Föreläsning 6

DD2390
Internetprogrammering
6 hp

Exempel

```
<html>
<head><title>Hälsning</title></head>
<h2>Välkomstprogram</h2>
<form action="slask.jsp">
<p>Vad heter du? <input type=text name="namn">
<p><input type=submit value="Skicka">
</form>
</html>
```

```
<html>
<head><title>Hälsning</title></head>
<body>
<center>
<h2>Hej <%=request.getParameter("namn")%> !!!</h2>
</center>
</body>
</html>
```

Innehåll

- Java Server Pages (JSP 2.3)
- Java DataBase Connectivity (JDBC 4.0)
- Java Naming and Directory Interface (JNDI 1.2)
- Model-View-Controller (MVC)

Fördefinierade variabler

- request (HTTPServletRequest)
- response (HTTPServletResponse)
- out (PrintWriter)
- session (HTTPSession)
- application (ServletContext)
- pageContext (PageContext)

Java Server Pages (JSP)

- Består av tre delar
  - Inbäddad jsp-kod
  - JavaBönor
  - Tag-bibliotek (JSTL)
- En jsp-sida är en html-sida med inbäddad javakod som har .jsp suffix.
- Den inbäddade koden separeras från html med speciella taggar <% *jsp-kod* %>
- Man har tillgång till hela javas klassbibliotek
- Vid körning översätts hela jsp-sidan till en servlet och kompileras, detta sker första gången sidan anropas, därefter anropas servlet:en.
- Nackdel: Många olika JSP-motorer innebär icke standardiserade felmeddelanden (inget radnummer vanligt)

PageContext

  - pageContext.getOut();
  - pageContext.getRequest();
  - pageContext.getResponse();
  - getServletContext() (=application scope)
  - getSession() (=session scope)
- från JspContext
  - findAttribute();
  - getAttribute();
  - removeAttribute();
  - setAttribute();

JSP-syntax

- Direktiv
- Deklarationer
- Scriptlets
- Fördefinierade variabler

JSP syntax -Scriptlets

- Scriptlets används till att exekvera javakod-block
- Exempel på syntax

  <% … java kod … %>
- Vilken syntaxriktig javakod som helst!

JSP-syntax - Direktiv

- Placeras i början av JSP-filen.

- `<%@ include file="filnamn" %>`
  - Inkluderar den angivna sidan **utan** att den exekveras
- `<%@ page errorpage="html/jsp filnamn" %>`
  - Returnerar den definierade sidan om ett fel inträffar i JSP-koden
  - Exempel: `<%@ page errorpage="error.html" %>`
- `<%@ page import="package name(s)" %>`
  - Definierar paketen som ska importeras för koden
  - Exempel: `<%@ page import="java.io.*,java.util.*"%>`

Skillnaden mellan <%! och <%

- Sidglobala varibler kan och bör deklareras inom

  ”<%! ... %>” taggar dvs:

```
<%! String s1 = "hej"; %>
<HR>
<%= s1 %>
```

- Då JSP-sidor kompileras om till servlets blir kod inom
  - deklarationer till instansvariabler/instansmetoder
  - Scriptlets till lokal kod i doGet, doPost etc.

JSP syntax - Deklarationer

- Deklarationer är till för att definiera instansvariabler och instansmetoder, hela jsp-sidan kan betraktas som ett objekt.
- Placeras innanför <%! taggar

Exempel

```
<html>
<head><title>Exempel 1</title></head>
<body><center>
<%!
public int utfall;
public void kasta(){
        utfall = 1 + (int)(Math.random()*6);
}
%>


<h2>Tärningskast</h2>
<%
kasta();
%>
Tärningens utfall: <%=utfall%>
<form><input type=submit value="Kasta"></form>
</center></body></html>
```

```
<html>
<head><title>Exempel 2</title></head>
<body><center>
<%!
class Dice {
    public int utfall;
    public void kasta(){
        utfall = 1 + (int)(Math.random()*6);
    }
}
%>
<h2>Tärningskast</h2>
<%
Dice d = new Dice();
d.kasta(); %>
Tärningens utfall: <%=d.utfall%>
<form><input type=submit value="Kasta"></form>
</center></body></html>
```

Javabönor

- Hitills har vi adderat jsp-kod till html-sidorna vilket har några nackdelar:
  - det blir snabbt oöverskådligt och svårt att felsöka
  - webbdesignern måste kunna Java
- Man önskar separera presentation (html) från logik (t e x jsp) vilket kan åstadkommas med javabönor.
- En javaböna är identiskt med en vanlig javaklass men följer JavaBean designmönster:
  - public konstruktor utan argument
  - publika set och get metoder för att sätta/läsa egenskaper

