F10

Webbteknologier

EDA095 Nätverksprogrammering

Roger Henriksson

Datavetenskap

Lunds universitet

Dynamiska webbsidor

- HTML är statisk. En sida får sitt utseende bestämt när en webbdesigner skapar den.
- Ofta vill man ha mera dynamiska webbsidor:
  - Svar på en databasförfrågan.
  - Konstant uppdaterade webbsidor.
  - Dialog med användaren.
  - Animeringar.
  - Kontroll av inmatad information i ett formulär.

Serversidan eller klientsidan?

Klientsidan

- JavaScript

Serversidan

- CGI – CommonGateway Interface
- JSP (Java Server Pages) och Servlets
- ASP – Active Server Pages
- PHP – “PHP:Hypertext Preprocessor”

Varför inte en helt specialskriven webbserver?

Servlets

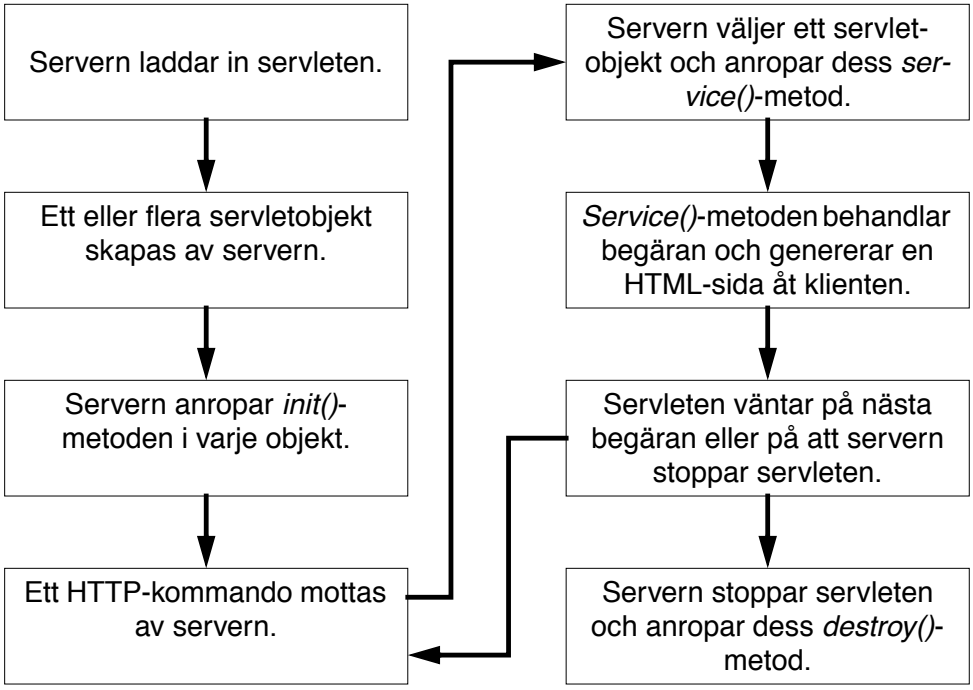
Servlet?

Applet - “liten applikation”, Servlet - “liten server”

Exekveras på servern:

- Skrivna i Java.
- Systemoberoende.
- Skapar inte ny operativsystemsprocess varje gång. Effektivt!
- Startar inte om för varje HTTP-begäran. Kan komma ihåg information från gång till gång.

En servlets liv



Klassöversikt

Paket

```
import javax.servlet.*;       // Standard i J2EE
import javax.servlet.http.*; // Standard i J2EE
import java.io.*;
```

Klasser/interface

- HttpServlet – superklass för webbservlets.
- HttpServletRequest – klientens HTTP-begäran.
- HttpServletResponse – servletens svar till klienten.
- ServletConfig – information om servern.

HttpServlet

Ett servletobjekt skapas av webbservern.

Initialisering

Implementera en av nedanstående:

```
public void init(ServletConfig config)
                                    throws ServletException;
public void init();
```

Terminering

```
public void destroy();
```

Implementera denna för att till exempel stänga databasuppkopplingar/stänga öppna filer när servleten avslutas av servern.

service()

När servern tar emot ett HTTP-kommando anropas:

```
protected void service(HttpServletRequest request,
                       HttpServletResponse response)
                  throws ServletException,
IOException;
```

- request – information om klientens begäran
- response – används för att skicka svar till klienten.

