16. ShoppingCartServlet.java>

위에서 제시된 HTTP 서블릿은 세션을 기본적으로 사용하고 있고, 현재 쇼핑몰의 상태를 유지하기 위해 FORM 속성 중 HIDDEN 속성을 사용하는 예에 대하여 자세히 보여주고 있습니다. 위의 예를 실행시키기 위해 기본적으로 URL 을 사용하므로, GET 방식을 지원해야 하며, 내부적으로 폼을 이용하여 POST 방식을 사용하므로 POST 방식을 지원해 주어야 합니다. 위의 자바 서블릿을 실행시킨후 "장바구니에 추가" 버튼을 누르면 다음과 같이 기존의 세션에 유지되어 있는 정보에 새로 추가된 물품에 대한 정보를 추가하게 됩니다. 다음에 나오는 그림은 장바구니에 추가 후의 결과를 보여주는 페이지입니다.

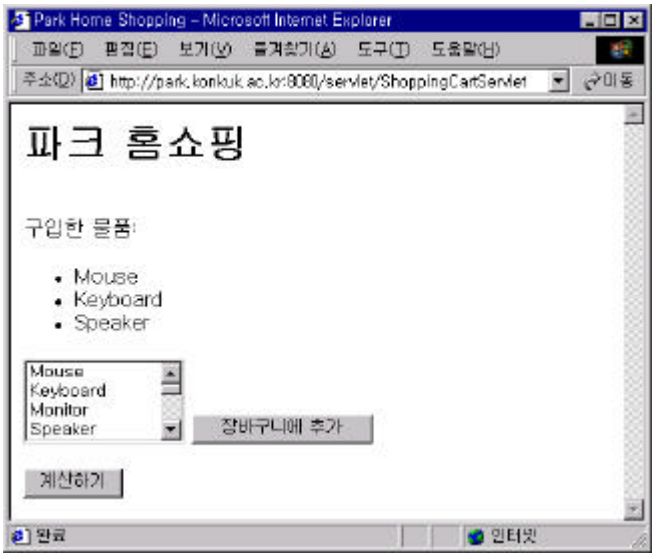


그림 8. 처음 선택한 물품을 추가한 결과 페이지

다음에 나오는 그림은 두 번째 선택한 물품을 추가한 후의 결과 페이지입니다.

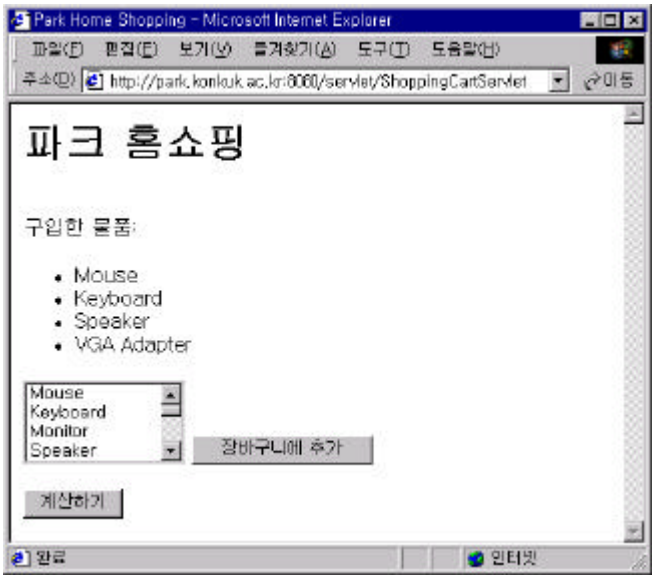


그림 9. 두 번째 선택한 물품을 추가한 결과 페이지

다음에 나오는 그림은 “계산하기” 버튼을 누른 후, 계산된 결과 페이지입니다.

