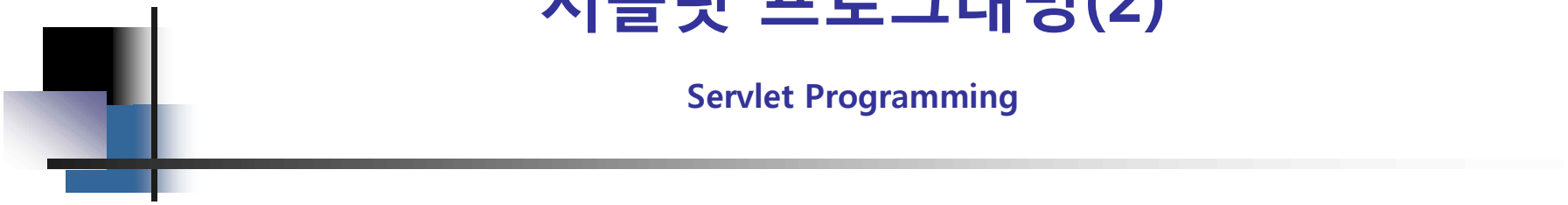


# 서블릿 프로그래밍(2)

## Servlet Programming

남 광 우

# 파일 업로드의 구현

- 파일 첨부 Request의 형태
  - MultiPart 형태로 encoding 되어 서블릿으로 전송됨
  - Multipart의 내용
    - Content Type
      - req.getContentType 메소드를 이용하여 내용에 해당하는 데이터의 MIME 형식을 얻을 수 있음. 이 때 MIME 형식이 "multipart/form-data" 일 경우에는 각 항목을 구분하기 위한 구분자를 얻을 수 있음
    - Seperator : ----------
      - 폼의 항목을 구분하기 위한 구분줄
    - Content-Disposition
      - 폼의 각 항목에 대한 설명줄. 이 라인을 분석하여 항목의 이름을 얻을 수 있음. 만약, 항목이 파일일 경우에는 파일의 이름(filename="...")도 얻을 수 있음.
    - 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 내용
    - 종료 Seperator : ----------
      - 마지막 종료를 나타냄

# 파일 업로드의 구현

- 파일 첨부 upload처리 : O'Reilly MultipartRequest 이용
  - O'Reilly MultipartRequest 이용
  - MultipartRequest 클래스
    - O'Reilly의 'Java Servlet Programming' 책의 예제를 확장
    - JSP/Servlet에서 현재 File Upload 처리를 위해 가장 많이 사용
    - 현재 2002/11/01 버전 배포
  - MultipartRequest 클래스의 다운로드 및 Deployment
    - 다운로드
      - http://www.servlets.com/cos/index.html 또는 WebDB 강의 게시판
      - 매뉴얼 : http://www.servlets.com/cos/javadoc/com/oreilly/servlet/MultipartRequest.html 또는 http://aboutjsp.com/lec/multipart.jsp 참조
    - Deployment
      - 압축 푼뒤, classes 디렉토리 아래를 tomcat의 classes 디렉토리에 통째로 복사
      - 또는, lib/cos.jar를 tomcat의 lib 디렉토리에 복사

# 파일 업로드의 구현

- MultipartRequest 클래스의 주요 생성자
  - MultipartRequest( HttpServletRequest request, String saveDirectory)
    - 특정 request를 다루기 위한 MultipartRequest 생성, 1M(default) 한계의 파일을 받아 saveDirectory에 저장
  - MultipartRequest(HttpServletRequest request, String saveDirectory, int maxPostSize)
    - maxPostSize 크기의 한계를 갖는 파일을 받아 saveDirectory에 저장
  - MultipartRequest(HttpServletRequest request, String saveDirectory, int maxPostSize, FileRenamePolicy policy)
    - FileRenamePolicy에 따라 일괄적으로 받은 파일 이름을 rename
  - MultipartRequest( request, saveDirectory, maxPostSize, String encoding, FileRenamePolicy policy)
    - maxPostSize 크기의 한계를 갖는 파일을 받아 saveDirectory에 저장
    - Encoding에 따라 파일 이름 encoding