Ofta vill vi göra olika saker beroende på typ av begäran:

```
if (request.getMethod().equals("GET")) {
  ...
} else {
   if (request.getMethod().equals("POST")) {
    ...
```

Alternativ till service()

Standardimplementationen av service() undersöker vilken typ av kommando klienten skickade (GET/POST/HEAD etc) och anropar en av:

```
protected void doGet(HttpServlerRequest request,
                     HttpServletResponse response)
                          throws ServletException,IOException;
protected void doPost(HttpServlerRequest request,
                      HttpServletResponse response)
                          throws ServletException,IOException;
```

Likadant för:

doHead
doPut
doDelete

HttpServletRequest

Information om klientens begäran.

Metoder

```
public String getParameter(String name);
```

Returnerar värdet för angiven parameter, t.ex. innehållet i ett textfält i ett HTML-formulär.

```
public String getRemoteAddr();
public String getRemoteHost();
```

IP-nummer / namn på klientdatorn.

```
public String getMethod();
```

Typ av begäran (GET, POST, etc.).

HttpServletResponse

Används för att skicka svar till klienten.

1. Ange MIME-typen för svaret:
   ```
   response.setContentType("text/html");
   ```
2. HTML-koden skickas genom en ström:
   ```
   PrintWriter output = response.getWriter();
   ```
3. Skriv HTML-koden till strömmen.
4. Stäng strömmen:
   ```
   output.close();
   ```

Tillståndsinformation

Kommunikationen mellan klient och webbserver är ofta en dialog - jmf en webbshop.

Servern måste hålla reda på vad som hänt under dialogen, dvs hålla reda på ett tillstånd.

Tillstånd kan lagras i servleten, men:

Vi måste kunna skilja på olika klienter!

- Gömda fält i formulär
  ```
  <input type="hidden" name="number" value="42">
  ```
- Cookies
- HttpSession

Klassen Cookie

Namn/värdepar som lagras på klienten.

Paket

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

Konstruktor

```
public Cookie(String name, String value);
```

Metoder

```
public String getName();
public String getValue();
```

Skriva/läsa cookies

Metoder i klasserna HttpServletRequest/ HttpServletResponse.

HttpServletRequest

```
public Cookie[] getCookies();
```

Returnerar en vektor med samtliga cookies från denna webbplats.