Exempel: JSP/JavaBean

```
<%@ page import="java.util.ArrayList" %>

<jsp:useBean class="bean.Forum" id="forum" scope="application"/>
<jsp:useBean class="bean.User" id="user" scope="session"/>
<jsp:useBean class="bean.Post" id="post" scope="request"/>

<jsp:setProperty name="forum" property="*"/>
<jsp:setProperty name="user" property="*"/>
<jsp:setProperty name="post" property="*"/>

<% if(session.isNew()){ %>
<h1>Ny session!</h1>
<form>
Nickname<input type="text" name="nickname"><br>
Email<input type="text" name="email">
<input type="submit"></form>

<%
}
if(request.getParameter("email")!=null){
%>
<h1>Ny användare</h1>
<form>
Text: <input type="text" name="text">
<input type="submit"></form>
```

Fördefinierade JavaBean taggar

- jsp:useBean tag används för att referera till en javaBean på en JSP-sida, syntax:
- *<jsp:useBean id = "valbart namn att referera till bönan" scope = "page|request|session|application" class = "namnet på .class-filen" />*
- scope (räckvidd):
  - page/request: bönan är sidglobal
  - session: Om bönan är skapad återanvänds den, om inte skapas den och sparas i HTTP-sessionsobjektet
  - application: webbserverns livscykel
- För att sätta en egenskap används setProperty
  - *<jsp:setProperty name="tidigare valt namn att referera till bönan" property="instansvariabel|*">*
- För att läsa en egenskap används getpropery
  - *<jsp:getProperty name="tidigare valt namn att referera till bönan" property="instansvariabel(oftast)">*

Exempel: JSP/JavaBean

```
<%
}
if(request.getParameter("text")!=null){
%>
<h1>Nytt inlägg</h1>
<%
post.setNickname(user.getNickname());
forum.addPost(post);
ArrayList posts = forum.getPosts();
for(int i = 0; i < posts.size(); i++){
post = (bean.Post)posts.get(i);
%>
<b><%=post.getText()%></b><br>
<i><%=post.getNickname()%></i><br>
<%
}
%>
<form>
Text: <input type="text" name="text">
<input type="submit"></form>
<%
}
%>
```

Exempel

```
<html>
<head><title>Exempel 3</title></head>
<body>
<center>

<jsp:useBean class="bean.DiceBean" id="db"
scope="session"/>
<% db.kasta(); %>
<h2>Tärningskast</h2>
Tärningens utfall:<p>
<img src="<jsp:getProperty name="db"
property="utfall"/>.gif">
<form><input type=submit value="Kasta"></form>
</center>
</body>
</html>
```

```
package bean;

public class DiceBean{
   private int utfall;

   public DiceBean(){
   }

   public void kasta(){
     utfall = 1 + (int)(Math.random()*6);
   }

   public int getUtfall(){
     return utfall;
   }
}
```

JDBC

- Ett API för att kommunicera med databaser från javaapplikationer
- JDBC API:t är en del av JDK som arbetar mot den databasdrivrutin som finns konfigurerad sedan tidigare
- Typer av drivrutiner
  - SimpleDataSource
  - ConnectionPoolDataSource
  - XADataSource
- Använd inte DriverManager.getConnection() utan slå upp datakällan via JNDI.
- Relevanta paket:
  - java.sql
  - javax.sql

javax.sql.Datasource

- En abstraktion av DB-drivrutinen, liknande DriverManager men erhåller en referens till DB-drivrutinen via JNDI, p s s slipper man DB-specifik kod.
- Datasource är ett Interface och implementeras av Driverförsäljaren, finns tre kategorier av dessa
- Man får normalt tag på ett Datasource objekt genom en JNDI-uppslagning.
- Är en abstraktion av drivrutinen som placeras under tomcat/lib och konfigureras i tomcat/conf/context.xml

JDBC - klasserna

- Dessa tre klasser är viktigast att känna till i JDBC:
  - `Connection`
  - `Statement (PreparedStatement)`
  - `ResultSet`

Typer

- javax.sql.DataSource
  - Vanlig ”simpel”
  - Identisk med DriverManager
- javax.sql.ConnectionPoolDataSource
  - Stödjer Connection pooling
- javax.sql.XADataSource
  - Distribuerad och stödjer Connection pooling

