

# 서블릿의 스코프

: 바인딩된 속성에 대한 접근 범위

- 종류

**ServletContext** : 애플리케이션 전체에 대해 접근

**HttpSession** : 브라우저에서만 접근


**HttpServletRequest** : 요청/응답 사이클에서만 접근

각 객체의 setAttribute 메서드를 이용해 속성 바인딩 작업

```
String ctxMsg = "context에 바인딩됩니다.";
String sesMsg = "session에 바인딩됩니다.";
String reqMsg = "request에 바인딩됩니다.";

ServletContext ctx = getServletContext();
HttpSession session = request.getSession();
ctx.setAttribute("context", ctxMsg);
session.setAttribute("session", sesMsg);
request.setAttribute("request", reqMsg);
```

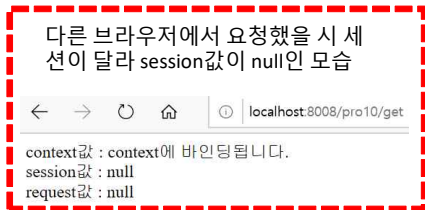
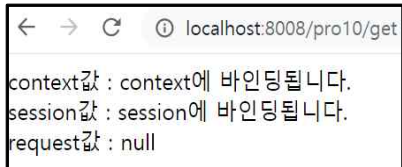
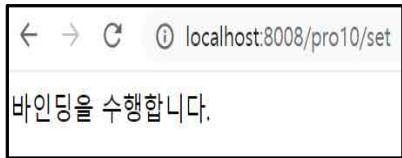
setAttribute(String name,  
Object value)로 바인딩



getAttribute 메서드를 이용해 속성 이름으로 값을 가져와 브라우저로 출력

```
ServletContext ctx = getServletContext();  
HttpSession sess = request.getSession();  
  
String ctxMesg = (String)ctx.getAttribute("context");  
String sesMesg = (String)sess.getAttribute("session");  
String reqMesg = (String)request.getAttribute("request");  
  
out.print("context값 : " + ctxMesg + "<br>");  
out.print("session값 : " + sesMesg + "<br>");  
out.print("request값 : " + reqMesg + "<br>");
```

# 바인딩과 요청 출력 결과 모습



# 서블릿의 URL 패턴

: 실제 서블릿의 매핑 이름

```
@WebServlet("/first/test") //정확히 이름까지 일치하는 URL 패턴
```

```
String context = request.getContextPath(); //컨텍스트 이름 가져오기  
String url = request.getRequestURL().toString(); //전체 URL 가져오기  
String mapping = request.getServletPath(); //서블릿 매핑 이름 가져오기  
String uri = request.getRequestURI(); //URI 가져오기
```

```
out.println("<html>");
```

/first/ 디렉터리 이름으로 시작하는 요청

```
@WebServlet("/first/*") //디렉터리 이름만 일치하는 URL 패턴
```

매핑 이름에 상관없이 확장자가 .do

```
@WebServlet("*.do") //확장자만 일치
```

모든 요청 URL 패턴

```
@WebServlet("/*") //모든 요청 URL
```

## /first/test 요청시

← → ↻ ⓘ localhost:8008/pro10/first/test

TestServlet1입니다.  
컨텍스트명 : /pro10  
전체경로 : http://localhost:8008/pro10/first/test  
매핑명 : /first/test  
URI : /pro10/first/test

## /first/\* 요청시

← → ↻ ⓘ localhost:8008/pro10/first/base

TestServlet2입니다.  
컨텍스트명 : /pro10  
전체경로 : http://localhost:8008/pro10/first/base  
매핑명 : /first  
URI : /pro10/first/base

## /\*.do 요청시

← → ↻ ⓘ localhost:8008/pro10/base.do

TestServlet3입니다.  
컨텍스트명 : /pro10  
전체경로 : http://localhost:8008/pro10/base.do  
매핑명 : /base.do  
URI : /pro10/base.do

## /\* 요청시

← → ↻ ⓘ localhost:8008/pro10/second/base

TestServlet3입니다.  
컨텍스트명 : /pro10  
전체경로 : http://localhost:8008/pro10/second/base  
매핑명 :  
URI : /pro10/second/base

## Filter API

: 브라우저에서 서블릿에 요청하거나 응답할 때 미리 요청이나 응답과 관련해 여러가지 작업을 처리하는 기능

<request에 한글 인코딩 설정 모습>

```
request.setCharacterEncoding("utf-8");
```



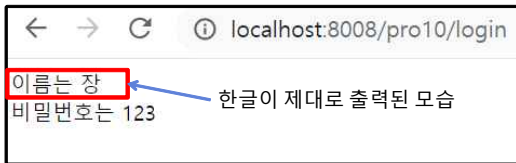
# EncoderFilter 클래스 작성

```
@WebFilter("/*")
public class EncoderFilter implements Filter {
    ServletContext context;

    public void init(FilterConfig fConfig) throws ServletException {
        System.out.println("utf-8 인코딩.....");
        context = fConfig.getServletContext();
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException { //doFilter() 안에서 실제 필터 기능 구현
        System.out.println("doFilter 호출");
        request.setCharacterEncoding("utf-8"); //한글 인코딩 설정 작업
        String context = ((HttpServletRequest)request).getContextPath(); //웹 애플리케이션의 컨텍스트 이름을 가져온다
        String pathinfo = ((HttpServletRequest)request).getRequestURI(); //웹 브라우저에서 요청한 요청 URI를 가져온다
        String realPath = request.getRealPath(pathinfo); //요청 URI의 실제 경로를 가져온다
        String msg = " Context 정보:" + context + "\n URI 정보 : " + pathinfo + "\n 물리적 경로: " + realPath;
        System.out.println(msg);
        long begin = System.currentTimeMillis(); //요청 필터에서 요청 처리 전의 시각
        chain.doFilter(request, response); //다음 필터로 넘기는 작업 수행
    }
}
```