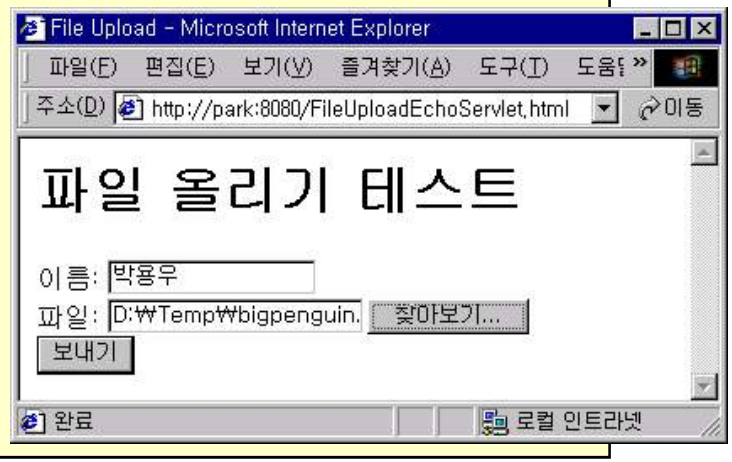
# 파일 업로드의 구현

- MultipartRequest 클래스의 주요 메소드
  - Enumeration  getParameterNames()
    - MultipartRequest의 각 파트를 구성하는 Parameter 이름들을 반환
  - String[] getParameterValues(java.lang.String name)
    - 지정된 name의 파라미터 값들을 String[] 형태로 반환
  - String getParameter(java.lang.String name)
    - 특정 이름의 파라미터 값을 반환
  - String getContentType(java.lang.String name)
    - 특정 파일의 content type을 반환
  - Enumeration getFileNames()
    - 업로드되 파일들의 이름을 String의 Enumeration으로 반환
  - File getFile(java.lang.String name)
    - 업로드 되어 저장된 파일의 객체를 반환
  - String getFilesystemName(java.lang.String name)
    - 특정 파일의 파일 이름을 반환(getOriginalFileName:원래파일이름 반환)

# 파일 업로드의 구현

- 파일 올리기
  - Form을 이용하여 파일을 전송
    - Input Type = FILE 로 설정
    - enctype=multipart/form-data
      - 업로드 할 파일의 형식을 지정하기 위해 사용
      - Servlet에서 multipart 데이터 처리를 위한 구현 필요

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

<BODY>
<H1>파일 올리기 테스트</H1>
<P>
<FORM method="POST"
    action="http://localhost: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>
```



# 파일 업로드의 구현

- 파일 업로드 예제 1
  - MultipartRequest 이용하여 1개의 파일 업로드

```
String savePath="/usr/local/tomcat/webapps/ROOT/test/upload"; // 저장할 디렉토리 (절대경로)

int sizeLimit = 5 * 1024 * 1024 ; // 5메가까지 제한 넘어서면 예외발생

try{

    MultipartRequest multi=new MultipartRequest(request, savePath, sizeLimit, new DefaultFileRenamePolicy());
    Enumeration formNames=multi.getFileNames();  // 폼의 이름 반환
    String formName=(String)formNames.nextElement(); // 자료가 많을 경우엔 while 문을 사용
    String fileName=multi.getFilesystemName(formName); // 파일의 이름 얻기

    if(fileName == null) {   // 파일이 업로드 되지 않았을때
        out.print("파일 업로드 되지 않았음");
    } else {  // 파일이 업로드 되었을때
        fileName=new String(fileName.getBytes("8859_1"),"euc-kr"); // 한글인코딩 - 브라우져에 출력
        out.print("User Name : " + multi.getParameter("name") + "<BR>");
        out.print("Form Name : " + formName + "<BR>");
        out.print("File Name  : " + fileName);
    } // end if

} catch(Exception e) {
    out.print("예외 상황 발생..! ");
}
```

# 파일 업로드의 구현

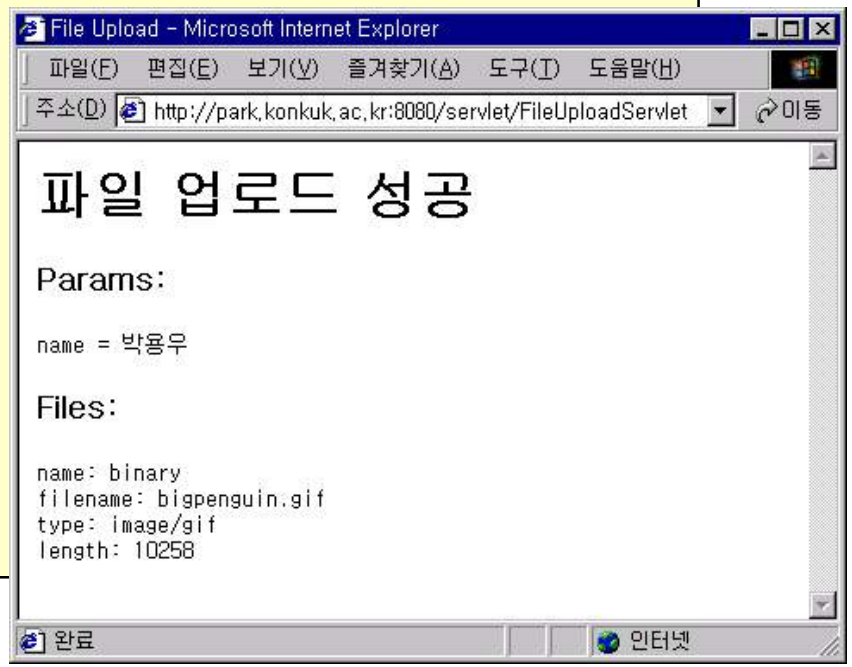
- HTML Form : 파일 업로드 서블릿 구현
  - MultipartRequest 이용하여 여러 개의 파일 업로드