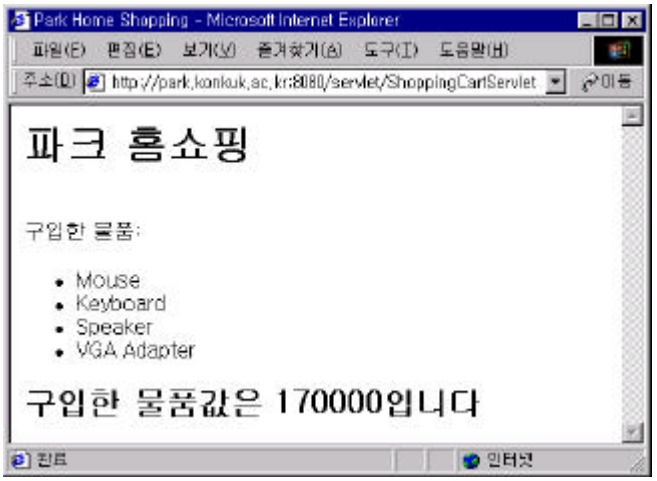


그림 10. 장바구니에 포함된 물품에 대한 계산한 후의 결과 페이지

1.3.6HttpUtils 클래스

HttpUtils 클래스는 HTTP 서블릿을 작성할 때 유용하게 사용할 수 있는 메소드들을 제공해 줍니다.

HttpUtils 클래스에서 제공해 주는 메소드들을 살펴보면, 다음과 같습니다.

- HttpUtils(): HttpUtils 객체를 생성합니다.
- static java.lang.StringBuffer getRequestURL(HttpServletRequest req): HttpServletRequest 객체 내에 있는 정보를 이용하여 클라이언트가 요청을 생성하기 위해 사용항 URL 을 얻습니다.
- static java.util.Hashtable parsePostData(int len, ServletInputStream in): 클라이언트가 HTTP POST 방식과 application/x-www-form-urlencoded MIME 타입으로 서버에게 보낸 HTML 로부터 데이터를 얻습니다.
- static java.util.Hashtable parseQueryString(java.lang.String s): 클라이언트가 서버에게 보낸 쿼리 문자열로부터 데이터를 얻습니다.

먼저, getRequestURL 메소드는 요청된 서버내의 URL 을 되돌려 줍니다. 다음으로, getQueryString 메소드는 GET 방식으로 전달되어 온 HTTP 데이터를 다루기 쉬운 형태로 바꿔주는 메소드이고, parsePostData 메소드는 POST 방식으로 전달된 데이터를 쉽게 다룰 수 있도록 해 주는 메소드입니다. 다시 말해서, getQueryString 메소드는 GET 방식으로 전달된 "이름=값" 쌍으로 구성된 데이터를 이름(key)과 값(value)을 저장하는 Hashtable 객체에 저장해 주고, parsePostData 메소드는 POST 방식으로 전달된 "이름=값" 쌍으로 구성된 데이터를 이름과 값을 저장하는 Hashtable 객체에 저장해 주는 것입니다.