DatabasURL:er

- **URL:en bidrar med nödvändig information**
  - Signalerar att det är en DB-URL
  - Identifierar databasens sub-protokoll
  - Lokaliserar databasen
- **Generell syntax**
  - **`jdbc : <driverns sub-protokoll> : <db sökvägen>`**

JDBC Arkitekturen

- **Java applikation**
  - Skapad av Java utvecklaren
  - Alla anrop till databasen enligt JDBC API:t
- **JDBC API**
  - Tillhandahållen av JavaSoft
  - Fungerar som länk mellan applikationen och drivrutinen
- **JDBC drivrutinen**
  - Tillhandahållen av DB företaget eller tredje part
  - Konverterar JDBC kod till DB specifika databaskommandon

Exempel på DatabasURL:er

- ODBC Datakälla
  jdbc:odbc:test
- PostgreSQL Databas
  jdbc:postgresql://pgsql0.nada.kth.se:5432/stene
- mySQL datakälla (egen db)
  jdbc:mysql://localhost:3306/test
- mySQL datakälla (nada)
  jdbc:mysql://mysql-vt2015.csc.kth.se:3306/stene
- Titta i tomcat/conf/context.xml

Java Naming and Directory Interface(JNDI)

- För att distribuerade applikationers komponenter skall kunna hitta varandra behövs någon tjänst som hjälper till med detta, en s k namngivningstjänst (Naming Service)
- JNDI mappar namn mot objekt (jämför med DNS).
- JNDI är ett interface som är beroende av en underliggande implementation t.ex. LDAP för att fungera
- JNDI använder ett fåtal objekt, främst Context & InitialContext
- Ett Context objekt har metoder för att binda namn till objekt, lista existerande namn, ta bort och döpa om

Exempel: JDBC

```
package bean;
import java.util.ArrayList;
import java.sql.*;
import javax.naming.*;
import javax.sql.*;

public class ForumDB{

   private Connection conn;

    public ForumDB(){
  try{
     Context initCtx = new InitialContext();
     Context envCtx = (Context) initCtx.lookup("java:comp/env");
     DataSource ds = (DataSource)envCtx.lookup("jdbc/db");
     Connection conn = ds.getConnection();
  }
  catch(SQLException e){
  }
  catch(NamingException e){
  }

   }
```

javax.naming.InitalContext

- root-context som kan innehålla objekt och andra Context
- Startpunkten för en JNDI-uppslagning

Exempel: JDBC

```
public void close(){
   try{
      conn.close();
   }
   catch(SQLException e){
   }
    }
```

javax.naming.Context

- Viktigaste metoder:
  - **`void bind(String stringName, Object object)`**
  - **`void unbind(String name)`**
  - **`void rebind(String name, Object obj)`**
  - **`void rename(String oldName, String newName)`**
  - **`Object lookup(String stringName)`**
  - **`NamingEnumeration list(String name)`**
  - **`NamingEnumeration listBindings(String name)`**

Exempel: JDBC

```
public void addPost(Post p){
      Statement stmt = null;
      ResultSet rs = null;
      String query = null;
      int id = 0;
      try{
         stmt = conn.createStatement();
         rs = null;
         query = "select max(id)+1 as id from posts";
         rs = stmt.executeQuery(query);
         if(rs.next())
        id = rs.getInt("id");
         rs.close();
      }
      catch(SQLException e){
      }

      query = "insert into posts values(?,?,?)";
      try{
         PreparedStatement pstmt = conn.prepareStatement(query);
         pstmt.setInt(1,id);
         pstmt.setString(2,p.getText());
         pstmt.setString(3,p.getNickname());
         int tupler = pstmt.executeUpdate();
      }
      catch(SQLException e){
      }
   }
```

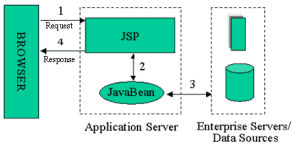
Exempel: JDBC

```
public ArrayList getPosts(){
      ArrayList list = new ArrayList();
      try{
         Statement stmt = conn.createStatement();
         ResultSet rs = null;
         String query = "select * from posts";
         rs = stmt.executeQuery(query);
         while(rs.next()){
        Post p = new Post();
        p.setText(rs.getString("text"));
        p.setNickname(rs.getString("nickname"));
        list.add(p);
         }
      }
      catch(SQLException e){
      }
      return list;
   }
}
```

Designmönster

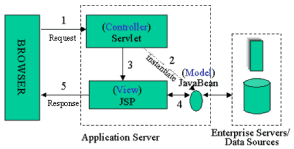
- Omedvetet eller ej har ni hittils oberoende av designmönster haft ungefär följande
  1. Hantera inkommande HTTP-request
  2. Instansiera objekt och / eller anropa metoder i dessa / redan befintliga objekt
  3. Generera HTTP-response
- Servlets är bra på (1), JavaBeans på (2) och JSP-sidor är lämpliga för (3)

Model - 1



•Den metod vi främst använt hittils
•I JSP-sidan blandas presentation (View) och validering av input (Control)
•En eller flera JavaBönor innehåller datastrukturen (Model)
•Ibland används inte ens JavaBönor och då blir koden mycket rörig.