```
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();

   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>");
```

# 파일 업로드의 구현

- HTML Form : 파일 올리기 구현

```
        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>");
    }
}
```



# Image 생성 및 전송

- 동적으로 이미지 생성하기
  - 동적으로 GIF/JPEG 파일을 생성하여 서블릿을 통해 전송/저장
  - 이미지 생성 순서
    - BufferedImage 클래스를 사용하여 원하는 크기의 버퍼 이미지를 생성
    - 생성된 버퍼 이미지와 관련된 Graphics 객체를 구함
    - Graphics 객체를 사용하여 원하는 이미지를 그림
    - ImageIO 클래스를 사용하여 버퍼 이미지를 GIF 내지 JPEG 이미지 파일로 변환

```
int width = 200; // 이미지 세로 크기
   int height = 150; // 이미지 가로 크기

   BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);

   Graphics g = image.getGraphics();
   // 또는
   // Graphics2D g = image.createGraphics();

   ... // Graphics 객체를 사용하여 원하는 이미지를 그린다.

   // 파일을 생성한다.
   File outFile = new File("c:₩temp.jpg");
   ImageIO.write(image, "jpg", outFile);
```

# Image 생성 및 전송

- 이미지 생성과 전송
  - 방법
    - 앞의 예제에서 파일 대신 ServletResponse의 OutputStream에 Write

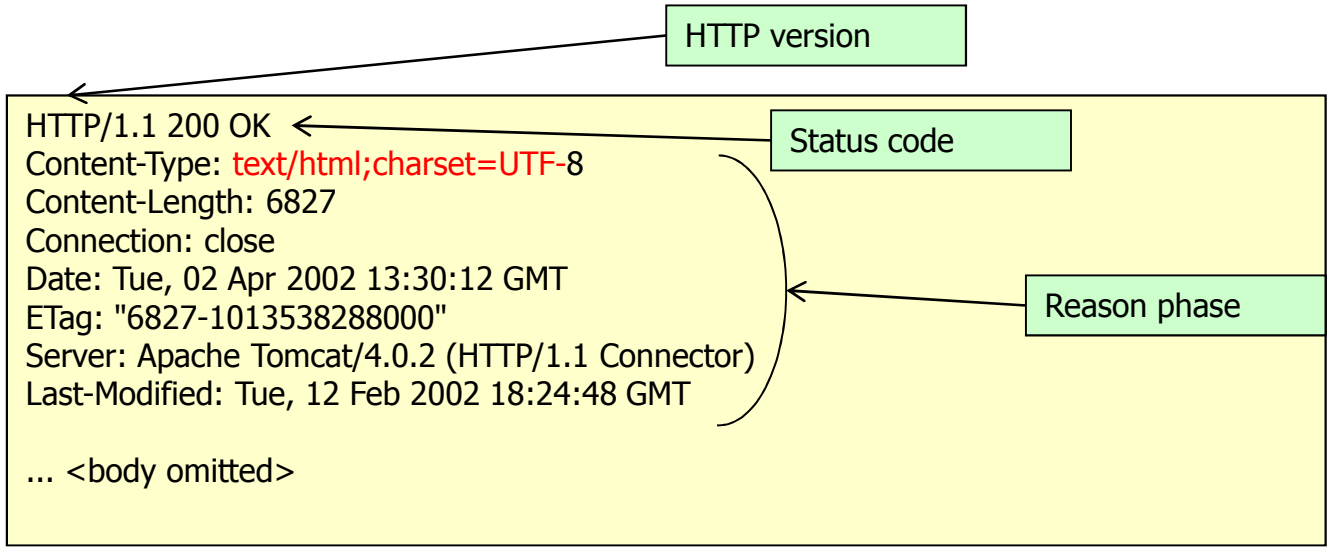
```
public class ImageCreator extends HttpServlet {
        public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                        throws IOException, ServletException {

                        int width = 200;
                        int height = 100;

                        response.setContentType("image/jpeg");
                        ServletOutputStream outStream = response.getOutputStream();

                        BufferedImage image = BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);
                        Graphics g = image.getGraphics();

                        g.drawString( "Hello World", 10, 50 );

                        ImageIO.write(image, "jpg", outStream);

