Web Development using Java, JSP, and Web Services

Introduction to web development using XHTML and CSS

Lecture #1–#2

Material

During this course, we will cover server side web development with **JavaServer Pages**.

JSP is an exciting technology that lets developers dynamically generate XHTML, XML and other types of documents in response to Web client requests. It also creates an easy way to provide users with interfaces to web services.

Course literature

This course uses the following freely available resources as course literature:

- Lecture slides,
- Core Servlets and JavaServer Pages, 1st Ed. by Marty Hall,
- More Servlets and JavaServer Pages, 1st Ed. by Marty Hall,
- Thinking in Java, 3rd Ed. by Bruce Eckel and
- any extra links posted on the web site.

You are encouraged to support the authors by buying the printed versions of the books. All the resources listed above are used with permission.

Staff

This course has one lecturer (P-O) and one TA (Emanuel), apart from myself (Lars) as today's lecturer. You may get help via e-mail or IRC chat, as well as visiting during office hours. See the web page for office hours and when we will be in the chat room. E-mail questions related to the lectures to P-O, and questions on the assignments to Emanuel via either:

- `p-o+5dv093@cs.umu.se`,
- `emanuel+5dv093@cs.umu.se`

The chat room is `#5DV093` at the IRC server `irc.acc.umu.se`. You may also use MSN (p-o_5dv093@cs.umu.se and sn0wsmyrf@hotmail.com, respectively).

Important email information

Questions that may be of interest to all students attending the course will be sent to the course mailing list. The mailing list will send the mail to your CS mail address (`username@cs.umu.se`). Make sure that you either get into the habit of checking this address daily or forward it to your main account!

Support (`http://support.cs.umu.se/`) has information on how to handle your mail.

Course registration

To register for the course, send an email to the following address stating your name and Swedish "personnummer" — those with "Villkor 10" also need to provide proof that they may attend the course:

`yvonne@cs.umu.se`

Registering is **very** important, since we cannot grade your assignments unless you have registered. Do it as soon as possible!

Development environment

Your solutions to the assignments will be written in JSP.

You can choose to either solve the tasks on our department's servers, or install a package on your own computer. The latter can be used offline. Should you choose to do so, download it from the web page and install Java.

On Friday 13th June, there will be a group exercise aimed at getting to know the lab environment. Bring your laptops if you intend to install the package!

Distance education

As stated on the web page, it is possible to complete this course from other locations than Umeå. It is of course highly recommended to attend the lectures if possible, but it is **not** a formal requirement.

XHTML

XHTML (eXtensible HyperText Markup Language) is one of the core technologies that you must master in this course. Web browsers request XHTML documents from a server, interpret the response, and display the interpretation on–screen.

Today's coverage of XHTML

Unlike last year, the course no longer requires that students have taken a course in XHTML programming and CSS–based web design. Thus, we need to cover these topics here in this introduction. Experience tells us that most students who choose this course are at least somewhat familiar with (X)HTML, and we will keep a high pace.

If you study 5DV092, you will recognise some slides. The material covered here will be from a different perspective, and more focused on what you primarily need to know as server–side programmers, leaving actual web design to others.

XHTML is written in plain text

An XHTML document is simply a text file containing instructions to the web browser on what to display on-screen. We use a simple text editor (not a word processor such as Microsoft Word) to write the XHTML commands and save the file as plain text. It is common to use the either the suffix `.xhtml` or `.html` for (X)HTML documents.

Some browsers, Firefox included, treat `.xhtml` documents differently. This will become clear to you when you learn about how servers serve documents to clients, and later on when you understand the difference in mimetypes.

XHTML tags

The way that you as an author use XHTML is by placing content in **tags** (markup). Each tag has a semantic meaning, such as the tag in the following figure, which creates a paragraph (<p>) of class "foo" containing the word "hello". In XHTML, tags are case sensitive and must be written in lowercase.

```
<p class="foo">hello</p>
```

attribute value content

opening tag

closing tag

XHTML 1.0 versions

There are three versions of XHTML 1.0:

- XHTML 1.0 Strict – the most strict standard, allows no presentational information
- XHTML 1.0 Transitional – compatibility version, that allows some presentational information to ease the transition from HTML to XHTML
- XHTML 1.0 Frameset – used to divide an XHTML page in two or more frames

During this course, we will use XHTML 1.0 **Strict** combined with CSS.

