Even More Javascript & XML

CS174

Chris Pollett

Oct 15, 2008.

Outline

- Moving Elements
- Visibility
- Dynamic Content
- Timers
- Introduction to XML

Moving Elements

- We have previously talked about absolute and relative positioning.
- Today we'll consider this in the context of moving elements around the screen.
- Consider the following Javascript function:

```
function moveIt(id, newTop, newLeft)
{
    myStyle = document.getElementById(id).style;
    myStyle.top = newTop + "px"; /* notice how CSS properties are
       properties of the style object*/
    myStyle.left = newLeft + "px";
}
```

- This could be used to move an element to a specific location on the screen.

Element Visibility, Colors and Fonts

- You can control the visibility of an XHTML element using the CSS property.

```
<div id="test" style="visibility: hidden">hi there</div>
<input type="button" onclick='show("test")' />
<script type="text/javascript">
    function show(id)
    {
      myStyle = document.getElementById(id).style;
      myStyle.visibility = "visible";
    }
</script>
```

- In a similar fashion one can change other CSS properties in response to events. For instance, if you like you could change the color or font type or size.

Dynamic Content

- It is sometimes useful to be able to give help information when a person hovers over a form element. This can be achieved as follows:

```
<script type="text/javascript" >
        msgs = ["<p>mesg0</p>", "<p>mesg1</p>"]
       function showMessage(evt, num)
       {
          box = document.getElementById("adviceBox");
          box.innerHTML = msgs[num];
          myStyle =box.style;
          myStyle.position = "absolute";
          myStyle.top = evt.clientY +10 +"px";
          myStyle.left =  evt.clientX +10 +"px";
          myStyle.visibility = "visible";
       }
     function hideMessage()
    {
          myStyle = document.getElementById("adviceBox").style;
          myStyle.visibility = "hidden";
    }
</script>
<div id="adviceBox" style="visibility:hidden" />
<input type="text" onmouseover="showMessage(event, 1)" onmouseout="hideMessage()" />
```

Timers

- It is sometimes useful to update the contents of a page every so many milliseconds.
- Javascript support this by using either the functions setInterval or setTimeout.
- For example,

```
setTimeout(myCallback, repeatTimeInMilliSec);
// myCallback is the Javascript function you would like
   called
```

- To stop the timer you can use clearTimeout/clearInterval.

Introduction to XML

- Recall that HTML was originally specified as an SGML (Standard Generalized Markup Language) doctype.
- HTML only provides limited semantic information about a document. You can tell if something is in <h1> tags that is probably important, but not much else.
- Starting in 1998 a stripped down version of SGML called XML (extensible markup language) was developed to make it easier to create new tag-based mark-up languages where the tags can be used to carry whatever semantic information is desired.

The Syntax of XML

- A new XML language can be specified in one of two ways:
  - Give a DTD (Document Type Definition) -- this is closer to the SGML way of specifying languages.
  - Give an XML schema -- unlike DTDs such schemas are also XML documents so can easily be parse with XMLParsers. Further schemas can be more detailed.
- In both a DTD and a schema, one specifies:
  - What tag elements exist in the language.
  - What subelements or data a given element is allowed to contain.
  - What attributes an element has and what their values can be.
- Tags are case-sensitive in XML, and every tag must have a close tag, although the abbreviation <element /> works as an implicit close tag.

XML Document Structure and Entities

- Let's look at XML document:

```
<patient type="out of state">
    <![CDATA[This is data that will not be parsed by the XML parser even if it have
    tags in it like this: <tag>]]>
      <name><first>John</first><last>Smith</last></name>
      <insurerID>&kaiser;</insurerID>
</patient>
```

- Two files are usually associated with such a document:
  - one specifies what tags will be used with this document.
  - another specifies how those tags should be styled.
- One can abbreviate parts of a document using entities. For example, &kaiser; above might abbreviate some string of numbers.
- Associated with any XML document is a *document entity*. This can be thought of as the root of the tree associated with the tags of the document. This entity thus can represent the whole document or the parts of it which do not change.
- Entities can also be used to reference binary data such as images.

Document Type Definitions

- A DTD is used to specify:
  1. what tags will be used in a document .
  2. what kind of data or subtags these tags can hold.
  3. what attributes the tags can have and what values for these attributes are legal.
  4. what entities can occur in the document .
- DTDs can be *internal* (occur in the XML document itself) or *external* (occur in some separate file).
- A DTD consists of a sequence of declarations enclosed in the block of a DOCTYPE markup declaration.
- Items 1-4 above correspond to the declaration of type <!ELEMENT …> (1 and 2), <!ATTLIST …>, and <!ENTITY …>
- There is also a tag called <!NOTATION …>which is sometimes used.

<!ELEMENT ..>

- Basic syntax of the tag is:

  <!ELEMENT element_name (list of names of child elements) SYSTEM Location NDATA NotationName > .

- Some examples:

  <!ELEMENT memo (from, to, date) > <![CDATA[SYSTEM and NDATA don't have to used]>

  <!ELEMENT dept_script SYSTEM "dept.php" NDATA "php" >
  <!NOTATION php SYSTEM "/usr/bin/php" >

  <!ELEMENT person(parent+, age, spouse?, sibling*) > <![CDATA[+ is one than one, ? is optional, and * is 0 or more]>

  <!ELEMENT element_name (#PCDATA) > ![CDATA[pcdata = parsable character data can also use EMPTY for no sub-tags or character data or ANY if you want to allow everything]>

<!ATTLIST ..>

- The general way to specify a list of attributes is:

<!ATTLIST element_name

attribute_name_1 attribute_type [default_value] … attribute_name_n attribute_type [default_value] >

- For example,

<!ATTLIST airplane places CDATA “4”>

<!ATTLIST airplane engine_type CDATA #REQUIRED><![CDATA[must have the field]>

<!ATTLIST airplane price CDATA #IMPLIED><![CDATA[no default value is given]>

<!ATTLIST airplane manufacturer CDATA #FIXED “cessna”><![CDATA[all instances must have the same value]>

<!ENTITY ..>

- Entities come in two flavors:
  - General entities that can be referenced anywhere in an XML document
  - Parameter entities which can be referenced only in markup declarations
- The general form of an entity declaration is:

  <!ENTITY [%] entity_name “entity value” >

  <![CDATA[% is used when it is a parameter entity]>
- For example,

  <!ENTITY cp “Chris Pollett” >

  <!ENTITY cool_pic SYSTEM “/usr/local/cool_pic.jpg”>

Example DTD

<?xml version = “1.0” encoding =“utf-8” ?>
<!-- plane.dtd fragment -->
<!ELEMENT planes_for_sale (ad+)>
<!ELEMENT ad (year, make, model, color, price?, seller) >
…
<!ELEMENT seller (#PCDATA)>

<!ATTLIST seller phone CDATA #REQUIRED>

<!ENTITY c “cessna” >