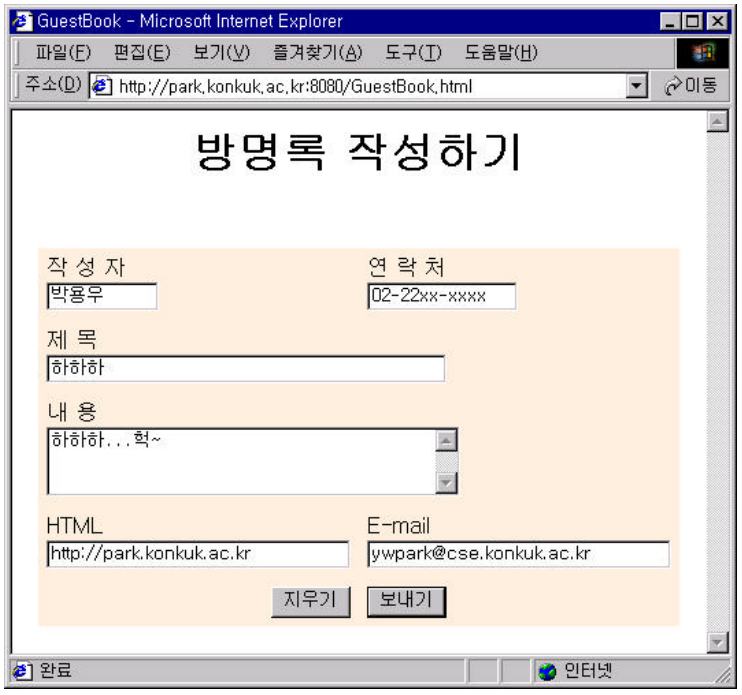
1.4자바 서블릿 예제 프로그램

1.4.1방명록 서블릿

자바 서블릿을 이용하여 방명록을 작성할 때, 다음과 같은 과정을 거치면 됩니다. 먼저, 방명록 요청을 처리할 자바 서블릿 프로그램을 작성하여 컴파일하고, HTML 페이지 내의 FORM 태그 부분에 CGI 프로그램을 연결하는 것과 똑같이 자바 서블릿 프로그램을 연결해 줍니다. 마지막으로, 자바 서버에서 이 자바 서블릿을 위한 초기화 정보를 얻을 수 있도록 하기 위해, servlet.properties 파일에 방명록 서블릿을 위한 정보를 등록해 줍니다. 다음에 나오는 세 개의 파일은 위의 세 가지 과정에서 생성되어야 하는 파일들입니다. 먼저, 방명록을 작성하기 위한 HTML 페이지 입니다.

```
<HTML>
<HEAD><TITLE>GuestBook</TITLE></HEAD>
<BODY>
<CENTER>
```

```
<H1>방명록 작성하기</H1>
<FORM action=http://park.konkuk.ac.kr:8080/servlet/guestbook method=post>
  <input type=hidden name=Function value=Write>
   <center>
  <table bgcolor=#faeddc cellspacing=0 cellpadding=6>
      <tr>
        <td>작 성 자<br><input type=text name=Author size=10></td>
        <td>연 락 처<br><input type=text name=Phone size=14></td>
      </tr>
     <tr><td colspan=2>제 목<br><input type=text name=Title size=40></td></tr></td></tr>
     <tr><td colspan=2>내 용<br><textarea name=Content rows=3 cols=40></textarea></td></tr>
     <tr>
       <td>HTML<br><input type=text name=HTML size=30></td>
        <td>E-mail<br><input type=text name=E_Mail size=30></td>
     </tr>
     <tr>
       <br>
       <td align=right><input type=reset  value=지우기></td>
      <td align=left> <input type=submit value=보내기></td>
     </tr>
   </table>
</FORM>
</BODY>
</HTML>

/*
 *
```




```
 */
```

<프로그램 17. GuestBook.html>

다음은 방명록 서블릿의 소스 코드입니다.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class GuestBookServlet extends HttpServlet implements SingleThreadModel {
    String guestBookPath;

   public void init(ServletConfig config)
   throws ServletException {
        super.init(config);

       guestBookPath = getInitParameter("guestBookPath");
       if(guestBookPath == null) {
           Enumeration initParams = getInitParameterNames();
            System.err.println("초기 파라메터: ");

            while(initParams.hasMoreElements()) {
                System.err.println(initParams.nextElement());
            }
            System.err.println("디렉토리를 설정하세요.");
           throw new UnavailableException (this, "디렉토리 설정이 잘못되었습니다!");
        }
    }

   public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
        res.setContentType("text/html; charset=euc-kr");
       PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Writing</TITLE>");
        out.println("</HEAD>");

        out.println("<BODY>");

        try {
           String functionStr = req.getParameterValues("Function")[0];

            if(functionStr.equalsIgnoreCase("Write")) {  // 방명록 기능 중 쓰기 기능 수행
              FileWriter guestBookFile = new FileWriter(guestBookPath, true);  // 방명록 파일
              PrintWriter toFile = new PrintWriter(guestBookFile);

                toFile.println("<BEGIN>");

                  TimeZone tz = new SimpleTimeZone(9*60*60*1000, "KST");  // 시간,날짜 자동
생성
              Date today = new Date();
                TimeZone.setDefault(tz);
              toFile.println("Date: " + today.toString());

                Enumeration values = req.getParameterNames(); // 방명록의 각 항목을 파일에 기
록
              while(values.hasMoreElements()) {
                 String name = (String)values.nextElement();
                 String value = req.getParameterValues(name)[0];
```

```
                    if(name.compareTo("submit") != 0) {
                        toFile.println(name + ": " + toKSC5601(value));
                    }
                }
                toFile.println("<END>");

                guestBookFile.close();

                out.println("<H1>");
                out.println("Success....");
                out.println("</H1>");
            } else {
                out.println("<H1>");
              out.println("Fail...." + functionStr + "..." + guestBookPath);
                out.println("</H1>");
             }
        } catch(IOException e) {  // 실패
             e.printStackTrace();
             out.println("<H1>");
             out.println("파일에 기록하는 중 문제가 발생했습니다.");
             out.println("</H1>");
         }
         out.println("</BODY>");
         out.println("</HTML>");

         out.close();
    }

   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws IOException, ServletException {
        doGet(req, res);
    }

   public static String toKSC5601( String str )
        throws UnsupportedEncodingException {
         if(str == null) {
             return(null);
         }
        return(new String(str.getBytes("8859_1"), "KSC5601"));
    }
}

/*
 *
```


```
 */
```

<프로그램 18. GuestBookServlet.java>

다음은 방명록 서블릿의 초기 설정을 위한 servlet.properties 파일입니다.

