Web Technologies — Week 6

Mikheil Rukhaia

International Black Sea University,
Tbilisi, Georgia

November 12, 2015

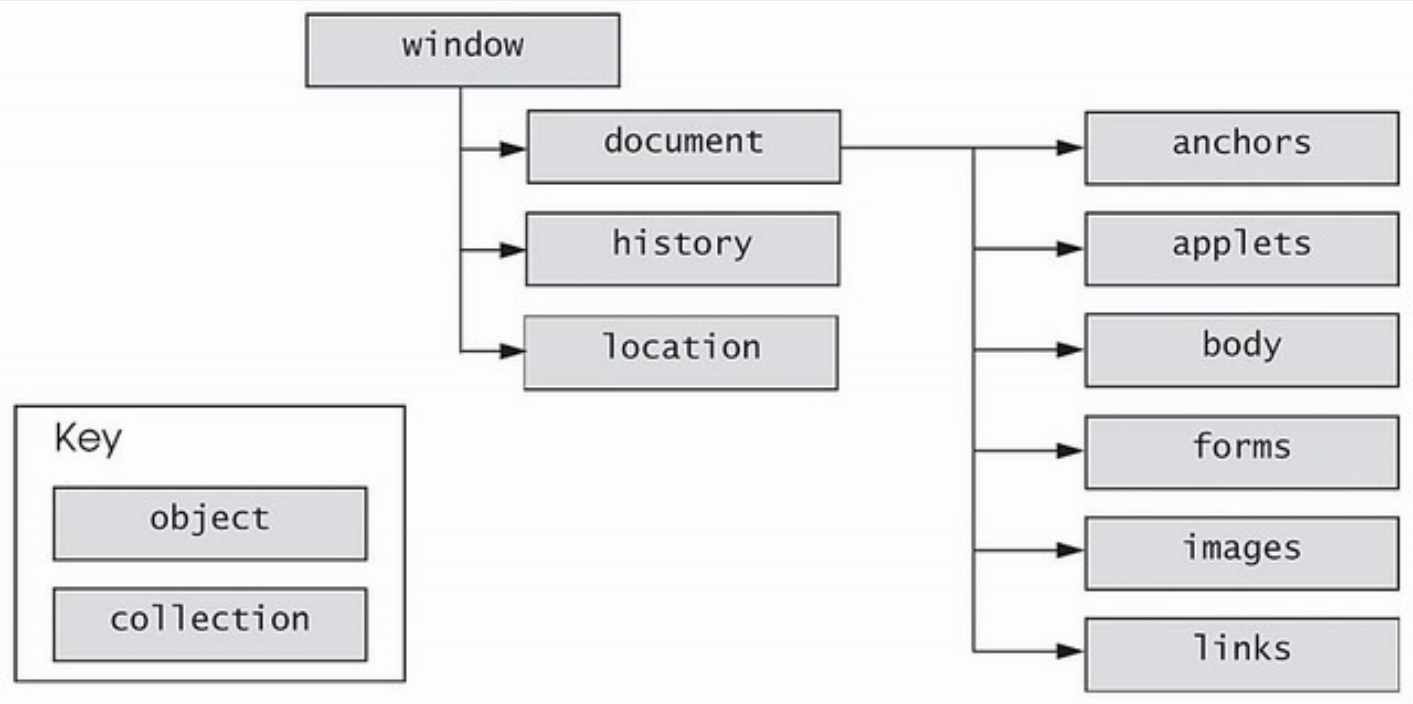
Outline

Document Object Model

Introduction

- Document Object Model, DOM, is a collection of properties and methods to access content of HTML files.
- DOM is not part of JavaScript, it is general Application Programming Interface, API, for all programming languages.
- The way browsers implement DOM is different, but the result of using it is the same.

Objects



- To access elements or their attributes use the full path to it, e.g. `window.location = document.links[0].href`

Objects (ctd.)

- Objects have properties and methods.
- Properties tell something about objects (correspond to attributes of elements).
- Methods perform some actions (e.g. `submit()`, `reset()`).