All XHTML documents must, at the very least, contain the following:

- A **document type declaration** that identifies that the document is, in fact, XHTML and should be interpreted as such and
- the **html** tag as the one containing all other tags. These are placed in one of the following sections:
  - the **head** section, containing information such as the **title** (also required) of the page and other information that is not displayed in the main window of the web browser and
  - the **body**, containing the elements (tags and their content) that shall be displayed on screen.

DOCTYPEs

We have three versions of XHTML (strict, transitional and frameset) and we need to tell the web browser which one should be used for the current page. The following line, placed on the first line in the file, identifies the document as XHTML 1.0 Strict:

```
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The others are written in the same way, exchanging “strict” in both places for the version that should be used instead.

XHTML root tag

The root tag of an XHTML document should look as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="en" lang="en">
```

The “lang” parts set which language the page is written in. Swedish pages should use “sv” instead of “en”.

XHTML skeleton

The smallest possible legal XHTML 1.0 Strict document is similar to the one presented in the following example. Use it as a starting point for your upcoming assignments!

`example-xhtml-skeleton.xhtml`

XHTML tags

XHTML 1.0 contains the same tags as HTML 4.01, but requires valid XML syntax. Chiefly, this means that XHTML requires all tags to be **well-formed** (opened and closed and properly nested). A single erroneous tag invalidates the entire page. The following is valid XHTML, and specifies that we want to make some text in a paragraph (<p>) bold (<b>).

```
<p>Normal text. <b>This part is bold</b>.</p>
```

Use automated tools for validating your solutions whenever possible. W3C offers the most commonly used validator for such purposes.

XHTML tags continued

In XHTML, most tags are opened and later closed using a closing tag. Some tags are known as **singular tags**, where no content would be appropriate. The `img` tag for including images is one such tag. Singular tags are closed immediately, like so:

```
<img src="circle.gif" alt="A red circle" />
```

Note the slash at the end, immediately closing the tag.

XHTML tags continued

HTML attribute minimisation is not permitted in XHTML, so while this was valid HTML:

```
<option selected>...
```

it must be written this way in XHTML:

```
<option selected="selected">...
```

Keep this in mind if you study the source code of other pages and want to adapt it to your own use.

XHTML 1.0 Strict

HTML, XHTML's predecessor, allowed authors to mix presentation information and tags for structuring the content of a site. This lead to hard–to–maintain code.

Tags that were deemed entirely presentational, such as `font` and `center` are not permitted for use in XHTML 1.0 Strict. Instead, we must use XHTML in combination with CSS to specify the appearance of our pages.

A more complete example

The following example shows some of the basic building blocks that you use will use during the course. Study it carefully, and return to it if you get stuck.

The example is called `example-xhtml-basics.xhtml`.

You are encouraged to visit the following address to study XHTML on your own:

`http://www.w3schools.com/xhtml/`

XHTML tags

We will not present all XHTML tags here, since that takes too long and is beyond the scope of this course. You are recommended to visit the following site for a complete reference:

`http://www.w3schools.com/tags/`

CSS

CSS (Cascading Style Sheets) is an important concept in modern web design. However, we do not have time to cover this advanced technology here. Therefore, we devote some time to the **very basics** of CSS.

We will start by

CSS basics

CSS lets us apply rules that affect the way elements are presented on a web page. The basic format of CSS is:

```
selector {
  propertyA: valueA;
  propertyB: valueB;
  ...
}
```

The next slide contains a simple example.

CSS basic example

```
h1 {
  color: green;
}
```

The CSS above will change the color of all level one headers (<h1> elements) to green.

Linking to CSS in an XHTML file

There are many ways of joining CSS and XHTML. The one you should use is to write a line like the following in the `<head>` section of your page. This keeps the XHTML file perfectly clean from everything related to presentational information.

```
<link rel="stylesheet" href="foo.css"
type="text/css" />
```

As future server–side programmers and administrators, you will appreciate the reduced bandwidth costs that this technique provides: the CSS file can be cached on the client.