```
# $Id: servlets.properties,v 1.2 1999/04/02 02:04:22 duncan Exp $

# Define servlets here

# <servletname>.code=<servletclass>
# <servletname>.initparams=<name=value>,<name=value>

snoop.code=SnoopServlet
snoop.initparams=initarg1=foo,initarg2=bar
snoop2.code=SnoopGenericServlet
snoop2.initparams=initarg1=foo,initarg2=bar
hello.code=HelloGenericServlet
hello.initparams=name=Yongwoo's Park
hello2.code=HelloGenericServlet2
hello2.initparams=
jsp.code=com.sun.jsp.runtime.JspServlet

# guestbook servlet
# 방명록을 기록할 파일
# D:\\Java\\jsdk2.1\\webpages\\guestbook\\GuestBook.txt
guestbook.code=GuestBookServlet
guestbook.initparams=guestBookPath=D:\\Java\\jsdk2.1\\webpages\\guestbook\\GuestBook.txt
```

<프로그램 19. servlets.properties >

다음은 실제 기록된 방명록 파일입니다.

```
<BEGIN>
Date: Wed Oct 27 04:00:55 GMT+09:00 1999
HTML: http://park.konkuk.ac.kr
Author: 박용우
Title: 하하하
Phone: 02-22xx-xxxx
Content: 하하하...헉~
E_Mail: ywpark@cse.konkuk.ac.kr
Function: Write
<END>
```

<프로그램 20. D:\\Java\\jsdk2.1\\webpages\\guestbook\\GuestBook.txt >

1.4.2SMTP(Send Mail Transfer Protocol) 서블릿

일반적으로 E-mail 을 보내려 할 때, 메일 서버와 메일 클라이언트는 SMTP(Simple Mail Transfer Protocol)에서 제공하는 다음과 같은 규약을 이용하여 통신을 합니다.

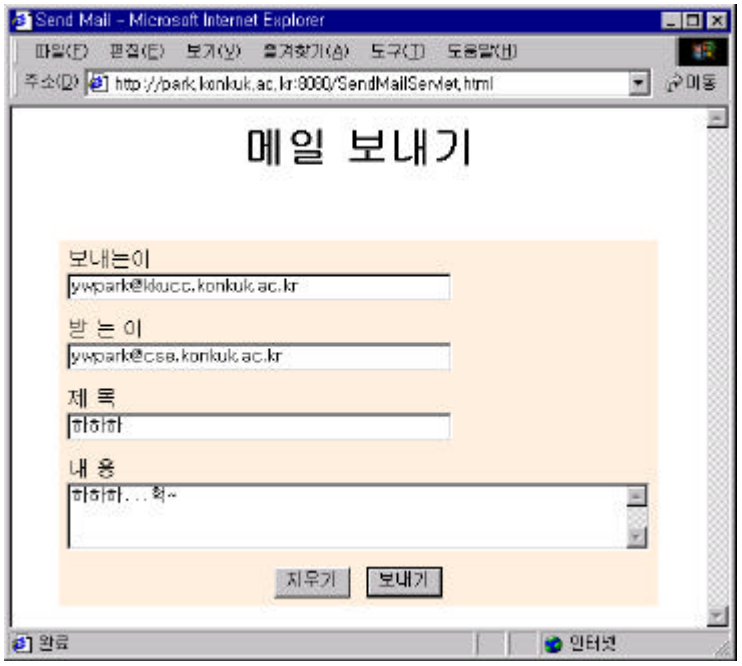
프로토콜	설 명
HELO localhost	SMTP 서버와 통신을 초기화 한다.
MAIL FROM: 보내는 사람 주소	보내는 사람의 전자우편 주소를 지정한다.
RCPT TO: 받는 사람 주소	받는 사람의 전자우편 주소를 지정한다.
DATA	내용이 송신될 것을 알린다.
SUBJECT: 제목	제목을 지정한다.
내용	내용을 보낸다.
.	내용의 끝을 나타낸다.
QUIT	연결을 종료한다.

먼저, SMTP 서버에게 클라이언트가 있는 호스트에 대한 정보를 알려주고, 송신자와 수신자에 대한 전자우편 주소를 알려주며, 데이터를 보냅니다. 이 때, 데이터는 제목(SUBJECT)과 내용으로 나뉘며, 내용의 끝은 마침표('.')만 있는 한 라인으로 나타냅니다.

자바 서블릿을 이용하여 메일을 작성할 때, 다음과 같은 과정을 거치면 됩니다. 먼저, 메일전송 요청을 처리할 자바 서블릿 프로그램을 작성하여 컴파일하고, HTML 페이지 내의 FORM 태그 부분에 CGI 프로그램을 연결하는 것과 똑같이 자바 서블릿 프로그램을 연결해 줍니다. 마지막으로, 자바 서버에서 이 자바 서블릿에 대한 초기화 정보를 얻을 수 있도록 하기 위해, servlet.properties 에 해당 자바 서블릿을 등록해 줍니다. 다음에 나오는 세 개의 파일은 위의 세 가지 과정에서 생성되어야 하는 파일들입니다. 먼저, 메일의 내용을 작성하기 위한 HTML 페이지 입니다.