The `document` object

- Properties:
  - `lastModified` – the date is sent by the web server (if not, either current date or 1 January 1970).
  - `referrer` – the URL of page users came from. Empty if there is no referrer.
  - `title` – the content of `<title>`.
- Methods:
  - `getElementById("id")` – returns only one element or null.
  - `getElementsByTagName("name")` – returns array of the elements or null.
  - `getElementsByClassName("class")` – returns array of the elements or null.
  - `write("string")` – adds text or elements into a document.
  - `writeln("string")` – same as `write`, but adds newline character at the end.

HTML Elements

- Contents of HTML elements can be changed using `innerHTML` property.

- Example:

```
<p id="p"> a paragraph </p>
<script>
document.write("this line is added after p");
document.getElementById("p").innerHTML =
  "replacing the text inside p";
</script>
```

Form elements

- Each form element has `elements` collection.
- To access the from element use path to it.
  (e.g. `document.forms[0].elements[0]`)
- It is easier to name the elements and access them via its name.
  (e.g. `document.aForm.anElement`)

Properties of form elements

Property	Applies to	Purpose
`name`	All elements	Access the name attribute
`type`	All elements	Returns type of the control
`value`	All elements	Access the value attribute of the element or content of a text input
`form`	All elements	Returns a reference to the containing form
`disabled`	All except hidden ones	Disable/enable the element
`checked`	Checkboxes, radio buttons	Check/uncheck the element
`length`	Select boxes	Number of selectable options
`selectedIndex`	Select boxes	Index number of the selected item

Methods of form elements

Method	Applies to	Purpose
`blur()`	All except hidden	Takes focus away from currently active element to next in tabbing order
`focus()`	All except hidden	Gives focus to the element
`click()`	All except text	Simulates the mouse click
`select()`	Text elements except hidden	Selects the text in the element

The `images` collection

- `images` is a collection of all images of the document.
- Images can be accessed similarly to form elements using their index or name.
- Properties include all attributes of `<img>` element and `complete` property, which indicates whether an image has loaded successfully.
- Example:

```
<img src="one.gif" name="img1"
 onmouseover=
   "document.images.img1.src='two.gif';"
 onmouseout=
   "document.images.img1.src='one.gif';"
/>
```

JavaScript Syntax

Adding to HTML

- JavaScript is added to HTML pages inside `<script type="text/javascript">` element.
- The `type` attribute can be omitted, but this may cause problems in some cases (e.g. user has changed the default scripting language in his browser, etc.).
- To embed external `.js` file, use `<script type="text/javascript" src="URL" />`
- If script is specified inside the `head`, then it executes when event triggers it.
- If script is specified inside the `body`, then it executes as the page loads.

Comments

- comments in JavaScript can be written in two ways.
- Use `// comment` to write one-line comments.
- Use `/* comment */` to write multi-line comments.
- It is a good practice do document the code with small comments.

Variables

- Variables are used to store information.
- To declare a variable, give it a name and assign a value.
- Later you can use the variable by calling its name.
- Example:

```
name = "Mikheil Rukhaia"
document.write(name)
i = 1
url = document.links[i].href
window.location = url
```

Variables (ctd.)

- Variable names are case-sensitive.
- Variable names must begin with a letter or the underscore character.
- Try to use descriptive names for your variables.
- Local variables are those defined inside functions and global variables are those defined outside functions.
- Local variables in different functions can have the same name.
- Avoid giving global and local variables the same name as this might create errors.

Operators

- Arithmetic operators:

Symbol	Description	Example `x=10`	Result
+	Addition	`x+5`	15
−	Subtraction	`x-2`	8
*	Multiplication	`x*3`	30
/	Division	`x/2`	5
%	Division remainder	`x%3`	1
++	Increment	`x++`	11
−−	Decrement	`x--`	9

- Concatenation operator: + is also used as a concatenation operator for strings.

Operators (ctd.)

- Assignment operators:

Symbol	Example	Equivalent
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

- Logical operators:

Symbol	Name	Description
&&	And	True iff both operands are true.
||	Or	False iff both operands are false.
!	Not	True if operand is false.