HttpServletResponse

```
public void addCookie(Cookie cookie);
```

JSP – Java Server Pages

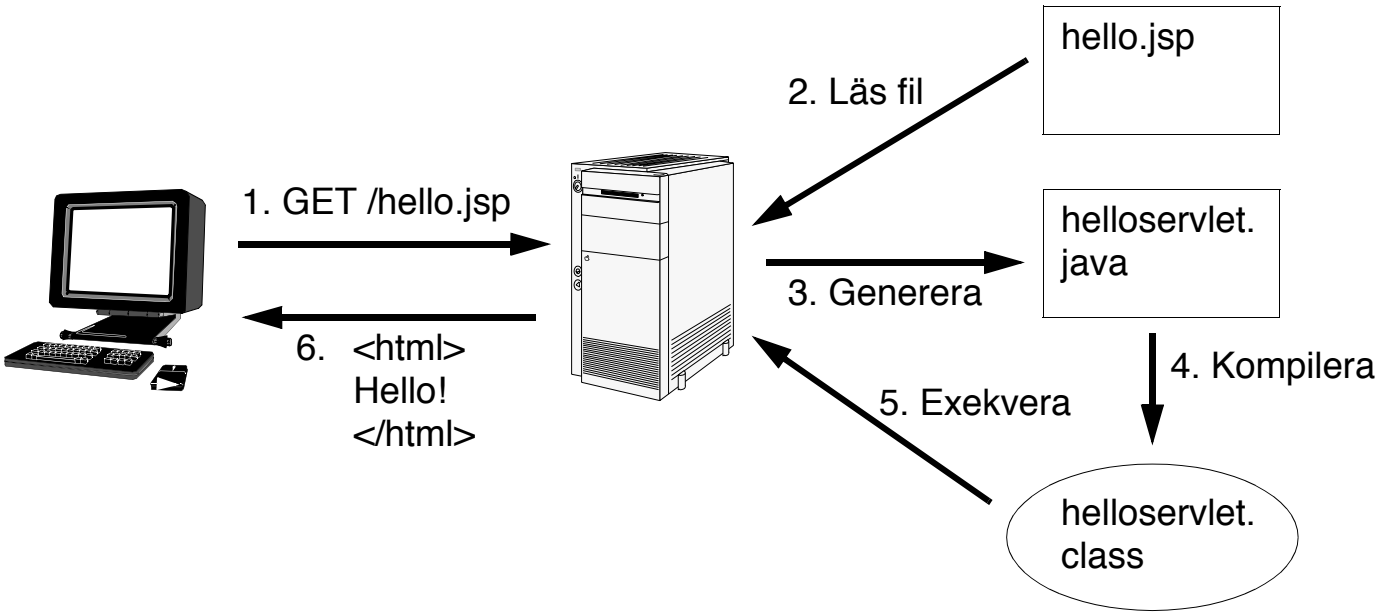
Idé

“Programkoden i HTML-koden” istället för “HTML-koden i programkoden”.

Implementation

- En JSP-fil (.jsp) är en HTML-fil med några extra element i.
- JSP-elementen (“tags”) anger var dynamisk HTML-kod ska infogas i dokumentet och hur den ska genereras.
- När en klient begär JSP-filen tolkas innehållet av servern och görs automatiskt om till en servlet första gången.

Översättning av JSP-fil



JSP – Tags

Direktiv

Anvisningar för översättningen till servlet. "<%@ ... %>"

Deklarationer

Deklarationer av attribut motsvarande servletattribut.

"<%! ... %>"

Uttryck

Anger javauttryck vars värde stoppas in på sidan.

"<%= ... %>"

Scriptlets

Block av javakod som exekveras när JSP-sidan anropas.

"<% ... %>"

JSP – Direktiv

Styr översättningen till en servlet.

"page"
Styr servletens struktur: Importerar externa klasser, ändrar "content type", ändrar servletens superklass.

Exempel:

```
<%@ page import="java.util.*" %>
<%@ page contentType="text/plain" %>
```

"include"
Inkluderar andra JSP-filer vid översättningen.

JSP – Deklarationer

Används för att deklarera variabler som sedan kan användas i uttryck och i "scriptlets".

Exempel:

```
<%! int counter = 0; %>
<%! Date today = new Date(); %>
```

Flera deklarationer kan samlas:

```
<%!
  int counter = 0;
  Date today = new Date();
%>
```

JSP – Uttryck

Används för att stoppa in resultatet av en beräkning eller annat uttryck i HTML-koden.

Exempel:

```
<%= counter %>
<%= today.toString() %>
Pris (inkl. moms): <%= pris*1.25 %> kronor.
```

Kan innehålla godtyckliga javauttryck.

JSP – Scriptlets

Anger Javakod som ska exekveras när sidan hämtas.

Exempel:

```
<%
   total = 0.0;
   for(int i=0;i<myArray.length;i++) {
      total = total+myArray[i];
   }
%>
Average: <%= total/myArray.length %>
```

JSP – Implicita objekt

Ett antal standardobjekt finns alltid tillgängliga utan explicit deklaration.

`request` – HTTP-begäran från klienten.

`response` – HTTP-svaret till klienten.

`session` – HTTPSession-objekt associerat till den aktuella användaren/sessionen.

`application` – Refererar till "globala" objekt som ska delas mellan alla sessioner, t.ex. databasanslutning.

`out` – Objekt som används för att skriva till den utgående svarsströmmen (till klienten).

PHP – PHP: Hypertext Preprocessor

- Skriptspråk särskilt lämpligt för webbapplikationer.
- Öppen, gratis, programvara.
- HTML-kod med PHP-taggar inlagda där dynamiskt beteende behövs (jämför JSP).
- Vanlig och mycket spridd teknik:
  - CMS – Content Management Systems (ex LTH).
  - Wiki – Många wikiimplementationer i PHP.

