Föreläsning 1

2D1482
Webbservertillämpningsprogrammering
4p

Kursintroduktion

- Presentation
  - Kursledare: Sten Andersson, stene@nada.kth.se
- Anmälningslista
  - Alla som inte står med måste anmäla sig på sitt kansli.
- Kursmaterial
  - Skrivs ut från webben.
- Kurshemsida
  - http://www.nada.kth.se/kurser/kth/2D1482/serverv06/
- Schema
- Föreläsningar
- Laborationer (datorsalar, konton, handledare)
- Betyg
- Kurslitteratur
- OBS!!! Kontrollera att ni har fungerande UNIX-konton **innan** första laborationstillfället.

Mål

- Kursens mål: Att självständigt kunna utveckla webbserverapplikationer med J2EE.
- Vad är en webbserverapplikation? En serverbaserad applikation som tillhandahåller klienter (= webbläsare) **dynamiska** HTML-sidor.
- Vad är en dynamisk html-sida: En html-sida som genereras med en utformning skräddarsydd för varje klient som efterfrågar den.
- Vad är en webbserver? Ett program som kommunicerar med http (https) och vanligtvis använder port 80 för kommunikation. En webbserver är även passiv, d v s den svarar endast på förfrågningar.

J2EE

- J2EE erbjuder ett komponentbaserat angreppssätt för att designa, utveckla, sammanfoga och driftsätta enterprise applikationer.
- En J2EE-komponent är en abstraktion för en logisk och ibland fysisk uppdelning av en applikation i flera beståndsdelar, jämför gärna med OO. Man brukar ibland tala om både klient- och serverkomponenter med om detta inte nämns avses serverkomponenter.
- En J2EE-komponent är en fristående programmodul som tillsammans med andra J2EE-komponenter utgör en J2EE-applikation. En komponent består dels av java-klasser men även av andra associerade filer.
- Komponentegenskaper
  - möjliggör flerskiktslösningar
  - är distribuerbara
  - tillhandahåller transaktionsstöd och persistens
  - hanterar säkerhet
- Nuvarande version 1.4

Flerskiktslösningar

- En metod att stycka upp sin applikation i vanligtvis tre stycken lager
  - Presentationslager
  - Web & Affärslogiklager
  - Databaslager
- De två mellersta lagren är endast logiskt uppdelade, inte fysiskt, därför är det en definitionsfråga om man vill betrakta dem som två eller en.
- En komponent utgör ofta ett lager.

J2EE Server komponenter

- Webkomponenter
  - JSP / servlets
  - dynamiska html-sidor
  - Taglib
  - JavaBeans
- Affärskomponenter
  - Enterprise JavaBeans
  - EJB != JavaBeans
  - Entity EJB tillhandahåller datapersistens
  - Session EJB används för att representera klienter

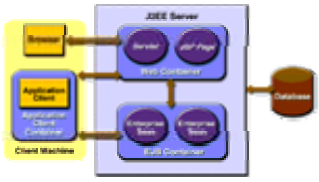
J2EE klient komponenter

- Webklient
  - HTML, XML
  - Applet
- Applikationsklient
  - Swing
  - AWT

Container

- Varje (server)komponent har ett gränssnitt mot den underliggande applikationsservern som kallas en container, innan en komponent kan användas måste den sammanfogas till en j2ee-modul och driftsättas inuti sin container.
  - EJB-container
  - Web-container
- Containern hanterar
  - Säkerhet
  - Transaktionshantering
  - JNDI
  - Kommunikation mellan komponenter
- En del funktionalitet i containern är konfigurerbar och som en följd av detta kan en j2ee-komponent uppvisa olika beteenden om den driftsätts på olika j2ee-servrar.