                        outStream.close();
        }
  }
```

HTML에서의 호출

```
<img src="http://some.com/../servlet/ImageCreator" width="200" height="100">
```

# Sending HTML Information

- HTTP Response의 구조
  - 기본 형식
    - <HTTP-version> <status-code> <reason-phrase>
  - HTTP-version
    - HTTP/1.0 또는 HTTP/1.1. 형태로 표현
  - 예 : response



# Sending HTML Information

- HTTP Response의 구조
  - status-code
    - 100–199 : informational
      - 100 Continue : 접수되었고, 처리 중. 클라이언트는 이 응답을 무시함
      - 101 Switching Protocols : 다른 프로토콜 또는 업그레이드 버전으로 변환함
    - 200–299 : successful page access
      - 200 OK : 요청이 성공적으로 처리됨
      - 201 Created : 요청에 응답하기 위한 자원이 생성되었음
      - 202 Non-Authoritative Information : 비 인증 정보
      - 204 No Content : 내용이 없음
    - 300–399 : redirect the request
      - 300 Multiple Choices : 복수 선택, 요청된 문서가 여러 곳에 있음
      - 301 Moved Permanently : 영구 이동
      - 307 Temporary Redirect : 임시적인 이동
    - 400–499 : client errors
      - 400 Bad Request : 잘못된 요구
      - 401 Unauthorized : 인증되지 않았음
      - 404 Not Found : 찾을 수 없음
      - 414 Request URI Too Long : GET URI가 너무 김
    - 500–599 : server errors
      - 500 Internal Server Error : 서버 내부 에러
      - 501 Not Implmented : 구현되지 않았음

# Sending HTML Information

- HTTP Response의 구조
  - Content-type
    - 현재 보내는 데이터의 종류를 알리는 표시
    - 예 : html 파일을 보낸다면  text/html

| 파 일 | 데이터 종류(Content-type) |
|---|---|
| HTML 문서 | text/html |
| 텍스트 화일 | text/plain |
| GIF 포맷 | image/gif |
| JPEG 포맷 | image/jpeg |
| 포스트 스크립트 | application/postscript |
| ZIP | application/x-zip-compressed |
| MPEG | video/mpeg |
| AVI | video/x-msvideo |

# Sending HTML Information

- 일반적인 HTML 보내기
  - 일반적인 문서의 Content-type은 text/html 형태의 문서로 구성됨
  - Content-type을 text/html로 설정

- 예 : hello again!!

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

public class HelloWorld extends HttpServlet {

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

    out.println("<HTML>");
    out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<BIG>Hello World</BIG>");
    out.println("</BODY></HTML>");
  }
}
```

Content Type 설정

# Sending HTML Information

- 일반적인 HTML 보내기 : 한글 처리 문제
  - 한글/언어의 설정
    - Content-type 에서 charset의 설정을 통해 한글 또는 다른 언어를 설정 가능

| KSC5601, KSC5636 | EUC_KR(euc-kr) | UNICODE |
|---|---|---|
| ▪ **KSC5601**<br>한글 완성형 표준(한글 2350자 표현)<br>2350자 이외의 문자 깨짐<br>한국공업표준 정보처리분야(C)의 5601 번 표준안<br>▪ **KSC5636**<br>영문자에 대한 표준<br>한국공업표준 정보처리분야(C)의 5636 번 표준안<br>기존 ASCII Code에서 역슬래시를 ₩(원) 표시 대체 | Bell Laboratories에서 확장된 유닉스 코드 제안<br>유닉스 상에서 영문자 이외의 문자를 지원하는 방법<br>EUC-KR(Exetend UNIX Code)<br><br>**▪한국어를 표현하는 방법**<br>영문: KSC5636 사용<br>한글: KSC5601 사용(KS-C-5601-1987)<br><br>**▪EUC_KR(euc-kr)**<br>EUC_KR = KSC5601 + KSC5636 | 국제 표준 다국어 표현 표준<br>Java에서 default로 사용<br><br>**▪ UTF-16**<br>유니코드 문자 인코딩 방식의 하나. 주로 사용되는 기본 다국어 평면에 속하는 문자들은 그대로 16비트 값으로 인코딩이 되고 그 이상의 문자는 특별히 정해진 방식으로 32비트로 인코딩<br><br>**▪ UTF-8**<br>유니코드를 위한 가변 길이 문자 인코딩 방식 중 하나로, 켄 톰프슨과 롭 파이크가 만듬<br>UTF-8 인코딩은 유니코드 한 문자를 나타내기 위해 1바이트에서 4바이트까지를 사용한다 |

