More Javascript

CS174

Chris Pollett

Sep 24, 2007.

Outline

- More Arrays
- Functions
- Constructors/Methods
- Pattern Matching
- Execution Environment
- Document Object Model

More Arrays

- The Array object in Javascript has several useful methods for manipulating arrays.
  - join --- can be used to make a string out of an array.

    var names = new Array(“Mary”, “Murray”, “Max”);

    var nstring = names.join(“:”);
  - concat --- can be used to add elements to an existing array.

    var a = [1, 2, 3];

    a.concat(4, 5);
  - slice(i,j) --- return a sublist from the i to the j element.
  - push, pop, shift, unshift --- stack-like operations

Functions

- A Javascript function definition consists of two parts:
  - a function header consisting of
    - the keyword function
    - an identifier
    - a parenthesized list of parameters
  - a compound statement

  For example,

```
function swap(i, j, a)
{
    var tmp=a[i]; /* explicitly defined variables have scope within the function
                     if I had declared the variable implicitly it would have global scope */
    a[i] = a[j]; a[j] = tmp;
}
```

  This function could be called with a syntax like:

```
swap(10, 5, b);
```

- A return statement can be used to return a value from a function.
- Functions are objects so can be assigned to variables.

```
var b = swap;
```

- The definition of a function does not need to list its arguments. One can obtain a list of arguments using the argument subobject of a Function.

```
function swap()
{ var i = this.arguments[0], j=this.arguments[1], a=this.arguments[2];
  //same code as before
}
```

Constructors

- Javascript constructors are special methods that create and initialize the properties for newly created objects. For example,

```
function car(new_make, new_model, new_year)
{
        this.make = new_make;
        this.model = new_model;
        this.year = new_year;
}
```

I could then create an object with

```
my_car = new car("Ford", "Contour SVT", "2000");
```

Methods

- To create methods, I can do things like the following way to create a display method to pretty print cars:

```
function display_car()
{
    document.write("Make:", this.make, "<br />");
    document.write("Model:", this.model, "<br />");
    document.write("Year:", this.year, "<br />");
}
function car(/*same as before*/)
{ //same as before
this.display = display_car;}
```

- The drawback of this is that each time we create a new car we have a separate pointer to display_car. Instead, we can use the prototype property of the car function. This will only create one pointer.

```
function car(/*same as before*/)
{ //same as before}
car.prototype.display = display_car;
```

More on the Prototype Property

- When Javascript looks up a property of a class, it:
  - first looks up property in the instance,
  - if it is not found then in looks in the prototype object of the identify of the given function object,
  - if it is not found there it looks at the prototype property of the class Object.

Pattern Matching

- Frequently in Javascript we will be manipulating strings using pattern matching, so it is useful to know what facilities are available for this.
- Javascript pattern matching is modeled on Perl's regular expressions.
- A pattern is an expression between / /.
- In such a pattern normal characters match themselves.
- In addition to normal characters there are special characters: \ | ( ) [] {} ^ $ * + ? .
- As an example:

```
var str = "Rabbits are furry";
var position = str.search(/bits/); /* returns position of first
    occurrence */
```

Pattern Special Characters

- . -- matches any single character. So /snow./ would match snows and snowy
- () -- used to control order of matching /(ab)*/ matches ab, abab, but not aab
- [] -- logical or of a group of patterns
  - [azf] matches an “a”, a “z”, or an “f”.
  - [a-d] - matches the range a,b,c,d
- ^ -- acts as negation or as a start of string anchor. So [^abc] is any character other than a, b, c; /^abc/ matches abc at the start of a string.
- $ -- acts acts an anchor to end of a string. /abc$/ matches abc at the end of a string.
- \ -- either can be used to escape characters (so \. would match a period), or for one of a list of special escape patterns such as \r \t, \n, \f or
  - \d -- match a digit,
  - \D -- match anything other than a digit
  - \w match a word character (alphanumeric)
  - \W match a not a word character.
  - \s match a single whitespace character
  - \S match a single nonwhitespace character

Yet more special characters

- * -- matches 0 or more occurrences of the pattern. For example /x*/ would match x, xx, xxx …
- + -- matches 1 or more occurrences of the pattern
- ? -- matches 0 or 1 occurrences of the pattern
- {} -- can be used to match exactly k occurrences:

  /yx{5}z/ matches yxxxxxz

Pattern Modifiers

- i -- makes the pattern case insensitive. For example,

  /Apple/i would match APPLE, aPple and apple.

- x -- allows whitespace to occur in the pattern

- g-- means do globally - we'll see this more on the next slide

More Pattern Methods

- replace -- replace the matched pattern with the given replacement string.

  var str="Fred, Freddie, Frederica";

  str.replace(/Fre/g, "Boyd");

  //notice use g to replace all occurences. The variable $1 is assigned by the match to the first matched substring, $2 to the second, etc.

- match -- returns an array of the pattern matched results

  var str= "3 and 4";

  var matches = str.match(/\d/g); //returns [3, 4]

- split -- splits a string into an array of substrings according to the pattern delimiter

  var str="grapes:apples:oranges"

  var fruit = str.split(":"); // [grapes, apples,oranges]

Execution Environment

- When a browser displays an XHTML document in a window, it will set up a Javascript Window object to represent information about the window.
- All Javascript variables are properties of some object. So implicitly defined globals on a page can be viewed as properties of the Window object.
- You can have more than one Window object if the browser opens more than one window.
- Every Window object has a property **document** which is the Document object representing the XHTML document it displays.
- Every document objects has a **forms** array each element of which represents a form (Form object) on the document.
- Each Form object has an elements array as a property which contains an array of form elements for the buttons , menus, etc on it.
- Document objects also have property arrays for anchors, links, images, and applets.

Document Object Model

- The Document Object Model is a model developed in the 90s for how the contents of an XHTML or XML document should be modeled by Javascript or other language's objects.
- Typically a document is modeled as a tree with roughly one node for each element type.
- DOM also described methods for getting, updating, and modifying elements.
- DOM Level 2 is the currently adopted standard, although there has been been development of DOM level 3.