Internet Software Technologies
JavaScript – part two

IMCNE

A.A. 2008/09

Gabriele Cecchetti

OBJECTS

Is JavaScript object-oriented? (1/2)

- It has objects which can contain data and methods that act upon that data.
- Objects can contain other objects.
- It does not have classes, but it does have constructors which do what classes do, including acting as containers for class variables and methods.
- It does not have class-oriented inheritance, but it does have prototype-oriented inheritance.

Is JavaScript object-oriented? (2/2)

- The two main ways of building up object systems are by inheritance (is-a) and by aggregation (has-a). JavaScript does both, but its dynamic nature allows it to excel at aggregation.
- Objects cannot have private variables and private methods: all members are public.
- But **JavaScript objects *can* have private variables and private methods.**

Hashtables

- To create a new hashtable and to assign it to a local variable, just write:
  ```
  var myHashtable=();
  ```
- Then to add, replace or retreve elements in the hashtable you can write:
  ```
  myHashtable[name]=“A name”
  ```
- The same operation can be written in a more efficient notation:
  ```
  myHashtable.name=“A name”;
  ```
- The dot notation can be used when the subscript is a string constant in the form of legal identifier (not use keywords for names!).

Hashtables === Objects !!

- In JavaScript, Objects and Hashtables are the same thing, so:
  ```
  var myHashtable = {};
  ```
  and
  ```
  var myHashtable = new Object();
  ```
  are equivalent.

JavaScript is fundamentally about *objects*

- Arrays are objects.
- Functions are objects.
- Objects are objects.
- So what are objects?
  - Objects are collections of name-value pairs.
    - The names are strings, and
    - the values are strings, numbers, booleans, and objects (including arrays and functions)
- Objects are usually implemented as hashtables so values can be retrieved quickly.

Object Constructors

- Objects can be produced by *constructors*, which are functions which initialize objects.
- Constructors provide the features that classes provide in other languages, including static variables and methods.
- Syntax:

```
function Container(param) {
   this.member = param;
}
```

Object instance

- So, if we construct a new object

  `var myContainer = new Container('abc');`

  then

  - `myContainer` is the reference to the object istance, and
  - `myContainer.member` is a object member which contains `'abc'`.

Public members of an object

- The members of an object are all *public* members.
- Any function can access, modify, or delete those members, or add new members.
- In our example:

  `myContainer.member`

  is a public member.

Example: defining an Object

```
function Car (b,m, c, e) {
  this.brand= b;    // public member
  this.model=m;     // public member
  this.color=c;     // public member
  this.engine=e;    // public member
  this.display=fun(b);   // public method
  }
function fun(s) {
  return “This is a “ + s
}
```

Example: creating Object istances

```
Ferrari2008 = new Car(“Ferrari”,”F2009”, “red”);
Ferrari2009 = new Car(“Ferrari”,”F2004”, “red”,
                        {“v10”,10});
Ferrari3000 = new Car(“Ferrari);
FerrariEmpty = new Car(); // members are undefined!
```

Example: initializing an object

- In the object literal notation, an object description is a set of comma-separated name/value pairs inside curly braces.
  - The names can be identifiers or strings followed by a colon, they cannot use reserved JavaScript keywords.
  - The values can be literals or expressions of any type.
- Example:

```
var FerrariX = { brand: "Ferrari", color: 'red'}
```

Example: add new members to an object

- Look at this example:

```
FerrariX.driver = "Michael Shumacher"
```

- Driver was not present in the constructor.
- JavaScript let you to add members at any time by assignment.

Public methods

- If a value is a function, we can consider it a *method*.
- When a method of an object is invoked, the this variable is set to the object.
- The method can then access the instance variables through the this variable.

Private members of an object (1/3)

- *Private* members are made by the constructor.
- Ordinary `vars` and parameters of the constructor becomes the private members.
- Example:

```
function Container(param) {
   this.member = param; // public member
   var secret = 3;      // private var
   var that = this;     // private var
}
```

- `param`, `secret` and `that` are 3 private instance variables. They are attached to the object,

Private members of an object (2/3)

- `param`, `secret` and `that` are attached to the object but they are not accessible to the outside, nor are they accessible to the object's own public methods.
- They are accessible to private methods.

Private members of an object (3/3)

- Private methods are inner functions of the constructor. Example:

```
function Container(param) {
    function dec() {                // private method
          if (secret > 0) {
          secret -= 1;
          return true;
          } else { return false;
          }
    }
    this.member = param;
    var secret = 3;
    var that = this;  // private parameter
}                     // which references the object
```

Privileged methods of an object (1/2)

- **Private methods cannot be called by public methods.**
- To make private methods useful, we need to introduce a **privileged method**.
- A *privileged* method is able to access the private variables and methods, and is itself accessible to the public methods and the outside.
- It is possible to delete or replace a privileged method, but it is not possible to alter it, or to force it to give up its secrets.