Container



Laborationer

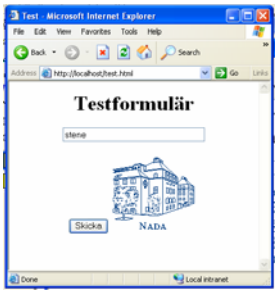
- Laboration 1: JSP/JavaBeans/Taglib
- Laboration 2: EJB
- Laboration 3: MVC
  - Dessa tre labbar utgör momentet LAB1
- Projekt
  - Utgör momentet LAB2

HTML

- Ett formateringsspråk för websidor vilket vanligtvis utgör användargränssnittet för webapplikationer.
- All information lagras i *taggar*, som har formatet: *<tag>Text</tag>*
- Alla html-dokument börjar med <html> och slutar med </html>. HTML-dokumentet är sedan grupperat i två delar, en inledande <head>…</head> där "runtomkring" information om dokumentet specificeras samt följande <body>…</body> som innehåller det "som visas"
- De flesta taggarna berör presentation men även variabler kan lagras i html-sidor m h a <form>
- En bra referens finns på www.w3schools.com

HTML-exempel

```
<html>

<head>
<title>Test</title>
</head>

<body>

<center>

<h1>Testformulär</h1>

<form method=get action="http://localhost:80/test.html">
<p><input type=text name=vem size=32>
<p><input type=hidden name=y value=2>
<p><input type=submit value="Skicka">
</form>
<img src="./nada.gif">

</center>

</body>

</html>
```



URL

- En URL (***U**niform **R**esource **L**ocator)* är en form av resurspekare som märker ut en resurs och sättet (protokollet) att kommunicera med den. Resursen är ofta en fil, men bör betraktas mer generellt (en datamängd).
- Följande standardiserade format används: *protokoll://värd[:port]/sökväg* t e x
  http://localhost:8080/index.html
  http://www.nada.kth.se
  ftp://ftp.sunet.se
- Om port ej anges är 80 standard för http.

HTTP

- HTTP är det protokoll med vilket större delen av all webtrafik (=HTML) levereras. Proceduren är följande:
  - Klienten öppnar en anslutning
  - Klienten skickar en förfrågan (= request)
  - Servern skickar ett svar (= response)
  - Servern stänger anslutningen
- Benämns tillståndslöst då ingen förbindelse upprätthålls mellan två förfrågningar utförda av en och samma klient.
- HTTP-protokollet är en specifikation för hur denna kommunikation ska ske och kan betraktas som en grammatisk kravspecifikation i två versioner
  - HTTP-request
  - HTTP-response
- Det är helt upp till utvecklaren av webbläsaren / webservern att implementera stödet för denna specifikation. Nuvarande version är 1.1.

HTTP-request

- Formatet för en http-request är:
  - initial line: <metod> <sökväg> <http-version>
  - header1: <variabel> : <värde>
  - header2: <variabel> : <värde>
  - header3: <variabel> : <värde>
  - <*blankrad*>
  - message: Eventuell data...
- <metod> = GET,POST,HEAD,PUT,DELETE,TRACE,OPTIONS
- Vanligaste metoder är GET och POST. Med GET blir formulärparametrar en del av URL:en och med post blir de en del av kroppen. I header:n skickar webläsaren med information om sig själv och dess miljö samt eventuella cookies.
- <sökväg> t e x /index.html
- <http-version> I stort sett alltid 1.1

GET-metoden

```
GET /test.html?x=stene&y=2 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shock
wave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application
/msword, */*
Referer: http://localhost/test.html
Accept-Language: sv
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1
.4322)
Host: localhost
Connection: Keep-Alive
Cookie: clientId=1

GET /nada.gif HTTP/1.1
Accept: */*
Referer: http://localhost/test.html?x=stene&y=2
Accept-Language: sv
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1
.4322)
Host: localhost
Connection: Keep-Alive
Cookie: clientId=1
```

POST-metoden

```
POST /test.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash,
application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://localhost/test.html
Accept-Language: sv
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Host: localhost
Content-Length: 11
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: clientId=1

x=stene&y=2

GET /nada.gif HTTP/1.1
Accept: */*
Referer: http://localhost/test.html
Accept-Language: sv
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Host: localhost
Connection: Keep-Alive
Cookie: clientId=1
```