Windows 한글 표현

웹 브라우저 한글 표현

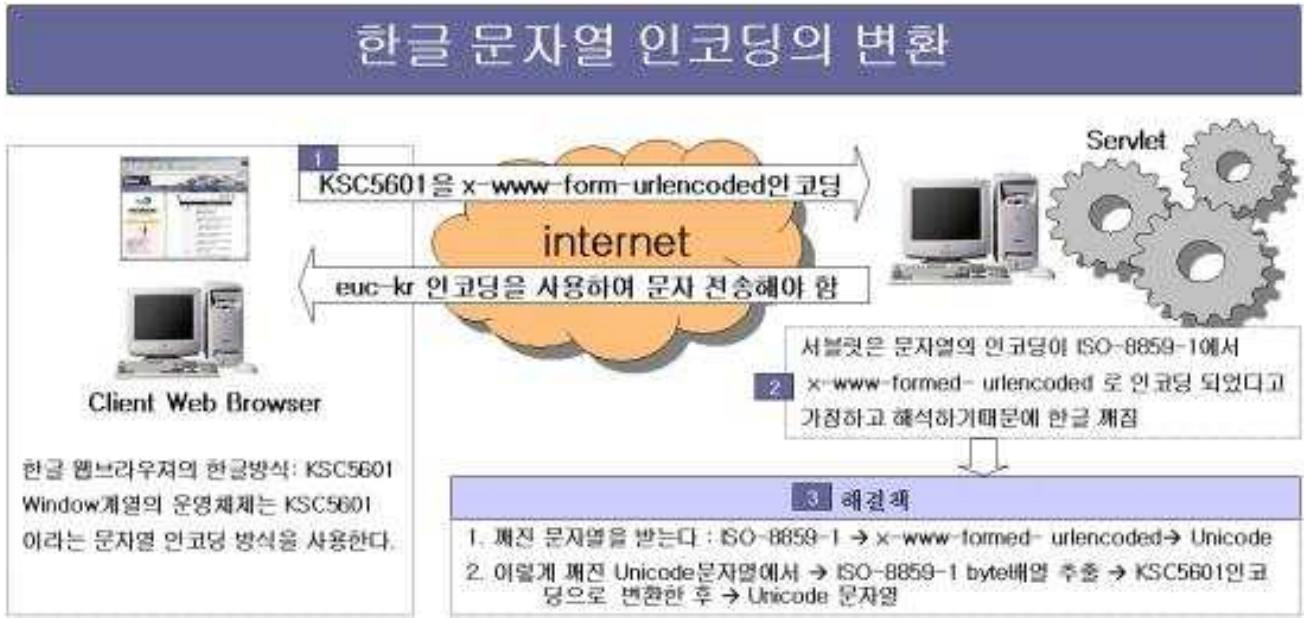
ISO-8859-1 : 서유럽 언어(라틴어) 표현 표준

MS949 : Windows 한글 표현

Java 한글 표현(최근 웹 브라우저)

# Sending HTML Information

- 일반적인 HTML 보내기 : 한글 처리 문제
  - 문제 원인
    - 웹 브라우저 – 서블릿 – 시스템(OS) 간의 한글 인코딩 방식의 차이



# Sending HTML Information

- 일반적인 HTML 보내기 : 한글 처리 문제
  - 웹 브라우저 -> 서블릿 전송
    - 웹 브라우저는 KSC5601 ->x-www-form-urlencoded 형식으로 변환하여 전송
      - 문제 : 서블릿은 라틴어 표준 인코딩 방식인 ISO-8859-1방식에서 웹 인코딩되었다고 간주하여 한글을 받음. 실제는 KSC5601로 인코딩 되어 있기 때문에 한글 깨짐.
      - 해결방안 : ISO-8859-1을 KSC5601로 변환하여 처리
  - 서블릿 -> 웹 브라우저 전송
    - 서블릿은 유니코드에서 자동으로 ISO-8859-1형식으로 변환해서 전송
      - 문제 : ISO-8859-1 은 라틴어 표준이므로, 한글이 깨짐
      - 해결방안 : 함께 사용될 수 있는 euc-kr로 변환하여 전송
  - Windows 파일 -> 서블릿 읽기
    - 윈도우는 KSC5601의 확장  완성으로 한글 표현(MS949)
      - 문제 : Java가 UNICODE를 기본으로 사용하므로 내부 구현에서 문제발생
      - 해결 방안 : 파일에서 읽을 때 UNICODE로 변환

# Sending HTML Information

- 일반적인 HTML 보내기 : 한글 처리 문제
  - 프로그램을 이용하여 해결하는 방법

```
package org.jabook.util;
public class HangulEncoder{
    public static String to8859_1(String ko){
        if(ko==null){
            return null;
        }
        try{
            return new String(ko.getBytes("KSC5601"),"8859_1");
        }catch(Exception e){return ko;}
    }
    public static String toKSC5601(String en){
        if(en==null){
            return null;
        }
        try{
            return new String(en.getBytes("8859_1"),"KSC5601");
        }catch(Exception e){return en;}
    }
}
```

# Sending HTML Information

- 일반적인 HTML 보내기 : 한글 처리 문제
  - 서블릿의 setCharacterEncoding() 사용하는 방법
    - Request로 들어오는 한글을 서블릿 내부에서 한글로 변환하여 처리하기 위해 request 파라미터 사용전에 setCharacterEncoding() 을 호출하는 방법

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        request.setCharacterEncoding("euc-kr");
        response.setContentType("text/html; charset=euc-kr");

        PrintWriter pw = new PrintWriter(response.getWriter());

        ...
    }
```

# 정보의 획득 – 서버 정보

- Servlet 서버의 정보 획득 메소드
  - servletRequest로 부터 획득
    - public String ServletRequest.getServerName()
      - 서버의 이름을 반환
    - public int ServletRequest.getServerPort()
      - 서블릿 포트 번호 반환
  - servletContext로 부터 획득
    - public String ServletContext.getServerInfo()
      - 서버에 대한 정보 반환
      - 서블릿 서버 이름과 버전
    - public Object ServletContext.getAttribute(String name)
      - 서버의 특정속성 설정을 반환

# 정보의 획득 – 서버 정보

- 예 : 서버정보 획득 서블릿