Privileged methods of an object (2/2)

- Privileged methods are assigned with `this` within the constructor.
- Example if we extend the previous example:

```
function Container(param) {
    function dec() { … }
    …
    var that = this;
    this.service = function () { // privileged method
          if (dec()) {
                return that.member; // return private
          } else {                  // member value
                return null;
          }
    };
}
```

What will happen running that program ?

- Calling `myContainer.service()` will return `'abc'` the first three times it is called.
- After that, it will return `null`.
- `service` calls the private `dec` method which accesses the private `secret` variable.
- `Service`
  - is available to other objects and methods, but
  - it does not allow direct access to the private members.

Closures

- This pattern of public, private, and privileged members is possible because JavaScript has *closures*.
- What this means is that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned.
- **Private and privileged members can only be made when an object is constructed.**
- Public members can be added at any time.

Objects properties can be enumerated (1/2)

- Syntax:

```
for (variable in object) {
   statements
}
```

- Example:

```
for (var n in FerrariX) {
   if (FerrariX.hasOwnProperty(n)) {
        document.writeln("<p>" + n + ": "
             + myHashtable[n] + "</p>");
   }
}
```

Objects properties can be enumerated (1/2)

- The result will be:

```
<p>brand: Ferrari</p>
<p>color: red</p>
<p>driver: Michael Shumacher</p>
```

A simple way to manipulate object members: with

- Syntax:

```
with (object) {
   statements
}
```

- Example

```
with (Ferrari) {
  driver = ”Kimi Raikkonen”;
}
```

  Updates the `driver` member.

Putting members in a new object

- There are two main ways of putting members in a new object:
  - In the constructor (as seen some slides before)
  - In the prototype (next slide)
    → This technique is usually used to add public methods.

Dynamic extension of object properties

- The prototype mechanism is used for inheritance.
- To add a method to all objects made by a constructor, add a function to the constructor's `prototype` with the syntax:

  ***objectName.propertyName = value*;**
- Examples:

```
FerrariX.wheels=6 // add this member only to
                     FerrariX istance while
FerrariX.prototype.wheels=4
  // add the member to the constructor and
  its inherited by all Childs.
```

Object operators

- **new***:*
  - *objectName* = **new** *objectType* (*param1* [, *param2*] …[,*paramn*])
- **delete**:
  - **delete** *objectName*;
  - **delete** *objectName.property*;
  - **delete** *objectName[index]*
- **in**:
  - *propertyNameorNumber* **in** *ObjectName*
- **instanceof**:
  - *objectName* **instanceof** *objectType*
- **this**:
  - **this**[.property]

Object Hierarchies

- Built-in objects
  - `String`
  - `RegExp,`
  - `Array,`
  - `Date,`
  - `Math,`
  - `Boolean,`
  - `Number, and`
  - `Function`
- Client objects (referred to HTML elements)
  - Client objects follow tree hierarchy of the HTML DOM.

BUILT-IN OBJECTS

Built-in Objects

- `Array`
- `Boolean`
- `Date`
- `String`

- `Number`
- `Math`
- `RegExp`
- `Function`

Arrays - Introduction

- Arrays in JavaScript are also hashtable objects.
- This makes them very well suited to sparse array applications.
- When you construct an array, you do not need to declare a size.
- Arrays grow automatically, much like Java vectors.
- The values are located by a key, not by an offset.
- This makes JavaScript arrays very convenient to use, but not well suited for applications in numerical analysis.

Arrays elements and index

- Arrays are not typed.
- They can contain numbers, strings, booleans, objects, functions, and arrays.
- You can mix strings and numbers and objects in the same array.
- You can use arrays as general nested sequences, much as s-expressions.
- The first index in an array is usually zero.

Arrays and Objects

- The main difference between objects and arrays is the `length` property.
- The `length` property is always 1 larger than the largest integer key in the array.
- There are two ways to make a new array:

```
var myArray = [];
var myArray = new Array();
```

Array length

- When a new item is added to an array and the subscript is an integer that is larger than the current value of length, then the length is changed to the subscript plus one.
- This is a convenience feature that makes it easy to use a for loop to go through the elements of an array.