Operators (ctd.)

- Comparison operators:

Symbol	Description	Example
==	Equal	`1==2` returns false
!=	Not equal	`1!=2` returns true
>	Grater than	`1>2` returns false
<	Less than	`1<2` returns true
>=	Grater than or equal	`1>=2` returns false
<=	Less than or equal	`1<=2` returns true

Functions

- Function is a collection of code, that is executed when an event fires or a call to that function is made.
- Functions are defined using the `function` keyword followed by function name, argument list and body code.
- Function body code is written between curly braces.
- Use `return` keyword to return the result of the function.
- Example:

```
function area(width, height) {
  return width * height
}
```

Functions (ctd.)

- To call a function, use its name and specify the arguments.

- Example:

```
Area: <span id="result"> </span>
<form name="frmArea" action="">
 Width: <input type="text" name="width"/>
 Height: <input type="text" name="height"/>
 <input type="button" value="Calculate"
  onclick="document.getElementById('result')
   .innerHTML = area(
       document.frmArea.width.value,
       document.frmArea.height.value)" />
</form>
```

Conditional statements

- Conditional statements allow to take different actions depending upon different statements.

- Conditional statement has the form:

```
if (cond1) {
  // code to be executed if cond1 is true
} else if (cond2) {
  /* code to be executed if cond1 is false
     and cond2 is true */
} else {
  /* code to be executed if both
     conditions are false */
}
```

- There can be arbitrary number of `else if` blocks, but at most one `else` block.

Looping

- Loops are used to execute block of code a number of times.
- while loop runs the same block of code while or until a condition is true.
- do while loop works in a similar manner but runs at least once.
- for loop runs the same block of code a specified number of times.
- All these loops are equivalent, they can simulate each other.

Looping (ctd.)

- Syntax of while loop:

```
while (condition) {
  // code to be executed
}
```

- Syntax of do while loop:

```
do {
  //code to be executed
} while (condition)
```

- Syntax of for loop:

```
for (counter; condition; change_counter) {
  // code to be executed
}
```

Looping (ctd.)

- Loops, if not defined properly, can run forever.
- Use `break` statement to exit loop conditionally.
- `break` statement can be used to exit infinite loops as well.
- Example:

```
i = 0
while (true) {
  document.write(i); i++;
  if (i == 100) break;
}
for (j = 0; j < i; j++,i--) {
  document.write("i="+i+", j="+j)
  if (i/10 == j) break;
}
```

Built-in Objects

Events

Event	Purpose
`onload`	Document has finished loading.
`onunload`	Document is unloaded, or removed, from a window or frameset.
`onclick`	Button on pointing device has been clicked over the element.
`ondblclick`	Button on pointing device has been double-clicked over the element.
`onmousedown`	Button on pointing device is held down over the element.
`onmouseup`	Button on pointing device has been released over the element.
`onmouseover`	Pointer has been moved onto the element.
`onmousemove`	Pointer has been moved while over the element.

Events (ctd.)

Event	Purpose
`onmouseout`	Pointer has been moved off the element.
`onkeypress`	A keyboard key is pressed over the element.
`onkeydown`	A keyboard key is held down over an element.
`onkeyup`	A keyboard key is released over an element.
`onfocus`	Element receives focus in either ways.
`onblur`	Element loses focus.
`onsubmit`	A form is submitted.
`onreset`	A form is reset.
`onselect`	User selects some text in a text field.
`onchange`	A control loses input focus and its value has been changed since gaining focus.

Strings

Method	Purpose
`anchor(name)`	Creates an anchor element.
`link(URL)`	Creates a link in the document.
`big()`	Displays text as if in `<big>`.
`small()`	Displays text as if in `<small>`.
`bold()`	Displays text as if in `<bold>`.
`italics()`	Displays text as if in `<i>`.
`strike()`	Displays text as if in `<strike>`.
`fixed()`	Displays text as if in `<tt>`.
`sub()`	Displays text as if in `<sub>`.
`sup()`	Displays text as if in `<sup>`.
`fontcolor(color)`	Displays text in the specified color.
`fontsize(size)`	Displays text in the specified size.
`toLowerCase()`	Converts a string to lowercase.
`toUpperCase()`	Converts a string to uppercase.