Struktur/taggar

HTML-kod med PHP-taggar insprängda.

```
<?php
    ...
?>
```

All text utanför PHP-taggarna skickas oförändrad till klienten.

Datatyper

Dynamiska typer, ingen variabeldeklaration behövs.

boolean	Sant/falskt.
integer	Heltal.
float	Flyttal.
string	Strängar.
array	Vektorer.
object	Objekt.
resource	Referens till externa resurser, t.ex. filer.
NULL	Tomt värde.

Språkkonstruktioner

Några språkkonstruktioner:

```
$a = $b*3;  // Tilldelning, uttryck

echo “Svar: “.$a.”\n”; // Utskrift,
                        strängkonkatenering

if ($a==0) { ... } else { ... } // If-sats

for($i=0;$i<10;$i++) { ... } // For-sats
```

Vektorer

Vektorer är associativa! (Egentligen en “ordered map”.)

Index kan vara av godtycklig typ.

```
$arr = array();    // Tom array
$arr[“roger”] = “duktig”;
$arr[2] = 45;
$arr[] = “NP”; // Index: högsta numeriska
               // index + 1
```

Iterator över vektorn:

```
foreach($arr as $key => $value) {
   echo $key.”:”.$value.”\n”;
}
```

Funktioner

```
<?php
function sum($arg_1, $arg_2, $arg_3)
{
   echo “Example function\n”.
   $retval = $arg_1+$arg_2+$arg_3;
   return $retval;
}
?>

Summan är: <?php echo sum(2,5,8); ?>
```

Biblioteksfunktioner

Stor mängd inbyggda biblioteksfunktioner.

Betoning på webbrelaterade uppgifter.

Audio	*Autentisering*	*Datum/tid*
Komprimering	*Kreditkorts-betalningar*	*Filhantering*
Teckenkodning	*Stränghantering*	*Bildmanipulering*
Mail	*Matematik*	*Nätverk*
Kryptering	*Databaser (SQL)*	*Processhantering*

Koppling till webbservern

Request

Parametrar accessas genom “superglobala” variabler, t.ex.

`$_GET`	Parametrar vid GET-anrop. Vektor.
`$_PUT`	Parametrar vid PUT-anrop. Vektor.
`$_COOKIE`	Cookies. Vektor.

Response

HTML-kod skrivs till standard output.

En elektronisk shoppinglista

Fallstudie:

Implementera en webbaserad tjänst för att hantera en familjs gemensamma shoppinglista.

- Lista över dagligvaror som saknas i hemmet.
- Enkel inläggning av varor från Internetanslutna enheter.
- Stöd både för att skapa listan och vid shoppingtillfället.

DEMO – shoppinglist.php

Design

Lagring

- Textfil med en rad per artikel.

Webbsida

Formulär med:

- Checkbox för varje artikel.
- Textinmatningsfält för ny artikel

Implementation

PHP-skript:

1. Läs in artiklar från fil.
2. Om checkbox ikryssad: tag bort motsvarande artikel.
3. Om textinmatningsfält ifyllt: lägg till artikel.

Ajax – Asynchronous JavaScript and XML

- Samling av relaterade tekniker för interaktiv webb.
- Förbättrar svarstider genom att HTML-sidor genereras lokalt mha JavaScript istället för på servern. Endast små datamängder överförs.
- Data/skript överförs asynkront i bakgrunden.
- Bygger på JavaScript och (ofta, men inte alltid) XML för överföring av data till/från servern.
- Sidans struktur kan manipuleras dynamiskt.

Exempel:

- Sidor med sökresultat. Endast själva sökresultatet behöver överföras/genereras och bytas ut på sidan.

angular + AJAX

```
protected get(url: string, parameters ?: URLSearchParams):
Observable<any> {
   if (!parameters) { parameters = new URLSearchParams(); }
   return this.http.get(url, {search: parameters})
       .map((response) => { return FetchJSON.extractData(response); })
       .catch((error) => { return FetchJSON.handleError(error); });
 }
 protected static extractData(response: Response): any {
  if (response.headers.get('Content-Type') &&
       response.headers.get('Content-
Type').toLowerCase().indexOf('application/json') >= 0) {
     try {
       let data = response.json();
```

REST

Representational State Transfer

- URI = resurs
- HTTP kommandon
  - POST, GET, PUT och DELETE.