```
public class ServerSnoop extends GenericServlet {

  public void service(ServletRequest req, ServletResponse res)
                        throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();

    out.println("req.getServerName(): " + req.getServerName());
    out.println("req.getServerPort(): " + req.getServerPort());
    out.println("getServletContext().getServerInfo(): " +
                getServletContext().getServerInfo());
    out.println("getServerInfo() name: " +
                getServerInfoName(getServletContext().getServerInfo()));
    out.println("getServerInfo() version: " +
                getServerInfoVersion(getServletContext().getServerInfo()));
    out.println("getServletContext().getAttribute(₩"attribute₩"): " +
                getServletContext().getAttribute("attribute"));
  }

  private String getServerInfoName(String serverInfo) {
    int slash = serverInfo.indexOf('/');
    if (slash == -1) return serverInfo;
    else return serverInfo.substring(0, slash);
  }

  private String getServerInfoVersion(String serverInfo) {
    int slash = serverInfo.indexOf('/');
    if (slash == -1) return null;
    else return serverInfo.substring(slash + 1);
  }
}
```

# 정보의 획득 – 클라이언트 정보

- 클라이언트 정보 획득 메소드
  - ServletRequest로 부터 획득
    - public String ServletRequest.getRemoteAddr()
      - 클라이언트의 IP 어드레스를 반환
    - public String ServletRequest.getRemoteHost()
      - 클라이언트의 Domain Name을 반환
    - public String ServletRequest.getRemotePort()
      - 클라이언트의 IP Source Port를 반환
    - IP 주소 또는 Domain Name의 java Inet 변환
      - InetAddress remoteInetAddress =
        InetAddress.getByName(req.getRemoteAddr());

# 정보의 획득 – 클라이언트 정보

- 예 : 특정 Domain Name의 클라이언트 접근 거부

```
public class ExportRestriction extends HttpServlet {

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

    // ...Some introductory HTML...

    // Get the client's hostname
    String remoteHost = req.getRemoteHost();

    // See if the client is allowed
    if (! isHostAllowed(remoteHost)) {
      out.println("Access <BLINK>denied</BLINK>");  // filter out the blink!
    }
    else {
      out.println("Access granted");
      // Display download links, etc...
    }
  }

  private boolean isHostAllowed(String host) {
    return (host.endsWith(".com") ||            host.endsWith(".edu") ||
            host.endsWith(".net") ||            host.endsWith(".org") ||
            host.endsWith(".gov") ||            host.endsWith(".mil") ||
            host.endsWith(".us") ||             host.endsWith(".ca") ||
            (host.indexOf('.') == -1));  // no domain, assume OK
  }
}
```

# 정보의 획득 – 클라이언트 정보

- Local 정보 획득 메소드
  - ServletRequest로 부터 획득
    - public String getLocalName()
      - 현재 컴퓨터의 Local Name을 반환
    - public String getLocalAddr()
      - 현재 컴퓨터의 Local IP Address를 반환
    - public String getLocalPort()
      - 현재 컴퓨터의 서비스 port를 반환

# 서블릿 초기화와 정보 공유 –ServletConfig

- ServletConfig의 이용

DD(web.xml) 파일에서:

```
<servlet>
    <servlet-name>BeerParamTests</servlet-name>
    <servlet-class>TestInitParams</servlet-class>

    <init-param>
      <param-name>adminEmail</param-name>
      <param-value>likewecare@wickedlysmart.com</param-value>
    </init-param>