Strings (ctd.)

Method	Purpose
`length`	The number of characters in a string.
`charAt(index)`	The character at a specified position.
`indexOf(str)`	The position of the first occurrence of the specified string.
`indexOf(str,ind)`	Same as `indexOf()`, but search begins from the specified index.
`lastIndexOf(str)`	Same as `indexOf()`, but runs from right to left.
`lastIndexOf(str,ind)`	Same as `indexOf()`, but runs from right to left.
`substr(start,length)`	Returns the specified characters.
`substring(start,end)`	Returns the specified characters between the start and end index points.

Date

- The `Date()` constructor can take the following parameters to create date objects:
  - Milliseconds – value since 01/01/1970.
  - DateString – something like "May 8, 2014".
  - Tuple year, month, and day.
  - Tuple years, months, days, hours, minutes, seconds, and milliseconds.
- Note that January corresponds to 0, and Sunday too.
- When dates are subtracted, the result is returned in milliseconds.

Date (ctd.)

Method	Purpose
`get/setDay()`	The day from 0 to 6 (0=Sunday, 1=Monday, and so on).
`get/setMonth()`	The month from 0 to 11 (0=January, 1=February, and so on).
`get/setFullYear()`	The year in four digits.
`get/setYear()`	The year in two digits from 0 to 99.
`get/setHours()`	The hours from 0 to 23.
`get/setMinutes()`	The minutes from 0 to 59.
`get/setSeconds()`	The seconds from 0 to 59.
`toGMTString()`	Converts the object to a string, set to GMT time zone.
`toString()`	Converts the object to a string.

Math

- Built-in `Math` object provides everything to work with numbers.
- It has properties for mathematical constants.
- It has methods representing mathematical functions.
- Example:

```
pi = Math.PI;
Math.round(pi);
Math.exp(Math.random());
```

Array

- Arrays can be defined by directly specifying its elements, or by specifying its length.
- Indices of array start from 0.
- To access an element in array refer to it via array name and element index.
- Example:

```
gender = new Array("Man", "Woman");
document.write(gender[0]);
cars = new Array(3);
cars[0] = "BMW";
cars[1] = "Mercedes";
cars[2] = "Toyota";
```

Array (ctd.)

Method	Purpose
`length`	number of element in the array.
`concat()`	Joins two or more arrays to create one new one.
`join(sep)`	Joins all of the elements of an array together, separated by the specified character.
`reverse()`	Returns the reversed array.
`slice()`	Returns a specified part of the array.
`sort()`	Returns a sorted array.

Window

Property	Purpose
`closed`	True if a window has been closed.
`defaultStatus`	The default message displayed in status bar.
`frames`	An array containing references to all named child frames in the current window.
`history`	Contains URLs visited from that window.
`location`	The URL of the current window.
`name`	The windows name.
`status`	Temporary message displayed in the status bar.
`statusbar`	Visibility of the status bar.
`toolbar`	Visibility of the scrollbar.
`top`	A reference for the topmost browser window if several windows are open on the desktop.

Window (ctd.)

Method	Purpose
`alert()`	An alert box containing a message.
`blur()`	Removes focus from the current window.
`close()`	Closes the window.
`confirm()`	A dialog box asking users to confirm an action with either OK or Cancel.
`focus()`	Gives focus to window and brings it to the top.
`moveBy(hpx,vpx)`	Moves the window by the specified number of pixels in relation to current coordinates.
`moveTo(x,y)`	Moves the top left of the window to x,y point.
`open(URL, name)`	Opens a new browser window.
`print()`	Prints the content of the current window.
`prompt()`	Creates a dialog box for the user to enter an input.

Form Validation and Enhancement

Form validation

- JavaScript can be used to check the user input in forms before sending the form to the server.
- One way to achieve this is to check the full form `onsubmit`.
- It is also possible to check the content of a single control `onblur`.
- In general, it is not a good idea to validate forms via JavaScript as it might be disabled in the browser.
- In practice, it is best to check the input via JavaScript and validate it after submission on the server via PHP.