Model – 2 (MVC)



Designmål: Separera

•Datastruktur (**M**odel)

•Presentation (**V**iew)

•Validering av input (**C**ontrol)

Designproblem

En Servlet är lämplig för att hantera logik men olämplig för att hantera presentation p g a att all HTML läggs inuti Java-kod.

En JSP-sida är lämplig för att hantera presentation men olämplig för att hantera logik p g a att all Java-kod hamnar i en HTML-kod.

Exempel: MVC

Model (JavaBönor)
- Forum.java (application scope)
- User.java (session scope)
- Post.java (request scope / page scope)

View (JSP-sidor)
- forum_index.html
- forum_view.jsp

Controller (Servlet)
- ForumController.java

ForumController.java

```
import java.io.*;
import javax.servlet.ServletContext;
import javax.servlet.RequestDispatcher;// ny
import javax.servlet.ServletException; // ny
import javax.servlet.http.*;
import java.util.ArrayList;

public class ForumController extends HttpServlet {

   public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws IOException{

    PrintWriter out = response.getWriter();
    ServletContext sc = getServletContext();
    if(sc.getAttribute("forum")==null){
       sc.setAttribute("forum", new bean.Forum());
    }
```

ForumController.java

```
    if(request.getParameter("text")!=null){
       bean.User u = (bean.User)session.getAttribute("user");
       bean.Forum f = (bean.Forum)sc.getAttribute("forum");
       bean.Post p = new bean.Post();
       p.setText(request.getParameter("text"));
       p.setNickname(u.getNickname());
       f.addPost(p);
       RequestDispatcher rd = sc.getRequestDispatcher("/forum_view.jsp");
       try{
            rd.forward(request, response);
       }
       catch(ServletException e){
            out.println(e.getMessage());
       }
    }
    out.close();
   }
}
```

ForumController.java

```
    HttpSession session = request.getSession();
    if(session.isNew()){
       session.setAttribute("user", new bean.User());
       RequestDispatcher rd =
    sc.getRequestDispatcher("/forum_index.html");
       try{
            rd.forward(request, response);
       }
       catch(ServletException e){
            out.println(e.getMessage());
       }
    }
```

Forum_index.html

```
<html>
<head><title>Forum (inloggning)</title></head>
<body>
<form action="/ForumController">
Nickname<input type="text" name="nickname"><br>
Email<input type="text" name="email"><br>
<input type="submit"></form>
</body>
</html>
```

ForumController.java

```
    if(request.getParameter("email")!=null){
       bean.User u = (bean.User)session.getAttribute("user");
       u.setNickname(request.getParameter("nickname"));
       u.setEmail(request.getParameter("email"));
       RequestDispatcher rd =
    sc.getRequestDispatcher("/forum_view.jsp");
       try{
            rd.forward(request, response);
       }
       catch(ServletException e){
            out.println(e.getMessage());
       }
    }
```

Forum_view.jsp

```
<%@ page import="java.util.ArrayList" %>
<html><head><title>Forum (visa)</title></head><body>
<%
  bean.Forum f = (bean.Forum)pageContext.getServletContext().getAttribute("forum");
  bean.User u = (bean.User)pageContext.getSession().getAttribute("user");
  bean.Post p;
%>
<h4><%= u.getNickname() %>(<%= u.getEmail() %>)</h4>
<%
  ArrayList posts = f.getPosts();
  for(int i = 0; i < posts.size(); i++){
         p  = (bean.Post)posts.get(i);
%>
         <b> <%=p.getText()    %> </b><br>
         <i> <%=p.getNickname()%> </i><br>
<%}%>
<form action="/ForumController">
Text<input type="text" name="text"><br>
<input type="submit"></form>
</body></html>
```

web.xml

```
<servlet>
    <servlet-name>ForumController</servlet-name>
    <servlet-class>ForumController</servlet-class>
  </servlet>
<servlet-mapping>
    <servlet-name>ForumController</servlet-name>
    <url-pattern>/ForumController</url-pattern>
</servlet-mapping>
```