```
<HTML>
<HEAD><TITLE>Send Mail</TITLE></HEAD>
<BODY>
<CENTER>
<H1>메일 보내기</H1>
<FORM action=http://park.konkuk.ac.kr:8080/servlet/sendmail method=post>
  <table bgcolor=#faeddc cellspacing=0 cellpadding=6>
     <tr>
       <td>보내는이<br><input type=text name=mailFrom size=40></td>
       <td>받 는 이<br><input type=text name=rtcpTo size=40></td>
     </tr>
    <tr><td colspan=2>제 목<br><input type=text name=subject size=40></td></tr></td></tr></tr>
    <tr><td colspan=2>내 용<br><textarea name=body rows=3 cols=60></textarea></td></tr>
     <tr>
       <br>
       <td align=right><input type=reset  value=지우기></td>
      <td align=left> <input type=submit value=보내기></td>
     </tr>
  </table>
</FORM>
</BODY>
</HTML>
```

```
/*
 *
```




```
 */
```

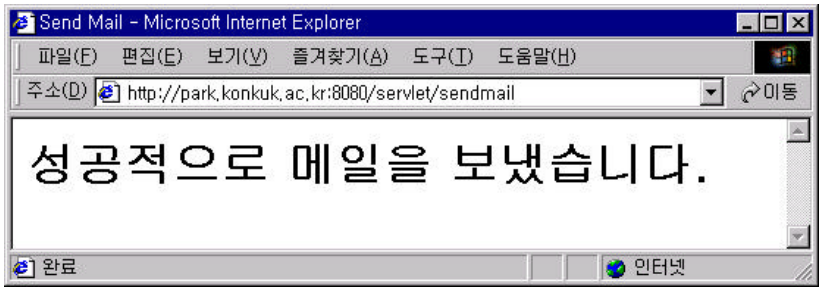
<프로그램 21. SendMailServlet.html>

다음은 SMTP **서블릿의 소스 코드입니다.**

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SendMailServlet extends HttpServlet implements SingleThreadModel {
    String smtpServer;
   PrintStream ps = null;
   BufferedReader dis = null;

   public void init(ServletConfig config) throws ServletException {
        super.init(config);

       smtpServer = getInitParameter("smtpServer");
       if(smtpServer == null) {
           Enumeration initParams = getInitParameterNames();
            System.err.println("초기 파라메터: ");

            while(initParams.hasMoreElements()) {
                System.err.println(initParams.nextElement());
            }
            System.err.println("SMTP 서버 정보가 있어야 합니다.");
```

```
            throw new UnavailableException (this, "Not given a SMTP server to send a mail!");
        }
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        res.setContentType("text/html; charset=euc-kr");
        PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Send Mail</TITLE>");
        out.println("</HEAD>");

        out.println("<BODY>");

        String END_OF_DATA = "\r\n.\r\n";
        Socket smtp = null;

        try {
            smtp = new Socket(smtpServer, 25);  // SMTP 서버의 25번 포트에 접속한다.
            OutputStream os = smtp.getOutputStream();
            ps = new PrintStream(os);
            InputStream is = smtp.getInputStream();
            dis = new BufferedReader(new InputStreamReader(is));
        } catch(IOException e) {
            System.out.println("Error connecting: " + e);
        }

        String   mailFrom = toKSC5601(req.getParameter("mailFrom"));
        String      rcptTo = toKSC5601(req.getParameter("rtcpTo"));
        String     subject = toKSC5601(req.getParameter("subject"));
        String        body = toKSC5601(req.getParameter("body"));

        out.println("<H1>");
    if(mailFrom == null || rcptTo == null|| subject == null || body == null) {
            out.println("mailFrom or rtcpTo or subject or body is empty...");
        } else {
            try {
                String localHost = InetAddress.getLocalHost().getHostName();  // Local 호스트의
이름을 얻는다.

                send("HELO " + localHost); receive();
                send("MAIL FROM: " + mailFrom); receive();
                send("RCPT TO: " + rcptTo); receive();
                send("DATA"); receive();
                send("SUBJECT: " + subject); receive();
                send(body+END_OF_DATA); receive();
                send("QUIT"); receive();
                smtp.close();

                out.println("성공적으로 메일을 보냈습니다.");
            } catch(IOException e) {
                out.println("메일을 보내는 중에 에러가 발생했습니다.");
            }
        }
        out.println("</H1>");

        out.println("</BODY>");
        out.println("</HTML>");
```

```
        out.close();
    }

    void send(String s) throws IOException {
        ps.println(s);
        ps.flush();

       System.out.println("Java request: " + s);
    }

   void receive() throws IOException {
       String readStr = dis.readLine();

       System.out.println("SMTP response: " + readStr);
    }

   public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws IOException, ServletException {
       doGet(req, res);
    }

   public static String toKSC5601( String str )
       throws UnsupportedEncodingException {
       if(str == null) {
           return(null);
        }
       return(new String(str.getBytes("8859_1"), "KSC5601"));
    }
}

/*
 *
```




```
 */
```