HTTP-response

- Formatet för en http-response är:
  - initial line: <http-version> <kod> <OK/Error>
  - header: <variabel> : <värde>
  - message: Eventuell data...
- Det finns fem grupper av koder:
  - 1xx = Informativ
  - 2xx = Lyckad
  - 3xx = Omdirigering
  - 4xx = Klientfel
  - 5xx = Serverfel
- I header:n bör alltid *Content-Type* ingå för att webbläsaren ska kunna skilja text från binär information etc. Dessa s k MIME-typer specificerar innehållet i *message*.
- OBS!!! Det är endast *message* (*body*) som syns när man väljer "view source" i IE.

HTTP-response (HttpServer.java)

```
HTTP/1.1 200 OK
Server : Slask 0.1 Beta
Content-Type: text/html
Set-Cookie: clientId=1; expires=Wednesday,31-Dec-05 21:00:00 GMT

<html>
...
</html>
```

HTTP-response (www.kth.se)

```
HTTP/1.1 200 OK
Date: Wed, 31 Aug 2005 00:25:49 GMT
Server: Apache/1.3.33 (Unix) mod_ssl/2.8.22 OpenSSL/0.9.7e
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
...
</html>
```

Cookies

- Då man i en webserverapplikation vanligtvis vill kunna skilja de olika klienterna från varandra och detta inte stöds av HTTP låter man servern spara en s k *cookie* hos klienten. Varje klient som besöker en webserver får en unik sessions-id-kaka.
- En cookie har ett namn och ett värde och sparas i webbläsarens minne. Servern definierar cookie:n i response header:n
  - Set-Cookie: < namn > = < värde >;
  - expires= < datum >;
  - domain= < domän >;
  - path= < bibliotek >
- Om klienten vid ett senare tillfälle försöker komma åt samma domän/sökväg skickas cookie:n med i request-header:n.

HttpClient

```
import java.io.*;
import java.net.*;

public class HttpClient{

    public static void main(String[] args) throws Exception{
        String host = args[0];
        int port = Integer.parseInt(args[1]);
        String fil = args[2];
        Socket s =
            new Socket(host,port);

        PrintStream utdata =
            new PrintStream(s.getOutputStream());
        utdata.println("GET /" + fil + " HTTP/1.1");
        s.shutdownOutput();

        BufferedReader indata =
            new BufferedReader(new InputStreamReader(s.getInputStream()));
        String str = "";
        while( (str = indata.readLine()) != null){
            System.out.println(str);
        }
        s.close();
    }
}
```

HttpServer 1 / 2

```
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

public class HttpServer{

    public static void main(String[] args) throws IOException{
        ServerSocket ss = new ServerSocket(80);
        while(true){
            Socket s = ss.accept();
            BufferedReader request =
                new BufferedReader(new InputStreamReader(s.getInputStream()));
            String str = request.readLine();
            System.out.println(str);
            StringTokenizer tokens =
                new StringTokenizer(str," ?");
            tokens.nextToken();
            String requestedDocument = tokens.nextToken();
            while( (str = request.readLine()) != null && str.length() > 0){
                System.out.println(str);
            }
            s.shutdownInput();
```

HttpServer 2 / 2

```
            PrintStream response =
                new PrintStream(s.getOutputStream());
            response.println("HTTP/1.0 200 OK");
            response.println("Server : Slask 0.1 Beta");
            if(requestedDocument.indexOf(".html") != -1)
                response.println("Content-Type: text/html");
            if(requestedDocument.indexOf(".gif") != -1)
                response.println("Content-Type: image/gif");

            response.println("Set-Cookie: clientId=1; expires=Wednesday,31-Dec-05 21:00:00 GMT");

            response.println();
            File f = new File("."+requestedDocument);
            FileInputStream infil = new FileInputStream(f);
            byte[] b = new byte[1024];
            while( infil.available() > 0){
                response.write(b,0,infil.read(b));
            }
            s.shutdownOutput();
            s.close();
        }
    }
}
```