Arrays

- *Syntax*:

  arrayObjectName = **new Array**(*element0*, *element1*, ..., *elementN*)
  arrayObjectName = **new Array**(*arrayLength*)
  arrayObjectName = [*element0*,*element1*,*element2*,….,*elementN*];

- Example:

```
colors = new Array(3);
colors[0] = ”red”;
colors[1] = ”yellow”;
colors[2] = ”green”;
```

Arrays literal notation

- It's similar to that for objects.

```
myList = ['oats', 'peas', 'beans', 'barley'];
emptyArray = [];
month_lengths = [31, 28, 31, 30, 31, 30, 31, 31, 30,
  31, 30, 31];
slides = [
  {url: 'slide0001.html', title: 'Looking Ahead'},
  {url: 'slide0008.html', title: 'Forecast'},
  {url: 'slide0021.html', title: 'Summary'}
];
```

- A new item can be added to an array by assignment.

```
a[i + j] = f(a[i], a[j]);
```

Array example (1/2)

```
<html><head>
<title>Array Object example</title>
<script type="text/JavaScript">
<!--
var myarray1= new
    Array("1","2","3");
printArray(myarray1,"Starting
    array");
function add() {
    myarray1[myarray1.length]
    =document.forms[0].text1.value;
    printArray(myarray1,"adding ");
}
function join() {
    myarray2=myarray1.join("+");
    printArray(myarray2,"joining ");
}
```

```
function reverse() {
    myarray1.reverse();
    printArray(myarray1,"reversing");
}
function shift() {
    myarray1.shift();
    printArray(myarray1,"shifting ");
}
function printArray(arr,comment) {
    for (var i=0; i<arr.length; i++)
    document.writeln(arr[i]);
    document.writeln("<br>"+
    comment);
}
// --></script></head>
```

Array example (2/2)

```
<body>
<form>
Insert new element
<input type="text" name="text1"><br>
<input type="button“ value="Insert" onClick="add()"> <br>
<input type="button" value="Join with + separator“ onClick="join()"><br>
<input type="button" value="Reverse" onClick="reverse()"><br>
<input type="button" value="Shift " onClick="shift()"><br>
</body>
</html>
```

Boolean object

- The Boolean object is an object wrapper for a boolean (true or false) value.
- You can explicitly define a Boolean via:

```
new Boolean([value])
```

- where the Value is the initial value of the Boolean object. Note that:
  - the value is converted to a boolean value, if necessary
  - If value is: not specified, 0, -0, null, false, NaN, undefined, or the empty string (""), the object is set to false. All other values, including any object or the string "false", create an object with a value of true.
- Example:

```
var guess = new Boolean(false) //false value
var guess = new Boolean(0) //false value
var guess = new Boolean(true) //true value
var guess = new Boolean("whatever") //true value
```

Boolean object: properties

`constructor` Specifies the function that created the object's prototype.

`prototype` Allows you to define properties on the Boolean that is shared by all Boolean objects.

Boolean object: methods

toString() Returns a string specifying the value of the Boolean, in this case, "true" or "false."

valueOf() Returns the primitive value of a Boolean object.

Date Objects

- There are fours ways of instantiating a date object:

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds,
  milliseconds) //most parameters here are optional.
```

- Not specifying causes 0 to be passed in.
- Here are a few examples of instantiating a date:

```
today = new Date()
birthday = new Date("March 11, 1985 09:25:00")
birthday = new Date(85,2,11)
birthday = new Date(85,2,11,9,25,0)
```

Data object property

`constructor` Returns a reference to the Date function that created the object.

`prototype` Allows you to add properties and methods to the object

Date object methods: to get Date

getFullYear()	Returns year in full 4 digit format (ie: 2004).
getYear()	Returns the year. Deprecated
getMonth()	Returns the month. (Range is 0-11)!
getDate()	Returns the day of the month (Range is 1-31)
getDay()	Returns the day of the week (Range is 0-6). 0=Sunday, 1=Monday, etc.
getHours()	Returns the hour (Range is 0-23).
getMinutes()	Returns the minutes. (Range is 0-59).
getSeconds()	Returns the seconds. (Range is 0-59).
getMilliseconds()	Returns the milliseconds. (Range is 0-999).
getTime()	Returns the number of milliseconds between 1/1/1970 (GMT) and the current Date object.
getTimezoneOffset()	Returns the offset between GMT and local time, in minutes.

Date object methods: to get UTC Date

getUTCFullYear()	Returns the full 4 digit year in Universal time.
getUTCMonth()	Returns the month in Universal time.
getUTCDate()	Returns the day of the month in Universal time.
getUTCDay()	Returns the day of the week in Universal time.
getUTCHours()	Returns the hour in Universal time.
getUTCMinutes()	Returns the minutes in Universal time.
getUTCSeconds()	Returns the seconds in Universal time.
getUTCMilliseconds()	Returns the milliseconds in Universal time.

Date Object methods: to set Date

setFullYear(year, [month], [day])	Sets the year (4 digit) of the Date object.
setYear(year)	Sets the year of the Date object. Deprecated
setMonth(month, [day])	Sets the month [0-11].
setDate(day_of_month)	Sets the day of the month [1-31].
setHours(hours, [minutes], [seconds], [millisec])	Sets the hour [0-23].
setMinutes(minutes, [seconds], [millisec])	Sets the minutes [0-59].
setSeconds(seconds, [millisec])	Sets the seconds [0-59].
setMilliseconds(milli)	Sets the milliseconds [0-999].
setTime(milli)	Sets the value of the Date object in terms of milliseconds elapsed since 1/1/1970 GMT.

Date Object methods: to set UTC Date

setUTCFullYear(year, [month], [day])	Sets the year of the Date object in Universal time.
setUTCMonth(month, [day])	Sets the month.
setUTCDate(day_of_month)	Sets the day of the month.
setUTCHours(hours, [minutes], [seconds], [millisec])	Sets the hours.
setUTCMinutes(minutes, [seconds], [millisec])	Sets the minutes.
setUTCSeconds(seconds, [millisec])	Sets the seconds.
setUTCMilliseconds(milli)	Sets the milliseconds.

Date Objects methods: converting Date

toGMTString() Converts a date to a string, using the GMT conventions. Deprecated.

toLocaleString() Converts a date to a string, using current locale conventions.

toLocaleDateString() Returns the date portion of the Date as a string, using current locale conventions.

toLocaleTimeString() Returns the time portion of the Date as a string, using current locale conventions.

toString() Converts a Date to human-readable string.

toUTCString() Converts a Date to human-readable string, in Universal time.

Data objects: other methods

parse(datestring) Returns the number of milliseconds in a date string since 1/1/1970. (datestring: a string containing the date/time to be parsed).

UTC(year, month, [day], [hours], [minutes], [seconds], [milli]) Returns the number of milliseconds in a date string since 1/1/1970, Universal time.

valueOf() Converts a Date to milliseconds. Same as getTime();.

Date objects: examples (1/2)

- To write out today's date for example, you would do:

```
<script type="text/javascript">
  var mydate= new Date()
  var theyear=mydate.getFullYear()
  var themonth=mydate.getMonth()+1
  var thetoday=mydate.getDate()

  document.write("Today's date is: ")
  document.write(theyear+"/"+themonth+"/"+thetoday)
</script>
```

- Output:

```
Today's date is: 2008/12/11
```

Date objects: examples (2/2)

- Calculate day of week of a date

```
birthday = new Date(1978,2,11)
weekday = birthday.getDay()
alert(weekday)
//alerts 6, or Saturday
```

- Set date to be a future date

```
var today=new Date()
today.setDate(today.getDate()+3)
//today now is set to be 3 days into the future
```

String Object

- The String object of JavaScript allows you to perform manipulations on a stored piece of text, such as extracting a substring, searching for the occurrence of a certain character within it etc.
- ***Syntax*** *for creating a String object:*

```
var myStr = new String(string);
```

String Object properties

- The string object has three properties and several methods:

  `constructor` : A reference to the function that created the object

  `length` : Returns the number of characters in a string

  `prototype` : Allows you to add properties and methods to the object
- Example:

```
<script type="text/javascript">
    var txt="Hello World!";
    document.write(txt.length);
</script>
```

- The output of the code above will be:

```
12
```

String Object methods (1/2)

anchor(name)	Returns the string with the tag <A name="name"> surrounding it.
big()	Returns the string with the tag <BIG> surrounding it.
blink()	Returns the string with the tag <BLINK> surrounding it.
bold()	Returns the string with the tag <B> surrounding it.
fixed()	Returns the string with the tag <TT> surrounding it.
fontcolor(color)	Returns the string with the tag <FONT color="color"> surround. it.
fontsize(size)	Returns the string with the tag <FONT size="size"> surround. it.
italics()	Returns the string with the tag <I> surrounding it.
link(url)	Returns the string with the tag <A href="url"> surrounding it.
small()	Returns the string with the tag <SMALL> surrounding it.
strike()	Returns the string with the tag <STRIKE> surrounding it.
sub()	Returns the string with the tag <SUB> surrounding it.
sup()	Returns the string with the tag <SUP> surrounding it.

String methods: charAt and charCodeAt(x)

charAt(x) Returns the character at the "x" position within the string.

- Example: in the string "Hello world!", we will return the character at position 1:

```
<script type="text/javascript">
  var str="Hello world!";
  document.write(str.charAt(1));
</script>
```

- The output of the code above will be:

```
e
```

charCodeAt(x) Returns the Unicode value of the character at position "x" within the string. Previous example using `charCodeAt(x)` is: `101`

String methods: concat(v1, v2,...)

concat(v1, v2,...) Combines one or more strings (arguments v1, v2 etc) into the existing one and returns the combined string. Original string is not modified.

- **Example**
- In the following example we will create two strings and show them as one using `concat()`:

```
<script type="text/javascript">
  var str1="Hello ";
  var str2="world!"; document.write(str1.concat(str2));
</script>
```

- The output of the code above will be:

```
Hello world!
```

String methods: fromCharCode(c1, c2,...)

fromCharCode(c1, c2,...) Returns a string created by using the specified sequence of Unicode values (arguments c1, c2 etc). Method of String object, not String instance.

String methods: indexOf(substr, [start])

indexOf(substr, [start]) **Searches and (if found) returns the index number of the searched character or substring within the string**. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is 0.

- **Example:** we will do different searches within a `"Hello world!"` string:

```
<script type="text/javascript">
  var str="Hello world!";
  document.write(str.indexOf("Hello") + " ");
  document.write(str.indexOf("World") + " ");
  document.write(str.indexOf("world")); </script>
```

- The output of the code above will be:

```
0 -1 6
```

String methods: lastIndexOf(substr, [start])

lastIndexOf(substr, [start]) **Searches and (if found) returns the index number of the searched character or substring within the string**. Searches the string from end to beginning. If not found, -1 is returned. "Start" is an optional argument specifying the position within string to begin the search. Default is string.length-1.

String methods: match(regexp)

match(regexp) Executes a **search for a match within a string based on a regular expression**. It returns an array of information or null if no match is found.

- Example: we will do different searches within a "Hello world!" string:

```
<script type="text/javascript">
  var str="Hello world!";
  document.write(str.match("world") + " ");
  document.write(str.match("World") + " ");
  document.write(str.match("worlld") + " ");
  document.write(str.match("world!")); </script>
```

- The output of the code above will be:

```
world null null world!
```

String methods: replace(regexp,replacetext)

replace(regexp, replacetext) **Searches and replaces the regular expression portion (match) with the replaced text instead**.

- **Example:** we will perform a global and case-insensitive match, and the word Microsoft will be replaced each time it is found, independent of upper and lower case characters:

```
<script type="text/javascript">
  str="Abc ABC abc abC";
  document.write(str.replace(/abc/gi, "CDE"));
</script>
```

- The output of the code above will be:

```
CDE CDE CDE CDE
```

String methods: search(regexp)

search(regexp) Tests for a **match in a string**. It returns the index of the match, or -1 if not found.

String methods: extracting string portions

slice(start, [end]) Returns a **substring of the string** based on the "start" and "end" index arguments, NOT including the "end" index itself. "End" is optional, and if none is specified, the slice includes all characters from "start" to end of string.

substring(from, [to]) Returns the **characters in a string** between "from" and "to" indexes, NOT including "to" inself. "To" is optional, and if omitted, up to the end of the string is assumed.

substr(start, [length]) Returns the **characters in a string** beginning at "start" and through the specified number of characters, "length". "Length" is optional, and if omitted, up to the end of the string is assumed.

Extracting string portions examples

- Example:

```
<script type="text/javascript">
  var str="Hello happy world!";
  document.write(str.slice(7,11) + " ");
  document.write(str.substring(7,12) + " ");
  document.write(str.substr(7,5));
</script>
```

- The output of the code above will be:

```
  Happy Happy Happy
```

String methods: split(delimiter, [limit])

split(delimiter, [limit]) Splits a string into many according to the specified delimiter, and returns an array containing each element. The optional "limit" is an integer that lets you specify the maximum number of elements to return.

- Example: we will split up a string in different ways:

```
<script type="text/javascript">
  var str="How are you doing today?";
  document.write(str.split(" ") + "<br>");
  document.write(str.split("") + "<br>");
  document.write(str.split(" ",3)); </script>
```

- The output of the code above will be:

```
How,are,you,doing,today?
H,o,w, ,a,r,e, ,y,o,u, ,d,o,i,n,g, ,t,o,d,a,y,?
How,are,you
```

String methods: changing case letters

toLowerCase() Returns the string with all of its characters converted to lowercase.

toUpperCase() Returns the string with all of its characters converted to uppercase.