</servlet>
```

DD의 <servlet> 항목 안에 <param-name>과 <param-value>를 작성하면 됩니다.

서블릿 코드에서:

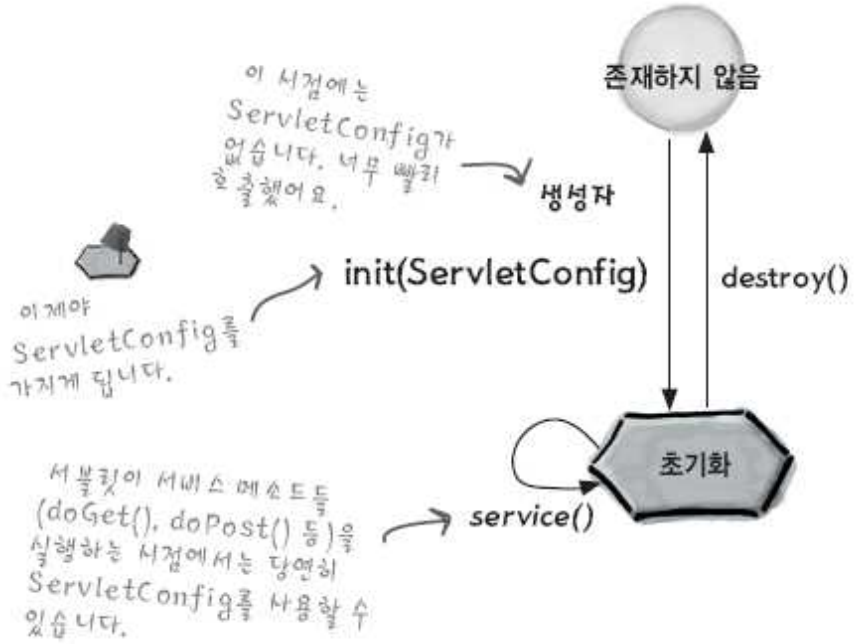
```
out.println(getServletConfig().getInitParameter("adminEmail"));
```

모든 서블릿에는 상속받은 getServletConfig()가 있습니다.

getServletConfig() 메소드의 리턴값은 ...음...뭐였더라... 잠시만요... 커닝 좀 하고... 아 예... ServletConfig입니다. ServletConfig에는 getInitParameter() 메소드가 있습니다.

# 서블릿 초기화와 정보 공유 –ServletConfig

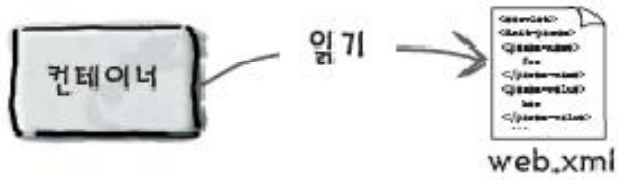
- ServletConfig의 이용
  - 서블릿 초기화가 된 다음에 초기화 파라미터를 사용할 수 있다
  - 컨테이너는 DD에서 서블릿 초기화 파라미터를 읽어, 이 정보를 ServletConfig로 넘겨줌
  - 그 다음 ServletConfig를 서블릿의 init() 메소드에 제공



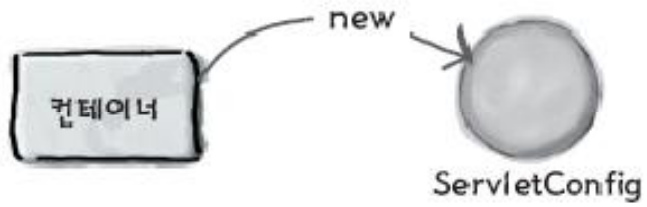
# 서블릿 초기화와 정보 공유 –ServletConfig

- ServletConfig의 동작 단계
  - 서블릿 초기화할 때 단 한번만 서블릿 초기화 파라미터를 읽음

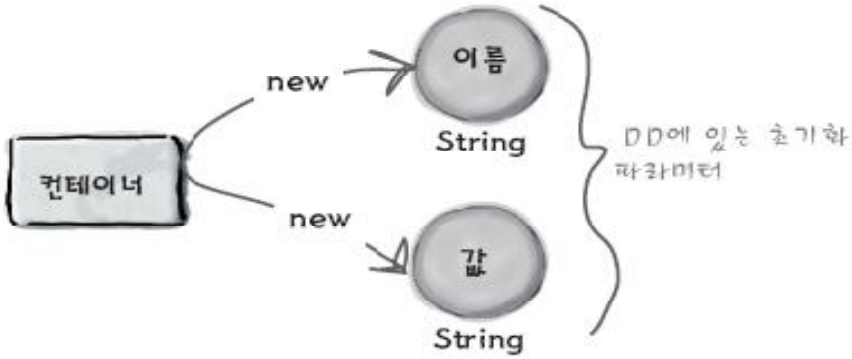
① 컨테이너는 배포 서술자(DD)를 읽습니다. 물론 초기화 파라미터((init-param))도 읽겠지요.



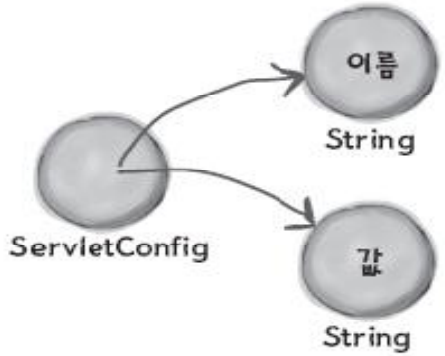

② 컨테이너는 새로운 ServletConfig 인스턴스를 만듭니다(서블릿당 하나씩 만듭니다).




③ 컨테이너는 초기화 파라미터에 있는 값들을 이름/값의 쌍의 형식으로 읽어들입니다. 여기서는 하나의 쌍만 있다고 가정해봅시다.

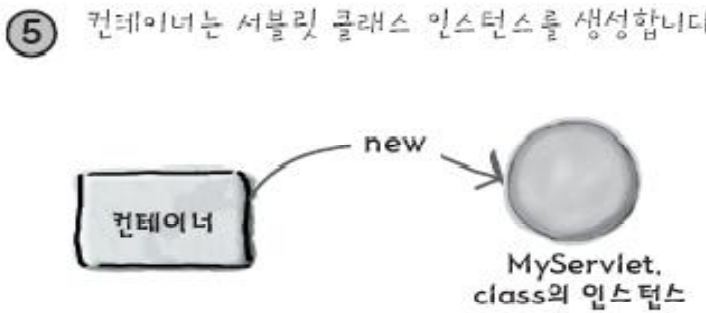


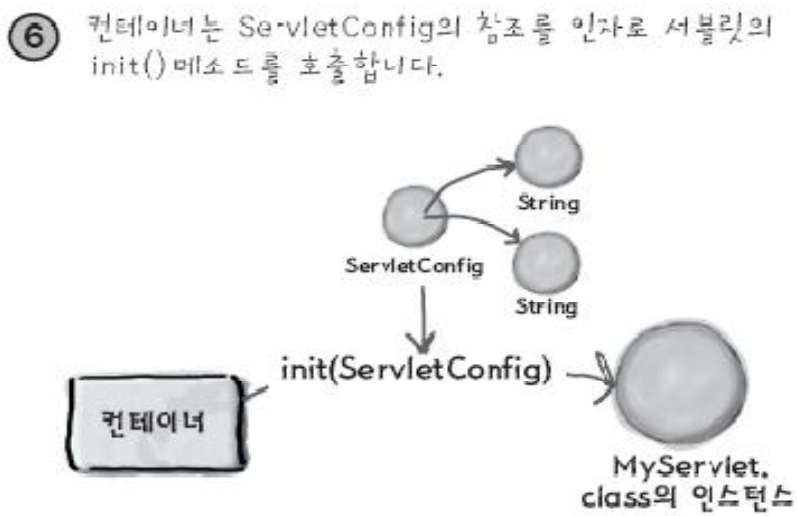

④ 컨테이너는 ServletConfig 객체에 이름/값으로 된 초기화 파라미터를 설정합니다.



# 서블릿 초기화와 정보 공유 –ServletConfig

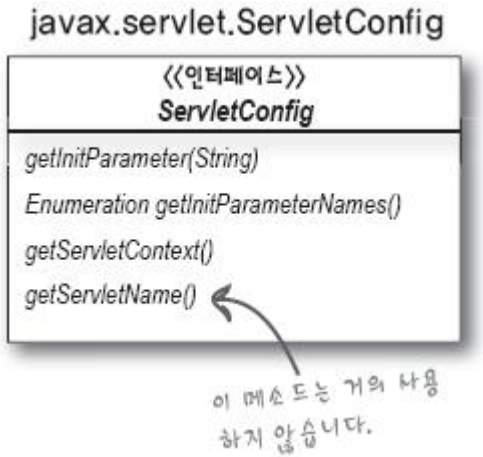
- ServletConfig의 동작 단계



# 서블릿 초기화와 정보 공유 –ServletConfig

- ServletConfig 이용하기



# 서블릿 초기화와 정보 공유 –ServletConfig

DD 파일(web.xml):

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
  <servlet>
    <servlet-name>BeerParamTests</servlet-name>
    <servlet-class>com.example.TestInitParams</servlet-class>
    <init-param>
      <param-name>adminEmail</param-name>
      <param-value>likewecare@wickedlysmart.com</param-value>
    </init-param>
    <init-param>
      <param-name>mainEmail</param-name>
      <param-value>blooper@wickedlysmart.com</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>BeerParamTests</servlet-name>
    <url-pattern>/Tester.do</url-pattern>
  </servlet-mapping>
</web-app>
```