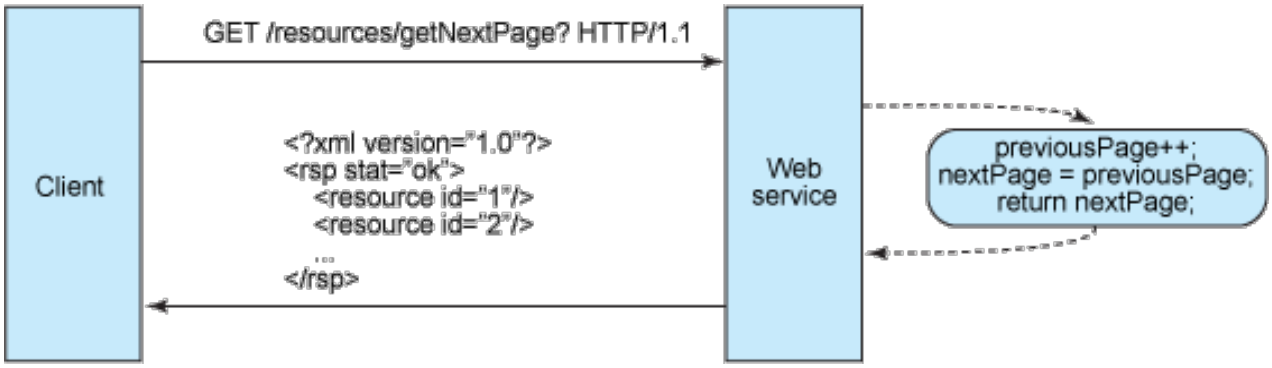
RESTful

- client-server
- tillståndslöst
- cachebart
- nivåer (möjliggör lastbalansering på server)
- enhetligt interface
  - resursidentifiering
  - manipulera resurs genom representation
  - självförklarande
  - hypermedia

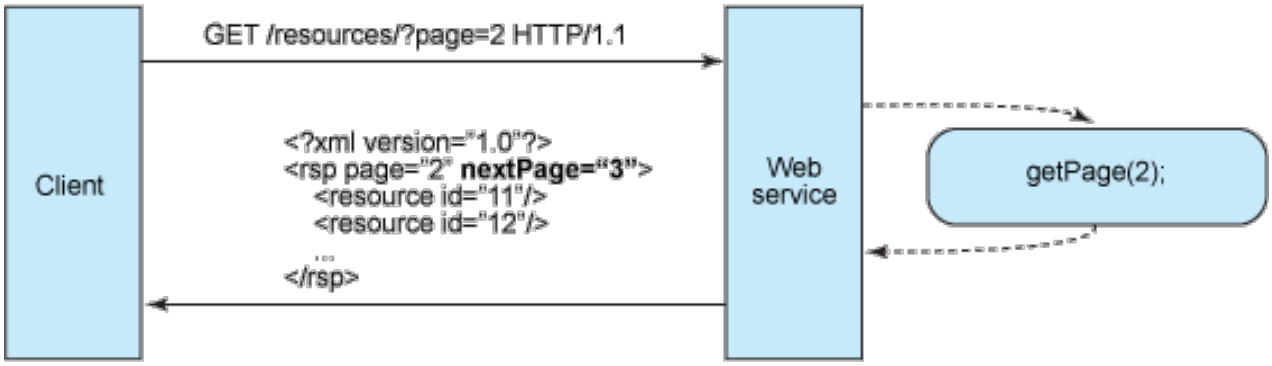
CRUD

- Operationer:
  - Create - POST
  - Read - GET
  - Update - PUT/PATCH
  - Delete - DELETE

Tillstånd


Client
GET /resources/getNextPage? HTTP/1.1
<?xml version="1.0"?>
<rsp stat="ok">
<resource id="1"/>
<resource id="2"/>
...
</rsp>
Web service
previousPage++;
nextPage = previousPage;
return nextPage;

Tillståndslös



Cache

- GET
  - If-Modified-Since header
- svar
  - 403 - Not Modified