Basic CSS selectors

There are many selectors in CSS that allows the developer to match elements based on their relationship to other nodes, but we will for time reasons look only at the most basic ones — the others are left for independent study.

- Type selector,
- Ancestor selector,
- Class and ID selector and
- combined selectors.

Type selector

The example we just saw used the type selector: we specify a tag name, and the rules are applied to all tags of the specified type. It's as simple as that!

Type selectors are very common, since we usually want all headers of a given level to look the same, and so on. The concept is similar to document templates in Word or LaTeX.

Ancestor selector

The basic format of the ancestor selector is `S1 S2`, where both `S1` and `S2` are selectors. This expression will match only such elements `S2` that have `S1` as an ancestor. For example, this will match only such `<a>` tags that are somewhere in a `<li>` (in English: a hyperlink contained in a list item):

```
li a { ...rules here... }
```

The list of ancestors may be of any length.

Class selector

We can for almost all elements in XHTML specify a class attribute, such as:

```
<h1 class="important">Important header</h1>
<p>Unimportant text</p>
<p class="important">Important text</p>
```

To match only such tags that are of the class `important`, we can write:

```
.important { ...rules here... }
```

ID selector

Tags in XHTML can be given an ID, that is, a unique identifier. Many elements can belong to the same class, but only one element can have a given ID:

```
<h1 id="foo">Header with ID</h1>
```

To match the only object that has an ID of `foo`, we write the following CSS selector:

```
#foo { ...rules here... }
```

Combining selectors

Selectors can be combined — for instance, the following will match **only paragraphs** of the class `foo`. Headers of the same class will not be matched.

```
p.foo { ...rules here... }
```

Properties and values

We cannot possibly list all CSS properties and their corresponding values here, as there are too many of them. Suffice to say that we can change **everything** about how a page, and its elements, is displayed.

See `http://www.w3schools.com/css/css_reference.asp` for a complete reference.

Container layout

XHTML has support for two main types of containers: `<div>` and `<span>`. The basic principle of CSS–based web design is to put related content in a container, and use CSS to place them on the page, giving them the desired look (background colour, fonts and so forth).

Use the following resource to learn the basics of CSS–based web design:

`http://www.w3schools.com/css/css_positioning.asp`

Container example

A small example shows how containers may be used. Download the source and play around with it on your own, to see what effects you can create. Also, sneek a peek at 5DV092's lectures on CSS design if you wish. The example is called:

`example-css-layout.xhtml`

XHTML Forms

In this course, we will acquire user input using different controls: text areas, radio boxes or items from drop down menus, and so forth. Collectively, these controls are known as forms. Generally, one or more of these controls are placed in a form, including a button for submitting the data to the server (where it is parsed and some action takes place).

All related controls must be placed in an enclosing `<form>` tag. There may be several unrelated forms on one page.

<form>

The <form> tag requires an attribute called action, which specifies the name of the resource (server side program) to which the data in the form shall be sent. For instance, if we have a page called sendEmail.jsp our form that uses it should be declared as such:

```
<form action="sendEmail.jsp">
...
</form>
```

Upon submitting this form, the data is sent to the specified web page for processing.

<form> continued

Form data is sent to the resource specified by `action` using either a method called GET or one called POST. The default is GET — if we want to specify which method should be used explicitly, we can do so in the `<form>` declaration:

```
<form action="sendEmail.jsp" method="post">
...
</form>
```

We will cover the differences between the two methods later.

<input>

User data can be entered via the <input> tag. As the attribute type, we specify what kind of input control we want to display: button, checkbox, file, hidden, image, password, radio, reset, submit or text. The default is text, meaning a box where a single line of text can be entered.

The submit type is special, since when clicked, it will make the web browser submit the data in the form to the form's action target.

<input> continued

One should always use the `name` attribute for controls, as it will be sent along with the data in the form. It will thus be easy to parse. An example:

```
<form action="sendEmail.jsp" id="email">
  <input type="text" name="email_address" />
  <input type="submit" value="Send e-mail" />
</form>
```

The code above will create a form with a single line input text field and a button with the text "Send e-mail". When clicked, it will send what the user entered in the text field to `sendEmail.jsp` and it will be easy to see that it was in the "email_address" field, since we gave it a name.