# 서블릿 초기화와 정보 공유 –ServletConfig

**서블릿 코드:**

```
package con.example;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class TestInitParams extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("test init parameters<br>");

        Enumeration e = getServletConfig().getInitParameterNames();
        while(e.hasMoreElements()) {
           out.println("<br>param name = " + e.nextElement() + "<br>");
        }
        out.println("main email is " + getServletConfig().getInitParameter("mainEmail"));
        out.println("<br>");
        out.println("admin email is " + getServletConfig().getInitParameter("adminEmail"));
    }
}
```

# 서블릿 초기화와 정보 공유 –ServletConfig

DD 파일(web.xml):

```
<servlet>
    <servlet-name>BeerParamTests</servlet-name>
    <servlet-class>TestInitParams</servlet-class>
</servlet>

<context-param>
    <param-name>adminEmail</param-name>
    <param-value>clientheaderror@wickedlysmart.com</param-value>
</context-param>
```

<servlet> 항목 안에 있던 <init-param>은 모두 들어 냈습니다.

<context-param>은 전체 애플리케이션을 위한 항목입니다. 따라서 <servlet> 항목 안에 들어가지 않습니다. <context-param>을 <web-app> 항목에 포함시키되 <servlet> 항목 안에다 두지는 마세요.

<param-name>과 <param-value>는 서블릿 초기화 파라미터와 마찬가지로 이름/값의 쌍입니다. 단 이 항목은 <init-param> 항목이 아니라 <context-param>에 포함됩니다.