<프로그램 22. SendMailServlet.java>

다음은 SMTP **서블릿의 초기 설정을 위한** servlet.properties **파일입니다**.

```
# $Id: servlets.properties,v 1.2 1999/04/02 02:04:22 duncan Exp $

# Define servlets here

# <servletname>.code=<servletclass>
# <servletname>.initparams=<name=value>,<name=value>
```

```
snoop.code=SnoopServlet
snoop.initparams=initarg1=foo,initarg2=bar
snoop2.code=SnoopGenericServlet
snoop2.initparams=initarg1=foo,initarg2=bar
hello.code=HelloGenericServlet
hello.initparams=name=Yongwoo's Park
hello2.code=HelloGenericServlet2
hello2.initparams=
jsp.code=com.sun.jsp.runtime.JspServlet

# guestbook servlet
# 방명록을 기록할 파일
# D:\\Java\\jsdk2.1\\webpages\\guestbook\\GuestBook.txt
guestbook.code=GuestBookServlet
guestbook.initparams=guestBookPath=D:\\Java\\jsdk2.1\\webpages\\guestbook\\GuestBook.txt

# sendmail servlet
sendmail.code=SendMailServlet
sendmail.initparams=smtpServer=cse.konkuk.ac.kr
```

<프로그램 23. servlets.properties >

다음은 SMTP 서블릿을 실행한 결과, 자바 서버에 기록된 내용입니다.

```
JSDK WebServer Version 2.1
Loaded configuration from file:D:\Java\jsdk2.1/default.cfg
endpoint created: park.konkuk.ac.kr/203.252.134.126:8080
com.sun.web.core.InvokerServlet: init
SendMailServlet: init
Java request: HELO park
SMTP response: 220 cse.konkuk.ac.kr ESMTP Sendmail 8.9.2/8.9.1; Wed, 27 Oct 1999
 04:21:32 +0900 (KST)
Java request: MAIL FROM: ywpark@kkucc.konkuk.ac.kr
SMTP response: 250 cse.konkuk.ac.kr Hello park.konkuk.ac.kr [203.252.134.126], p
leased to meet you
Java request: RCPT TO: ywpark@cse.konkuk.ac.kr
SMTP response: 250 ywpark@kkucc.konkuk.ac.kr... Sender ok
Java request: DATA
SMTP response: 250 ywpark@cse.konkuk.ac.kr... Recipient ok
Java request: SUBJECT: 하하하
SMTP response: 354 Enter mail, end with "." on a line by itself
Java request: 하하하...헉~
.

SMTP response: 250 EAA11863 Message accepted for delivery
Java request: QUIT
SMTP response: 221 cse.konkuk.ac.kr closing connection
```

<프로그램 24. 자바 서버 로그 >