# 필터를 거친 출력 결과



doFilter 호출

Context 정보: /pro10

URI 정보: /pro10/login

물리적 경로: C:\Users\wkddb\spring\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\pro10\pro10\login

작업 시간: 2ms

## 여러 가지 서블릿 관련 Listener API

- `HttpSessionBindingListener` 이용해 로그인 접속자수 표시
- `HttpSessionListener` 이용해 로그인 접속자수 표시

## HttpSessionBindingListener 이용해 로그인 접속자수 표시

```
String user_id = request.getParameter("user_id");
String user_pw = request.getParameter("user_pw");
LoginImpl loginUser = new LoginImpl(user_id, user_pw); //이벤트 핸들러를 생성한 후 세션에 저장
if(session.isNew()) {
    session.setAttribute("loginUser", loginUser); //세션에 바인딩 시 LoginImpl의 valueBound()메서드 호출
}

out.println("<head>");
out.println("<script type='text/javascript'>");
out.println("setTimeout('history.go(0);', 5000)"); //setTimeout() 함수를 이용해 5초마다 서버릿에 재요청하여
out.println("</script>"); //현재 접속자 수 표시
out.println("</head>");
out.println("<html><body>");
out.println("아이디는 " + loginUser.user_id + "<br>");
out.println("총 접속자수는 " + LoginImpl.total_user + "<br>"); //브라우저로 접속자수 출력
out.println("</body></html>");
```

# LoginImpl 클래스 작성

```
public class LoginImpl implements HttpSessionBindingListener { //HttpSessionBindingListener를 구현해 세션에
    String user_id; //바인딩 시 이벤트 처리
    String user_pw;
    static int total_user = 0; //세션에 바인딩 시 1씩 증가

    public LoginImpl() {

    }

    public LoginImpl(String user_id, String user_pw) {
        this.user_id = user_id;
        this.user_pw = user_pw;
    }

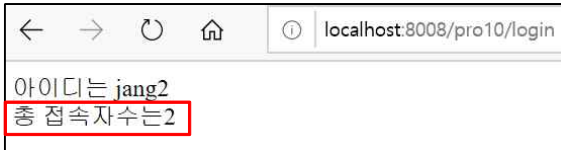
    @Override
    public void valueBound(HttpSessionBindingEvent arg0) { //세션에 저장 시 접속자수를 증가
        System.out.println("사용자 접속");
        ++total_user;
    }

    @Override
    public void valueUnbound(HttpSessionBindingEvent arg0) { //세션에서 소멸 시 접속자수 감소
        System.out.println("사용자 접속 해제");
        total_user--;
    }
}
```

## 크롬에서 로그인 한 결과



## 다른 브라우저에서 로그인 한 결과



## HttpSessionListener 이용해 로그인 접속자수 표시

```
public class LoginTest extends HttpServlet {  
    ServletContext context = null;  
    List user_list = new ArrayList(); //로그인한 접속자 ID를 저장하는 ArrayList
```

```
    LoginImpl loginUser = new LoginImpl(user_id, user_pw); //LoginImpl 객체를 생성한 후 전송된 ID와 비밀번호를 저장  
    if (session.isNew()) {  
        session.setAttribute("loginUser", loginUser);  
        user_list.add(user_id);  
        context.setAttribute("user_list", user_list); //최초로그인시 접속자 ID를 ArrayList에 차례로 저장한 후 다시  
    } //context 객체에 속성으로 저장
```

```
    out.println("<html><body>");  
    out.println("아이디는 " + loginUser.user_id + "<br>");  
    out.println("총 접속자수는 " + LoginImpl.total_user + "<br><br>");  
    out.println("접속 아이디:<br>");  
    List list = (ArrayList) context.getAttribute("user_list"); //context객체의 ArrayList를 가져와 접속자 ID를 차례로  
    for (int i = 0; i < list.size(); i++) { //브라우저로 출력  
        out.println(list.get(i) + "<br>");  
    }  
    out.println("<a href='logout?user_id=" + user_id + "'>로그아웃 </a>"); //로그아웃 클릭 시 서버릿 logout으로 접속자 ID를  
    out.println("</body></html>"); //전송해 로그아웃
```

## 로그아웃 클래스 작성

```
String user_id = request.getParameter("user_id");    //user_list에서 삭제할 ID를 가져온다

session.invalidate();    //로그아웃시 세션 소멸

List user_list = (ArrayList) context.getAttribute("user_list");
user_list.remove(user_id);
context.removeAttribute("user_list");
context.setAttribute("user_list", user_list);    //user_list에서 로그아웃한 접속자 ID를 삭제한 후
out.println("<br>로그아웃 했습니다.");    //다시 user_list를 컨텍스트에 저장
```



# 세션 생성과 소멸 시 이벤트 처리 핸들러 작성

@WebListener

```
public class LoginImpl implements HttpSessionListener {  
    String user_id;  
    String user_pw;  
    static int total_user = 0;  
}
```

HttpSessionBindingListener를 제외한 Listener를 구현한 모든 이벤트 핸들러는 반드시 애너테이션을 이용해서 Listener로 등록

```
@Override  
public void sessionCreated(HttpSessionEvent se) {           //세션 생성 시 이벤트 처리  
    System.out.println("세션 생성");  
    ++total_user;  
}  
  
@Override  
public void sessionDestroyed(HttpSessionEvent se) {         //세션 소멸 시 이벤트 처리  
    System.out.println("세션 소멸");  
    --total_user;  
}
```

## 결과 정보 표시