<fieldset>

To group related controls in a graphical box, we use the <fieldset> tag. It does not effect the data in the form, it is merely a graphical hint to your users that the controls are related.

Use the <legend> tag in the fieldset to show a title.

<select>

The <select> tag creates a drop down menu, where the user can select a single choice. The choices are listed as <option> tags within the <select> tag:

```
<select name="gender">
  <option value="female">Female</option>
  <option value="male">Male</option>
</select>
```

Accessibility: <label>

Most forms have labels in addition to the controls, and these labels should be marked up as such. Give each control a unique ID, and mark the label text with the label tag. In the label tag, specify the "for" attribute and let its value equal the chosen unique ID, as such:

```
<label for="email">Your email addr:</label>
<input type="text" name="mailaddr" id="email" />
```

Note that "name" does not have to equal "id", but it the common practise.

A large example form

See `example-form.xhtml` for an example of many of the things that a form can do.

Difference between `GET` and `POST`

`GET`:

- General "retrieve resource from web server" command
- Parameters (data) are stored in the query string
  - Links and bookmarks to query strings containing parameters can be made
  - Search engines can index the results of such links

Difference between `GET` and `POST`

`POST`:

- Used for letting users submit content
- Parameters (data) are stored in request

The general rule is: if the data is such that the user has entered it himself, and it should update a resource (i.e. database), POST should be used.

If the parameters only select what type of values that the user wants to read from a resource, or some other read–only operation, GET should be used.

Best practises — Forms

An entire lecture will be devoted to best practises in general, but the following basic ones are directly related to forms:

- include the name of the form in the names of the controls – it makes it easier to keep track of multiple forms on a single page,
- pick a naming convention with variables, and stick to it – some platforms ignore case, others (such as JSP) are case sensitive,
- avoid Swedish and other non-English characters in variable names.

<table>

Tables in XHTML consist of a number of **table rows**, containing a mix of **table headers** and **table data**. While not required to be valid XHTML, one should always give a table summary so that blind users may opt to skip listening to its contents (`summary` attribute of the `<table>` tag).

Table data (`<td>`) and headers (`<th>`) must be placed in a table row (`<tr>`).

<table> continued

To let a cell span over several columns, we can use the `colspan` attribute of the `<td>` tag. For instance, if we have a table with four columns and we want a cell to span the first three, we would write:

```
<td colspan="3">data</td>
```

`<table>` continued

Let's look at some examples of tables!

The file `example-tables.xhtml` shows a common use for tables.

The schedule page for this course is another good example of a table.

<table> continued

Tables have been abused more than any other tag in HTML – it has been used for layout purposes. This is incorrect, one should use CSS for positioning and layout. We will study how to do this during the next couple of lectures.

Tables are for **tabular data** only!

Very large tables

When presenting very large tables, one should separate the table into the sections **thead**, **tfoot** and **tbody** – in that order! The web browser can then render the header (typically column names) and the footer (typically a sum of the entire table) before it is done downloading all the data in the table body.

Note, however, that browser support for this is quite poor and that this technique is not used often in practise. Rather, data is split up in "pages" of a sensible size and parameters sent using GET is used to display only the requested data.

Accessible tables

We've already touched on the subject of how we should avoid using tables for layout purposes. We are now going to focus on what we need to do to ensure that tables containing tabular data can be accessible to the visually impaired, and to help search engines index the data correctly.

Even if we just use the `<th>` tag to display table headers, we have already done more than most web designers do!

<th> improvements

How do we know if a table header is the header for a row or a column in the table? We should always use the `scope` attribute for this reason (`scope` can be either `row` or `col`):

```
<table>
  <tr>
    <th scope="row">header</th>
    <td>cell</td>
  </tr>
</table>
```

<caption>

In addition to the `summary` attribute of tables, we should also always provide a caption that describes what data the table contains, to do so we use the `<caption>` tag within the table.

Accessible table example

To see all of the above tips put together, look at the file `example-table.xhtml`.

Summary

We have briefly looked at some basic XHTML combined with CSS. CSS–based web design is a huge topic, and students unfamiliar with it are recommended to take a course on it.

In particular, we have studied XHTML Forms and Tables and seen an example of how we can build a form and table easily.

Next time, we study Web Technologies and the basics of server–side web development in general.