Form validation (ctd.)

- The function that validate the form `onsubmit` must return a boolean value.
- If the value is `true` submission must proceed, otherwise stop.
- Example:

```
<form name="frm" onsubmit="return validate()">
 <!-- form controls -->
</form>
<script type="text/javascript">
 function validate() {
   if (/*everything is fine*/) return true;
   else return false;
 }
</script>
```

Form validation (ctd.)

- Example:

```
<form name="frm" action="login.php"
                 onsubmit="return validate()">
 Username <input type="text" name="un" />
 <br />
 Password <input type="password" name="pwd1" />
 <br />
 Confirm  <input type="password" name="pwd2" />
 <br />
 <input type="submit" value="LogIn"/>
</form>
```

Form validation (ctd.)

- (ctd.)

```
<script type="text/javascript">
 function validate() {
  if (frm.un.value.length == 0) {
   alert("Empty username");
   return false;
  }
  else if (frm.pwd1.value != frm.pwd2.value) {
   alert("Passwords do not match")
   return false;
  }
  else return true;
 }
</script>
```

Form enhancement

- There are numerous ways that do not validate the form, but help users to fill in the form with less effort.
- Here we consider several examples, like focus on the elements, auto-tabbing, disabling controls, and case conversion.
- You can do more with JavaScript depending on your needs and imagination.

Form enhancement (ctd.)

- It is very helpful to give focus on the first element when form loads.
- To do so, give the focus `onload` in the `body` tag.
- Example:
  ```
  <body onload="document.frm.un.focus()">
  ```
- Analogous way you can give the focus to an element having incorrect value from the `validation()` function.
- Example:
  ```
  if (frm.un.value.length == 0) {
    alert("Empty username");
    frm.un.focus();
    return false;  }
  ```

Form enhancement (ctd.)

- If input format is known, like date or credit card number, etc., auto-tabbing can be implemented using `focus()`.

- Example:

```
<form name="date">
 <input name="day" size="3" maxlength="2"
  onkeyup="if (this.value.length >= 2)
   this.form.month.focus();"/>
 <input name="month" size="3" maxlength="2"
  onkeyup="if(this.value.length>=2)
   this.form.year.focus();" />
 <input name="year" size="5" maxlength="4"
  onkeyup="if(this.value.length>=4)
   this.form.submit.focus();" />
 <input type="submit" value="OK" />
</form>
```

Form enhancement (ctd.)

- Sometimes you want to disable control until some other controls are filled in.
- For example, you disable the submit button when page loads and enable after the form is filled in correctly.
- More dynamic example can be to enable additional text input on users choice.

Form enhancement (ctd.)

- Example:

```
<body onload="document.frm.txt.disabled=true;">
<form name="frm">
 <input type="radio" name="rd" value=""
  onclick="handle(this.value);" /> None <br />
 <input type="radio" name="rd" value="other"
  onclick="handle(this.value);" /> Specify:
 <input type="text" name="txt" />
</form>
<script type="text/javascript">
 function handle(val) {
  if (val=="other")
   document.frm.txt.disabled = false;
  else document.frm.txt.disabled = true;
 }
</script>
```

Form enhancement (ctd.)

- Sometimes it is useful to convert user input to uppercase or lowercase letters, since JavaScript is case sensitive.
- There are several ways to do so, like `onblur`, `onchange`, and `onkeyup`.
- Example:

```
<input type="text"
 onkeyup="this.value=this.value.toUpperCase()"/>
```

Laboratory Work

Exercises

- Create GEL to EUR converter web form (let user enter the preferred rate).
- Improve the form by adding select boxes to convert between different currencies.
- Write a form which takes a number and calculates its factorial if it is positive. Otherwise returns 0.
  Recall: $n! = 1 * 2 * \cdots * n$.
- Put things together: add JavaScript to the student registration form, that checks whether student filled-in the form correctly.
- Use several enhancement techniques in the form by your will.

Discussion?!